TOSHIBA

32-Bit TX System RISC
TX39 Family

TMPR3927

TOSHIBA CORPORATION

MIPS16, application Specific Extensions and R3000A are a trademark of MIPS Technologies, Inc.

Table of Contents

Handling precautions

Handling Precautions

1. Using Toshiba Semiconductors Safely

TOSHIBA are continually working to improve the quality and the reliability of their products.

Nevertheless, semiconductor devices in general can malfunction or fail due to their inherent electrical sensitivity and vulnerability to physical stress. It is the responsibility of the buyer, when utilizing TOSHIBA products, to observe standards of safety, and to avoid situations in which a malfunction or failure of a TOSHIBA product could cause loss of human life, bodily injury or damage to property.

In developing your designs, please ensure that TOSHIBA products are used within specified operating ranges as set forth in the most recent products specifications. Also, please keep in mind the precautions and conditions set forth in the TOSHIBA Semiconductor Reliability Handbook.

2. Safety Precautions

This section lists important precautions which users of semiconductor devices (and anyone else) should observe in order to avoid injury and damage to property, and to ensure safe and correct use of devices.

Please be sure that you understand the meanings of the labels and the graphic symbol described below before you move on to the detailed descriptions of the precautions.

[Explanation of labels]

⚠DANGER	Indicates an imminently hazardous situation which will result in death or serious injury if you do not follow instructions.
⚠WARNING	Indicates a potentially hazardous situation which could result in death or serious injury if you do not follow instructions.
⚠CAUTION	Indicates a potentially hazardous situation which if not avoided, may result in minor injury or moderate injury.

[Explanation of graphic symbol]

Graphic symbol	Meaning
	Indicates that caution is required (laser beam is dangerous to eyes).

2.1 General Precautions regarding Semiconductor Devices

⚠CAUTION
Do not use devices under conditions exceeding their absolute maximum ratings (e.g. current, voltage, power dissipation or temperature). This may cause the device to break down, degrade its performance, or cause it to catch fire or explode resulting in injury. Do not insert devices in the wrong orientation. Make sure that the positive and negative terminals of power supplies are connected correctly. Otherwise the rated maximum current or power dissipation may be exceeded and the device may break down or undergo performance degradation, causing it to catch fire or explode and resulting in injury. When power to a device is on, do not touch the device's heat sink. Heat sinks become hot, so you may burn your hand. Do not touch the tips of device leads. Because some types of device have leads with pointed tips, you may prick your finger. When conducting any kind of evaluation, inspection or testing, be sure to connect the testing equipment's electrodes or probes to the pins of the device under test before powering it on. Otherwise, you may receive an electric shock causing injury. Before grounding an item of measuring equipment or a soldering iron, check that there is no electrical leakage from it. Electrical leakage may cause the device which you are testing or soldering to break down, or could give you an electric shock. Always wear protective glasses when cutting the leads of a device with clippers or a similar tool. If you do not, small bits of metal flying off the cut ends may damage your eyes.

2.2 Precautions Specific to Each Product Group

2.2.1 Optical semiconductor devices

⚠ DANGER

When a visible semiconductor laser is operating, do not look directly into the laser beam or look through the optical system. This is highly likely to impair vision, and in the worst case may cause blindness.
If it is necessary to examine the laser apparatus, for example to inspect its optical characteristics, always wear the appropriate type of laser protective glasses as stipulated by IEC standard IEC825-1.

⚠ WARNING

Ensure that the current flowing in an LED device does not exceed the device's maximum rated current.
This is particularly important for resin-packaged LED devices, as excessive current may cause the package resin to blow up, scattering resin fragments and causing injury.

When testing the dielectric strength of a photocoupler, use testing equipment which can shut off the supply voltage to the photocoupler. If you detect a leakage current of more than 100 μA, use the testing equipment to shut off the photocoupler's supply voltage; otherwise a large short-circuit current will flow continuously, and the device may break down or burst into flames, resulting in fire or injury.

When incorporating a visible semiconductor laser into a design, use the device's internal photodetector or a separate photodetector to stabilize the laser's radiant power so as to ensure that laser beams exceeding the laser's rated radiant power cannot be emitted.
If this stabilizing mechanism does not work and the rated radiant power is exceeded, the device may break down or the excessively powerful laser beams may cause injury.

2.2.2 Power devices

⚠ DANGER

Never touch a power device while it is powered on. Also, after turning off a power device, do not touch it until it has thoroughly discharged all remaining electrical charge.
Touching a power device while it is powered on or still charged could cause a severe electric shock, resulting in death or serious injury.

When conducting any kind of evaluation, inspection or testing, be sure to connect the testing equipment's electrodes or probes to the device under test before powering it on.
When you have finished, discharge any electrical charge remaining in the device.
Connecting the electrodes or probes of testing equipment to a device while it is powered on may result in electric shock, causing injury.

⚠WARNING

Do not use devices under conditions which exceed their absolute maximum ratings (current, voltage, power dissipation, temperature etc.).
This may cause the device to break down, causing a large short-circuit current to flow, which may in turn cause it to catch fire or explode, resulting in fire or injury.

Use a unit which can detect short-circuit currents and which will shut off the power supply if a short-circuit occurs.
If the power supply is not shut off, a large short-circuit current will flow continuously, which may in turn cause the device to catch fire or explode, resulting in fire or injury.

When designing a case for enclosing your system, consider how best to protect the user from shrapnel in the event of the device catching fire or exploding.
Flying shrapnel can cause injury.

When conducting any kind of evaluation, inspection or testing, always use protective safety tools such as a cover for the device.
Otherwise you may sustain injury caused by the device catching fire or exploding.

Make sure that all metal casings in your design are grounded to earth.
Even in modules where a device's electrodes and metal casing are insulated, capacitance in the module may cause the electrostatic potential in the casing to rise.
Dielectric breakdown may cause a high voltage to be applied to the casing, causing electric shock and injury to anyone touching it.

When designing the heat radiation and safety features of a system incorporating high-speed rectifiers, remember to take the device's forward and reverse losses into account.
The leakage current in these devices is greater than that in ordinary rectifiers; as a result, if a high-speed rectifier is used in an extreme environment (e.g. at high temperature or high voltage), its reverse loss may increase, causing thermal runaway to occur.
This may in turn cause the device to explode and scatter shrapnel, resulting in injury to the user.

A design should ensure that, except when the main circuit of the device is active, reverse bias is applied to the device gate while electricity is conducted to control circuits, so that the main circuit will become inactive.
Malfunction of the device may cause serious accidents or injuries.

⚠CAUTION

When conducting any kind of evaluation, inspection or testing, either wear protective gloves or wait until the device has cooled properly before handling it.
Devices become hot when they are operated. Even after the power has been turned off, the device will retain residual heat which may cause a burn to anyone touching it.

2.2.3 Bipolar ICs (for use in automobiles)

⚠CAUTION

If your design includes an inductive load such as a motor coil, incorporate diodes or similar devices into the design to prevent negative current from flowing in.
The load current generated by powering the device on and off may cause it to function erratically or to break down, which could in turn cause injury.

Ensure that the power supply to any device which incorporates protective functions is stable.
If the power supply is unstable, the device may operate erratically, preventing the protective functions from working correctly. If

protective functions fail, the device may break down causing injury to the user.

3. General Safety Precautions and Usage Considerations

This section is designed to help you gain a better understanding of semiconductor devices, so as to ensure the safety, quality and reliability of the devices which you incorporate into your designs.

3.1 From Incoming to Shipping

3.1.1 Electrostatic discharge (ESD)

When handling individual devices (which are not yet mounted on a printed circuit board), be sure that the environment is protected against electrostatic electricity. Operators should wear anti-static clothing, and containers and other objects which come into direct contact with devices should be made of anti-static materials and should be grounded to earth via an 0.5- to 1.0-MΩ protective resistor.

Please follow the precautions described below; this is particularly important for devices which are marked “Be careful of static.”.

(1) Work environment

- When humidity in the working environment decreases, the human body and other insulators can easily become charged with static electricity due to friction. Maintain the recommended humidity of 40% to 60% in the work environment, while also taking into account the fact that moisture-proof-packed products may absorb moisture after unpacking.
- Be sure that all equipment, jigs and tools in the working area are grounded to earth.
- Place a conductive mat over the floor of the work area, or take other appropriate measures, so that the floor surface is protected against static electricity and is grounded to earth. The surface resistivity should be 10^4 to 10^8 Ω/sq and the resistance between surface and ground, 7.5×10^5 to 10^8 Ω
- Cover the workbench surface also with a conductive mat (with a surface resistivity of 10^4 to 10^8 Ω/sq, for a resistance between surface and ground of 7.5×10^5 to 10^8 Ω) . The purpose of this is to disperse static electricity on the surface (through resistive components) and ground it to earth. Workbench surfaces must not be constructed of low-resistance metallic materials that allow rapid static discharge when a charged device touches them directly.
- Pay attention to the following points when using automatic equipment in your workplace:

  (a) When picking up ICs with a vacuum unit, use a conductive rubber fitting on the end of the pick-up wand to protect against electrostatic charge.

  (b) Minimize friction on IC package surfaces. If some rubbing is unavoidable due to the device’s mechanical structure, minimize the friction plane or use material with a small friction coefficient and low electrical resistance. Also, consider the use of an ionizer.

  (c) In sections which come into contact with device lead terminals, use a material which dissipates static electricity.

  (d) Ensure that no statically charged bodies (such as work clothes or the human body) touch the devices.

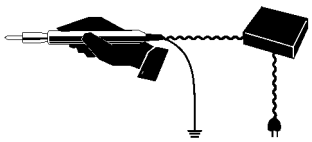
(e) Make sure that sections of the tape carrier which come into contact with installation devices or other electrical machinery are made of a low-resistance material.

(f) Make sure that jigs and tools used in the assembly process do not touch devices.

(g) In processes in which packages may retain an electrostatic charge, use an ionizer to neutralize the ions.

- Make sure that CRT displays in the working area are protected against static charge, for example by a VDT filter. As much as possible, avoid turning displays on and off. Doing so can cause electrostatic induction in devices.
- Keep track of charged potential in the working area by taking periodic measurements.
- Ensure that work chairs are protected by an anti-static textile cover and are grounded to the floor surface by a grounding chain. (Suggested resistance between the seat surface and grounding chain is 7.5×10^5 to $10^{12}\Omega$.)
- Install anti-static mats on storage shelf surfaces. (Suggested surface resistivity is 10^4 to 10^8 Ω/sq; suggested resistance between surface and ground is 7.5×10^5 to 10^8 Ω.)
- For transport and temporary storage of devices, use containers (boxes, jigs or bags) that are made of anti-static materials or materials which dissipate electrostatic charge.
- Make sure that cart surfaces which come into contact with device packaging are made of materials which will conduct static electricity, and verify that they are grounded to the floor surface via a grounding chain.
- In any location where the level of static electricity is to be closely controlled, the ground resistance level should be Class 3 or above. Use different ground wires for all items of equipment which may come into physical contact with devices.

(2) Operating environment



- Operators must wear anti-static clothing and conductive shoes (or a leg or heel strap).
- Operators must wear a wrist strap grounded to earth via a resistor of about 1 MΩ.
- Soldering irons must be grounded from iron tip to earth, and must be used only at low voltages (6 V to 24 V).
- If the tweezers you use are likely to touch the device terminals, use anti-static tweezers and in particular avoid metallic tweezers. If a charged device touches a low-resistance tool, rapid discharge can occur. When using vacuum tweezers, attach a conductive chucking pat to the tip, and connect it to a dedicated ground used especially for anti-static purposes (suggested resistance value: 10^4 to 10^8 Ω).
- Do not place devices or their containers near sources of strong electrical fields (such as above a CRT).

- When storing printed circuit boards which have devices mounted on them, use a board container or bag that is protected against static charge. To avoid the occurrence of static charge or discharge due to friction, keep the boards separate from one other and do not stack them directly on top of one another.

- Ensure, if possible, that any articles (such as clipboards) which are brought to any location where the level of static electricity must be closely controlled are constructed of anti-static materials.

- In cases where the human body comes into direct contact with a device, be sure to wear anti-static finger covers or gloves (suggested resistance value: 10^8 Ω or less).

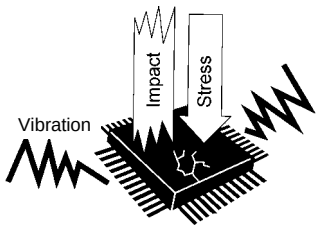
- Equipment safety covers installed near devices should have resistance ratings of 10^9 Ω or less.

- If a wrist strap cannot be used for some reason, and there is a possibility of imparting friction to devices, use an ionizer.

- The transport film used in TCP products is manufactured from materials in which static charges tend to build up. When using these products, install an ionizer to prevent the film from being charged with static electricity. Also, ensure that no static electricity will be applied to the product's copper foils by taking measures to prevent static occuring in the peripheral equipment.

3.1.2 Vibration, impact and stress

Handle devices and packaging materials with care. To avoid damage to devices, do not toss or drop packages. Ensure that devices are not subjected to mechanical vibration or shock during transportation. Ceramic package devices and devices in canister-type packages which have empty space inside them are subject to damage from vibration and shock because the bonding wires are secured only at their ends.



Plastic molded devices, on the other hand, have a relatively high level of resistance to vibration and mechanical shock because their bonding wires are enveloped and fixed in resin. However, when any device or package type is installed in target equipment, it is to some extent susceptible to wiring disconnections and other damage from vibration, shock and stressed solder junctions. Therefore when devices are incorporated into the design of equipment which will be subject to vibration, the structural design of the equipment must be thought out carefully.

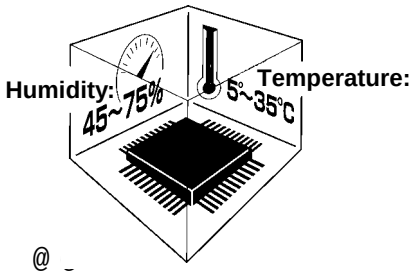
If a device is subjected to especially strong vibration, mechanical shock or stress, the package or the chip itself may crack. In products such as CCDs which incorporate window glass, this could cause surface flaws in the glass or cause the connection between the glass and the ceramic to separate.

Furthermore, it is known that stress applied to a semiconductor device through the package changes the resistance characteristics of the chip because of piezoelectric effects. In analog circuit design attention must be paid to the problem of package stress as well as to the dangers of vibration and shock as described above.

3.2 Storage

3.2.1 General storage

- Avoid storage locations where devices will be exposed to moisture or direct sunlight.
- Follow the instructions printed on the device cartons regarding transportation and storage.
- The storage area temperature should be kept within a temperature range of 5°C to 35°C, and relative humidity should be maintained at between 45% and 75%.
- Do not store devices in the presence of harmful (especially corrosive) gases, or in dusty conditions.



- Use storage areas where there is minimal temperature fluctuation. Rapid temperature changes can cause moisture to form on stored devices, resulting in lead oxidation or corrosion. As a result, the solderability of the leads will be degraded.
- When repacking devices, use anti-static containers.
- Do not allow external forces or loads to be applied to devices while they are in storage.
- If devices have been stored for more than two years, their electrical characteristics should be tested and their leads should be tested for ease of soldering before they are used.

3.2.2 Moisture-proof packing

Moisture-proof packing should be handled with care. The handling procedure specified for each packing type should be followed scrupulously. If the proper procedures are not followed, the quality and reliability of devices may be degraded. This section describes general precautions for handling moisture-proof packing. Since the details may differ from device to device, refer also to the relevant individual datasheets or databook.

(1) General precautions
Follow the instructions printed on the device cartons regarding transportation and storage.

- Do not drop or toss device packing. The laminated aluminum material in it can be rendered ineffective by rough handling.
- The storage area temperature should be kept within a temperature range of 5°C to 30°C, and relative humidity should be maintained at 90% (max). Use devices within 12 months of the date marked on the package seal.

- If the 12-month storage period has expired, or if the 30% humidity indicator shown in Figure 1 is pink when the packing is opened, it may be advisable, depending on the device and packing type, to back the devices at high temperature to remove any moisture. Please refer to the table below. After the pack has been opened, use the devices in a 5°C to 30°C. 60% RH environment and within the effective usage period listed on the moisture-proof package. If the effective usage period has expired, or if the packing has been stored in a high-humidity environment, bake the devices at high temperature.

Packing	Moisture removal
Tray	If the packing bears the “Heatproof” marking or indicates the maximum temperature which it can withstand, bake at 125°C for 20 hours. (Some devices require a different procedure.)
Tube	Transfer devices to trays bearing the “Heatproof” marking or indicating the temperature which they can withstand, or to aluminum tubes before baking at 125°C for 20 hours.
Tape	Deviced packed on tape cannot be baked and must be used within the effective usage period after unpacking, as specified on the packing.

- When baking devices, protect the devices from static electricity.

- Moisture indicators can detect the approximate humidity level at a standard temperature of 25°C. 6-point indicators and 3-point indicators are currently in use, but eventually all indicators will be 3-point indicators.

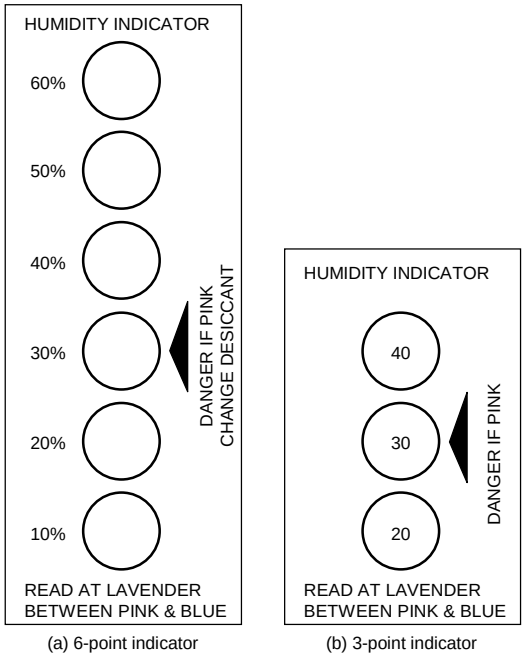


Figure 1 Humidity indicator

3.3 Design

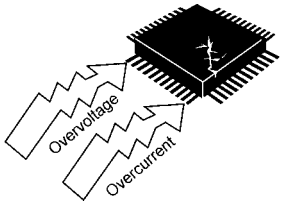
Care must be exercised in the design of electronic equipment to achieve the desired reliability. It is important not only to adhere to specifications concerning absolute maximum ratings and recommended operating conditions, it is also important to consider the overall environment in which equipment will be used, including factors such as the ambient temperature, transient noise and voltage and current surges, as well as mounting conditions which affect device reliability. This section describes some general precautions which you should observe when designing circuits and when mounting devices on printed circuit boards.

For more detailed information about each product family, refer to the relevant individual technical datasheets available from Toshiba.

3.3.1 Absolute maximum ratings

⚠CAUTION Do not use devices under conditions in which their absolute maximum ratings (e.g. current, voltage, power dissipation or temperature) will be exceeded. A device may break down or its performance may be degraded, causing it to catch fire or explode resulting in injury to the user.

The absolute maximum ratings are rated values which must not be exceeded during operation, even for an instant. Although absolute maximum ratings differ from product to product, they essentially concern the voltage and current at each pin, the allowable power dissipation, and the junction and storage temperatures.



If the voltage or current on any pin exceeds the absolute maximum rating, the device's internal circuitry can become degraded. In the worst case, heat generated in internal circuitry can fuse wiring or cause the semiconductor chip to break down.

If storage or operating temperatures exceed rated values, the package seal can deteriorate or the wires can become disconnected due to the differences between the thermal expansion coefficients of the materials from which the device is constructed.

3.3.2 Recommended operating conditions

The recommended operating conditions for each device are those necessary to guarantee that the device will operate as specified in the datasheet.
If greater reliability is required, derate the device's absolute maximum ratings for voltage, current, power and temperature before using it.

3.3.3 Derating

When incorporating a device into your design, reduce its rated absolute maximum voltage, current, power dissipation and operating temperature in order to ensure high reliability.
Since derating differs from application to application, refer to the technical datasheets available for the various devices used in your design.

3.3.4 Unused pins

If unused pins are left open, some devices can exhibit input instability problems, resulting in malfunctions such as abrupt increase in current flow. Similarly, if the unused output pins on a device are connected to the power supply pin, the ground pin or to other output pins, the IC may malfunction or break down.

Since the details regarding the handling of unused pins differ from device to device and from pin

to pin, please follow the instructions given in the relevant individual datasheets or databook.

CMOS logic IC inputs, for example, have extremely high impedance. If an input pin is left open, it can easily pick up extraneous noise and become unstable. In this case, if the input voltage level reaches an intermediate level, it is possible that both the P-channel and N-channel transistors will be turned on, allowing unwanted supply current to flow. Therefore, ensure that the unused input pins of a device are connected to the power supply (Vcc) pin or ground (GND) pin of the same device. For details of what to do with the pins of heat sinks, refer to the relevant technical datasheet and databook.

3.3.5 Latch-up

Latch-up is an abnormal condition inherent in CMOS devices, in which Vcc gets shorted to ground. This happens when a parasitic PN-PN junction (thyristor structure) internal to the CMOS chip is turned on, causing a large current of the order of several hundred mA or more to flow between Vcc and GND, eventually causing the device to break down.

Latch-up occurs when the input or output voltage exceeds the rated value, causing a large current to flow in the internal chip, or when the voltage on the Vcc (Vdd) pin exceeds its rated value, forcing the internal chip into a breakdown condition. Once the chip falls into the latch-up state, even though the excess voltage may have been applied only for an instant, the large current continues to flow between Vcc (Vdd) and GND (Vss). This causes the device to heat up and, in extreme cases, to emit gas fumes as well. To avoid this problem, observe the following precautions:

(1) Do not allow voltage levels on the input and output pins either to rise above Vcc (Vdd) or to fall below GND (Vss). Also, follow any prescribed power-on sequence, so that power is applied gradually or in steps rather than abruptly.

(2) Do not allow any abnormal noise signals to be applied to the device.

(3) Set the voltage levels of unused input pins to Vcc (Vdd) or GND (Vss).

(4) Do not connect output pins to one another.

3.3.6 Input/Output protection

Wired-AND configurations, in which outputs are connected together, cannot be used, since this short-circuits the outputs. Outputs should, of course, never be connected to Vcc (Vdd) or GND (Vss).

Furthermore, ICs with tri-state outputs can undergo performance degradation if a shorted output current is allowed to flow for an extended period of time. Therefore, when designing circuits, make sure that tri-state outputs will not be enabled simultaneously.

3.3.7 Load capacitance

Some devices display increased delay times if the load capacitance is large. Also, large charging and discharging currents will flow in the device, causing noise. Furthermore, since outputs are shorted for a relatively long time, wiring can become fused.

Consult the technical information for the device being used to determine the recommended load capacitance.

3.3.8 Thermal design

The failure rate of semiconductor devices is greatly increased as operating temperatures increase. As shown in Figure 2, the internal thermal stress on a device is the sum of the ambient temperature and the temperature rise due to power dissipation in the device. Therefore, to achieve optimum reliability, observe the following precautions concerning thermal design:

(1) Keep the ambient temperature (Ta) as low as possible.

(2) If the device's dynamic power dissipation is relatively large, select the most appropriate circuit board material, and consider the use of heat sinks or of forced air cooling. Such measures will help lower the thermal resistance of the package.

(3) Derate the device's absolute maximum ratings to minimize thermal stress from power dissipation.
$\theta ja = \theta jc + \theta ca$
$\theta ja = (Tj - Ta) / P$
$\theta jc = (Tj - Tc) / P$
$\theta ca = (Tc - Ta) / P$
in which θja = thermal resistance between junction and surrounding air (°C/W)
θjc = thermal resistance between junction and package surface, or internal thermal resistance (°C/W)
θca = thermal resistance between package surface and surrounding air, or external thermal resistance (°C/W)
Tj = junction temperature or chip temperature (°C)
Tc = package surface temperature or case temperature (°C)
Ta = ambient temperature (°C)
P = power dissipation (W)

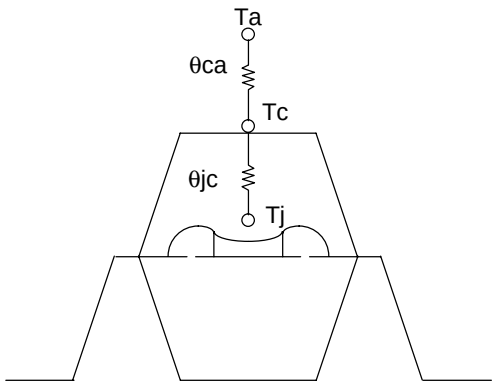


Figure 2 Thermal resistance of package

3.3.9 Interfacing

When connecting inputs and outputs between devices, make sure input voltage (V_{IL}/V_{IH}) and output voltage (V_{OL}/V_{OH}) levels are matched. Otherwise, the devices may malfunction. When connecting devices operating at different supply voltages, such as in a dual-power-supply system, be aware that erroneous power-on and power-off sequences can result in device breakdown. For details of how to interface particular devices, consult the relevant technical datasheets and databooks. If you have any questions or doubts about interfacing, contact your nearest Toshiba office or distributor.

3.3.10 Decoupling

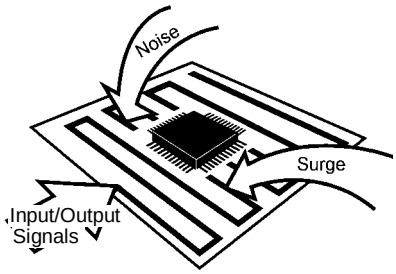
Spike currents generated during switching can cause Vcc (Vdd) and GND (Vss) voltage levels to fluctuate, causing ringing in the output waveform or a delay in response speed. (The power supply and GND wiring impedance is normally 50 Ω to 100 Ω.) For this reason, the impedance of power supply lines with respect to high frequencies must be kept low. This can be accomplished by using thick and short wiring for the Vcc (Vdd) and GND (Vss) lines and by installing decoupling capacitors (of approximately 0.01 μF to 1 μF capacitance) as high-frequency filters between Vcc (Vdd) and GND (Vss) at strategic locations on the printed circuit board.

For low-frequency filtering, it is a good idea to install a 10- to 100-μF capacitor on the printed circuit board (one capacitor will suffice). If the capacitance is excessively large, however, (e.g. several thousand μF) latch-up can be a problem. Be sure to choose an appropriate capacitance value.

An important point about wiring is that, in the case of high-speed logic ICs, noise is caused mainly by reflection and crosstalk, or by the power supply impedance. Reflections cause increased signal delay, ringing, overshoot and undershoot, thereby reducing the device's safety margins with respect to noise. To prevent reflections, reduce the wiring length by increasing the device mounting density so as to lower the inductance (L) and capacitance (C) in the wiring. Extreme care must be taken, however, when taking this corrective measure, since it tends to cause crosstalk between the wires. In practice, there must be a trade-off between these two factors.

3.3.11 External noise

Printed circuit boards with long I/O or signal pattern lines are vulnerable to induced noise or surges from outside sources. Consequently, malfunctions or breakdowns can result from overcurrent or overvoltage, depending on the types of device used. To protect against noise, lower the impedance of the pattern line or insert a noise-canceling circuit. Protective measures must also be taken against surges.



For details of the appropriate protective measures for a particular device, consult the relevant databook.

3.3.12 Electromagnetic interference

Widespread use of electrical and electronic equipment in recent years has brought with it radio and TV reception problems due to electromagnetic interference. To use the radio spectrum effectively and to maintain radio communications quality, each country has formulated regulations limiting the amount of electromagnetic interference which can be generated by individual products.

Electromagnetic interference includes conduction noise propagated through power supply and telephone lines, and noise from direct electromagnetic waves radiated by equipment. Different measurement methods and corrective measures are used to assess and counteract each specific type of noise.

Difficulties in controlling electromagnetic interference derive from the fact that there is no method available which allows designers to calculate, at the design stage, the strength of the electromagnetic waves which will emanate from each component in a piece of equipment. For this reason, it is only after the prototype equipment has been completed that the designer can take measurements using a dedicated instrument to determine the strength of electromagnetic interference waves. Yet it is possible during system design to incorporate some measures for the

prevention of electromagnetic interference, which can facilitate taking corrective measures once the design has been completed. These include installing shields and noise filters, and increasing the thickness of the power supply wiring patterns on the printed circuit board. One effective method, for example, is to devise several shielding options during design, and then select the most suitable shielding method based on the results of measurements taken after the prototype has been completed.

3.3.13 Peripheral circuits

In most cases semiconductor devices are used with peripheral circuits and components. The input and output signal voltages and currents in these circuits must be chosen to match the semiconductor device's specifications. The following factors must be taken into account.

(1) Inappropriate voltages or currents applied to a device's input pins may cause it to operate erratically. Some devices contain pull-up or pull-down resistors. When designing your system, remember to take the effect of this on the voltage and current levels into account.

(2) The output pins on a device have a predetermined external circuit drive capability. If this drive capability is greater than that required, either incorporate a compensating circuit into your design or carefully select suitable components for use in external circuits.

3.3.14 Safety standards

Each country has safety standards which must be observed. These safety standards include requirements for quality assurance systems and design of device insulation. Such requirements must be fully taken into account to ensure that your design conforms to the applicable safety standards.

3.3.15 Other precautions

(1) When designing a system, be sure to incorporate fail-safe and other appropriate measures according to the intended purpose of your system. Also, be sure to debug your system under actual board-mounted conditions.

(2) If a plastic-package device is placed in a strong electric field, surface leakage may occur due to the charge-up phenomenon, resulting in device malfunction. In such cases take appropriate measures to prevent this problem, for example by protecting the package surface with a conductive shield.

(3) With some microcomputers and MOS memory devices, caution is required when powering on or resetting the device. To ensure that your design does not violate device specifications, consult the relevant databook for each constituent device.

(4) Ensure that no conductive material or object (such as a metal pin) can drop onto and short the leads of a device mounted on a printed circuit board.

3.4 Inspection, Testing and Evaluation

3.4.1 Grounding

⚠CAUTION Ground all measuring instruments, jigs, tools and soldering irons to earth. Electrical leakage may cause a device to break down or may result in electric shock.

3.4.2 Inspection Sequence

⚠CAUTION

① Do not insert devices in the wrong orientation. Make sure that the positive and negative electrodes of the power supply are correctly connected. Otherwise, the rated maximum current or maximum power dissipation may be exceeded and the device may break down or undergo performance degradation, causing it to catch fire or explode, resulting in injury to the user.

② When conducting any kind of evaluation, inspection or testing using AC power with a peak voltage of 42.4 V or DC power exceeding 60 V, be sure to connect the electrodes or probes of the testing equipment to the device under test before powering it on. Connecting the electrodes or probes of testing equipment to a device while it is powered on may result in electric shock, causing injury.

(1) Apply voltage to the test jig only after inserting the device securely into it. When applying or removing power, observe the relevant precautions, if any.

(2) Make sure that the voltage applied to the device is off before removing the device from the test jig. Otherwise, the device may undergo performance degradation or be destroyed.

(3) Make sure that no surge voltages from the measuring equipment are applied to the device.

(4) The chips housed in tape carrier packages (TCPs) are bare chips and are therefore exposed. During inspection take care not to crack the chip or cause any flaws in it.
Electrical contact may also cause a chip to become faulty. Therefore make sure that nothing comes into electrical contact with the chip.

3.5 Mounting

There are essentially two main types of semiconductor device package: lead insertion and surface mount. During mounting on printed circuit boards, devices can become contaminated by flux or damaged by thermal stress from the soldering process. With surface-mount devices in particular, the most significant problem is thermal stress from solder reflow, when the entire package is subjected to heat. This section describes a recommended temperature profile for each mounting method, as well as general precautions which you should take when mounting devices on printed circuit boards. Note, however, that even for devices with the same package type, the appropriate mounting method varies according to the size of the chip and the size and shape of the lead frame. Therefore, please consult the relevant technical datasheet and databook.

3.5.1 Lead forming

⚠CAUTION

① Always wear protective glasses when cutting the leads of a device with clippers or a similar tool. If you do not, small bits of metal flying off the cut ends may damage your eyes.

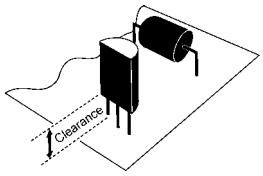
② Do not touch the tips of device leads. Because some types of device have leads with pointed tips, you may prick your finger.

Semiconductor devices must undergo a process in which the leads are cut and formed before the devices can be mounted on a printed circuit board. If undue stress is applied to the interior of a device during this process, mechanical breakdown or performance degradation can result. This is attributable primarily to differences between the stress on the device's external leads and the stress on the internal leads. If the relative difference is great enough, the device's internal leads, adhesive properties or sealant can be damaged. Observe these precautions during the lead-forming process (this does not apply to surface-mount devices):

(1) Lead insertion hole intervals on the printed circuit board should match the lead pitch of the device precisely.

(2) If lead insertion hole intervals on the printed circuit board do not precisely match the lead pitch of the device, do not attempt to forcibly insert devices by pressing on them or by pulling on their leads.

(3) For the minimum clearance specification between a device and a printed circuit board, refer to the relevant device's datasheet and databook. If necessary, achieve the required clearance by forming the device's leads appropriately. Do not use the spacers which are used to raise devices above the surface of the printed circuit board during soldering to achieve clearance. These spacers normally continue to expand due to heat, even after the solder has begun to solidify; this applies severe stress to the device.



(4) Observe the following precautions when forming the leads of a device prior to mounting.

- Use a tool or jig to secure the lead at its base (where the lead meets the device package) while bending so as to avoid mechanical stress to the device. Also avoid bending or stretching device leads repeatedly.
- Be careful not to damage the lead during lead forming.
- Follow any other precautions described in the individual datasheets and databooks for each device and package type.

3.5.2 Socket mounting

(1) When socket mounting devices on a printed circuit board, use sockets which match the inserted device's package.

(2) Use sockets whose contacts have the appropriate contact pressure. If the contact pressure is insufficient, the socket may not make a perfect contact when the device is repeatedly inserted and removed; if the pressure is excessively high, the device leads may be bent or damaged when they are inserted into or removed from the socket.

(3) When soldering sockets to the printed circuit board, use sockets whose construction prevents flux from penetrating into the contacts or which allows flux to be completely cleaned off.

(4) Make sure the coating agent applied to the printed circuit board for moisture-proofing purposes does not stick to the socket contacts.

(5) If the device leads are severely bent by a socket as it is inserted or removed and you wish to repair the leads so as to continue using the device, make sure that this lead correction is only performed once. Do not use devices whose leads have been corrected more than once.

(6) If the printed circuit board with the devices mounted on it will be subjected to vibration from external sources, use sockets which have a strong contact pressure so as to prevent the sockets and devices from vibrating relative to one another.

3.5.3 Soldering temperature profile

The soldering temperature and heating time vary from device to device. Therefore, when specifying the mounting conditions, refer to the individual datasheets and databooks for the devices used.

(1) Using a soldering iron

Complete soldering within ten seconds for lead temperatures of up to 260°C, or within three seconds for lead temperatures of up to 350°C.

(2) Using medium infrared ray reflow

- Heating top and bottom with long or medium infrared rays is recommended (see Figure 3).

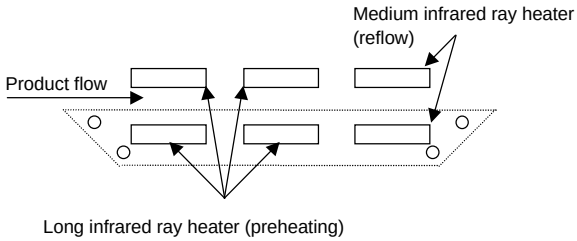


Figure 3 Heating top and bottom with long or medium infrared rays

- Complete the infrared ray reflow process within 30 seconds at a package surface temperature of between 210°C and 240°C.
- Refer to Figure 4 for an example of a good temperature profile for infrared or hot air reflow.

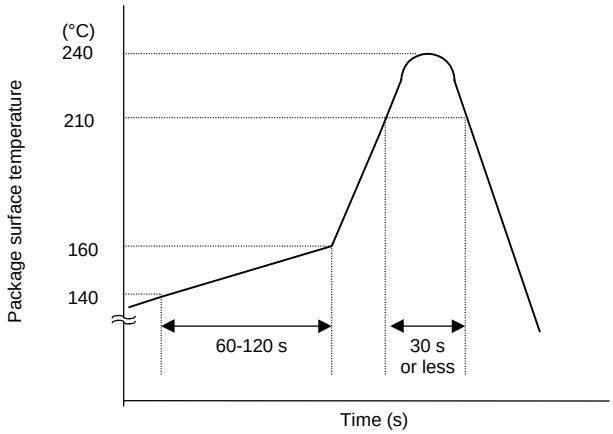


Figure 4 Sample temperature profile for infrared or hot air reflow

(3) Using hot air reflow

- Complete hot air reflow within 30 seconds at a package surface temperature of between 210°C and 240°C.
- For an example of a recommended temperature profile, refer to Figure 4 above.

(4) Using solder flow

- Apply preheating for 60 to 120 seconds at a temperature of 150°C.
- For lead insertion-type packages, complete solder flow within 10 seconds with the temperature at the stopper (or, if there is no stopper, at a location more than 1.5 mm from the body) which does not exceed 260°C.
- For surface-mount packages, complete soldering within 5 seconds at a temperature of 250°C or

less in order to prevent thermal stress in the device.

- Figure 5 shows an example of a recommended temperature profile for surface-mount packages using solder flow.

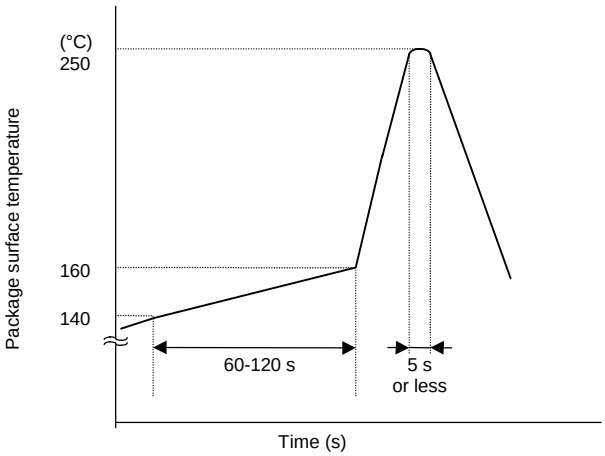


Figure 5 Sample temperature profile for solder flow

3.5.4 Flux cleaning and ultrasonic cleaning

(1) When cleaning circuit boards to remove flux, make sure that no residual reactive ions such as Na or Cl remain. Note that organic solvents react with water to generate hydrogen chloride and other corrosive gases which can degrade device performance.

(2) Washing devices with water will not cause any problems. However, make sure that no reactive ions such as sodium and chlorine are left as a residue. Also, be sure to dry devices sufficiently after washing.

(3) Do not rub device markings with a brush or with your hand during cleaning or while the devices are still wet from the cleaning agent. Doing so can rub off the markings.

(4) The dip cleaning, shower cleaning and steam cleaning processes all involve the chemical action of a solvent. Use only recommended solvents for these cleaning methods. When immersing devices in a solvent or steam bath, make sure that the temperature of the liquid is 50°C or below, and that the circuit board is removed from the bath within one minute.

(5) Ultrasonic cleaning should not be used with hermetically-sealed ceramic packages such as a leadless chip carrier (LCC), pin grid array (PGA) or charge-coupled device (CCD), because the bonding wires can become disconnected due to resonance during the cleaning process. Even if a device package allows ultrasonic cleaning, limit the duration of ultrasonic cleaning to as short a time as possible, since long hours of ultrasonic cleaning degrade the adhesion between the mold resin and the frame material. The following ultrasonic cleaning conditions are recommended:

Frequency: 27 kHz ~ 29 kHz

Ultrasonic output power: 300 W or less (0.25 W/cm^2 or less)

Cleaning time: 30 seconds or less

Suspend the circuit board in the solvent bath during ultrasonic cleaning in such a way that the ultrasonic vibrator does not come into direct contact with the circuit board or the device.

3.5.5 No cleaning

If analog devices or high-speed devices are used without being cleaned, flux residues may cause minute amounts of leakage between pins. Similarly, dew condensation, which occurs in environments containing residual chlorine when power to the device is on, may cause between-lead leakage or migration. Therefore, Toshiba recommends that these devices be cleaned. However, if the flux used contains only a small amount of halogen (0.05W% or less), the devices may be used without cleaning without any problems.

3.5.6 Mounting tape carrier packages (TCPs)

(1) When tape carrier packages (TCPs) are mounted, measures must be taken to prevent electrostatic breakdown of the devices.

(2) If devices are being picked up from tape, or outer lead bonding (OLB) mounting is being carried out, consult the manufacturer of the insertion machine which is being used, in order to establish the optimum mounting conditions in advance and to avoid any possible hazards.

(3) The base film, which is made of polyimide, is hard and thin. Be careful not to cut or scratch your hands or any objects while handling the tape.

(4) When punching tape, try not to scatter broken pieces of tape too much.

(5) Treat the extra film, reels and spacers left after punching as industrial waste, taking care not to destroy or pollute the environment.

(6) Chips housed in tape carrier packages (TCPs) are bare chips and therefore have their reverse side exposed. To ensure that the chip will not be cracked during mounting, ensure that no mechanical shock is applied to the reverse side of the chip. Electrical contact may also cause a chip to fail. Therefore, when mounting devices, make sure that nothing comes into electrical contact with the reverse side of the chip.
If your design requires connecting the reverse side of the chip to the circuit board, please consult Toshiba or a Toshiba distributor beforehand.

3.5.7 Mounting chips

Devices delivered in chip form tend to degrade or break under external forces much more easily than plastic-packaged devices. Therefore, caution is required when handling this type of device.

(1) Mount devices in a properly prepared environment so that chip surfaces will not be exposed to polluted ambient air or other polluted substances.

(2) When handling chips, be careful not to expose them to static electricity.
In particular, measures must be taken to prevent static damage during the mounting of chips. With this in mind, Toshiba recommend mounting all peripheral parts first and then mounting chips last (after all other components have been mounted).

(3) Make sure that PCBs (or any other kind of circuit board) on which chips are being mounted do not have any chemical residues on them (such as the chemicals which were used for etching the PCBs).

(4) When mounting chips on a board, use the method of assembly that is most suitable for maintaining the appropriate electrical, thermal and mechanical properties of the semiconductor devices used.

* For details of devices in chip form, refer to the relevant device's individual datasheets.

3.5.8 Circuit board coating

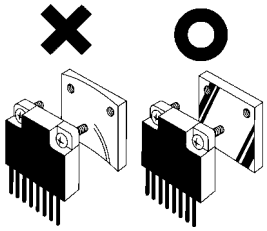
When devices are to be used in equipment requiring a high degree of reliability or in extreme environments (where moisture, corrosive gas or dust is present), circuit boards may be coated for protection. However, before doing so, you must carefully consider the possible stress and contamination effects that may result and then choose the coating resin which results in the minimum level of stress to the device.

3.5.9 Heat sinks

(1) When attaching a heat sink to a device, be careful not to apply excessive force to the device in the process.

(2) When attaching a device to a heat sink by fixing it at two or more locations, evenly tighten all the screws in stages (i.e. do not fully tighten one screw while the rest are still only loosely tightened). Finally, fully tighten all the screws up to the specified torque.

(3) Drill holes for screws in the heat sink exactly as specified. Smooth the surface by removing burrs and protrusions or indentations which might interfere with the installation of any part of the device.



(4) A coating of silicone compound can be applied between the heat sink and the device to improve heat conductivity. Be sure to apply the coating thinly and evenly; do not use too much. Also, be sure to use a non-volatile compound, as volatile compounds can crack after a time, causing the heat radiation properties of the heat sink to deteriorate.

(5) If the device is housed in a plastic package, use caution when selecting the type of silicone compound to be applied between the heat sink and the device. With some types, the base oil separates and penetrates the plastic package, significantly reducing the useful life of the device.
Two recommended silicone compounds in which base oil separation is not a problem are YG6260 from Toshiba Silicone.

(6) Heat-sink-equipped devices can become very hot during operation. Do not touch them, or you may sustain a burn.

3.5.10 Tightening torque

(1) Make sure the screws are tightened with fastening torques not exceeding the torque values stipulated in individual datasheets and databooks for the devices used.

(2) Do not allow a power screwdriver (electrical or air-driven) to touch devices.

3.5.11 Repeated device mounting and usage

Do not remount or re-use devices which fall into the categories listed below; these devices may cause significant problems relating to performance and reliability.

(1) Devices which have been removed from the board after soldering

(2) Devices which have been inserted in the wrong orientation or which have had reverse current applied

(3) Devices which have undergone lead forming more than once

3.6 Protecting Devices in the Field

3.6.1 Temperature

Semiconductor devices are generally more sensitive to temperature than are other electronic components. The various electrical characteristics of a semiconductor device are dependent on the ambient temperature at which the device is used. It is therefore necessary to understand the temperature characteristics of a device and to incorporate device derating into circuit design. Note also that if a device is used above its maximum temperature rating, device deterioration is more rapid and it will reach the end of its usable life sooner than expected.

3.6.2 Humidity

Resin-molded devices are sometimes improperly sealed. When these devices are used for an extended period of time in a high-humidity environment, moisture can penetrate into the device and cause chip degradation or malfunction. Furthermore, when devices are mounted on a regular printed circuit board, the impedance between wiring components can decrease under high-humidity conditions. In systems which require a high signal-source impedance, circuit board leakage or leakage between device lead pins can cause malfunctions. The application of a moisture-proof treatment to the device surface should be considered in this case. On the other hand, operation under low-humidity conditions can damage a device due to the occurrence of electrostatic discharge. Unless damp-proofing measures have been specifically taken, use devices only in environments with appropriate ambient moisture levels (i.e. within a relative humidity range of 40% to 60%).

3.6.3 Corrosive gases

Corrosive gases can cause chemical reactions in devices, degrading device characteristics. For example, sulphur-bearing corrosive gases emanating from rubber placed near a device (accompanied by condensation under high-humidity conditions) can corrode a device's leads. The resulting chemical reaction between leads forms foreign particles which can cause electrical leakage.

3.6.4 Radioactive and cosmic rays

Most industrial and consumer semiconductor devices are not designed with protection against radioactive and cosmic rays. Devices used in aerospace equipment or in radioactive environments must therefore be shielded.

3.6.5 Strong electrical and magnetic fields

Devices exposed to strong magnetic fields can undergo a polarization phenomenon in their plastic material, or within the chip, which gives rise to abnormal symptoms such as impedance changes or increased leakage current. Failures have been reported in LSIs mounted near malfunctioning deflection yokes in TV sets. In such cases the device's installation location must be changed or the device must be shielded against the electrical or magnetic field. Shielding against magnetism is especially necessary for devices used in an alternating magnetic field because of the electromotive forces generated in this type of environment.

3.6.6 Interference from light (ultraviolet rays, sunlight, fluorescent lamps and incandescent lamps)

Light striking a semiconductor device generates electromotive force due to photoelectric effects. In some cases the device can malfunction. This is especially true for devices in which the internal chip is exposed. When designing circuits, make sure that devices are protected against incident light from external sources. This problem is not limited to optical semiconductors and EPROMs. All types of device can be affected by light.

3.6.7 Dust and oil

Just like corrosive gases, dust and oil can cause chemical reactions in devices, which will adversely affect a device's electrical characteristics. To avoid this problem, do not use devices in dusty or oily environments. This is especially important for optical devices because dust and oil can affect a device's optical characteristics as well as its physical integrity and the electrical performance factors mentioned above.

3.6.8 Fire

Semiconductor devices are combustible; they can emit smoke and catch fire if heated sufficiently. When this happens, some devices may generate poisonous gases. Devices should therefore never be used in close proximity to an open flame or a heat-generating body, or near flammable or combustible materials.

3.7 Disposal of Devices and Packing Materials

When discarding unused devices and packing materials, follow all procedures specified by local regulations in order to protect the environment against contamination.

4. Precautions and Usage Considerations

This section describes matters specific to each product group which need to be taken into consideration when using devices. If the same item is described in Sections 3 and 4, the description in Section 4 takes precedence.

4.1 Microcontrollers

4.1.1 Design

(1) Using resonators which are not specifically recommended for use

Resonators recommended for use with Toshiba products in microcontroller oscillator applications are listed in Toshiba databooks along with information about oscillation conditions. If you use a resonator not included in this list, please consult Toshiba or the resonator manufacturer concerning the suitability of the device for your application.

(2) Undefined functions

In some microcontrollers certain instruction code values do not constitute valid processor instructions. Also, it is possible that the values of bits in registers will become undefined. Take care in your applications not to use invalid instructions or to let register bit values become undefined.

TMPR3927

1. Outline and Features

1.1 Outline

The TMPR3927A (hereinafter called “TX3927”) is a standard microcontroller of the TX39 family, which is one of Toshiba’s 32-bit TX System RISC families.

The TX3927 uses the TX39/H2 processor core as the CPU. The TX39/H2 processor core is a 32-bit RISC CPU core Toshiba developed based on the R3000A architecture of MIPS Technologies, Inc. (“MIPS”).

The TX3927 is designed for embedded applications. In addition to its TX39/H2 processor core, the TX3927 also incorporates peripheral circuits such as memory controllers, PCI and DMA controllers, serial and parallel ports, and timers/counters.

For information on the architecture of the TX39/H2 processor core, including the instruction set, refer to the following document:

32-bit TX System RISC, TX39/H2 Family Processor Core Architecture: Databook

R3000A is a trademark of MIPS Technologies, Inc.

1.2 Notation Used in This Manual

1.2.1 Numerical Notation

- Hexadecimal numbers are expressed as follows (example shown for decimal number 42): 0x2A
- KB (kilobyte): 2^{10} = 1,024 bytes
  MB (megabyte): 2^{20} = 1,024 × 1,024 = 1,048,576 bytes
  GB (gigabyte): 2^{30} = 1,024 × 1,024 × 1,024 = 1,073,741,824 bytes

1.2.2 Data Notation

- Byte: 8 bits
- Half word: 2 contiguous bytes (16 bits)
- Word: 4 contiguous bytes (32 bits)
- Double word: 8 contiguous bytes (64 bits)

Note: Chapter 12, "PCI Controller (PCIC)" uses notation based on the PCI specifications. Refer to the note given on page 12-1.

1.2.3 Signal Notation

- Active-low signals are indicated by adding an asterisk (*) at the end of the signal name. (Example: RESET*)
- A signal is "asserted" when it is driven to the active voltage level. A signal is "deasserted" when it is driven to an inactive voltage level.

1.2.4 Register Notation

- The bits of registers have any of the following attributes:

  R Read-only. The bit cannot be written.

  W Write-only. The value of the bit is undefined if read.

  R/W Read/Write

  R/WC Read/Write Clear. The bit can be read and written. However, a write of a 1 clears (resets to 0) the bit and a write of a 0 has no effect.

  R/WL Read/Write Local. The bit can be read from either the PCI bus or the TX39/H2 core. However, only the TX39/H2 core can write data to the bit.

- The notation <register name>.<bit/field name> is used to indicate a specific bit/field of a register.

  Example: CCFG. TOE

  CCFG.TOE refers to the Timeout Enable for Bus Error (TOE) field, located at bit 14 of the Chip Configuration Register (CCFG).

1.3 Features

(1) TX39/H2 processor core

- The TX39/H2 is a high-performance 32-bit microprocessor core Toshiba developed based on the R3000A architecture.
- 8K bytes of instruction cache (2-way set associative)
- 4K bytes of data cache (2-way set associative)
- Supports burst refill and cache locking functions
- Supports critical word first mode
- MMU containing transition lookaside buffer (TLB)
- Built-in hardware MAC unit for single-cycle DSP operation (throughput)
- Built-in debug support unit (DSU)

(2) SDRAM controller

- 8 channels (6 channels shared with ROMC)
- Supports SDRAM, DIMM flash, SGRAM, or SMROM memory
- Supports 16M/64M/128M/256M-bit SDRAM with 2/4 bank size availability
- Supports 16/32-bit data bus sizing on a per channel basis
- Supports single data rate (SDR) SDRAM
- Supports JEDEC standard 100-pin or 168-pin DIMM sockets for SDRAM
- Supports JEDEC standard 100-pin DIMM sockets for flash memory
- CKE-based timing control
- Refresh control
- Self-refreshing
- 66 MHz clock

(3) ROM Controller

- 8 channels (6 channels shared with SDRAMC)
- Supports ROM, page mode ROM, mask ROM, EPROM, E^2PROM, SRAM, and flash memory and I/O devices
- Supports memory sizes of 1M byte to 1G byte in 32-bit mode, and sizes of 1M byte to 512M bytes in 16-bit mode.
- Supports 16/32-bit data bus sizing on a per channel basis
- Supports full speed (66 MHz max.)/half speed (33 MHz max.) bus mode
- Supports selection between internal and external wait modes (ACK*/RDY)

(4) Timers/Counters

- 3-channel 24-bit up-counter
- Supports interval timer mode, pulse generator mode, and watchdog timer mode (only for timer 2).
- 2 timer output pins
- Supports external input clock

(5) Interrupt Controller

- Non-maskable interrupts (NMI)
- Maskable interrupts: 8 internal sources and 6 external sources
- Supports selection between edge- and level-triggered interrupts

(6) PCI Controller

- Full compliance with PCI Local Bus Specification Revision 2.1
- 32-bit PCI interface at 33 MHz
- Supports target mode (with streaming function) and initiator mode (without streaming function)
- Supports zero-wait-state read and write burst transfer for target mode
- FIFO to minimize memory controller latency
- Automatic mapping of PCI bus memory space to local bus (G-Bus) address space
- Supports enabling/disabling of internal arbiter function (for up to 4 external masters)
- External interrupt function
- Supports selection between internal and external clocks

(7) Direct memory access controller (DMAC)

- 4 channels
- Supports memory-to-memory and memory-to-I/O transfer
- Supports 8/16/32-bit wide I/O devices
- Supports internal/external transfer requests
- Supports dual address and single address transfer modes
- Supports word aligned memory-to-memory transfer using 4-word/8-word burst read/write
- Configurable address increments for both source and destination addresses

(8) Serial I/O ports

- 2-channel UART
- Built-in baud rate generator
- Modem flow control function
- 8-bit × 8 transmitter FIFO
- 13-bit (8 data bits and 5 status bits) × 16 receiver FIFO
- Supports selection between internal and external clocks

(9) Parallel I/O ports

- Up to 16 bi-directional I/O pins that can be read regardless of direction or mode (including one dedicated parallel port pin)
- Independent selection of direction of pins and choice between totem-pole and open-drain outputs
- 16-bit flag register

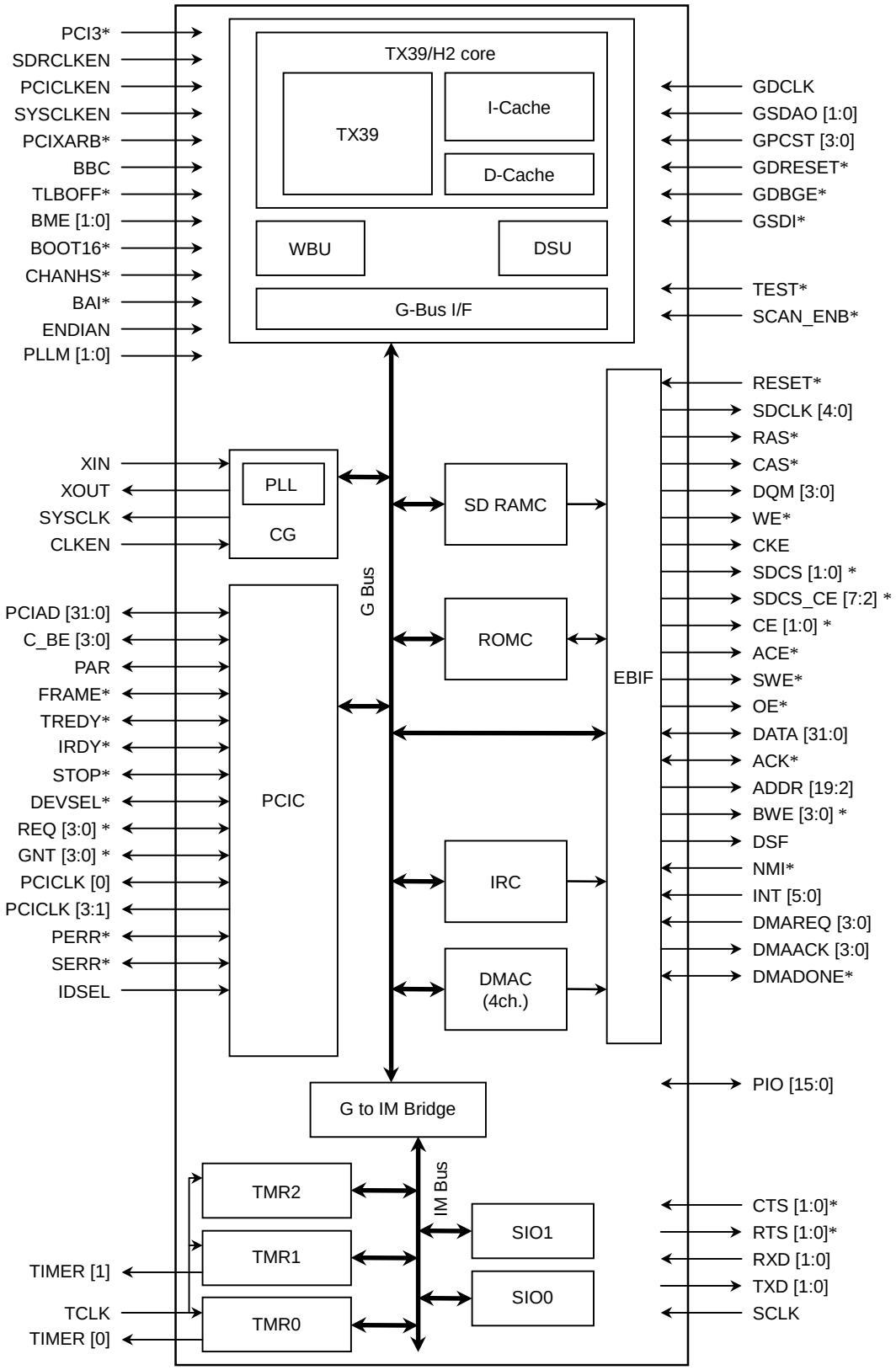
(10) Power Supply: 2.5 V (internal), 3.3 V (I/O)

(11) Maximum operating frequency: 133 MHz

(12) Package type: 240-pin QFP

2. Structure

2.1 Block Diagram



Note: Some pins are multiplexed with other functions.
For more information on multiplexed pins, refer to “3.3 Pin Multiplexing.”

Figure 2.1.1 TX3927 block diagram

3. Pins

3.1 Pinout

Table 3.1.1 TX3927 Pinout (1/2)

Pin No.	Signal	Pin No.	Signal	Pin No.	Signal	Pin No.	Signal
1	VSS2	31	VSS2	61	VSS	91	VSS2
2	DATA [30]	32	VSS	62	PCIAD [29]	92	DEVSEL*
3	DATA [23]	33	VDDS	63	PCIAD [28]	93	STOP*
4	DATA [31]	34	RTS [1] *	64	PCIAD [27]	94	PERR*
5	CE [1]	35	GPCST [2]	65	VSS	95	SERR*
6	VSS	36	GPCST [1]	66	PCIAD [26]	96	PAR
7	CE [0]	37	GPCST [0]	67	PCIAD [25]	97	C_BE [1]
8	ACE*	38	GDCLK	68	PCIAD [24]	98	VSS
9	ADDR [4]	39	GSDI	69	VSS	99	PCIAD [15]
10	ADDR [3]	40	GDRESET*	70	C_BE [3]	100	VDDS
11	ADDR [2]	41	GDBGE*	71	VDDS	101	PCIAD [14]
12	SYSCLK	42	PCICLK [3]	72	IDSEL	102	PCIAD [13]
13	VDD2	43	VDDS	73	PCIAD [23]	103	VSS
14	BWE [3] *	44	VSS	74	PCIAD [22]	104	PCIAD [12]
15	BWE [2] *	45	PCICLK [2]	75	VSS	105	PCIAD [11]
16	VSS	46	PCICLK [1]	76	PCIAD [21]	106	PCIAD [10]
17	BWE [1] *	47	PCICLK [0]	77	PCIAD [20]	107	VSS
18	BWE [0] *	48	VSS	78	PCIAD [19]	108	PCIAD [9]
19	OE*	49	GNT [3] *	79	VSS	109	PCIAD [8]
20	SWE*	50	GNT [2] *	80	PCIAD [18]	110	VDDS
21	SCLK	51	GNT [1] *	81	PCIAD [17]	111	C_BE [0]
22	RXD [0]	52	GNT [0] *	82	VDDS	112	VSS
23	TXD [0]	53	REQ [3] *	83	PCIAD [16]	113	PCIAD [7]
24	RTS [0] *	54	REQ [2] *	84	VSS	114	PCIAD [6]
25	CTS [0] *	55	REQ [1] *	85	C_BE [2]	115	PCIAD [5]
26	RXD [1]	56	REQ [0] *	86	FRAME*	116	VSS
27	TXD [1]	57	PCIAD [31]	87	IRDY*	117	PCIAD [4]
28	CTS [1] *	58	PCIAD [30]	88	VSS	118	PCIAD [3]
29	GSDAO [0]	59	VSS2	89	TRDY*	119	PCIAD [2]
30	VDD2	60	VDDS	90	VDD2	120	VDDS

Table 3.1.1 TX3927 Pinout (2/2)

Pin No.	Signal	Pin No.	Signal	Pin No.	Signal	Pin No.	Signal
121	VSS	151	VSS2	181	VSS2	211	VSS2
122	PCIAD [1]	152	DATA [12]	182	XIN	212	SDCS [0] *
123	PCIAD [0]	153	VSS	183	XOUT	213	SDCS [1] *
124	ACK*	154	DATA [5]	184	VDD2	214	SDCS_CE [2] *
125	DMAREQ [3]	155	DATA [13]	185	VDD2	215	SDCS_CE [3] *
126	DMAACK [3]	156	DATA [6]	186	PLLVDD	216	SDCS_CE [4] *
127	DMAREQ [2]	157	DATA [14]	187	FILTER [0]	217	SDCS_CE [5] *
128	DMAACK [2]	158	DATA [7]	188	FILTER [1]	218	VDDS
129	DMAREQ [0]	159	VDDS	189	PLLVSS	219	DMAACK [1]
130	VSS	160	DATA [15]	190	VSS2	220	DMAREQ [1]
131	VDDS	161	VSS	191	NMI*	221	DQM [2]
132	DMAACK [0]	162	DQM [0]	192	SCANENB*	222	VSS
133	DMADONE*	163	DQM [1]	193	CLKEN	223	DQM [3]
134	INT [3]	164	ADDR [5]	194	RESET*	224	DATA [16]
135	INT [2]	165	ADDR [6]	195	TEST*	225	DATA [24]
136	INT [1]	166	ADDR [7]	196	ADDR [18]	226	DATA [17]
137	INT [0]	167	ADDR [8]	197	ADDR [19]	227	DATA [25]
138	PIO [0]	168	VSS	198	RAS*	228	DATA [18]
139	DATA [0]	169	ADDR [9]	199	VSS	229	VSS
140	DATA [8]	170	VDDS	200	CAS*	230	VDDS
141	DATA [1]	171	ADDR [10]	201	SDCLK [0]	231	DATA [26]
142	DATA [9]	172	ADDR [11]	202	SDCLK [1]	232	DATA [19]
143	VSS	173	ADDR [12]	203	SDCLK [2]	233	DATA [27]
144	DATA [2]	174	ADDR [13]	204	VDDS	234	DATA [20]
145	DATA [10]	175	ADDR [14]	205	SDCLK [3]	235	DATA [28]
146	DATA [3]	176	VSS	206	SDCLK [4]	236	DATA [21]
147	VDDS	177	ADDR [15]	207	VSS	237	VSS
148	DATA [11]	178	ADDR [16]	208	CKE	238	DATA [29]
149	DATA [4]	179	ADDR [17]	209	WE*	239	DATA [22]
150	VDD2	180	VDDS	210	VDD2	240	VDDS

3.2 Pin Description

Signal Name	I/O	Description
System interface		
SYSCLK	O	System Clock Outputs a system clock in full speed mode (at one half the frequency of the TX39/H2 core) or half speed mode (at one quarter the frequency of the TX39/H2 core). The mode depends on the settings of boot signals CHANHS* (ADDR[15] pin) and BME[1:0] (ADDR[9:8] pins). Half speed mode is selected when CHANHS* is set to “0” or BME[1:0] are set to “10”. SYSCLK is not output when boot signal SYSCLKEN (ADDR[5] pin) is set to “0”.
DATA [31:0]	I/O	Data Data bus signals. In 16-bit bus mode, the DATA[15:0] pins are used. The pins are equipped with internal pull-up resistors.
ACK*	I/O	Acknowledge In external ACK* mode, this pin is used for an input signal for terminating bus operation. It can also be used as an RDY input (which can be set on a per channel basis). In internal ACK* mode, the pin is used to output a signal for notifying external devices of the termination of bus operation. The pin is equipped with an internal pull-up resistor.
RESET*	I	Reset Initializes the TX3927. The RESET* signal must remain low for at least 512 CPU clock cycles to initialize the internal circuitry. The pin is equipped with an internal pull-up resistor.
Clock signals		
XIN	I	Crystal Input Connect a crystal resonator or directly input an external clock signal. The clock frequency must match the multiplier specified with boot signals PLLM[1:0] (ADDT[3:2] pins). PLLM[1:0] = “01”: The multiplier is 2. In this case, input an external clock signal, leaving XOUT open. PLLM[1:0] = “11”: The multiplier is 16. In this case, connect a crystal resonator or input an external clock signal. PLLM[1:0] should not be set to “00” or “10”.
XOUT	O	Crystal Output Connect a crystal resonator. This pin must be left open when PLLM[1:0] = “01”.
CLKEN	I	Clock Enable Enables the internal clock generator of the TX3927. The CLKEN signal must remain low until the clock signal input from XIN (a crystal resonator or external clock input) becomes stable as the voltage settles. The pin is equipped with an internal pull-up resistor.

Signal Name	I/O	Description
Interrupt signals		
NMI*	I	Non Maskable Interrupt Non-maskable interrupt signal. The pin is equipped with an internal pull-up resistor.
INT [5:4]	I	Interrupt Request External interrupt request signals. INT[5] and INT[4] are multiplexed with CTS0 and RTS0, respectively. The pins are equipped with internal pull-up resistors.
INT [3:0]	I	Interrupt Request External interrupt request signals. The pins are equipped with internal pull-up resistors.
Timer interface		
TIMER [1:0]	O	Timer Pulse Width Output Timer output signals used in pulse generator mode. The pins output “1” in other modes. The pins are multiplexed with other functions. DMAREQ[3]/PIO[15]/TIMER[1] DMAACK[3]/PIO[11]/TIMER[0] DMADONE*/PIO[7]/TIMER[0]
TCLK	I	External Timer Clock Timer input clock signal. TMR0, TMR1, and TMR2 share this signal. The pin is multiplexed with PIO[13] and DMAREQ[2].
Memory interface		
SDCLK [4:0]	O	SDRAM Clock Out Outputs one half the frequency of the TX39/H2 core (e.g., 66 MHz when the TX39/H2 is operating at 133 MHz), which is used as a clock for SDRAM and SMROM, as well as for I/O devices that run in full speed bus mode. Only SDCLK[0] is equipped with an internal pull-up resistor. SDCLK is not output when boot signal SDRCLKEN (ADDR[4] pin) is set to “0.”
RAS*	O	Row Address Strobe RAS* signal for SDRAM, SMROM, and SGRAM.
CAS*	O	Column Address Strobe CAS* signal for SDRAM, SMROM, and SGRAM.
SDCS [7:0] *	O	Synchronous Memory Device Chip Select Chip select signals for SDRAM, SMROM, SGRAM, and 100-pin DIMM flash memory. The SDCS[7:2] pins are multiplexed with CE[7:2]. SDCS[7:6] are also multiplexed with DMAREQ[1]/PIO[11] and DMAACK[1]/PIO[10]. The SDCS[7:6] pins are equipped with internal pull-up resistors.
DQM [3:0] *	O	Data Mask During a write cycle, the DQM signals function as a data mask and can control individual bytes of the input data for SDRAM. During a read cycle, they control the SDRAM output buffers. The DQM signals also function as byte enable signals for DIMM flash memory during a write cycle.
WE*	O	Write Enable WE* signal for SDRAM, SMROM, and SGRAM.
CKE	O	Clock Enable CKE signal for SDRAM, SMROM, and SGRAM.
SWE*	O	SRAM Write Enable Write enable signal for SRAM and I/O devices.

Signal Name	I/O	Description
Memory interface		
ADDR [19:2]	O	Address Address signals. For ROM, SRAM, and I/O devices, ADDR can be used as 28 address signals with an external circuit (74377) and the ACE* signal. ADDR[29:20] are output to the ADDR[19:10] pins when ACE* is low. ADDR[19:5] are used for SDRAM, SMROM, SGRAM, and DIMM flash memory. All pins are equipped with internal pull-up resistors. The ADDR signals are also used as boot signals during a reset. The boot settings are latched into the TX3927 on the rising edge of RESET*.
OE*	O	Output Enable Output enable signal for ROM, SRAM, I/O devices, SMROM, DIMM flash memory.
CE [7:0] *	O	ROM Chip Enable Chip select signals for ROM, SRAM, and I/O devices. The CE[7:2] pins are multiplexed with SDCS[7:2]. Only CE[7:6] pins are equipped with internal pull-up resistors.
ACE*	O	ROM Address Clock Enable Clock enable signals used by an external circuit (74377) to latch ADDR[29:20], output from the multiplexed ADDR[19:10] pins.
BE [3:0] */ BWE [3:0] *	O	Byte Enable/Byte Write Enable BE[3:0] indicate a valid data position on the data bus DATA[31:0] during bus operation. In 16-bit bus mode, only BE[1:0]* are used. BWE[3:0]* indicate a valid data position on the data bus DATA[31:0] during write bus operation. In 16-bit bus mode, only BWE[1:0]* are used. The following shows the correspondence between BE[3:0]*/BWE[3:0]* and the data bus signals. BE[3]*/BWE[3]*: DATA[31:24] BE[2]*/BWE[2]*: DATA[23:16] BE[1]*/BWE[1]*: DATA[15:8] BE[0]*/BWE[0]*: DATA[7:0] Boot signal BBC (ADDR[6] pin) and the RCCRn.RBCn ROM controller bit determine whether the signals are used as BE[3:0] or BWE[3:0].
DSF	O	Define Special Function Signal used for SGRAM. The DSF pin is multiplexed with PIO[1].

Signal Name	I/O	Description
PCI interface		
PCIAD [31:0]	I/O	PCI Address and Data Multiplexed address and data bus.
C_BE [3:0]	I/O	Bus Command and Byte Enables Command and byte enable signals.
PAR	I/O	Parity Even parity signal for PCIAD[31:0] and C_BE[3:0]*.
FRAME*	I/O	Cycle Frame Indicates that bus operation is in progress.
IRDY*	I/O	Initiator Ready Indicates that the initiator is ready to complete data transfer.
TRDY*	I/O	Target Ready Indicates that the target is ready to complete data transfer.
STOP*	I/O	Stop The target sends this signal to the initiator to request termination of data transfer.
ID_SEL	I	Initialization Device Select Chip select signal used for configuration access.
DEVSEL*	I/O	Device Select The target asserts this signal in response to access from the initiator.
REQ [3:0] *	I/O	Request Signals used by the master to request bus mastership. In internal arbiter mode, REQ[3:0] are input signals. In external arbiter mode, REQ[0] is an output signal, REQ[1] is a flag interrupt output signal, and REQ[3:2] are not used.
GNT [3:0] *	I/O	Grant Indicates that bus mastership has been granted to the master. In internal arbiter mode, GNT[3:0] are output signals. In external arbiter mode, GNT[0] is an input signal and GNT[3:1] are not used.
PERR*	I/O	Parity Error Indicates a parity error in a bus cycle other than special cycles.
SERR*	I/O	System Error Indicates an address parity error, a data parity error in a special cycle, or a fatal error.
PCICLK [3:0]	I/O	PCI Clock PCI bus clock signals. PCICLK[0] is either used as an output from the TX3927 or as an input from an external device to the PCI controller of the TX3927. PCICLK[3:1] are tri-state outputs. Boot signal PCICLKEN (ADDR[18] pin) determines the usage of PCICLK[0]. When PCICLKEN is set to “1”, PCICLK[3:0] are used as outputs. When PCICLKEN is set to “0”, PCICLK[0] is used as an input and PCICLK[3:1] are not used.

Signal Name	I/O	Description
DMA interface		
DMAREQ [3:0]	I	DMA Request DMA transfer request signals from an external I/O device. The pins are multiplexed with PIO, TIMER, TCLK, SDCS[7], and CE[7]. Refer to “3.3 Pin Multiplexing” for details. The pins are equipped with internal pull-up resistors.
DMAACK [3:0]	O	DMA Acknowledge DMA transfer acknowledge signals to an external I/O device. The pins are multiplexed with PIO, TIMER, SDCS[6], and CE[6]. Refer to “3.3 Pin Multiplexing” for details. The pins are equipped with internal pull-up resistors.
DMADONE*	I/O	DMA Done DMADONE is either used as an output signal that reports the termination of DMA transfer or as an input signal that causes DMA transfer to terminate. The pin is equipped with an internal pull-up resistor.
SIO interface		
TXD [1:0]	O	SIO Transmit Data Serial data output signals. The TXD pins are multiplexed with PIO. The pins are equipped with internal pull-up resistors.
RXD [1:0]	I	SIO Receive Data Serial data input signals. The RXD pins are multiplexed with PIO. The pins are equipped with internal pull-up resistors.
RTS [1:0] *	O	Request to Send RTS* signals. The RTS pins are multiplexed with PIO, INT, DSF, and debug signal GPCST[3]. The pins are equipped with internal pull-up resistors.
CTS [1:0] *	I	Clear to Send CTS* signals. The CTS pins are multiplexed with PIO, INT, and debug signal GSDAO[1]. The pins are equipped with internal pull-up resistors.
SCLK	I	SIO Clock SIO clock input signal. SIO0 and SIO1 share this signal. The pin is equipped with an internal pull-up resistor.
Parallel interface		
PIO [15:0]	I/O	PIO Port The PIO[0] pin is dedicated to a PIO signal. The other pins, PIO[15:1], are multiplexed with DMA, interrupt input, timer, serial interface, and SGRAM DSF signals. The pins are equipped with internal pull-up resistors.

Signal Name	I/O	Description
Debug interface		
GDCLK	O	Debug Clock Signal Clock signal for an external real-time debug system.
GSDAO [1:0]	O	Serial Data and Address Output/Target PC Output signals for an external real-time debug system.
GPCST [3:0]	O	PC Trace Status PC trace output signals for an external real-time debug system.
GDRESET*	I	Debug Reset Reset signal for an external real-time debug system. The pin is equipped with an internal pull-up resistor.
GDBGE*	I	Debugger Enable Enable signal for an external real-time debug system. The pin is equipped with an internal pull-up resistor.
GSDI*	I	Serial Data input/Debug Interrupt Input signal for an external real-time debug system. The pin is equipped with an internal pull-up resistor.
Other signals		
TEST*	I	Test Test pin. Fix the pin to high. The pin is equipped with an internal pull-up resistor.
SCAN_ENB*	I	Scan Mode Test Control Test pin. Fix the pin to high. The pin is equipped with an internal pull-up resistor.
PLLVDD		Power supply pin for the PLL. Supply 2.5 V.
PLLVSS		Ground pin for the PLL.
VDD2		Power supply pin for the internal logic. Supply 2.5 V.
VSS2		Ground pin for the internal logic.
VDDS		Power supply pin for the I/O ports. Supply 3.3 V.
VSS		Ground pin for the I/O ports.

3.3 Pin Multiplexing

A total of 24 pins of the TX3927 have multiplexed functions. Table 3.3.1 shows the multiplexed pins. The functions of individual pins are selected in different ways. Table 3.3.2 shows the pins for which the PCFG control register determines the functions.

Table 3.3.1 Pin Multiplexing

Pin No.	Signal Name	Multiplexed Functions
217	SDCS_CE [5] *	SDCS [5] */CE [5] *
216	SDCS_CE [4] *	SDCS [4] */CE [4] *
215	SDCS_CE [3] *	SDCS [3] */CE [3] *
214	SDCS_CE [2] *	SDCS [2] */CE [2] *
52	GNT [0] *	GNT_0_OUT/GNT_IN
56	REQ [0] *	REQ_0_IN/REQ_OUT
55	REQ [1] *	REQ_1_IN/XINT_OUT
125	DMAREQ [3]	PIO [15] /DMAREQ [3] /TIMER [1]
126	DMAACK [3]	PIO [14] /DMAACK [3] /TIMER [0]
127	DMAREQ [2]	PIO [13] /DMAREQ [2] /TCLK
128	DMAACK [2]	PIO [12] /DMAACK [2]
220	DMAREQ [1]	PIO [11] /DMAREQ [1] /SDCS [7] */CE [7] *
219	DMAACK [1]	PIO [10] /DMAACK [1] /SDCS [6] */CE [6] *
129	DMAREQ [0]	PIO [9] /DMAREQ [0]
132	DMAACK [0]	PIO [8] /DMAACK [0]
133	DMADONE*	PIO [7] /DMADONE*/TIMER [0]
26	RXD [1]	PIO [6] /RXD [1]
27	TXD [1]	PIO [5] /TXD [1]
22	RXD [0]	PIO [4] /RXD [0]
23	TXD [0]	PIO [3] /TXD [0]
28	CTS [1] *	PIO [2] /CTS [1] */GSDAO [1]
34	RTS [1] *	PIO [1] /RTS [1] */DSF/GPCST [3]
25	CTS [0] *	INT [5] /CTS [0] *
24	RTS [0] *	INT [4] /RTS [0] *

The SDCS_CE[5:2]* chip select signals, corresponding to channels 5 to 2, respectively, are shared for both SDRAM and ROM. If a given SDRAM or ROM channel is enabled using the SDRAM or ROM controller, the corresponding pin is assigned to that channel. If SDRAM and ROM channels having an identical number are enabled at the same time, the signal on the corresponding SDCS_CE pin is asserted in both SDRAM and ROM bus cycles, resulting in undetermined memory data.

The PCI grant signals (GNT[3:0]*) and request signals (REQ[3:0]*) have different operations depending on whether the chip booted into external or internal PCI arbiter mode. In internal arbiter mode, REQ[3:0]* are input signals and GNT[3:0]* are output signals.

In external arbiter mode, REQ[0]* becomes a request signal output to the PCI module and GNT[0]* becomes a grant signal input from the PCI module. REQ[1]* becomes an interrupt output that can be controlled with the PIO module flag registers. REQ[3:2]* and GNT[3:1]*, which become output and input pins, respectively, are not used.

The other multiplexed pins are controlled by setting the bits of the pin configuration (PCFG) register, as detailed in Table 3.3.2. Usually, the pins are controlled in pairs, such as DMAREQ[3] and DMAACK[3].

CTS[1]* and RTS[1]* are also shared with debug support signals; their functions are determined by the GDBGE* pin as well as PCFG register bits.

The DMAREQ[1] and DMAACK[1] pins can be used for the SDCS_CE[7:6]* chip select signals, corresponding to channels 7 and 6, respectively. These chip select signals are, however, shared for both SDRAM and ROM. If a given SDRAM or ROM channel is enabled using the SDRAM or ROM controller, the corresponding pin is assigned to that channel. If SDRAM and ROM channels having an identical number are enabled at the same time, the signal on the corresponding SDCS_CE pin is asserted in both SDRAM and ROM bus cycles, resulting in undetermined memory data.

CTS[0]* and RTS[0]* are multiplexed with INT[5:4]. Both pins are always internally routed to the interrupt controller. Thus, if the serial interface function is selected, ensure that interrupts are not enabled. Otherwise, interrupts will occur every time CTS[0]* or RTS[0]* changes the state.

Table 3.3.2 Pin Functions Corresponding to The Settings of PCFG Control Register Bits

Pin/Function	Pin/Function	PCFG Control Bits		
DMAREQ [3]	DMAACK [3]	SELDMA [3]	SELTMR [1]	SELTMR [0]
PIO [15]	PIO [14]	0	0	0
PIO [15]	TIMER [0]	0	0	1
TIMER [1]	PIO [14]	0	1	0
TIMER [1]	TIMER [0]	0	1	1
DMAREQ [3]	DMAACK [3]	1	x	x
DMAREQ [2]	DMAACK [2]	SELDMA [2]		
PIO [13]	PIO [12]	0		
DMAREQ [2]	DMAACK [2]	1		
DMAREQ [1]	DMAACK [1]	SELDMA [1]	SELCS	
PIO [11]	PIO [10]	0	0	
SDCS_CE [7]*	SDCS_CE [6]*	0	1	
DMAREQ [1]	DMAACK [1]	1	x	
DMAREQ [0]	DMAACK [0]	SELDMA [0]		
PIO [9]	PIO [8]	0		
DMAREQ [0]	DMAACK [0]	1		
DMADONE*		SELDONE	SELTMR [2]	
PIO [7]		0	0	
TIMER [0]		0	1	
DMADONE*		1	x	
RXD [1]	TXD [1]	SELSIO [1]		
PIO [6]	PIO [5]	0		
RXD [1]	TXD [1]	1		
RXD [0]	TXD [0]	SELSIO [0]		
PIO [4]	PIO [3]	0		
RXD [0]	TXD [0]	1		
CTS [1] *	RTS [1] *	GDBGE*	SELSIOC [1]	SELDSF
PIO [2]	PIO [1]	1	0	0
PIO [2]	DSF	1	0	1
CTS [1] *	RTS [1] *	1	1	0
GSDAO [1]	GPCST [3]	0	x	x
CTS [0] *	RTS [0] *	SELSIOC [0]		
INT [5]	INT [4]	0		
CTS [0] *	RTS [0] *	1		

3.4 Initial Setting Signals

These initial setting signals (also known as boot signals) use the same pins as ADDR[19:2]. The signals are captured into the TX3927 on the rising edge of the RESET* or CLKEN signal to initialize the TX3927. PLLM[1:0] are latched on the rising edge of CLKEN and the others are latched on the rising edge of RESET*. All initial setting signals are pulled up in the TX3927.

Some of these signals are latched into registers. The R/W column of the following table shows the read/write attribute of the corresponding register bit. For details of the CCFG and PCFG registers, refer to Chapter 5, "Configuration." For details of the SDCCR0 register, refer to Chapter 8, "SDRAM Controller." For details of the RCCR0 register, refer to Chapter 9, "External Bus Controller."

Initial Setting Signal	Pin	Description	Register/Bit	R/W
PLLM [1:0]	ADDR [3:2]	PLL Multiplier Specifies the multiplier value for the internal clock generator. After the termination of a reset, the value cannot be changed. 00: Not to be set. 01: ×2 10: Not to be set. 11: ×16	CCFG/bit [5:4] (Chip Configuration register)	R
TLBOFF*	ADDR [19]	TLB Off Enables or disables TLB operation. 0: Disable (TLB Off) 1: Enable (TLB On)	CCFG/bit [17] (Chip Configuration register)	R/W (*1)
ENDIAN	ADDR [14]	ENDIAN Specifies the CPU endian mode. After the termination of a reset, the value cannot be changed. 0: Little endian 1: Big endian	CCFG/bit [2] (Chip Configuration register)	R
SYSCLKEN	ADDR [5]	System Clock Enable Enables or disables the system clock output (SYSCLK). 0: Disable 1: Enable	PCFG/bit [27] (Pin Configuration register)	R/W
CHANHS*	ADDR [15]	Boot with Half Speed Bus Specifies the frequency of the SYSCLK output. When BME[1:0] = "10", however, the SYSCLK output is forcibly placed in half speed bus mode. After the termination of a reset, the value cannot be changed. 0: The SYSCLK output frequency is one quarter of the CPU frequency (half speed bus mode). 1: The SYSCLK output frequency is one half of the CPU frequency (full speed bus mode).	CCFG/bit [1] (Chip Configuration register)	R
SDRCLKEN	ADDR [4]	SDRAM Clock Enable Enables or disables the output clock for SDRAM. 0: Disable 1: Enable	PCFG/bit [26:22] (Pin Configuration register)	R/W
BME [1:0]	ADDR [9:8]	Boot Memory Enable Specifies the type of memory device to boot from. 00: SMROM 01: DIMM flash memory 10: Half speed ROM 11: Full speed ROM	RCCR0/bit [4:3] or SDCCR0/bit [19:18], [17]	R/W
BOOT16*	ADDR [13]	Boot with 16-bit Bus Width Specifies the bus size for the boot device. 0: 16 bits 1: 32 bits	RCCR0/bit [7] (ROM channel control register 0)	R/W

Initial Setting Signal	Pin	Description	Register/Bit	R/W
BAI*	ADDR [7]	Boot ACK* Input Enables or disables an external ACK input signal for the boot device. 0: Enable external ACK input 1: Disable external ACK input (ACK is internally generated and output from the ACK pin.)	RCCR0/bit [12] (ROM channel control register 0)	R/W
BBC	ADDR [6]	Boot Byte Control Specifies whether the byte enable signals for the boot device are BWE [3:0] or BE [3:0]. 0: BE [3:0] (Byte Enable) 1: BWE [3:0] (Byte Write Enable)	RCCR0/bit [5] (ROM channel control register 0)	R/W
PCICLKEN	ADDR [18]	PCI Clock Enable Enables or disables the PCI output clock (PCICLK). 0: Disable (input) An external PCI clock must be input to the PCICLK[0] pin. 1: Enable (output)	PCFG/bit [21:18] (Pin Configuration register)	R/W
PCI3*	ADDR [17]	PCI Clock divisor Specifies the divisor value for the PCI output clock. 0: The PCI clock frequency is one third of the GBus clock frequency. 1: The PCI clock frequency is one half of the GBus clock frequency.	CCFG/bit [12] (Chip Configuration register)	R/W
PCIXARB*	ADDR [11]	PCI Arbiter Control Setting Specifies whether the TX3927 or an external bus master is responsible for PCI arbitration. After the termination of a reset, the value cannot be changed. 0: External bus master 1: PCI controller of TX3927	CCFG/bit [13] (Chip Configuration register)	R

Note 1: CCFG.TLBOFF can be written but it should not be changed from the initial value.

Note 2: Ensure that none of ADDR[16], ADDR[12], and ADDR[10] are driven to low during a reset.

Table 3.4.1 Correspondence Between Initial Setting Signals and Address Bus Pins

Pin No.	Initial Setting Signal	Address Bus Pin
197	TLBOFF*	ADDR [19]
196	PCICLKEN	ADDR [18]
179	PCI3	ADDR [17]
177	CHANHS*	ADDR [15]
175	ENDIAN	ADDR [14]
174	BOOT16*	ADDR [13]
172	PCIXARB*	ADDR [11]
169,167	BME [1:0]	ADDR [9:8]
166	BAI*	ADDR [7]
165	BBC	ADDR [6]
164	SYSCLKEN	ADDR [5]
9	SDRCLKEN	ADDR [4]
10,11	PLLM [1:0]	ADDR [3:2]

4. Address Mapping

4.1 Memory Mapping

The TX3927 supports up to 4G bytes of address space.

The memory map of the TX3927 is managed by the memory management unit (MMU) of the TX39/H2 processor core. Table 4.1.1 summarizes the address mapping of the TX3927. Figure 4.1.1 shows the address map in TLB OFF. Figure 4.1.2 shows the address map in TLB ON.

For more information on the memory map of the TX39/H2 processor core, refer to the *32-bit TX System RISC, TX39/H2 Family Processor Core Architecture: Databook.*

Table 4.1.1 TX3927 Address Mapping

Segment	Virtual Address	Physical Address	Cacheable Area		Mode
			TLB on	TLB off	
kseg2 (Reserved)	0xFFFF_FFFF 0xFFFF_0000	0xFFFF_FFFF 0xFFFF_0000	Cacheable	Uncacheable	Kernel
Kseg2 (Reserved)	0xFFFE FFFF 0xFF00 0000	0xFFFE FFFF 0xFF00 0000	Uncacheable	Uncacheable	Kernel
kseg2	0xFEFF_FFFF 0xC000_0000	0xFEFF_FFFF 0xC000_0000	Cacheable	Cacheable	Kernel
kseg1	0xBFFF_FFFF 0xA000_0000	0x1FFF_FFFF 0x0000_0000	Uncacheable	Uncacheable	Kernel
kseg0	0x9FFF_FFFF 0x8000_0000	0x1FFF_FFFF 0x0000_0000	Cacheable	Cacheable	Kernel
Kuseg (Reserved)	0x7FFF_FFFF 0x7F00_0000	0xBFFF_FFFF 0xBF00_0000	Cacheable	Uncacheable	Kernel/user
kuseg	0x7EFF_FFFF 0x0000_0000	0xBEFF_FFFF 0x4000_0000	Cacheable	Cacheable	Kernel/user

In the TX3927, the memory controllers (SDRAMC and ROMC) are used to specify the address (physical address) in the memory (SDRAM, ROM, etc.) or I/O device to be connected.

The boot memory device is assigned channel 0 of ROMC or SDRAMC. Immediately after a reset, both channels have a base address of 0x1FC0_0000 (physical address), from which instruction fetch is started.

The top area of kseg2 (0xFF00_0000-0xFFFE_FFFF) is reserved. The registers of the TX3927 built-in modules are allocated in this area.

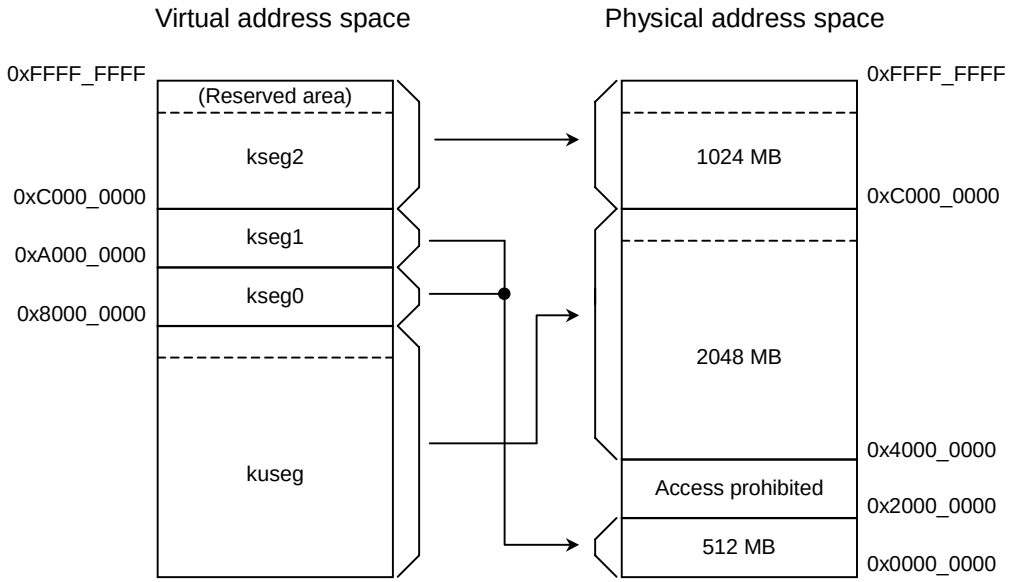


Figure 4.1.1 Memory Address Map (TLB off)

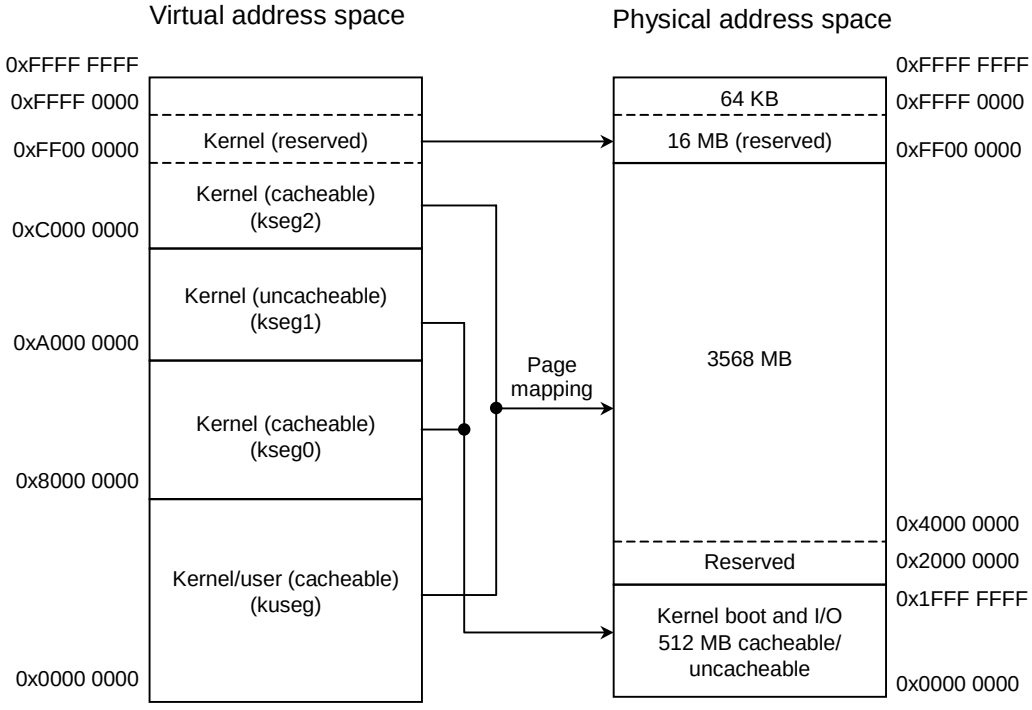


Figure 4.1.2 Memory Address Map (TLB on)

4.2 Register Mapping

Among addresses 0xFFFE_0000 to 0xFFFE_FFFF, only the addresses shown in the table can be accessed. Accessing any other address may cause the bus to be locked.

Table 4.2.1 TX3927 Register Mapping (1/6)

Address	Register Symbol	Register Name
PIO		
0xFFFE_F524	XPIOMASKEXT	PIO External Interrupt Mask Register
0xFFFE_F520	XPIOMASKCPU	PIO CPU Interrupt Mask Register
0xFFFE_F51C	XPIOINT	PIO Interrupt Control Register
0xFFFE_F518	XPIOPOL	PIO Flag Polarity Control Register
0xFFFE_F514	XPIOFLAG1	PIO Flag 1 Register
0xFFFE_F510	XPIOFLAG0	PIO Flag 0 Register
0xFFFE_F50C	XPIOOD	PIO Open Drain Control Register
0xFFFE_F508	XPIODIR	PIO Direction Control Register
0xFFFE_F504	XPIODI	PIO Input Data Register
0xFFFE_F500	XPIODO	PIO Output Data Register
SIO1		
0xFFFE_F420	SIRFIFO1	Receiver FIFO 1
0xFFFE_F41C	SITFIFO1	Transmitter FIFO 1
0xFFFE_F418	SIBGR1	Baud Rate Control Register 1
0xFFFE_F414	SIFLCR1	Flow Control Register 1
0xFFFE_F410	SIFCR1	FIFO Control Register 1
0xFFFE_F40C	SICISR1	Status Change Interrupt Status Register 1
0xFFFE_F408	SIDISR1	DMA/Interrupt Status Register 1
0xFFFE_F404	SIDICR1	DMA/Interrupt Control Register 1
0xFFFE_F400	SILCR1	Line Control Register 1
SIO0		
0xFFFE_F320	SIRFIFO0	Receiver FIFO 0
0xFFFE_F31C	SITFIFO0	Transmitter FIFO 0
0xFFFE_F318	SIBGR0	Baud Rate Control Register 0
0xFFFE_F314	SIFLCR0	Flow Control Register 0
0xFFFE_F310	SIFCR0	FIFO Control Register 0
0xFFFE_F30C	SICISR0	Status Change Interrupt Status Register 0
0xFFFE_F308	SIDISR0	DMA/Interrupt Status Register 0
0xFFFE_F304	SIDICR0	DMA/Interrupt Control Register 0
0xFFFE_F300	SILCR0	Line Control Register 0

Table 4.2.1 TX3927 Register Mapping (2/6)

Address	Register Symbol	Register Name
TMR2		
0xFFFE_F2F0	TMTRR2	Timer Read Register 2
0xFFFE_F240	TMWTMR2	Watchdog Timer Mode Register 2
0xFFFE_F230	—	(Reserved)
0xFFFE_F220	TMCCDR2	Divider Register 2
0xFFFE_F210	TMITMR2	Interval Timer Mode Register 2
0xFFFE_F20C	—	(Reserved)
0xFFFE_F208	TMCPRA2	Compare Register A2
0xFFFE_F204	TMTISR2	Timer Interrupt Status Register 2
0xFFFE_F200	TMTCR2	Timer Control Register 2
TMR1		
0xFFFE_F1F0	TMTRR1	Timer Read Register 1
0xFFFE_F140	—	(Reserved)
0xFFFE_F130	TMPGMR1	Pulse Generator Mode Register 1
0xFFFE_F120	TMCCDR1	Divider Register 1
0xFFFE_F110	TMITMR1	Interval Timer Mode Register 1
0xFFFE_F10C	TMCPRB1	Compare Register B1
0xFFFE_F108	TMCPRA1	Compare Register A1
0xFFFE_F104	TMTISR1	Timer Interrupt Status Register 1
0xFFFE_F100	TMTCR1	Timer Control Register 1
TMR0		
0xFFFE_F0F0	TMTRR0	Timer Read Register 0
0xFFFE_F040	—	(Reserved)
0xFFFE_F030	TMPGMR0	Pulse Generator Mode Register 0
0xFFFE_F020	TMCCDR0	Divider Register 0
0xFFFE_F010	TMITMR0	Interval Timer Mode Register 0
0xFFFE_F00C	TMCPRB0	Compare Register B0
0xFFFE_F008	TMCPRA0	Compare Register A0
0xFFFE_F004	TMTISR0	Timer Interrupt Status Register 0
0xFFFE_F000	TMTCR0	Timer Control Register 0
CCFG		
0xFFFE_E010	PDCR	Power Down Control Register
0xFFFE_E00C	TEAR	Timeout Error Address Register
0xFFFE_E008	PCFG	Pin Configuration Register
0xFFFE_E004	CRIR	Chip reversion ID Register
0xFFFE_E000	CCFG	Chip Configuration Register

Table 4.2.1 TX3927 Register Mapping (3/6)

Address	Register Symbol	Register Name
PCIC		
0xFFFE_D158	IPCICBE	Initiator Indirect Command/Byte Enable Register
0xFFFE_D154	IPCIDATA	Initiator Indirect Data Register
0xFFFE_D150	IPCIADDR	Initiator Indirect Address Register
0xFFFE_D14C	IOMAS	Initiator I/O Mapping Address Size Register
0xFFFE_D148	MMAS	Initiator Memory Mapping Address Size Register
0xFFFE_D144	ISCDP	Initiator Special Cycle Data Port Register
0xFFFE_D140	IIADP	Initiator Interrupt Acknowledge Data Port Register
0xFFFE_D13C	ICRD	Initiator Configuration Data Register
0xFFFE_D138	ICA	Initiator Configuration Address Register
0xFFFE_D134	PCISTATIM	PCI Status Interrupt Mask Register
0xFFFE_D130	LBIM	Local Bus Interrupt Mask Register
0xFFFE_D12C	LBSTAT	Local Bus Status Register
0xFFFE_D128	LBC	Local Bus Control Register
0xFFFE_D124	MBAS	Target Memory Base Address Size Register
0xFFFE_D120	IOBAS	Target I/O Base Address Size Register
0xFFFE_D11C	PBACS	PCI Bus Arbiter Current State Register
0xFFFE_D118	CPCIBGS	Current PCI Bus Grant Status Register
0xFFFE_D114	CPCIBRS	Current PCI Bus Request Status Register
0xFFFE_D110	BM	Broken Master Register
0xFFFE_D10C	PBAPMIM	PCI Bus Arbiter/Park Master Interrupt Mask Register
0xFFFE_D108	PBAPMS	PCI Bus Arbiter/Park Master Status Register
0xFFFE_D104	PBAPMC	PCI Bus Arbiter/Park Master Control Register
0xFFFE_D100	REQ_TRACE	Request Trace Register
0xFFFE_D0EA	PWMNGSR	Power Management Support Register
0xFFFE_D0E0	PWMNGR	Power Management Register
0xFFFE_D0D0	TBL	Target Burst Length Register
0xFFFE_D0CC	SC_BE	Special Cycle Byte Enable Register
0xFFFE_D0C8	SC_MSG	Special Cycle Message Register
0xFFFE_D0C4	TLBIOMAR	Target Local Bus I/O Mapping Address Register
0xFFFE_D0C0	TLBMMAR	Target Local Bus Memory Mapping Address Register
0xFFFE_D0BC	TLBIAP	Target Local Bus IFIFO Address Register
0xFFFE_D0B8	TLBOAP	Target Local Bus OFIFO Address Register
0xFFFE_D0A8	PCIRRDT	PCI Read Retry Discard Timer Register
0xFFFE_D0A4	PCIRRT_CMD	PCI Read Retry Timer Command Register
0xFFFE_D0A0	PCIRRT	PCI Read Retry Tag Register
0xFFFE_D09C	TCCMD	Target Current Command Register
0xFFFE_D098	TIM	Target Interrupt Mask Register
0xFFFE_D094	TSTAT	Target Status Register
0xFFFE_D090	TC	Target Control Register
0xFFFE_D068	ILBIOMA	Initiator Local Bus I/O Mapping Address Register
0xFFFE_D064	ILBMMAR	Initiator Local Bus Memory Mapping Address Register
0xFFFE_D060	IPBIOMAR	Initiator PCI Bus I/O Mapping Address Register
0xFFFE_D05C	IPBMMAR	Initiator PCI Bus Memory Mapping Address Register
0xFFFE_D04C	RRT	Retry/Reconnect Timer Register
0xFFFE_D048	IIM	Initiator Interrupt Mask Register
0xFFFE_D044	ISTAT	Initiator Status Register
0xFFFE_D040	—	(Reserved)
0xFFFE_D03F	IL	PCI Interrupt Line Register

Table 4.2.1 TX3927 Register Mapping (4/6)

Address	Register Symbol	Register Name
0xFFFE_D03E	IP	PCI Interrupt Pin Register
0xFFFE_D03D	MG	Minimum Grant Register
0xFFFE_D03C	ML	Maximum Latency Register
0xFFFE_D037	CAPPTR	Capabilities Pointer
0xFFFE_D030	—	(Reserved)
0xFFFE_D02E	SSVID	Subsystem Vendor ID Register
0xFFFE_D02C	SVID	System Vendor ID Register
OxFFFE_D028	—	(Reserved)
0xFFFE_D014	MBA	Target Memory Base Address Register
0xFFFE_D010	IOBA	Target I/O Base Address Register
0xFFFE_D00F	CLS	Cache Line Size Register
0xFFFE_D00E	MLT	Master Latency Timer Register
0xFFFE_D00D	HT	Header Type Register
0xFFFE_D00C	—	(Reserved)
0xFFFE_D00B	RID	Revision ID Register
0xFFFE_D00A	RLPI	Register Level Programming Interface Register
0xFFFE_D009	SCC	Sub-Class Code Register
0xFFFE_D008	CC	Class Code Register
0xFFFE_D006	PCICMD	PCI Command Register
0xFFFE_D004	PCISTAT	PCI Status Register
0xFFFE_D002	VID	Vendor Identification Register
0xFFFE_D000	DID	Device Identification Register
IRC		
0xFFFE_C0A0	IRCSR	Interrupt Current Status Register
0xFFFE_C080	IRSSR	Interrupt Source Status Register
0xFFFE_C060	IRSCR	Interrupt Status Control Register
0xFFFE_C040	IRIMR	Interrupt Mask Register
0xFFFE_C02C	IRILR7	Interrupt Level Register 7
0xFFFE_C028	IRILR6	Interrupt Level Register 6
0xFFFE_C024	IRILR5	Interrupt Level Register 5
0xFFFE_C020	IRILR4	Interrupt Level Register 4
0xFFFE_C01C	IRILR3	Interrupt Level Register 3
0xFFFE_C018	IRILR2	Interrupt Level Register 2
0xFFFE_C014	IRILR1	Interrupt Level Register 1
0xFFFE_C010	IRILR0	Interrupt Level Register 0
0xFFFE_C008	IRCR1	Interrupt Control Mode Register 1
0xFFFE_C004	IRCR0	Interrupt Control Mode Register 0
0xFFFE_C000	IRCER	Interrupt Control Enable Register

Table 4.2.1 TX3927 Register Mapping (5/6)

Address	Register Symbol	Register Name
DMA		
0xFFFE_B0A8	—	(Reserved)
0xFFFE_B0A4	MCR	DMA Master Control Register
0xFFFE_B0A0	TDHR	DMA Temporary Data Holding Register
0xFFFE_B09C	DBR7	DMA Data Buffer Register 7
0xFFFE_B098	DBR6	DMA Data Buffer Register 6
0xFFFE_B094	DBR5	DMA Data Buffer Register 5
0xFFFE_B090	DBR4	DMA Data Buffer Register 4
0xFFFE_B08C	DBR3	DMA Data Buffer Register 3
0xFFFE_B088	DBR2	DMA Data Buffer Register 2
0xFFFE_B084	DBR1	DMA Data Buffer Register 1
0xFFFE_B080	DBR0	DMA Data Buffer Register 0
0xFFFE_B07C	CSR3	DMA Status Register Channel 3
0xFFFE_B078	CCR3	DMA Control Register Channel 3
0xFFFE_B074	DAI3	DMA Destination Address Increment Register Channel 3
0xFFFE_B070	SAI3	DMA Source Address Increment Register Channel 3
0xFFFE_B06C	CNAR3	DMA Count Register Channel 3
0xFFFE_B068	DAR3	DMA Destination Address Register Channel 3
0xFFFE_B064	SAR3	DMA Source Address Register Channel 3
0xFFFE_B060	CHAR3	DMA Chain Address Register Channel 3
0xFFFE_B05C	CSR2	DMA Status Register Channel 2
0xFFFE_B058	CCR2	DMA Control Register Channel 2
0xFFFE_B054	DAI2	DMA Destination Address Increment Register Channel 2
0xFFFE_B050	SAI2	DMA Source Address Increment Register Channel 2
0xFFFE_B04C	CNAR2	DMA Count Register Channel 2
0xFFFE_B048	DAR2	DMA Destination Address Register Channel 2
0xFFFE_B044	SAR2	DMA Source Address Register Channel 2
0xFFFE_B040	CHAR2	DMA Chain Address Register Channel 2
0xFFFE_B03C	CSR1	DMA Status Register Channel 1
0xFFFE_B038	CCR1	DMA Control Register Channel 1
0xFFFE_B034	DAI1	DMA Destination Address Increment Register Channel 1
0xFFFE_B030	SAI1	DMA Source Address Increment Register Channel 1
0xFFFE_B02C	CNAR1	DMA Count Register Channel 1
0xFFFE_B028	DAR1	DMA Destination Address Register Channel 1
0xFFFE_B024	SAR1	DMA Source Address Register Channel 1
0xFFFE_B020	CHAR1	DMA Chain Address Register Channel 1
0xFFFE_B01C	CSR0	DMA Status Register Channel 0
0xFFFE_B018	CCR0	DMA Control Register Channel 0
0xFFFE_B014	DAI0	DMA Destination Address Increment Register Channel 0
0xFFFE_B010	SAI0	DMA Source Address Increment Register Channel 0
0xFFFE_B00C	CNAR0	DMA Count Register Channel 0
0xFFFE_B008	DAR0	DMA Destination Address Register Channel 0
0xFFFE_B004	SAR0	DMA Source Address Register Channel 0
0xFFFE_B000	CHAR0	DMA Chain Address Register Channel 0

Table 4.2.1 TX3927 Register Mapping (6/6)

Address	Register Symbol	Register Name
ROMC		
0xFFFE_901C	RCCR7	ROM Channel Control Register 7
0xFFFE_9018	RCCR6	ROM Channel Control Register 6
0xFFFE_9014	RCCR5	ROM Channel Control Register 5
0xFFFE_9010	RCCR4	ROM Channel Control Register 4
0xFFFE_900C	RCCR3	ROM Channel Control Register 3
0xFFFE_9008	RCCR2	ROM Channel Control Register 2
0xFFFE_9004	RCCR1	ROM Channel Control Register 1
0xFFFE_9000	RCCR0	ROM Channel Control Register 0
SDRAMC		
0xFFFE_8034	SDCSMRS2	SGRAM Load Color Register
0xFFFE_8030	SDCSMRS1	SGRAM Load Mask Register
0xFFFE_802C	SDCCMD	SDRAM Command Register
0xFFFE_8028	SDCTR3	SDRAM Timing Register 3 (for SMROM)
0xFFFE_8024	SDCTR2	SDRAM Timing Register 2 (for DIMM flash memory)
0xFFFE_8020	SDCTR1	SDRAM Timing Register 1 (for SDRAM/SGRAM)
0xFFFE_801C	SDCCR7	SDRAM Channel Control Register 7
0xFFFE_8018	SDCCR6	SDRAM Channel Control Register 6
0xFFFE_8014	SDCCR5	SDRAM Channel Control Register 5
0xFFFE_8010	SDCCR4	SDRAM Channel Control Register 4
0xFFFE_800C	SDCCR3	SDRAM Channel Control Register 3
0xFFFE_8008	SDCCR2	SDRAM Channel Control Register 2
0xFFFE_8004	SDCCR1	SDRAM Channel Control Register 1
0xFFFE_8000	SDCCR0	SDRAM Channel Control Register 0

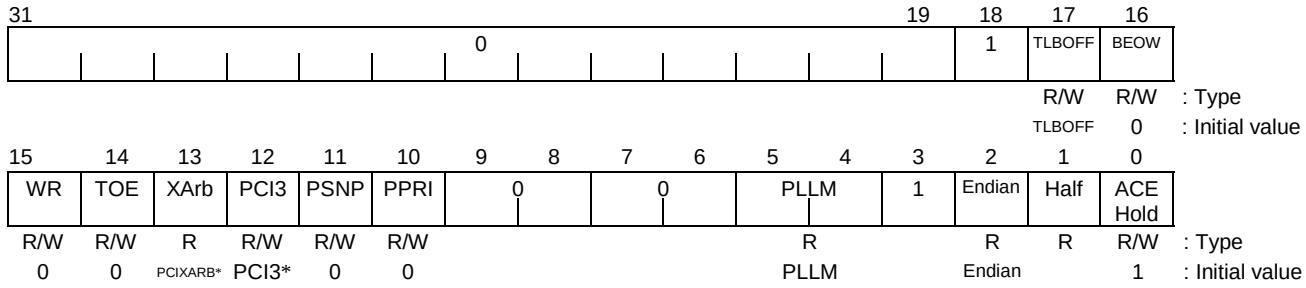
Note: All addresses listed above comply with big-endian address formatting.

5. Configuration

This chapter describes the four registers that are mapped to addresses 0xFFFE_E000 to 0xFFFE_E00F.

5.1 Chip Configuration Register (CCFG) 0xFFFE_E000

The TX3927 has software-configurable functions and pins. These functions and pins are set up using the chip configuration register (CCFG) and the pin configuration register (PCFG). They are also subjected to boot configuration using boot pins.



Bits	Mnemonic	Field Name	Description
17	TLBOFFB	TLB Off	TLB Off Enables or disables TLB operation. Latched from ADDR[19] during a reset. This bit is write-enabled but its value should not be changed after a reset. 0: TLB off (disable) 1: TLB on (enable)
16	BEOW	Bus Error	Bus Error on Write 0: Clear the bit. 1: Indicate that a bus error has occurred during a write operation by the TX39/H2 core.
15	WR	Watchdog Timer Reset	Watchdog Timer for Reset/NMI Specifies the action to be taken when TMR2 generates a watchdog timer interrupt. 0: Generates a non-maskable interrupt. 1: Causes a reset.
14	TOE	Timeout Enable for Bus Error	Timeout Enable for Bus Error Enables or disables the bus error timeout function. 0: Disable the timeout function. 1: Enable the timeout function. Refer to “7.2.1 Bus error” for details.
13	PCIXARB	PCI Arbiter	Internal or External PCI arbiter. Indicates whether the internal arbiter is used as a PCI arbiter. Latched from ADDR[11] during a reset. 0: External arbiter 1: Internal arbiter
12	PCI3B	PCI Clock Divider	PCI Clock Divider Control Specifies the PCI controller clock frequency. Latched from ADDR[17] during a reset. 0: The PCI clock frequency is 1/3 of the G-Bus clock frequency. 1: The PCI clock frequency is 1/2 of the G-Bus clock frequency.

Figure 5.1.1 Chip Configuration Register (1/2)

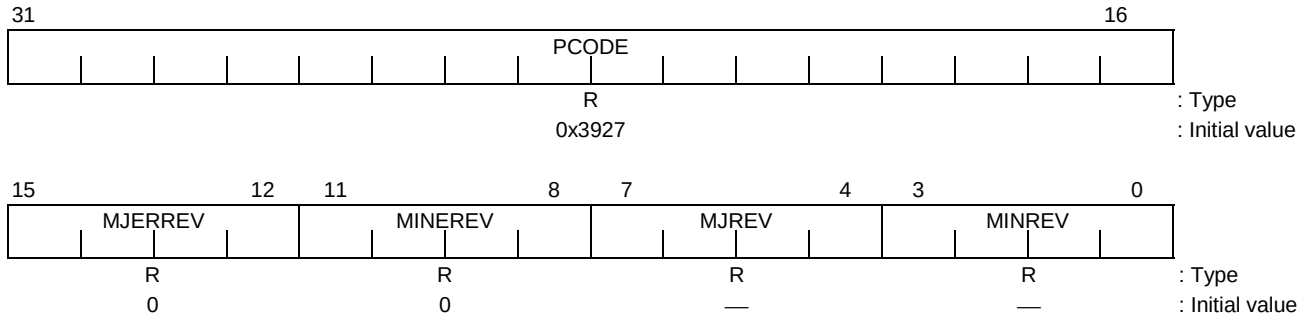
Bits	Mnemonic	Field Name	Description
11	PSNP	PCI Snoop	PCI Bus Request Snoop Enables or disables the TX39/H2 core cache snoop function while the PCI has bus mastership. If data cache is used in write back mode, the snoop function must be disabled. 0: Disable the cache snoop function. 1: Enable the cache snoop function.
10	PPRI	PCI Arbitration Priority	Select PCI Arbitration Priority Specifies the priority of the DMAC to PCI. 0: DMA has priority over PCI in arbitration. 1: Setting prohibited.
5 : 4	PLLM	PLL Multiplier	PLL Multiplier Indicates the PLL multiplier value. Latched from ADDR[3:2] during a reset. When PLLM[1] is set to “1”, a crystal having a frequency in the range of 6.25 to 8.33 MHz should be used. When PLLM[1] is set to “0”, a crystal oscillator should be used and connected to the XIN pin as a clock input signal. The XOUT pin should be left open. 00: Don’t use. 01: ×2 10: Don’t use.s 11: ×16
2	ENDIAN	Endian	Current Endian Setting of G-Bus Indicates the endian mode. Latched from ADDR[14] during a reset. 0: Little endian 1: Big endian The setting of this bit is not affected by the state of the RE (reverse endian) bit of the TX39/H2 core status register.
1	HALF	Half-Speed	System Clock Half-Speed Mode Indicates the SYSCLK output frequency. The state of this bit is determined from the values of ADDR[15] and ADDR[9:8] during a reset. Half speed mode is selected when ADDR[15] is “0” or when ADDR[9:8] is “10”. Otherwise, full speed mode is selected. 0: Full speed (same frequency as G-Bus clock frequency) 1: Half speed (half of G-Bus clock frequency)
0	ACEHOLD	ACE Hold	ACE* Address Hold Specifies whether the address is held for one clock cycle after ACE* is deasserted. 0: The address changes in the same clock cycle as ACE*. In this mode, an address hold time of 1 ns is guaranteed. 1: The address is held for one clock cycle after the rising edge of ACE*.

Note: For the bits other than those defined above, write the values shown in the figure.

Figure 5.1.1 Chip Configuration Register (2/2)

5.1.1 Chip Revision ID Register (CRIR) 0xFFFE_E004

The chip revision ID register shows the revision of the TX3927.

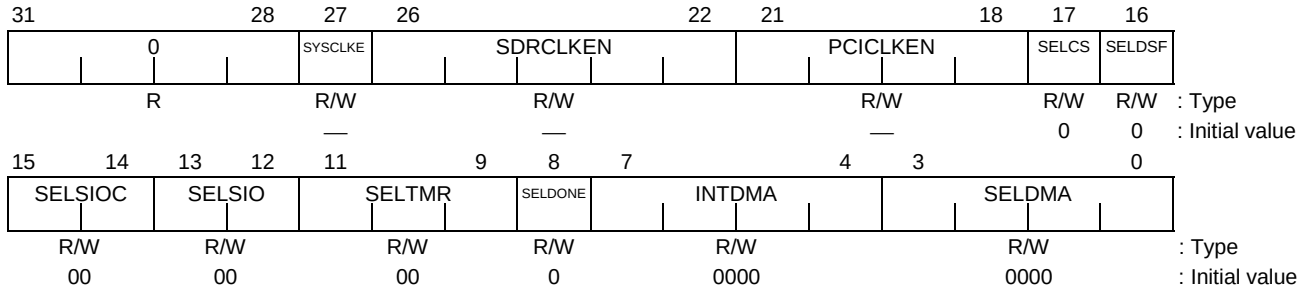


Bits	Mnemonic	Field Name	Description
31 : 16	PCODE	Product Code	Product Code (fixed value: 0x3927) Indicates the product number.
15 : 12	MJERREV	Major Extra Code	Major Extra Code Implementation Revision (fixed value: 0x0) Indicates the major extra code.
11 : 8	MINEREV	Minor Extra Code	Minor Extra Code Implementation Revision (fixed value: 0x0) Indicates the minor extra code.
7 : 4	MJREV	Major Revision	Major Implementation Revision Indicates the major revision. Contact Toshiba technical staff for information on the value.
3 : 0	MINREV	Minor Revision	Minor Implementation Revision Indicates the minor revision. Contact Toshiba technical staff for information on the value.

Figure 5.1.2 Chip Revision ID Register

5.1.2 Pin Configuration Register (PCFG) 0xFFFE_E008

The TX3927 has software-configurable functions and pins. These functions and pins are set up using the chip configuration register (CCFG) and the pin configuration register (PCFG). Refer also to "3.3 Pin Multiplexing" for information on pin multiplexing.



Bits	Mnemonic	Field Name	Description
27	SYSCLKEN	System Clock Enable	System Clock Enable Enables or disables output from the SYSCLK pin. Latched from ADDR[5] at the rising edge of RESET. After a reset, the bit may be read and written by the CPU. 1: Enable SYSCLK. 0: Disable SYSCLK.
26 : 22	SDRCLKEN [4 : 0]	SDRAM Clock Enable	SDRAM Clock Enable Individually enables or disables SDRAM clock output on each of the SDCLK[4:0] pins. Latched from ADDR[4] at the rising edge of RESET. After a reset, the bits may be read and written individually by the CPU. 1: Enable SDRCLK. 0: Disable SDRCLK.
21 : 18	PCICLKEN [3 : 0]	PCI Clock Enable	PCI Clock Enable Individually enables or disables PCI clock output on each of the PCICLK[3:0] pins. Latched from ADDR[18] at the rising edge of RESET. After a reset, the bits may be read and written individually by the CPU. 1: Enable PCICLK. 0: Disable PCICLK.
17	SELCS	Select CS	Select DMA/SDCS_CE Function (initial value: 0) Used in conjunction with SELDMA[1] to select the functions of DMAREQ[1]/PIO[11]/SDCS_CE[7] and DMAACK[1]/PIO[10]/SDCS_CE[6]. Refer to "3.3 Pin Multiplexing" and Table 5.1.1 for details of pin multiplexing.
16	SELDSF	Select DSF	Select DSF Function (initial value: 0) Used in conjunction with SELSIOC[1] to select the function of the RTS*[1]/PIO[1]/DSF pin. Refer to "3.3 Pin Multiplexing" and Table 5.1.1 for details of pin multiplexing.
15 : 14	SELSIOC [1 : 0]	Select SIO Control	Select SIO Control Pins (initial value: 00) SELSIOC[1]: Used in conjunction with SELDSF to select the functions of CTS[1]/PIO[2] and RTS[1]/PIO[1]/DSF. SELSIOC[0]: Used in conjunction with SELDSF to select the functions of CTS[0]/INT[5] and RTS[0]/INT[4]. Refer to "3.3 Pin Multiplexing" and Table 5.1.1 for details of pin multiplexing.

Figure 5.1.3 Pin Configuration Register (1/2)

Bits	Mnemonic	Field Name	Description
13 : 12	SELSIO [1 : 0]	Select SIO Function	Select SIO/PIO function select (initial value: 00) SELSIO[1]: Selects the functions of RXD[1]/PIO[6] and TXD[1]/PIO[5]. SELSIO[0]: Selects the functions of RXD[0]/PIO[4] and TXD[0]/PIO[3]. 0: PIO pin 1: SIO pin Refer to “3.3 Pin Multiplexing” and Table 5.1.1 for details of pin multiplexing.
11 : 9	SELTMR [2 : 0]	Select TMR Function	Select TIMER/PIO Function Select (initial value: 000) SELTMR[2]: Used in conjunction with SELDONE to select the function of DMADONE/TIMER[0]/PIO[7]. SELTMR[1]: Used in conjunction with SELDMA[3] to select the function of DMAREQ[3]/TIMER[1]/PIO[15]. SELTMR[0]: Used in conjunction with SELDMA[3] to select the function of DMAACK[3]/TIMER[0]/PIO[14]. Refer to “3.3 Pin Multiplexing” and Table 5.1.1 for details of pin multiplexing.
8	SELDONE	Select DMADONE	Select DMADONE*/PIO (initial value: 0) Used in conjunction with SELTMR[2] to select the function of DMADONE/TIMER[0]/PIO[7]. Refer to “3.3 Pin Multiplexing” and Table 5.1.1 for details of pin multiplexing.
7 : 4	INTDMA [3 : 0]	Select Internal DMA Request	Internal/External DMA Source connection (initial value: 0000) Specifies whether the external I/O or internal SIO is used as a DMA request source. INTDMA[3]: 0: DMAREQ/ACK[3] connects to external pins. 1: DMAREQ/ACK[3] connects to SIO[1] SITXDREQ/ACK. INTDMA[2]: 0: DMAREQ/ACK[2] connects to external pins. 1: DMAREQ/ACK[2] connects to SIO[0] SITXDREQ/ACK. INTDMA[1]: 0: DMAREQ/ACK[1] connects to external pins. 1: DMAREQ/ACK[1] connects to SIO[1] SIRXDREQ/ACK. INTDMA[0]: 0: DMAREQ/ACK[0] connects to external pins. 1: DMAREQ/ACK[0] connects to SIO[0] SIRXDREQ/ACK.
3 : 0	SELDMA [3 : 0]	Select DMA Function	Select DMA/PIO/TIMER Function (initial value: 0000) SELDMA[3]: Used in conjunction with SELTMR[1:0] to select the functions of DMAREQ[3]/PIO[15]/TIMER[1] and DMAACK[3]/PIO[14]/TIMER[0]. SELDMA[2]: Used to select the functions of DMAREQ[2]/PIO[13] and DMAACK[2]/PIO[12]. SELDMA[1]: Used in conjunction with SELCS to select the functions of DMAREQ[1]/PIO[11]/SDCS_CE[7] and DMAACK[1]/PIO[10]/ SDCS_CE[6]. SELDMA[0]: Used to select the functions of DMAREQ[0]/PIO[9] and DMAACK[0]/PIO[8]. Refer to “3.3 Pin Multiplexing” and Table 5.1.1 for details of pin multiplexing.

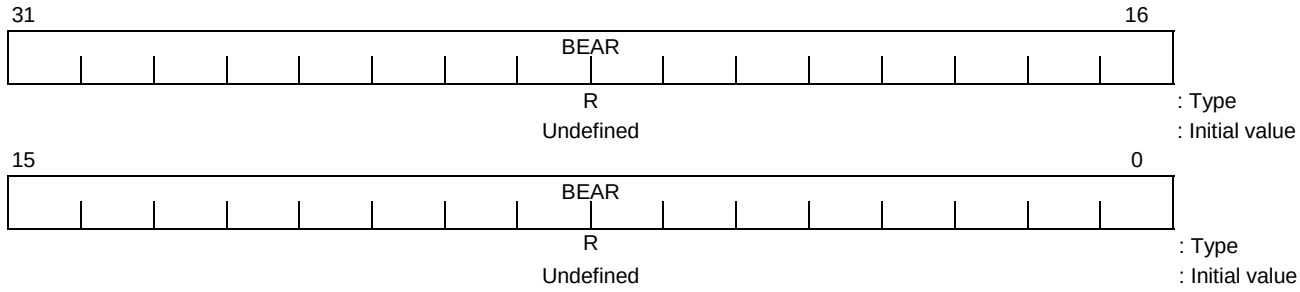
Note: For the bits other than those defined above, write the values shown in the figure.

Figure 5.1.3 Pin Configuration Register (2/2)

Table 5.1.1 Pin Functions Corresponding to the Settings of PCFG Control Register Bits

Pin/Function	Pin/Function	PCFG Control Bits		
DMAREQ [3]	DMAACK [3]	SELDMA [3]	SELTMR [1]	SELTMR [0]
PIO [15]	PIO [14]	0	0	0
PIO [15]	TIMER [0]	0	0	1
TIMER [1]	PIO [14]	0	1	0
TIMER [1]	TIMER [0]	0	1	1
DMAREQ [3]	DMAACK [3]	1	x	x
DMAREQ [2]	DMAACK [2]	SELDMA [2]		
PIO [13]	PIO [12]	0		
DMAREQ [2]	DMAACK [2]	1		
DMAREQ [1]	DMAACK [1]	SELDMA [1]	SELCS	
PIO [11]	PIO [10]	0	0	
SDCS_CE [7]	SDCS_CE [6]	0	1	
DMAREQ [1]	DMAACK [1]	1	x	
DMAREQ [0]	DMAACK [0]	SELDMA [0]		
PIO [9]	PIO [8]	0		
DMAREQ [0]	DMAACK [0]	1		
DMADONE*		SELDONE	SELTMR [2]	
PIO [7]		0	0	
TIMER [0]		0	1	
DMADONE*		1	x	
RXD [1]	TXD [1]	SELSIO [1]		
PIO [6]	PIO [5]	0		
RXD [1]	TXD [1]	1		
RXD [0]	TXD [0]	SELSIO [0]		
PIO [4]	PIO [3]	0		
RXD [0]	TXD [0]	1		
CTS* [1]	RTS* [1]	GDBGE*	SELSIOC [1]	SELDSF
PIO [2]	PIO [1]	1	0	0
PIO [2]	DSF	1	0	1
CTS* [1]	RTS* [1]	1	1	0
GSDAO [1]	GPCST [3]	0	x	x
CTS* [0]	RTS* [0]	SELSIOC [0]		
INT [5]	INT [4]	0		
CTS* [0]	RTS* [0]	1		

5.1.3 Timeout Error Address Register (TEAR) 0xFFFE_E00C



Bits	Mnemonic	Field Name	Description
31 : 0	BEAR	Bus Error Address	Bus Error Address This register latches the address on the bus when a bus error occurs.

Figure 5.1.4 Timeout Error Address Register

6. Clocks

6.1 Clock Generator

The CPU clock of the TX3927 uses a frequency that is a multiple of the externally input clock frequency. The multiplier value can be chosen between two and sixteen, using boot signals PLLM[1:0] (ADDR[3:2]). If PLLM[1:0] = "01" at the rising edge of the CLKEN signal, the multiplier value is two. If PLLM[1:0] = "11" at the rising edge of the CLKEN signal, the multiplier value is sixteen.

The TX3927 has five internal clock sources: CORECLK, GBUSCLK, SYSCLK, PCICLK, and IMCLK. Table 6.1.1 lists their frequencies and functions.

Table 6.1.1 Frequencies and Functions of TX3927 Internal Clocks

Clock			Operating Frequency fc = (Externally Input Clock Frequency) × (Multiplier)		Function	Block Used
			Full Speed Bus Mode	Half Speed Bus Mode		
CORECLK			fc	fc	Reference clock for TX39/H2 CPU core	TX39/H2 core
GBUSCLK			fc/2	fc/2	G-Bus reference clock	DMAC, SDRAMC, IRC, part of EBIF, part of ROMC, PCIC (G-Bus interface)
SYSCLK			fc/2	fc/4	G-Bus clock (reference clock for half speed bus mode)	ROMC, part of EBIF
PCICLK	PCICLKEN = High	PCI3* = Low	fc/6	fc/6	PCI bus reference clock	PCIC (PCI bus interface)
		PCI3* = High	fc/4	fc/4		
	PCICLKEN = Low		PCICLK input	PCICLK input		
IMCLK			fc/4	fc/4	IM-Bus reference clock	SIO, TMR

6.2 System Control Clock (SYSCLK)

The internal clock generator of the TX39/H2 core generates this signal. The system control clock (SYSCLK) operates 1/2 or 1/3 the frequency of the TX39/H2 processor core. SYSCLK provides timing for system control interface signals.

The boot setup value which determines the frequency of SYSCLK is captured into the TX3927 in synchronization with the internal clock. Therefore, if half speed mode is selected, SYSCLK may operate at full speed for several clocks before entering half speed mode.

6.3 Power-Down Mode

The TX3927 provides power-down mode, in which it enters a standby state with the internal clocks stopped, including the PLL.

6.3.1 Operation

The TX3927 uses an externally input clock (XIN) to control the power-down logic because power-down causes the internal clock signals from the clock generator to be stopped. Note that the input clock used to control the power-down logic is not frequency-multiplied by the PLL of the clock generator. When the power-down trigger bit (PDN) of the power-down control register (PDCR) is set, the power-down logic waits for 20 (if the multiplier is 16) or 132 (if the multiplier is 2) XIN clock cycles before deasserting the CLKEN control input signal to stop the output of clock signals. Then, the power-down logic waits 16 (if the multiplier is 16) or 128 (if the multiplier is 2) XIN clock cycles before turning off the PLL. All on-chip clocks will be halted with the exception of the PCI clock (if it is supplied by an external device) and the XIN clock. Once all clocks are shut down, they will remain in this state until an interrupt is generated from any of the specified interrupt pins, including NMI. The power-down mask bits (PDCR.PDNMSK[6:0]) of the power-down control register determine which external interrupt sources are used for resume from power-down mode. Each mask bit corresponds to an interrupt source. Setting a bit enables the corresponding interrupt source as an interrupt for terminating the power-down state. If an interrupt is generated, the power-down logic turns on the PLL, and then, after the delay specified with the power-up time counter (PUTCV) of the power-down control register (PDCR), enables the output of internal clocks, causing the system to resume normal operation. Table 6.3.1 shows the delay required for power-down in terms of the number of SDCLK cycles. Table 6.3.2 shows the delay required for power-up (value added to the PCVTV count) in terms of the number of externally input clock cycles.

Note: Poorly written software may unexpectedly cause the power-down logic to disable the clocks, with no easy way to re-enable them. If software disables all interrupt inputs, then there is no way to exit power-down mode other than a special reset sequence or a power cycle. The special reset cycle requires the same reset timing as the power-up reset. After the power-down trigger bit is set, any interrupt received will cause the TX3927 to exit from power-down mode and restore normal operation with the clocks operating. If a wakeup event occurs before the PLL is disabled, clocks will be re-enabled immediately after three clock cycles.

Table 6.3.1 Power-Down Delay

PLLM [1 : 0]	Number of Clock Cycles Required Before the Output Clocks Stop
01	132 (Xin clock cycles) + 12 to 24 (CORECLK cycles)
11	20 (Xin clock cycles) + 12 to 24 (CORECLK cycles))

Table 6.3.2 Power-Up Delay

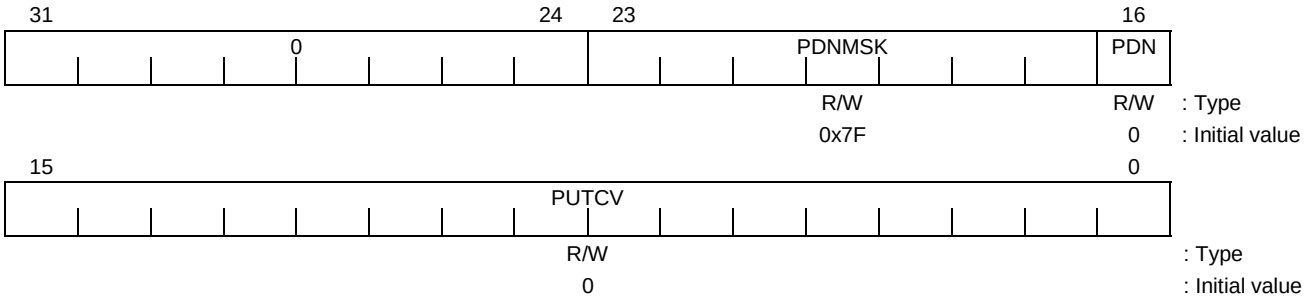
PLLM [1 : 0]	Number of Clock Cycles Required Before the Output Clocks Restart
01	PUTCV (Xin clock cycles) + 24 (CORECLK cycles)
11	PUTCV (Xin clock cycles) + 24 (CORECLK cycles)

Transition to power-down mode is controlled using a program. The program must be executed from cache to ensure that there are no active bus cycles during the power-down or power-up process.

6.3.2 Register

The power-down control register (PDCR) is used to control the power-down process. Bit 16 of this register is the power-down trigger bit (PDN), which is cleared to "0" by a reset. A subsequent transition from "0" to "1" will initiate the power-down sequence. The bit is not cleared automatically, so software must clear this bit before starting another power-down sequence. Bits [23:17] (PDNMSK[6:0]) contain a 7-bit mask for the interrupt input pins. Bit 17 masks external interrupt 0, bit 18 masks external interrupt 1, and so on. A "0" value in any mask bit prevents that interrupt pin from starting the power-up sequence while a "1" allows it. Bits [15:0] contain a value that determines the dominant delay between the PLL being enabled and the input clocks from the clock generator being activated. The total power-up delay is the sum of this value plus 24 XIN clock cycles.

6.3.2.1 Power-Down Control Register (PDCR) 0xFFFE_E010



Bits	Mnemonic	Field Name	Description
23 : 17	PDNMSK [6 : 0]	Power-Down Mask	Power Down Mask bit (initial value: 0x7F) Specifies which external interrupt signals are used to terminate power-down mode. A bit is allocated to each interrupt source. Setting 1 enables the corresponding interrupt signal to terminate power-down mode. PDNMSK [6] = NMI PDNMSK [5] = INT [5] PDNMSK [4] = INT [4] PDNMSK [3] = INT [3] PDNMSK [2] = INT [2] PDNMSK [1] = INT [1] PDNMSK [0] = INT [0] 1: Enable interrupt. 0: Disable interrupt
16	PDN	Power-Down Trigger	Power Down Trigger (initial value: 0) This bit is a trigger for the power-down sequence. It is cleared to "0" upon reset and a "0" to "1" transition will initiate the power-down sequence. This bit does not clear to "0" automatically when power-down mode is terminated. Clear the bit to "0" before starting a next power-down sequence.
15 : 0	PUTCV	Power-Up Time Counter	Power Up Time Counter Value (initial value: 0x0000) Specifies the delay between the PLL being enabled and the outputs of the clock generator being enabled. The total wait time is this value plus 3.5 XIN clock cycles. Ensure that the power-up sequence is performed when the XIN clock is stable. XIN can be 1/16 or 1/2 of the CPU core clock depending on the selected PLL multiplier value.

Note: For the bits other than those defined above, write the values shown in the figure.

Figure 6.3.1 Power-Down Control Register

7. Bus Operation

In the TX3927, the TX39/H2 CPU core normally operates as the bus master, but the DMAC and PCIC can also acquire bus mastership to initiate bus operation. The bus master accesses external memory and other devices via the memory controllers in the TX3927.

7.1 Bus Mastership

In the TX3927, the DMAC and PCIC can become the bus master as well as the TX39/H2 CPU core. Although the TX39/H2 core normally operates as the bus master, when necessary, the DMAC and PCIC can also acquire internal bus mastership. The DMAC and PCIC operate as the bus master under the following conditions:

- DMAC: When performing I/O to memory, memory to I/O, or memory to memory DMA transfer.
- PCIC: When performing memory to PCI or PCI to memory data transfer.
  The PCIC uses the TX3927's built-in memory controllers (SDRAMC and ROMC) to access memory.

7.1.1 Snoop Function

The DMAC and PCIC can use the snoop function.

- When cache is used in write back mode

  The snoop function is not available. Set the CCFG.PSNP and CCRn.SNOP bits to "0".

- When cache is used in write through mode

  The snoop function is available. To use the snoop function, set the CCFG.PSNP and CCRn.SNOP bits to "1".

7.1.2 Relationship Between the Endian Mode and Data Bus

The TX3927 supports both big endian and little endian modes. Which mode is used depends on the following:

- Initial setting signal ENDIAN (ADDR[14])
- LE bit of DMAC MCR register (bit 2)

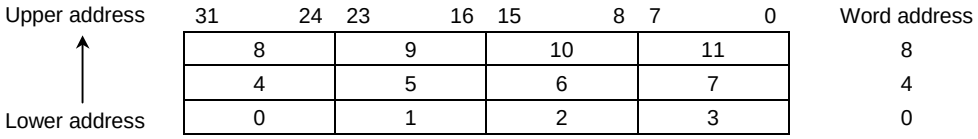
The above signal and bit must specify the same endian mode.

The PCI bus also supports the PCIC byte swap function, providing the following choices:

- Straight data output without swap
- Byte swap

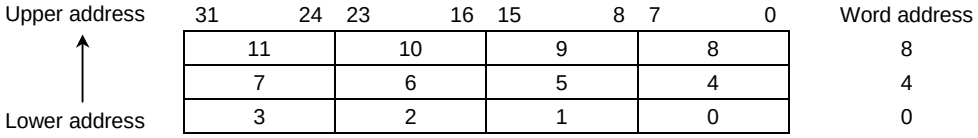
For details, refer to "12.4.12 Byte swap function".

The following describes the data structure in registers and memory. The TX3927 supports 32-bit word, 16-bit half word, and 8-bit byte data sizes. The byte sequence depends on the endian mode. Figure 7.1.1 shows the byte sequence within a word and the sequence of multiple words in little endian mode.



- Byte 0 is the highest byte (bits 31-24).
- The address of the word is specified with the address of the highest byte (byte 0).

(a) Big endian



- Byte 0 is the lowest byte (bits 7-0).
- The address of the word is specified with the address of the lowest byte (byte 0).

(b) Little endian

Figure 7.1.1 Big Endian and Little Endian

The TX3927 supports 16-bit and 32-bit memory bus sizes. Table 7.1.1 shows the relationship among the data size, bus size, endian mode, and data positions.

Table 7.1.1 Relationship among Data Size, Bus Size, Endian Mode, and Data Positions

Data Size	Start Address	Bus Size	Bus Cycle Address	Data							
				Big Endian				Little Endian			
				DATA[31]-DATA[24]	DATA[23]-DATA[16]	DATA[15]-DATA[8]	DATA[7]-DATA[0]	DATA[31]-DATA[24]	DATA[23]-DATA[16]	DATA[15]-DATA[8]	DATA[7]-DATA[0]
8 bits	4n + 0	16 bits	4n + 0	xxxx	xxxx	b7 – b0	xxxx	xxxx	xxxx	xxxx	b7 – b0
		32 bits	4n + 0	b7 – b0	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	b7 – b0
	4n + 1	16 bits	4n + 0	xxxx	xxxx	xxxx	b7 – b0	xxxx	xxxx	b7 – b0	xxxx
		32 bits	4n + 0	xxxx	b7 – b0	xxxx	xxxx	xxxx	xxxx	b7 – b0	xxxx
	4n + 2	16 bits	4n + 2	xxxx	xxxx	b7 – b0	xxxx	xxxx	xxxx	xxxx	b7 – b0
		32 bits	4n + 0	xxxx	xxxx	b7 – b0	xxxx	xxxx	b7 – b0	xxxx	xxxx
	4n + 3	16 bits	4n + 2	xxxx	xxxx	xxxx	b7 – b0	xxxx	xxxx	b7 – b0	xxxx
		32 bits	4n + 0	xxxx	xxxx	xxxx	b7 – b0	b7 – b0	xxxx	xxxx	xxxx
16 bits	4n + 0	16 bits	4n + 0	xxxx	xxxx	b15 – b8	b7 – b0	xxxx	xxxx	b15 – b8	b7 – b0
		32 bits	4n + 0	b15 – b8	b7 – b0	xxxx	xxxx	xxxx	xxxx	b15 – b8	b7 – b0
	4n + 2	16 bits	4n + 2	xxxx	xxxx	b15 – b8	b7 – b0	xxxx	xxxx	b15 – b8	b7 – b0
		32 bits	4n + 0	xxxx	xxxx	b15 – b8	b7 – b0	b15 – b8	b7 – b0	xxxx	xxxx
32 bits	4n + 0	16 bits	4n + 0 4n + 2	xxxx xxxx	xxxx xxxx	b31 – b24 b15 – b8	b23 – b16 b7 – b0	xxxx xxxx	xxxx xxxx	b15 – b8 b31 – b24	b7 – b0 b23 – b16
		32 bits	4n + 0	b31 – b24	b23 – b16	b15 – b8	b7 – b0	b31 – b24	b23 – b16	b15 – b8	b7 – b0

7.2 Bus Operation

The TX3927 performs bus operation for the following four device controllers. Before attempting to access a particular external device, you must set the corresponding controller with the device's address space and the number of desired wait states. For details of bus operation, refer to the relevant chapters.

External Device	Controller	Detailed Description
SDRAM, SGRAM, SMROM, DIMM flash memory	SDRAMC	Chapter 8
ROM, SRAM, flash memory	ROMC	Chapter 9
Memory, I/O	DMAC	Chapter 10
PCI devices	PCIC	Chapter 12

The SDRAMC and ROMC operate in the same way regardless of whether the bus master is the TX39/H2, DMA controller (DMAC), or PCI controller (PCIC). The DMAC does not directly access external memory or I/O devices, but access them via the SDRAMC or ROMC. When the TX39/H2 or DMAC accesses any device on the PCI, the PCIC acts as the memory controller for the PCI bus. When an external PCI device accesses memory or other devices attached to the TX3927, the PCIC acts as the bus master and accesses the TX3927's local memory via the SDRAMC or ROMC. If the data cache is used in write back mode, however, the TX39/H2 core does not support the snoop function, so the snoop function must be disabled for the PCIC and DMAC.

Bus Master	Memory Controller	External Device
TX39/H2	SDRAMC, ROMC	Local memory, I/O
	PCIC (initiator)	PCI device
DMAC	SDRAMC, ROMC	Local memory, I/O
	PCIC (initiator)	PCI device
PCIC (target) (external PCI device)	SDRAMC, ROMC	Local memory, I/O

Set the address space for each memory controller to ensure that the SDRAMC, ROMC, and PCIC acting as memory controllers will not simultaneously access the same address space.

The program must ensure that the TX3927 will only initiate bus operation for accesses to the address areas which have been properly set up in the controllers listed above (or to TX3927 internal registers). Set the address area for the device you wish to access in the appropriate controller.

The TX3927 offers two bus speed modes: full speed bus mode and half speed bus mode. In full speed bus mode, the external bus operates at the same frequency as the G-Bus (1/2 the CPU frequency). In half speed bus mode, the external bus operates at 1/2 the G-Bus frequency (1/4 the CPU frequency). Boot signal CHANHS* (ADDR[15] pin) selects the speed of SYSCLK output. Full speed mode is selected if CHANHS* is high at the rising edge of RESET*. Half speed mode is selected if CHANHS* is low at the rising edge of RESET*. The ROMC also contains a register used to select either mode for each channel.

7.2.1 Bus Error

When the CCFG.TOE bit is set to 1 to enable bus cycle timeout, the TX3927 generates a bus error if the access is invalid and no acknowledgement is returned (there is no external pin for bus errors).

A bus error will occur if no acknowledgement is received within 512 G-bus clock cycles. A bus error (timeout error) occurs when an external ACK* is not asserted and when the TX3927 accesses an address that is not set in the appropriate memory controller.

When a bus error occurs during TX39/H2 core read operation, the core generates a bus error exception. When a bus error occurs during TX39/H2 core write operation, the core generates a non-maskable interrupt exception. When a bus error occurs during DMAC bus operation, the DMAC terminates transfer and sets the status bit. If interrupts are enabled, an interrupt is also generated.

When a bus error occurs during PCIC bus operation, the PCIC terminates the bus cycle normally and sets the status bit. If interrupts are enabled, an interrupt is also generated.

8. SDRAM Controller

8.1 Features

The SDRAM controller (SDRAMC) generates the required control signals to interface to SDRAM, DIMM flash memory, SMROM, or SGRAM. The SDRAM controller has eight channels that can operate independently. It supports memory sizes of up to 1 G-byte in various bus configurations.

Features of the SDRAM controller include:

(1) Clock frequency: 50 to 66 MHz (1/2 the frequency of the TX39/H2 core)

(2) Designed for timing of −10 or faster speed grade SDRAMs/SGRAMs and −15 (66 MHz) to −20 (50 MHz) speed grade SMROMs

(3) Eight independent memory channels. The chip select signals for channels 2 to 7 are, however, shared with ROM controller channels 2 to 7. Note that both the SDRAM and ROM channels cannot be enabled at the same time.

(4) Each memory channel configurable for SDRAM, DIMM SDRAM, DIMM flash memory, SMROM, or SGRAM. Supports JEDEC standard 100-pin and 168-pin memory DIMMs for SDRAM and 100-pin DIMMs for flash memory.

(5) Data bus width: 16-bit/32-bit selectable per channel (only 32-bit for SMROM and SGRAM)

(6) Critical word first access. This allows for improved performance under cache miss conditions where the desired data is not located on an aligned boundary. The word data that has caused a cache miss is read first and used by the CPU to process the instruction while the remaining cache line is refilled. The SDRAMC supports this feature for all memory types.

(7) Optionally boot after a reset from DIMM flash memory or SMROM. Optionally enable the row address matching feature for SMROM to eliminate time consuming activate commands.

(8) SDRAM/SGRAM memory access timing:
Single read: 8 cycles Burst read: 8 + (n – 1) cycles
Single write: 6 cycles Burst write: 6 + (n – 2) cycles
where n is the number of data transfers within a burst operation. With slow write bursts enabled, each additional access for write bursts adds 2 cycles.

(9) A given channel base address is located on an appropriate channel size boundary.
(This is the normal use; the hardware allows more options, but they are not recommended.)

(10) SDRAM/SGRAM and SMROM timing latencies are programmable. The timing parameters can be programmed according to the clock frequency and the speed grade of the device used, thus optimizing memory performance for a particular system.

(11) SDRAM/SGRAM burst length: 1 for 32-bit and 16-bit memory
SMROM burst length: 4 for 32-bit memory

Note: The SMROM burst length must be 4-word aligned except when the critical word first feature is enabled.

(12) Optionally increment the address by a programmable value during burst reads or writes. The default burst address increments by 1 word. This feature works with the DMA controller in single address mode to allow more efficient transfers. (This feature is not supported for SMROMs.)

(13) G-Bus burst lengths of 4, 8, 16, and 32 words. There is no alignment requirement for SDRAM, SGRAM, and DIMM flash memory so bursts that cross page, bank, and channel boundaries are supported. Bursts crossing a page are always 8 address bits or 256 words. A refresh cycle is held off during any boundary crossing. Bursts that begin outside of the memory address area and end inside or vice-versa might produce

unpredictable results possibly including a bus hang.

(14) Arbitrary byte write for single or burst writes. This feature is controlled with the BWE signal.

(15) Premature termination of an internal bus cycle with a bus error or GDRESET* will allow memory operation to complete with no loss of data. Termination of an internal bus cycle with RESET* will, however, terminate memory operation immediately.

(16) Programmable refresh period

(17) SDRAM/SGRAM refresh mode: Auto refresh or self-refresh

(18) Low power modes: Self-refresh or precharge power-down for SDRAM/SGRAM and low power mode for SMROM. Automatic exit from the power-down modes is supported.

(19) Addressing mode: Sequential only

(20) Auto precharge is used for burst writes only. Precharge commands are used for other types of access.

(21) Serial Presence Detect (SPD) to determine the configuration of DIMMs may be implemented by using two programmable I/O (PIO) pins. The user should implement the serial protocol through software.

(22) High fanout

In order to support additional loading on the data bus, two selectable data read-back paths and an optional slow write burst mode are supported. For reads, the user can choose between the path where data is latched using a feedback clock to better maintain timing coherency with the data and the path that bypasses that latching stage. For slow burst writes, two clocks are used for each write instead of one.

(23) SDR (Single Data Rate) SDRAMs/SGRAMs are supported. DDR (Double Data Rate) devices are not supported.

8.2 SDRAM Block Diagram

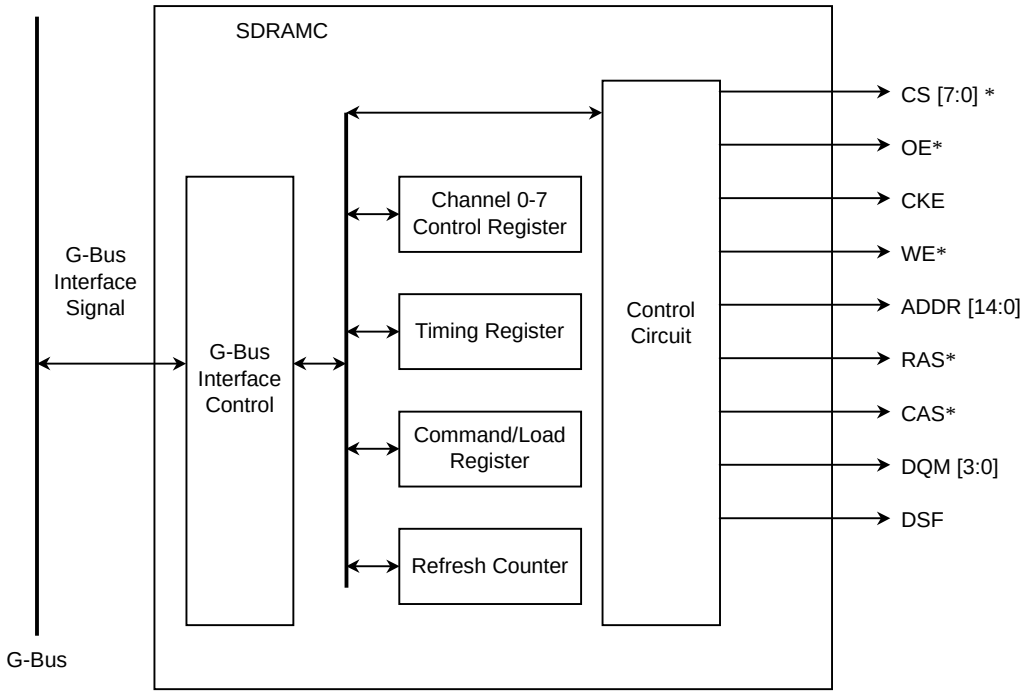


Figure 8.2.1 TX3927 SDRAMC Block Diagram

8.3 Memory Configuration

Table 8.3.1 shows the SDRAM, SMROM, and SGRAM configurations the SDRAM controller supports.

Table 8.3.1 Supported Memory Configurations

	Memory Size	Banks	Row Addresses	Column Addresses
SDRAM	16 Mbit			
	1 M × 16	2	11	8
	2 M × 8	2	11	9
	4 M × 4*	2	11	10
	64 Mbit			
	2 M × 32	2	11	9
	2 M × 32	2	12	8
	4 M × 16	2	11	10
	4 M × 16	2	13	8
	8 M × 8	2	13	9
	16 M × 4*	2	13	10
	64 Mbit			
	2 M × 32	4	11	8
	4 M × 16	4	12	8
	8 M × 8	4	12	9
	16 M × 4*	4	12	10
	128 Mbit			
	4 M × 32	4	12	8
	8 M × 16	4	12	9
	16 M × 8	4	12	10
	32 M × 4*	4	12	11
	256 Mbit			
	8 M × 32	4	13	8
	16 M × 16	4	13	9
	32 M × 8	4	13	10
	64 M × 4*	4	13	11
SMROM	32 Mbit			
	1 M × 32	NA	13	7
	64 Mbit			
	2 M × 32	NA	14	7
	2 M × 32	NA	13	8
SGRAM	8 Mbit			
	256 K × 32	2	9	8
	16 Mbit			
	512 K × 32	2	10	8

* These configurations are supported logically but there are bus loading issues that might preclude their use.

Table 8.3.2 Maximum Memory (using x8 devices or larger)

Memory Type	Maximum/Channel (MB)	Maximum for 8 Channels
SDRAM	128	1 GB
DIMM FLASH	32	256 MB
SMROM	8	64 MB
SGRAM	2	16 MB

8.4 Registers

8.4.1 Register Mapping

Table 8.4.1 SDRAM Control Registers

Address	Register Symbol	Register Name
0xFFFE_8000	SDCCR0	SDRAM Channel Control Register 0
0xFFFE_8004	SDCCR1	SDRAM Channel Control Register 1
0xFFFE_8008	SDCCR2	SDRAM Channel Control Register 2
0xFFFE_800C	SDCCR3	SDRAM Channel Control Register 3
0xFFFE_8010	SDCCR4	SDRAM Channel Control Register 4
0xFFFE_8014	SDCCR5	SDRAM Channel Control Register 5
0xFFFE_8018	SDCCR6	SDRAM Channel Control Register 6
0xFFFE_801C	SDCCR7	SDRAM Channel Control Register 7
0xFFFE_8020	SDCTR1	SDRAM Shared Timing Register 1 (for SDRAM/SGRAM)
0xFFFE_8024	SDCTR2	SDRAM Shared Timing Register 2 (for DIMM flash memory)
0xFFFE_8028	SDCTR3	SDRAM Shared Timing Register 3 (for SMROM)
0xFFFE_802C	SDCCMD	SDRAM Command Register
0xFFFE_8030	SDCSMRS1	SGRAM Load Mask Register
0xFFFE_8034	SDCSMRS2	SGRAM Load Color Register

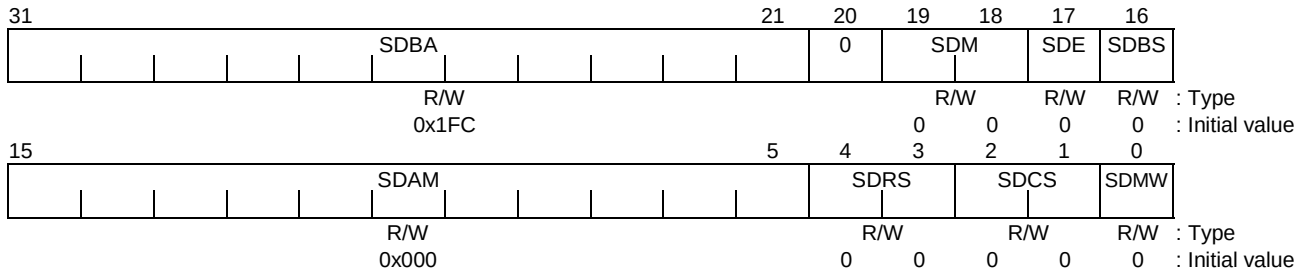
Note 1: Any “Reserved” designation means unpredictable results.

Note 2: All registers are readable and are word addressable only.

Note 3: t_{CK} = clock period

Channels 2 to 7 share chip select signals with ROM controller channels 2 to 7. Ensure that the same channel is not enabled simultaneously for the SDRAM and ROM controllers.

8.4.2 SDRAM Channel Control Registers (SDCCR0-SDCCR7) 0xFFFE_8000 (ch.0)
0xFFFE_8004 (ch.1)
0xFFFE_8008 (ch.2)
0xFFFE_800C (ch.3)
0xFFFE_8010 (ch.4)
0xFFFE_8014 (ch.5)
0xFFFE_8018 (ch.6)
0xFFFE_801C (ch.7)



Note : When DIMM flash memory or SMROM is selected as the boot memory device, the contents of the channel 0 register are initialized by boot setting. Refer to “3.4 Initial Setting Signals” for information on boot setting.

Bits	Mnemonic	Field Name	Description
31 : 21	SDBA	Base Address	Base Address (initial value: 0x1FC) Base address for the channel 0.
19 : 18	SDM	Memory Type	Memory Type (initial value: 00) Memory type for the channel 0. 00: SDRAM 01: DIMM flash memory 10: SMROM 11: SGRAM
17	SDE	Enable	Enable (initial value: 0) Enables the channel 0. 0: Disable 1: Enable
16	SDBS	Number of Banks	Number of Banks (initial value: 0) Selects the number of SDRAM/SGRAM banks or the address mapping type for DIMM flash memory. 0: 2 SDRAM/SGRAM banks or address mapping type 0 for DIMM flash memory 1: 4 SDRAM/SGRAM banks or address mapping type 1 for DIMM flash memory
15 : 5	SDAM	Address Mask	Address Mask (initial value: 0x000) Specifies which bits of the base address (SDBA field) are valid in address comparison. 0: Valid bit (to be compared) 1: Invalid bit (to be ignored)
4 : 3	SDRS[1]	Row Size	Row Size (initial value: 00) Selects the row size of memory. 00: 2048 rows (11-bit) 01: 4096 rows (12-bit) 10: 8192 rows (13-bit) 11: Reserved

Figure 8.4.1 SDRAM Channel Control Registers (1/2)

Bits	Mnemonic	Field Name	Description
2 : 1	SDCS[1]	Column Size	Column Size (initial value: 00) Selects the column size of memory. 00: 256 words (8-bit) 01: 512 words (9-bit) 10: 1024 words (10-bit) 11: 2048 words (11-bit)
0	SDMW[2]	Memory Width	Memory Width (initial value: 0) Selects the memory bus width. 0: 32-bit 1: 16-bit

Note 1: These fields are used for SDRAMs or SGRAMs only. They are ignored for all other memory types.

Note 2: The memory width field should always be set to 0 for SMROM or SGRAM.

Figure 8.4.1 SDRAM Channel Control Registers (2/2)

- Note on using the base address (SDBA) and address mask (SDAM)
  The address mask (SDAM) specifies whether the corresponding address bits of the base address (SDBA) are used to determine the address space for the channel. Note that the address space can only be allocated within specified boundaries. For example, the following setting allocates the address space in the range of 0xA000_0000 to 0xA1FF_FFFF, instead of 0xA100_0000 to 0xA2FF_FFFF.

  Example:

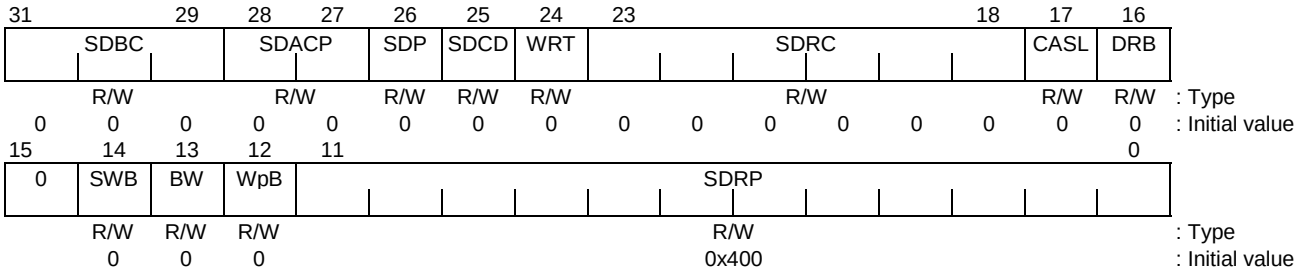
  SDCCR: 0x010301ea

  -SDBA = 0000 0001 000b

  -SDAM = 0000 0001 111b

  Mask these address bits

8.4.3 SDRAM Timing Register 1 (for SDRAM/SGRAM) (SDCTR1) 0xFFFE_8020



Bits	Mnemonic	Field Name	Description
31 : 29	SDBC	Bank Cycle Time	Bank Cycle Time (t_{RC}) (initial value: 000) Selects the bank cycle time for memory. 000: 5 t_{CK} (reset) 100: 9 t_{CK} 001: 6 t_{CK} 101: 10 t_{CK} 010: 7 t_{CK} 110: Reserved 011: 8 t_{CK} 111: Reserved
28 : 27	SDACP	Active Command Period	Active Command Period (t_{RAS}) (initial value: 00) Selects the active command period for memory. 00: 3 t_{CK} (reset) 01: 4 t_{CK} 10: 5 t_{CK} 11: 6 t_{CK}
26	SDP	Precharge Time	Precharge Time (t_{RP}) (initial value: 0) Select the precharge time for memory. 0: 2 t_{CK} (reset) 1: 3 t_{CK}
25	SDCD	RAS-CAS Delay	RAS to CAS Delay (t_{RCD}) (initial value: 0) 0: 2 t_{CK} (reset) 1: 3 t_{CK}
24	WRT	Write Recovery Time	Write Recovery Time (t_{WR}) (initial value: 0) Selects the write recovery time for memory. 0: 1 t_{CK} (reset) 1: 2 t_{CK}
23 : 18	SDRC	Refresh Counter	Refresh Counter (initial value: 000000) Decremented at each refresh. When the refresh circuit is activated with a non-zero value loaded, this field functions as a down-counter which stops at 0. A non-zero value must be reloaded to begin another countdown. Used for memory initialization.
17	CASL	CAS Latency	CAS Latency (t_{CASL}) (initial value: 0) Selects the CAS latency. 0: 2 t_{CK} (reset) 1: 3 t_{CK}
16	DRB	Data Read Bypass	Data Read Bypass (initial value: 0) Selects which data read-back path is used. 0: Use the feedback clock to latch data into the register (reset). 1: Bypass the data latching stage.

Figure 8.4.2 SDRAM Timing Register 1 (for SDRAM/SGRAM) (1/2)

Bits	Mnemonic	Field Name	Description
14	SWB	Slow Write Burst	Slow Write Burst (t_{SWB}) (initial value: 0) Enables slow write bursts. 0: Burst writes occur every t_{CK} (reset). 1: Burst writes occur every other t_{CK}.
13	BW	SGRAM Block Write	SGRAM Block Write (initial value: 0) Enables SGRAM block write. 0: Disable (reset) 1: Enable
12	WpB	SGRAM Write Per Bit	SGRAM Write Per Bit (initial value: 0) 0: Disable (reset) 1: Enable
11 : 0	SDRP	Refresh Period	Refresh Period (initial value: 0x400) Specifies the number of clock cycles between refresh cycles. Refresh is enabled only if at least one channel is enabled for SDRAM/SGRAM. The timing register should be programmed before any of the channels are enabled. The initial value is 0x400, which causes refresh cycles to occur at intervals of 15.5 μs (at 66 MHz).

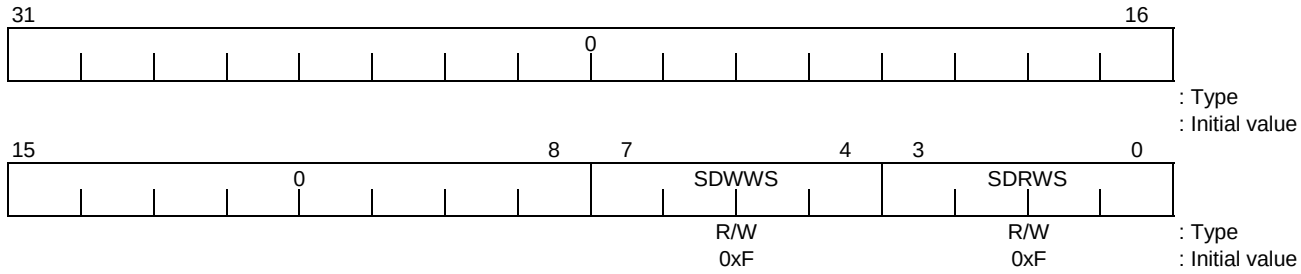
Note 1: The mode register setting cycle time, t_{RSC}, is set to $3t_{CK}$ for convenience to satisfy the SMROM MRS recovery time.

Note 2: t_{RC} is used only for the refresh cycle time. For reads and writes, including bursts, t_{RC} is always satisfied because it is subsumed by the condition $t_{RAS} + t_{RP} + 1t_{CK} \geq t_{RC}$.

Note 3: A combination of $t_{RCD} = 2t_{CK}$ and $t_{RAS} = 6t_{CK}$ is not allowed.

Figure 8.4.2 SDRAM Timing Register 1 (for SDRAM/SGRAM) (2/2)

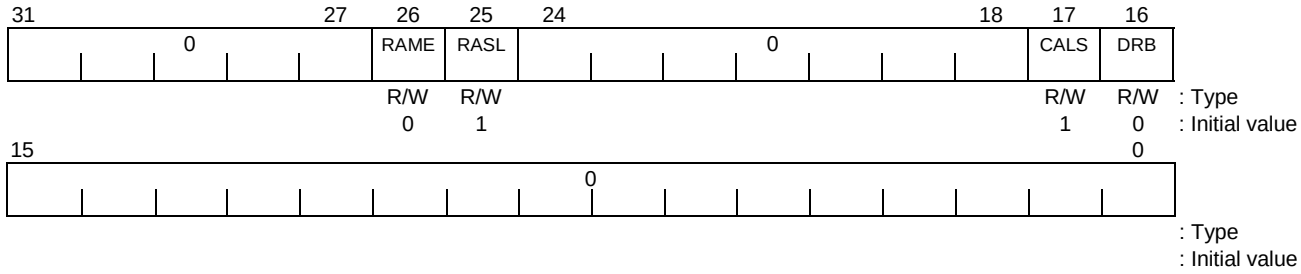
8.4.4 SDRAM Timing Register 2 (for DIMM Flash Memory) (SDCTR2) 0xFFFE_8024



Bits	Mnemonic	Field Name	Description
7 : 4	SDWWS	Write Wait States	Write Wait States (initial value: 0xF) Specifies the number of wait states used for writes to DIMM flash memory. 0000: $1t_{CK}$ 0001: $2t_{CK}$: 1110: $15t_{CK}$ 1111: $16t_{CK}$
3 : 0	SDRWS	Read Wait States	Read Wait States (initial value: 0xF) Specifies the number of wait states used for reads from DIMM flash memory. 0000: $1t_{CK}$ 0001: $2t_{CK}$: 1110: $15t_{CK}$ 1111: $16t_{CK}$

Figure 8.4.3 SDRAM Timing Register 2 (for DIMM Flash Memory)

8.4.5 SDRAM Timing Register 3 (for SMROM) (SDCTR3) 0xFFFE_8028



Bits	Mnemonic	Field Name	Description
26	RAME	Row Address Match Enable	Row Address Match Enable (initial value: 0) Enables the feature to check if the current row address matches the previous read row address on the same channel. If a match occurs then the activate command is skipped and only the read command is executed. 0: Disable (reset) 1: Enable
25	RASL	RAS Latency	RAS Latency (t_{RCD}) (initial value: 1) Selects the RAS latency. 0: $1t_{CK}$ 1: $2t_{CK}$ (reset)
17	CASL	CAS Latency	CAS Latency (t_{CASL}) (initial value: 1) Selects the CAS latency. 0: $4t_{CK}$ 1: $5t_{CK}$ (reset)
16	DRB	Data Read Bypass	Data Read Bypass (initial value: 0) Selects which data read-back path is used. 0: Use the feedback clock to latch data into the register. 1: Bypass the data latching stage.

Note 1: The mode register setting recovery time is set to $3t_{CK}$

Note 2: t_{RC} is subsumed by t_{RCD} and t_{CAS}

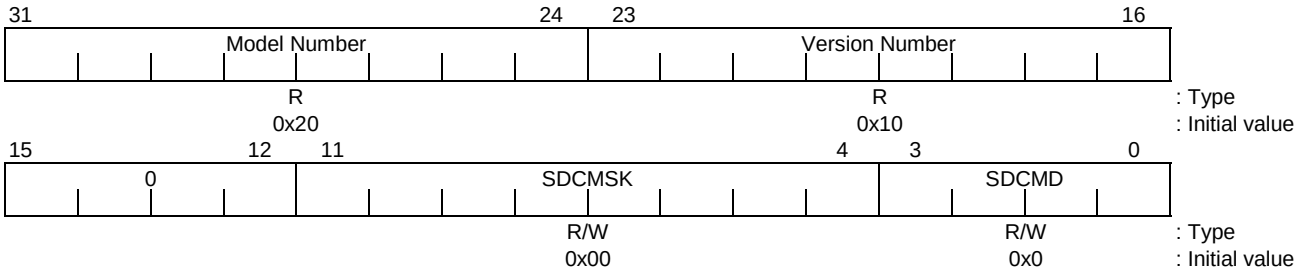
Note 3: t_{VCVC}, valid CAS to valid CAS time, is set to $4t_{CK}$

Note 4: The following table lists the current parameter settings:

Speed	t_{RCD} (t_{CK})	t_{CASL} (t_{CK})
50 MHz	1	4
66 MHz	2	5

Figure 8.4.4 SDRAM Timing Register 3 (for SMROM)

8.4.6 SDRAM Command Register (SDCCMD) 0xFFFE_802C



Bits	Mnemonic	Field Name	Description
31 : 24	Model Number	Model Number	Model Number (initial value: 0x20) Shows the model number. The TX3927's model number is 0x20. The field is read-only.
23 : 16	Version Number	Version Number	Version Number (initial value: 0x10) Shows the version number. The TX3927's version number is 0x10. The field is read-only.
11 : 4	SDCMSK	Channel Mask	Channel Mask (initial value: 0x00) A "1" enables the command for the corresponding channel. The command is executed in parallel on all enabled channels.[1] Bit 11: Channel 7 Bit 10: Channel 6 Bit 9: Channel 5 Bit 8: Channel 4 Bit 7: Channel 3 Bit 6: Channel 2 Bit 5: Channel 1 Bit 4: Channel 0
3 : 0	SDCMD	Command	Command (initial value: 0x0) Selects the command for memory. 0x0: NOP command 0x1: Set the SDRAM/SGRAM mode register. 0x2: Set the SMROM mode register. 0x3: Precharge all SDRAM/SGRAM banks (PALL). 0x4: Enter low power mode. 0x5: Enter power-down mode. 0x6: Exit low power/power-down mode. 0x7-0xf: Reserved Commands will only be executed on channels with a memory type of SDRAM, SMROM, or SGRAM. Low power mode is self-refresh mode for SDRAMs/SGRAMs and low power mode for SMROMs. Power-down mode is precharge power-down mode for SDRAMs/SGRAMs and low power mode for SMROMs.

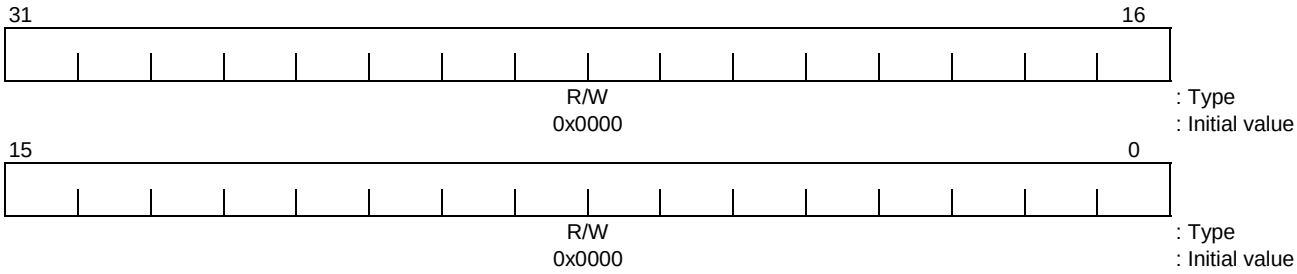
Note 1: The mask bits are also used for the SGRAM load mask and load color registers.

Any write cycle for the command register will wait until the command is executed. Therefore, memory cycles cannot be executed until the command cycle is executed. The command cannot start until the SDRAMC becomes idle.

First use the channel control register to enable the target channel before executing a command for the channel. Commands will only be executed on channels enabled with the channel control register.

Figure 8.4.5 SDRAM Command Register

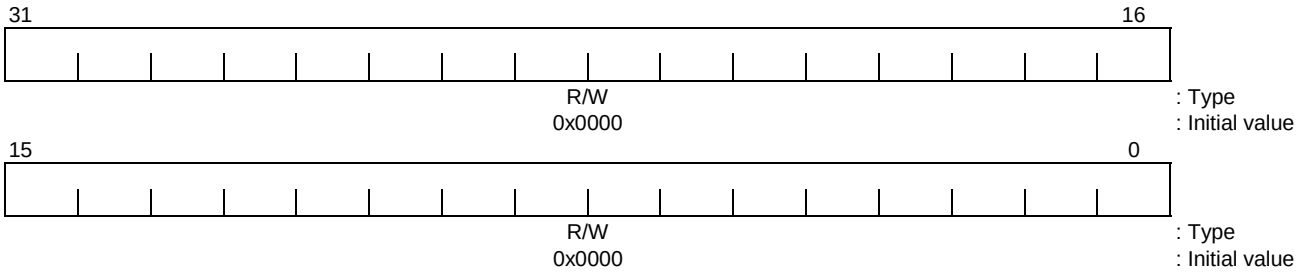
8.4.7 SGRAM Load Mask Register (SDCSMRS1) 0xFFFE_8030



Bits	Mnemonic	Field Name	Description
31 : 0	SDCSMRS1	SGRAM Load Mask	SGRAM Load Mask Register 1 (initial value: 0x0000_0000) A write to this register loads the SGRAM mask register with 32-bit data. The mask bit(s) for the target channel(s) must be enabled in the command register and the corresponding control register(s) must be set with a memory type of SGRAM. To enable the mask bits, perform a NOP command so the external SGRAM is not affected by an inadvertent command. In the same way as with a command operation, the write starts only when the SDRAMC is idle. This register always reads as 0.

Figure 8.4.6 SGRAM Load Mask Register

8.4.8 SGRAM Load Color Register (SDCSMRS2) 0xFFFE_8034



Bits	Mnemonic	Field Name	Description
31 : 0	SDCSMRS2	SGRAM Load Color	SGRAM Load Color Register 2 (initial value: 0x0000_0000) A write to this register loads the SGRAM color register with 32-bit data. The mask bit(s) for the target channel(s) must be enabled in the command register and the corresponding control register(s) must be set with a memory type of SGRAM. To enable the mask bits, perform a NOP command so the external SGRAM is not affected by an inadvertent command. In the same way as with a command operation, the write starts only when the SDRAMC is idle. This register always reads as 0.

Figure 8.4.7 SGRAM Load Color Register

8.5 Operation

8.5.1 TX3927 Signals for Different Memory Types

Table 8.5.1 Control Signals for Memory Types

Description	Signal Name	SDRAM/SGRAM		DIMM Flash Memory		SMROM	
		Read	Write	Read	Write	Read	Write[1]
Chip select	CS* [7 : 0]	L	L	L	L	L	L
Output enable	OE*	NU	NU	L	H	L[2]	H[2]
Clock enable	CKE	V	V	NU	NU	V	V
Address	ADDR [19 : 5]	V	V	V	V	V	V
Data	DATA [31 : 0]	V	V	V	V	V	V
Row address strobe	RAS*	L	L	NU	NU	L	L
Column address strobe	CAS*	L	L	NU	NU	L	L
Write enable	WE*	H	L	NU	NU	H[3]	L[3]
Data mask	DQM [3 : 0]	L	V	H[4]	V	NU[5]	NU[5]
Define special function	DSF	L	V[6]	NU	NU	NU	NU
Byte/write enable	BWE [3 : 0]	NU	NU	NU	NU	NU	NU
Serial presence detect	SCL, SDA	V[7]	V[7]	V	V	NU	NU

(V = valid, H = logic high, L = logic low, NU = not used)

Note 1: The only SMROM write operations are the Mode Register Set (MRS) and Burst Stop (precharge) commands.

Note 2: Connected to the DQM pin of the SMROM.

Note 3: Connected to the mode register setting pin, MR*, of the SMROM.

Note 4: DQM[3:0] are forcibly driven to high. The signals are not used as mask bits but as byte write enables only.

Note 5: DQM[3:0] are kept high during SMROM accesses.

Note 6: Asserted only for special SGRAM functions.

Note 7: Used only for DIMMs.

- NU signals may or may not be physically connected to the device.
- D[31:0], SCL, and SDA are handled by other modules.
- CS*[1:0] are dedicated to the SDRAM controller while CS*[7:2] are shared with the ROM controller chip select signals. Ensure that the SDRAM and ROM channels having an identical number are not enabled at the same time. OE* is shared with the ROM controller output enable. OE* is used for enabling the output buffers of DIMM flash memory and SMROM during a read access to DIMM flash memory or SMROM.

8.5.2 SDRAM Operation

(1) Initialization and refresh

The TX3927 command register provides the ability to generate the cycles required for initialization and the software can mix and match the cycles with whatever timing it determines is appropriate so that initialization sequences can be very flexible.

The CAS-before-RAS (CBR) refresh cycles during initialization are generated by the normal refresh circuit. If they should be completed faster than would result from the normal refresh interval, the software can set the refresh period to a much smaller time during initialization. After the required number of CBR refresh cycles have occurred, the software should then write the normal value into the refresh period field.

The refresh counter field in the SDRAM timing register provides a convenient way to count the number of refreshes to the memory. Thus software does not need to implement special timing loops to wait for the appropriate number of refreshes to occur.

Re-initialization of the memory is also supported. Again, software has complete flexibility in the implementation.

Note that at least one channel must be enabled and configured for SDRAM (or SGRAM) to enable the refresh circuit. Once a channel is enabled, the refresh circuit loads the refresh period into its counter to start operation. The refresh circuit reloads the refresh period each time it counts down to zero. Any change in the refresh cycle will, therefore, take effect in the next refresh cycle.

A refresh request has precedence over any other type of SDRAM controller access request. Any pending memory access request will be held off and serviced as soon as the refresh completes. If, however, the refresh period is set to a very small value, perhaps either by accident or for testing, potentially all memory accesses will be locked out by continuous refreshes and the SDRAMC will no longer respond. To guard against this possibility, the TX3927 first handles the pending memory access request before starting a refresh cycle in response to a second refresh request. This allows a register or memory access to slip in between refreshes.

(2) Precharging

- A burst or single read or a single write is terminated with a precharge active bank command (PBank). A burst write is terminated with an auto-precharge command, or with a PBank command if the write crosses boundaries.

The timing parameters “write recovery time” and “time from last data input to low precharge,” represented with t_{WR} and t_{DPL} (or t_{RDL}) in datasheets, respectively, are particularly important for write cycles. These parameters are controlled with the write recovery time (WRT) bit of SDRAM timing register 1 (SDCTR1).

The WRT bit should be set to a value appropriate to the SDRAM used. Otherwise, auto-precharge occurs with wrong timing, causing a problem during an operation with a sequence of consecutive accesses. When using the SDRAM with the write recovery time set to 1 clock cycle, make sure that it can operate normally. If the write recovery time is set to 2 clock cycles, SDRAMs configured to run with 1 clock cycle can also operate normally.

- A bank boundary crossing is treated as a page crossing because a PBank command is issued before the new bank is activated.

- The PALL command works for any memory configuration on any channel where the command is issued.

(3) Reading and writing

- For single accesses, the basic number of cycles, 8 cycles for read and 6 cycles for write, are increased as follows:

Table 8.5.2

		Read	Write
t_{RCD}	2	No additional cycles	No additional cycles
	3	1 additional cycle	1 additional cycle
t_{CASL}	2	No additional cycles	No additional cycles
	3	1 additional cycle	No additional cycles

- For consecutive accesses to SDRAM, reads and writes may operate slightly slower depending on the programmed timing latencies.

- In all types of accesses a bus acknowledge is returned as soon as possible. The SDRAMC state machine then returns to an IDLE state according to the programmed timings and what type of cycle is completing (read or write). Any precharge or recovery cycle required for SDRAM is performed automatically without the need of special coding. This may, however, prevent back-to-back operations from starting immediately. Basically, the TX3927 handles accesses to SDRAM as quickly as possible.

- Some burst write operations cross a page boundary. An extra t_{RCD} cycle is inserted to determine the page boundary.

- The following burst accesses are supported in the TX3927.

Table 8.5.3 Burst Accesses Supported

Number of words	4	8	16	32
CPU RD	Yes	Yes	Yes	Yes
CPU WR	Yes	No	No	No
PCI RD	Yes	Yes	Yes	No
PCI WR	Yes	Yes	Yes	No
DMA RD	Yes	Yes	Yes	Yes
DMA WR	Yes	Yes	Yes	Yes

- Slow write burst mode, once enabled, applies to all burst write accesses and to single writes to 16-bit memory. Slow writes set up the data bus and DQM bits two clocks before they are used. All other signals to SDRAM are set up one clock before they are used.

(4) Addressing considerations

- Table 8.5.4 shows the SDRAM address mappings for 32-bit memory. B0 is used for the "2 bank" select. B1 is used for the "4 bank" select. L/H indicates low or high, depending on whether auto-precharge is used.

Table 8.5.4 Address Mapping for 32-bit SDRAM (1/2)

Row address width = 11 Column address width = 8															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	2	3	4	5	6	7	8	9	L/H	L/H	L/H	19	20	22	21
Row address	10	11	12	13	14	15	16	17	18	19	20	19	20	22	21

Row address width = 11 Column address width = 9															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	2	3	4	5	6	7	8	9	21	22	L/H	22	22	22	22
Row address	10	11	12	13	14	15	16	17	18	19	20	22	22	22	22

Row address width = 11 Column address width = 10															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	2	3	4	5	6	7	8	9	21	22	L/H	23	22	22	23
Row address	10	11	12	13	14	15	16	17	18	19	20	23	22	22	23

Row address width = 12 Column address width = 8															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	2	3	4	5	6	7	8	9	22	23	L/H	21	22	23	22
Row address	10	11	12	13	14	15	16	17	18	19	20	21	22	23	22

Row address width = 12 Column address width = 9															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	2	3	4	5	6	7	8	9	22	23	L/H	21	23	24	23
Row address	10	11	12	13	14	15	16	17	18	19	20	21	23	24	23

Row address width = 12 Column address width = 10															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	2	3	4	5	6	7	8	9	22	23	L/H	21	24	25	24
Row address	10	11	12	13	14	15	16	17	18	19	20	21	24	25	24

Table 8.5.4 Address Mapping for 32-bit SDRAM (2/2)

Row address width = 12 Column address width = 11															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	2	3	4	5	6	7	8	9	22	23	L/H	24	25	26	25
Row address	10	11	12	13	14	15	16	17	18	19	20	21	25	26	25

Row address width = 13 Column address width = 8															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	2	3	4	5	6	7	8	9	23	24	L/H	21	22	24	23
Row address	10	11	12	13	14	15	16	17	18	19	20	21	22	24	23

Row address width = 13 Column address width = 9															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	2	3	4	5	6	7	8	9	23	24	L/H	21	22	25	24
Row address	10	11	12	13	14	15	16	17	18	19	20	21	22	25	24

Row address width = 13 Column address width = 10															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	2	3	4	5	6	7	8	9	23	24	L/H	21	22	26	25
Row address	10	11	12	13	14	15	16	17	18	19	20	21	22	26	25

Row address width = 13 Column address width = 11															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	2	3	4	5	6	7	8	9	23	24	L/H	25	22	27	26
Row address	10	11	12	13	14	15	16	17	18	19	20	21	22	27	26

- Table 8.5.5 shows the SDRAM address mappings for 16-bit memory. B0 is used for the "2 bank" select. B1 is used for the "4 bank" select (or ADDR18 if using 2 banks). L/H indicates low or high, depending on whether auto-precharge is used.

Table 8.5.5 Address Mapping for 16-bit SDRAM (1/2)

Row address width = 11 Column address width = 8															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	1	2	3	4	5	6	7	8	20	21	L/H	20	21	21	20
Row address	9	10	11	12	13	14	15	16	17	18	19	20	21	21	20

Row address width = 11 Column address width = 9															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	1	2	3	4	5	6	7	8	20	21	L/H	21	21	21	21
Row address	9	10	11	12	13	14	15	16	17	18	19	21	21	21	21

Row address width = 11 Column address width = 10															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	1	2	3	4	5	6	7	8	20	21	L/H	22	21	21	22
Row address	9	10	11	12	13	14	15	16	17	18	19	22	21	21	22

Row address width = 12 Column address width = 8															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	1	2	3	4	5	6	7	8	21	22	L/H	20	21	22	21
Row address	9	10	11	12	13	14	15	16	17	18	19	20	21	22	21

Row address width = 12 Column address width = 9															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	1	2	3	4	5	6	7	8	21	22	L/H	20	22	23	22
Row address	9	10	11	12	13	14	15	16	17	18	19	20	22	23	22

Row address width = 12 Column address width = 10															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	1	2	3	4	5	6	7	8	21	22	L/H	20	23	24	23
Row address	9	10	11	12	13	14	15	16	17	18	19	20	23	24	23

Table 8.5.5 Address Mapping for 16-bit SDRAM (2/2)

Row address width = 12 Column address width = 11															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	1	2	3	4	5	6	7	8	21	22	L/H	23	24	25	24
Row address	9	10	11	12	13	14	15	16	17	18	19	20	24	25	24

Row address width = 13 Column address width = 8															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	1	2	3	4	5	6	7	8	22	23	L/H	20	21	23	22
Row address	9	10	11	12	13	14	15	16	17	18	19	20	21	23	22

Row address width = 13 Column address width = 9															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	1	2	3	4	5	6	7	8	22	23	L/H	20	21	24	23
Row address	9	10	11	12	13	14	15	16	17	18	19	20	21	24	23

Row address width = 13 Column address width = 10															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	1	2	3	4	5	6	7	8	22	23	L/H	20	21	25	24
Row address	9	10	11	12	13	14	15	16	17	18	19	20	21	25	24

Row address width = 13 Column address width = 11															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	1	2	3	4	5	6	7	8	22	23	L/H	24	21	26	25
Row address	9	10	11	12	13	14	15	16	17	18	19	20	21	26	25

- If the same address is enabled for more than one channel, the address becomes active for the channel having the smallest number. The following shows the order of priority for the channels: ch0 ⇒ ch1 ⇒ ch2 ⇒ ch3 ⇒ ch4 ⇒ ch5 ⇒ ch6 ⇒ ch7

- 16-bit to 32-bit memory and 32-bit to 16-bit memory channel crossings are supported.

- 16-bit memory must be placed on the low-order data bus, D0-D15. Likewise the low-order data mask signals DQM0* and DQM1* are used. In big endian mode, the high-order half word D16-D31 is written first and the low-order half word D0-D15 is written last. In little endian mode, the low-order half word D0-D15 is written first and the high-order half word D16-D31 is written last. A bys cycle to 16-bit memory will always generate two external accesses, no matter how the byte enables are set. The high-order data mask signals DQM2* and DQM3* are held high.

- The setting of the base address (SDCCRn.SDBAn) and the address mask (SDCCRn.SDAMn) determine whether the channel is selected. However, the address bits being asserted to the SDRAM(s) depend solely on the row/column sizes set in the control registers (SDCCRn.SDRSn/SDCSn). Hence, there could be overlap between low-order bits used in channel selection and high-order bits of the asserted address.

- The internal registers of the TX3927 are located in a reserved address space (0xFF00_0000-0xFFFE_FFFF). If an SDRAMC memory channel is mapped to overlay the register space, unpredictable results may occur.

(5) Bus error and abnormal cycle termination

- No bus error is returned by the SDRAMC.

- If a write is performed to an address where no register is allocated in the SDRAMC registers address area (0xFFFE_8000-0xFFFE_8FFF), the bus cycle is completed normally without any adverse effects. A read from such an address reads all 0's.

- The SDRAMC immediately aborts its current operation under two conditions: a bus error (generated by some other module) or a debug reset. If either case occurs, the current SDRAM clock cycle completes, the remaining SDRAMC operation is aborted, a PALL command is issued, and the SDRAMC returns to its idle state. The SDRAMC will hold off any new accesses until it returns to idle and then respond normally. The contents of the SDRAM remain intact for recovery or debugging.

- A write to the command register for executing a command cycle causes any write cycles to this register to wait until the command is executed. Therefore, aborting such a register access is supported. If a register access is aborted, the current operation in the clock cycle completes and the control signals return to their idle state.

- Refreshes are unaffected by the cycle aborts, even if a refresh was in progress when the abort occurred.

(6) Power-down modes

Software can disable and enable CKE to the SDRAMs at any time by issuing the appropriate command in the command register. The commands Enter L/H Power Mode and Enter Power Down Mode turn off CKE and the command Exit L/H Power/Power Down Mode turns on CKE. Low power mode puts the SDRAMs in self-refresh mode and power-down mode puts the SDRAMs in precharge power-down mode, where no refresh is performed and data is at risk.

If one of the power-down modes is entered, the clocks to the SDRAMs are turned off. Also, the SDRAMC resets and disables the refresh counter of the internal refresh circuit.

Exiting the power-down modes can be done either by using the command register as described above or by an automatic process. The auto-exit process is executed whenever CKE is turned off and an SDRAM (or SMROM) access is attempted. This is transparent to software: CKE is automatically turned on and the bus access completes normally. Thus the TX3927 can begin fetching instructions or data immediately after returning from one of its power-down states.

Whenever the power-down modes are exited, a pause of $10t_{CK}$ occurs before the next operation initiates. This satisfies the SDRAM timing requirements for the first access after power-down exits.

The TX3927 also provides HALT and DOZE modes for power reduction. In these modes, the CPU stops but the SDRAMC and other peripheral circuits continue to operate. Therefore, software must setup the memory by executing the proper channel commands before HALT or DOZE mode is entered.

(7) Debug mode

- As mentioned above ("8.5.1 (5) Bus error and abnormal cycle termination"), debug reset will gracefully terminate the current SDRAM operation and the memory contents are available for debugging.

- The SDRAMC executes normally in debug mode.

8.5.3 DIMM Flash Memory Operation

(1) Initialization, boot, and refresh

DIMM flash memory requires no initialization. The devices are ready to read or write with the reset period of 16 clock cycles maximum.

Boot from DIMM flash memory is selected by setting boot signals BME[1:0] (ADDR[9:8]) to "01" on the rising edge of the reset signal. SDRAM channel 0 is used as the boot channel. The default base address in the control register (SDCCR0) is 0x1FC0_0000. If the DIMM flash memory address uses up to an address bit of 21 as its MSB row address, then the start address at the DIMM flash memory will be 0. The other fields initialized in the control register (SDCCR0) are memory type (SDM0) = 01 (DIMM flash memory), enable (SDE0) = 1, and memory width (SDMW0) = 0 or 1 (32-bit or 16-bit depending on at the value of boot signal BOOT16* (ADDR[13])).

The refresh circuit is only enabled if there is an SDRAM or SGRAM enabled on at least one SDRAMC channel. Refresh is inhibited to any channel used for DIMM flash memory.

Another significant difference between the SDRAM and flash memory refresh operations is that burst accesses to DIMM flash memory are interruptible by refresh requests. This is because under certain configurations a flash memory burst access can exceed the normal SDRAM refresh period, thereby potentially missing one of two contiguous refresh requests. For example, a 32-word burst read from 16-bit flash memory with 16 wait states and a system clock of 50 MHz takes 28.1 μs to complete [(44 clock cycles × 32 reads −3 clock cycles) × 20 ns], thus exceeding a 15.6 μs refresh period. To prevent this, the TX3927 always handles a refresh request first, even if it occurs during a burst access to DIMM flash memory.

(2) Precharging

Precharging has no meaning to flash memory. Likewise, neither do the other commands in the command register. A command executed to flash memory is treated as a NOP since the CS* for the flash memory channel is not asserted. Thus if a flash memory channel is selected via the enable and mask bits or even if flash memory is the only memory in the system, memory accesses are disabled until the execution of the command is completed.

In general, the ADDR, RAS*, CAS*, and WE* signals, which are shared for SDRAM and DIMM flash memory channels, will become active during SDRAM accesses and command execution. However, only the CS* signals for the appropriate SDRAM channels will be asserted. The flash memory remains in a standby state until its CS* is asserted.

(3) Reading and writing

- Programmable wait states in the SDRAM timing register for DIMM flash memory (SDCTR2) are used to tune read and write accesses to the speed grade of the flash memory.

- The same burst accesses as those for SDRAM are supported for DIMM flash memory. Burst writes to DIMM flash memory do not make sense since DIMM flash memory writes generally require a timed interval to complete. However, the feature is supported for future implementations.

- If the CPU or other circuits attempt burst accesses to DIMM flash memory, the SDRAMC treats them as single accesses on the external bus.

- During non-DIMM flash memory accesses, the CS* signals are always kept high to keep the flash memory devices in a NOP state.

- Slow write burst mode is not available for DIMM flash memory. If the feature is enabled (by setting the SDTCR1.SWB1 bit to "1"), it has no effect on the DIMM flash memory accesses. The feature can be enabled only for SDRAMs in the same system.

- Data read bypass has the same effect on DIMM flash memory accesses. That is, DIMM flash memory accesses follow the current setting of the data read bypass bit (DRB1) in the SDRAM timing register (SDTCR1).

(4) Addressing considerations

- Table 8.5.6 shows the Type 0 address mappings for DIMM flash memory. ADDR5-ADDR7 are "don't cares" for row address specification. ADDR16-ADDR17 are available for expansion.

Table 8.5.6 Type 0 Address Mappings for DIMM Flash Memory

Type 0 addressing for 32-bit flash memory															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15	16	17	18 (B1)	19 (B0)
Column address	2	3	4	5	6	7	8	9	10	11	12	25	26	14	13
Row address	10	11	12	15	16	17	18	19	20	21	22	25	26	24	23

Type 0 addressing for 16-bit flash memory															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15	16	17	18 (B1)	19 (B0)
Column address	1	2	3	4	5	6	7	8	9	10	11	24	25	13	12
Row address	10	11	12	14	15	16	17	18	19	20	21	24	25	23	22

- Table 8.5.7 shows the Type 1 address mappings for DIMM flash memory. ADDR5-ADDR7 and ADDR18-ADDR19 are “don’t cares” for row address specification. ADDR17 is available for expansion.

Table 8.5.7 Type 1 Address Mappings for DIMM Flash Memory

Type 1 addressing for 32-bit flash memory															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15	16	17	18 (B1)	19 (B0)
Column address	2	3	4	5	6	7	8	9	19	20	10	24	25	12	11
Row address	10	11	12	13	14	15	16	17	18	21	22	23	25	12	11

Type 1 addressing for 16-bit flash memory															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15	16	17	18 (B1)	19 (B0)
Column address	1	2	3	4	5	6	7	8	18	19	9	23	24	11	10
Row address	10	11	12	12	13	14	15	16	17	20	21	22	24	11	10

- The setting of the base address (SDCCRn.SDAAn) and the address mask (SDCCRn.SDAMn) determine whether the channel is selected. However, the row size (SDCCRn.SDRSn) and column size (SDCCRn.SDCSn) have no effect on the flash memory. These fields are ignored.

- All other addressing features are the same as those for SDRAM. All boundary crossings are supported (in particular, page and channel boundaries). If the same address is enabled for more than one channel, the address becomes active for the channel having the smallest number. The following shows the order of priority for the channels:
  ch0 ⇒ ch1 ⇒ ch2 ⇒ ch3 ⇒ ch4 ⇒ ch5 ⇒ ch6 ⇒ ch7

(5) Bus error and abnormal cycle termination

- Same as those for SDRAM with one exception. The aborting access has a different effect in that the DIMM flash memory cycle terminates immediately, regardless of whether the abort occurs in read or write operation (the DIMM flash memory access takes several clock cycles to complete). This is necessary since the byte enable and data bus (including the deasserting of OE*) are required immediately by the bus master.

(6) Power-down modes

- Same as those for SDRAM except that bus cycles can still occur as long as the SDRAMC is running even if SDCLK is not being output. DIMM flash memory devices are asynchronous devices and do not use the CKE or external clock signals.

(7) Debug mode

- Same as that for SDRAM.
  Refer to “8.5.1 (7) Debug mode.”

8.5.4 SMROM Operation

(1) Initialization, boot, and refresh

SMROM is self-initialized at power-on and is ready to begin delivering data. The device does not drive the bus until both CS* and OE* are asserted. Its default values are RAS latency (t_{RCD}) = 2, CAS latency (t_{CASL}) = 5, and burst length = 4 or 8.

These default values are required for proper operation during SMROM boot. The timing latencies match those set in the SMROM timing register for SMROM (SDCTR3) during initialization. And although the SDRAMC operates with a burst length of 4, single reads are performed during the initial boot so a setting of burst length = 8 is acceptable. A burst setting of 8 imposes a requirement of t_{RC} = 10 at 66 MHz on SMROM but this is also acceptable since t_{RC} = 11 for single reads with the default timing latencies.

SMROM boot is selected by setting boot signals BME[1:0] (ADDR[9:8] pins) to "00" during a reset. As the boot sequence proceeds the first step in changing any of the default settings is to change the burst length from 8 to 4. This is done by executing the MRS command (the burst length is automatically set to 4 and the current, default, timing latencies are used). It is important not to reprogram the timing latencies either in the shared timing register or in the SMROM until the operations can be properly synchronized.

Once the burst length is set to 4, then the cache can be enabled and burst reads can be performed. If the boot sequence continues from SMROM and the timing latencies are to be changed, then the code to do this should be in cache. The cached instructions to perform an update to the SDRAM timing register (SDCTR3) and an MRS are then unaffected by changing the SMROM timing latencies. Once the operations are complete the timing latencies are synchronized between the SDRAMC and the SMROM and normal read cycles can commence.

On the other hand, if the boot sequence has switched to RAM code fetches before the timing latencies are changed, then there is more flexibility in changing the timing register (SDCTR3) and the mode register in the SMROM.

Memory channel 0 is used as the boot channel. The default base address in the control register (SDCCR0) is 0x1FC0_0000. Since the 32M-bit SMROM uses address bit 21 as its MSB row address, the boot start address in SMROM will be 0. However, for 64M-bit SMROM devices, address bit 22 (=1) is the MSB and the boot start address will be 0x400000. The other fields initialized in the control register (SDCCR0) are memory type (SDM0) = 10, enable (SDE0) = 1, and memory width (SDMW0) = 0 (32-bit).

The refresh circuit is only enabled if there is an SDRAM enabled on at least one memory channel. Refresh is inhibited to any channel used for SMROM.

(2) Reading and writing

- Burst accesses must be 4-word aligned. The TX39/H2 core (except in critical word first mode) and the DMA controller in dual address mode always align accesses to 4-word boundaries. Note that PCI burst accesses to SMROM are not always aligned to 4-word boundaries.

- Writes to SMROM are illegal. If a write is attempted, the SDRAMC ignores it and does not return a bus acknowledge. As a result, the bus is locked and a bus timeout will occur.

- Since the burst length is 4, single reads are terminated with the burst stop command (precharge command). Thus the SMROM asserts only a single word and then places the data bus in the high-impedance state.

- If the row address match feature is enabled, the current row address is compared to the previously read row address. If a match is detected, the same channel is assumed so the activate command (RAS* cycle) is skipped and only the read command (CAS* cycle) is executed. This works for single or burst reads. At 66 MHz, two clock cycles are saved for each read access since $t_{RCD} = 2t_{CK}$. At 50 MHz, one clock cycle is saved for each read access since $t_{RCD} = 1t_{CK}$.

(3) Addressing considerations

- Table 8.5.8 shows the SMROM address mapping for 32-bit memory. The basic mapping is shown in dark gray for a 13-bit row and 7-bit column device. The next two address bits, A22 and A23, are provided for larger column or row space. The number of address bits to be used depends on the SMROM device.

Table 8.5.8 Address Mapping for 32-bit SMROM

Row address width = 13 (14, 15) Column address width = 7 (8, 9)															
Address bit ADDR [19 : 5]	5	6	7	8	9	10	11	12	13	14	15 (AP)	16	17	18 (B1)	19 (B0)
Column address	2	3	4	5	6	7	8	22	22	23	19	20	21	22	23
Row address	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

- If the row address match feature is enabled, the match circuit checks all fifteen row address bits.

(4) Bus error and abnormal cycle termination

- Same as those for SDRAM except that SMROM operates with a burst length of 4. Read operation is aborted with a precharge command but the CAS latency of $4t_{CK}$ or $5t_{CK}$ is incurred before the SMROM data pins are placed in the high-impedance state.

(5) Power-down modes

- Same as those for SDRAM except that the only SMROM power-down mode is low power mode. The auto-exit feature is also implemented in the same way as with SDRAM.

8.5.5 SGRAM Operation

SGRAM operation is very similar to that of SDRAM. In fact, SGRAM devices can be used as a replacement for SDRAMs in a system by connecting their DSF pins to a low level. There are some differences regarding their size and special features which are described next.

(1) Addressing

The available SGRAM memory sizes are 8M and 16M bits (the data width is 32 bits), thus supporting a smaller addressing range than SDRAMs. The SGRAM devices are used by configuring the control register (SDCCRn) with a row size of 11 bits and a column size of 8 bits. The TX3927 supports both of these SGRAM memory sizes. For the 8M-bit device, ADDR5-ADDR13 and ADDR16 of the TX3927 are connected to A0-A8 and BA, respectively, of the device. For the 16M-bit device, ADDR5-ADDR14 and ADDR17 are connected to A0-A9 and BA, respectively, of the device. In both cases, the precharge bit ADDR13 or ADDR14 is set to low or high in the column address depending on whether auto-precharge is enabled.

In order to get contiguous SGRAM memory from one channel to the next, the 16M-bit device should be used. (The 8M-bit device is smaller than the base address selection field in the control register.)

(2) Write per bit (WPB) and block write (BW) features

Each feature is enabled by setting the corresponding bit to “1” in the SDRAM timing register (SDCTR1). Once enabled the feature will affect write accesses to all channels that are enabled with SGRAM. If both features are enabled then both features are applied to the same write access.

The write per bit function is initialized with the loading of the mask register in the SGRAM. This uses a Special Mode Register Set (SMRS) command, i.e., the Load Mask command of the TX3927. Similarly, the write block function is initialized with the Load Color command. These commands can be executed at any time to update the internal registers of the SGRAM. Note that the channel mask bits (SDCMSK) in the command register (SDCCMD) should be correctly set before the SMRS command is executed.

Unlike SDRAMs, there are unique timings associated with a BW that must be observed. (The WPB feature is unencumbered by special constraints.) The two parameters of concern are t_{BPL}, block write to precharge delay, and t_{BWC}, block write cycle time. The first parameter governs how quickly a precharge can be executed after a BW and varies from $1t_{CK}$ to $2t_{CK}$ depending on the clock rate of the device. The second parameter governs how quickly contiguous BWs can be executed during a burst and varies from $1t_{CK}$ to $2t_{CK}$ depending on the clock rate of the device.

t_{BPL} applies to single writes and burst writes and is controlled by setting the write recovery time in the SDRAM timing register (SDCTR1). For burst writes with auto-precharge, this parameter should be set to a value appropriate for the memory used, in the same way as with SDRAMs.

t_{BWC} applies to burst writes and, in general, is fixed to $1t_{CK}$. However, if the slow write option is enabled in the SDRAM timing register (SDCTR1), then t_{BWC} becomes $2t_{CK}$.

The DMA address increment feature can be used in conjunction with a BW.

The DSF signal is asserted during the execution of an SMRS command and at the appropriate times during the execution of the WPB and BW features. The DSF pin is shared with a PIO feature (PIO[1]) so it must be enabled properly by the TX3927 pin configuration register (PCFG). The DSF signal should only be expected to be asserted during clock cycles in which it is used by the SGRAM.

8.5.6 Notes on Programming

- Make sure the SDRAM timing register for SDRAM/SGRAM (SDCTR1) is programmed with the correct timing values before the MRS command is issued. Make sure the SDRAM timing register for DIMM flash memory (SDCTR2) is updated with the correct values to allow for efficient accesses. The SDRAM timing register for SMROM (SDCTR3) requires special consideration during boot, as described in "8.5.3(1) Initialization, boot, and refresh."

- The three SDRAM timing registers (SDCTR1 to SDCTR3) contain timing parameters that apply, respectively, to all SDRAM/SGRAM, DIMM flash memory, and SMROM devices in the system. Thus they must be programmed to satisfy the requirements for the slowest devices.

- The TX3927 has a write buffer. The TX39/H2 core does not write to memory directly at a store instruction. It writes only to the write buffer and continues to handle the next instruction. The write buffer writes data to the target memory when the bus is empty. Thus there are a time gap between a store instruction being executed and the data actually being written to memory. To guarantee that the execution of a command from the command register (SDCCMD) (or the SGRAM load mask (SDCSMRS1) or load color (SDCSMRS2) register) has completed in the TX3927, the write buffer must be flushed. This can be accomplished by simply reading back the command register after it is written (the read-back from an address recently written does not complete until the write buffer has been flushed). The "lw" instruction that performs the read-back causes the pipeline to stall, thus ensuring the command will be executed in a correct sequence.

- Note that the boot address in 64M-bit SMROM devices will be non-zero (0x400000) while it will 0 for 32M-bit SMROM devices. Refer to the SMROM section for details.

8.6 Timing Diagrams

The timing waveforms shown in this section are subjected to the following restriction:

(1) For single 32-bit writes, ACK* is asserted at the same time data is written to memory. However, for burst writes or a 32-bit write to 16-bit memory, ACK* is asserted two clock cycles prior to the writing of data to memory. For all reads, ACK* asserts on the same cycle that data is valid and read from memory. ACK* is also shown for a DIMM flash memory write, but it is asserted at the end of the bus cycle.

8.6.1 SDRAM Single Read Operation in 32-bit Bus Mode

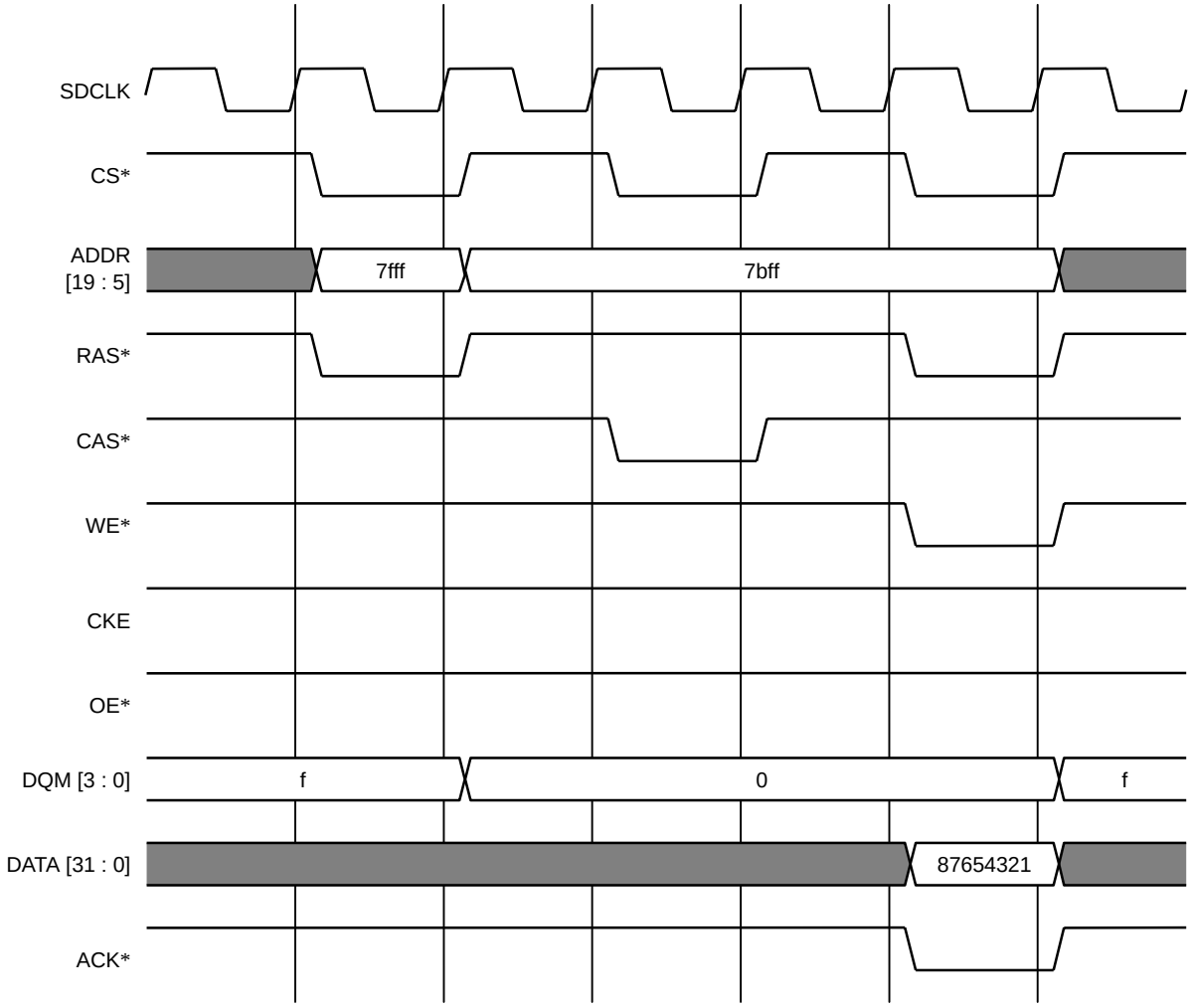


Figure 8.6.1 Single Read from 32-bit SDRAM (t_{CLK} = 15, t_{RCD} = 2, t_{CASL} = 2)

8.6.2 SDRAM Single Write Operation in 32-bit Bus Mode

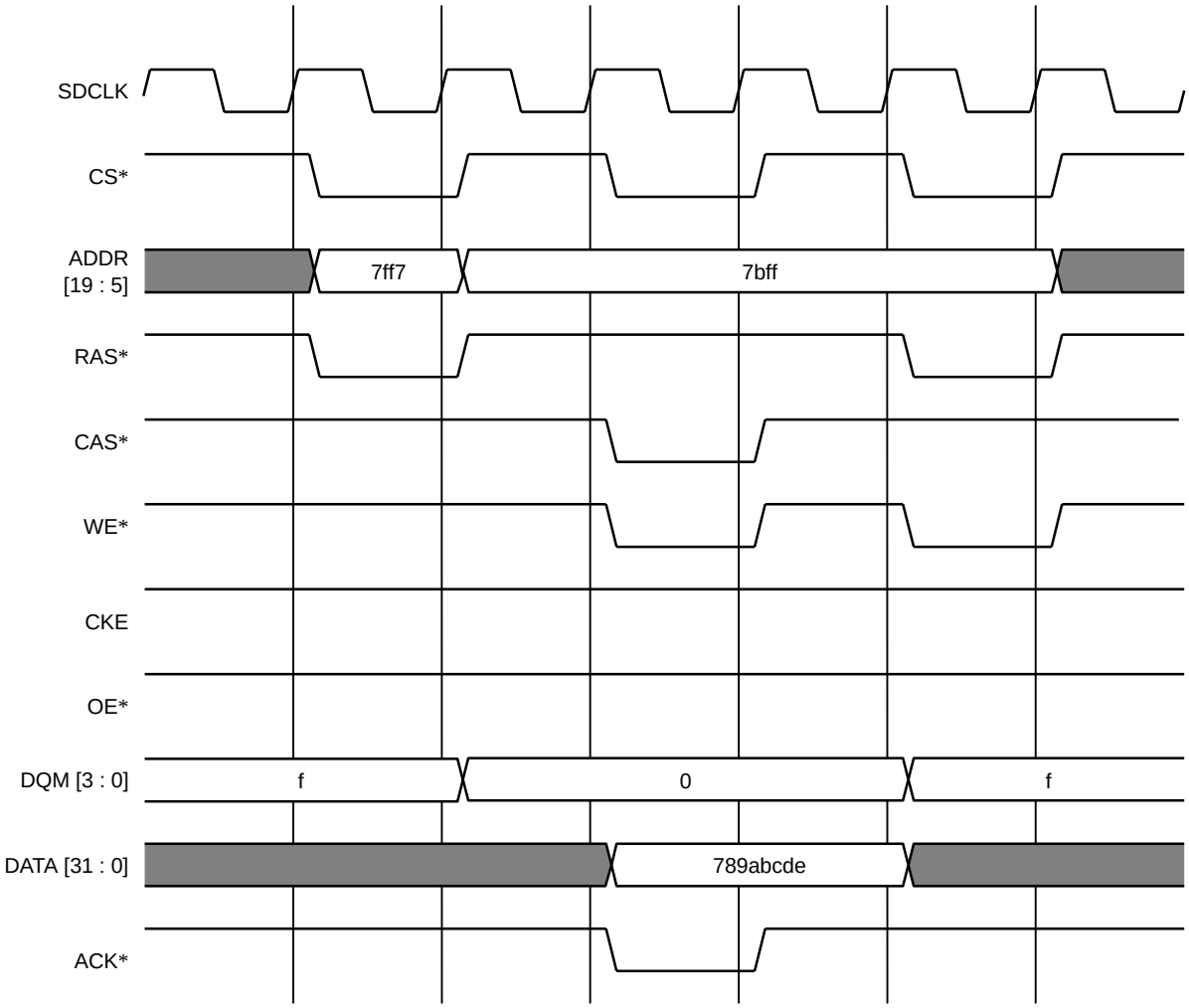


Figure 8.6.2 Single Write to 32-bit SDRAM (t_{CLK} = 15, t_{RCD} = 2, t_{RAS} = 4)

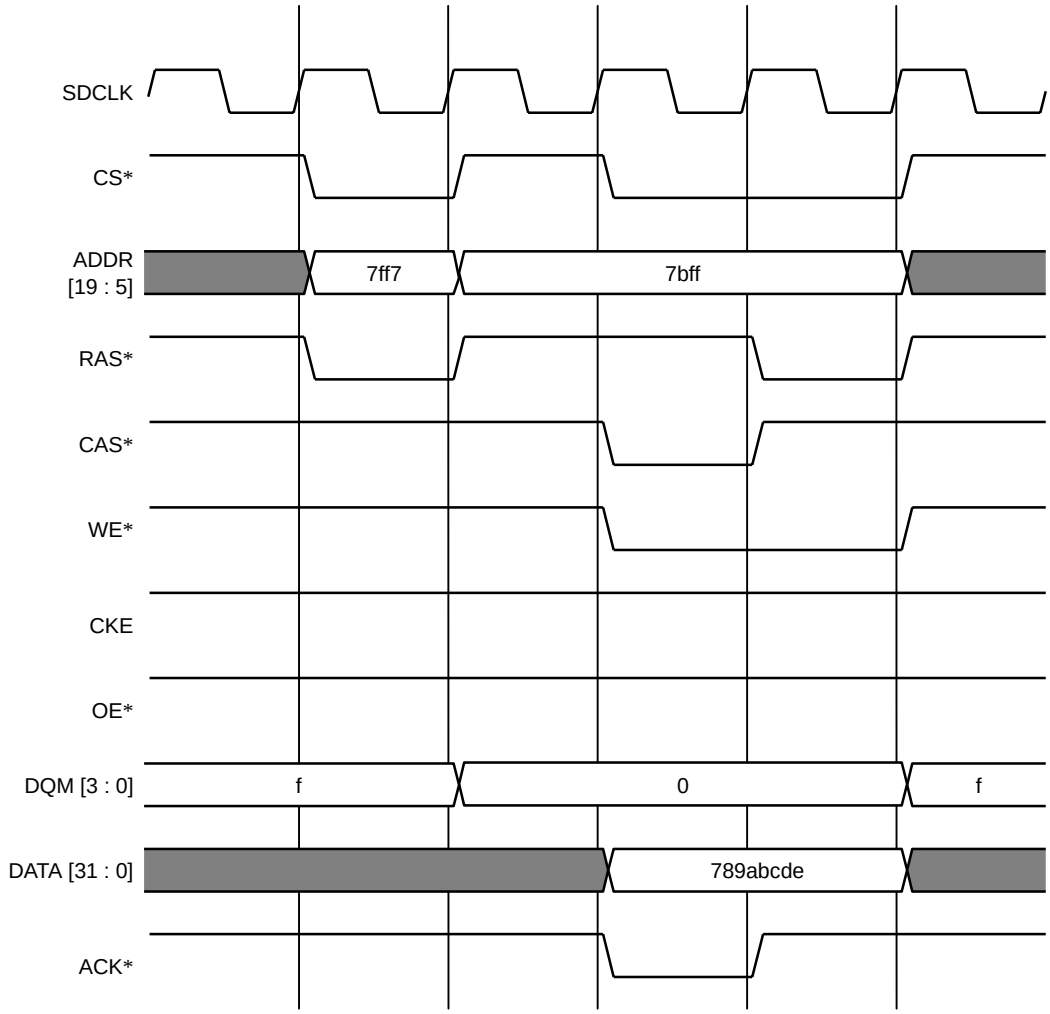


Figure 8.6.3 Single Write to 32-bit SDRAM (t_{CLK} = 15, t_{RCD} = 2, t_{RAS} = 3)

8.6.3 SDRAM Burst Read Operation in 32-bit Bus Mode

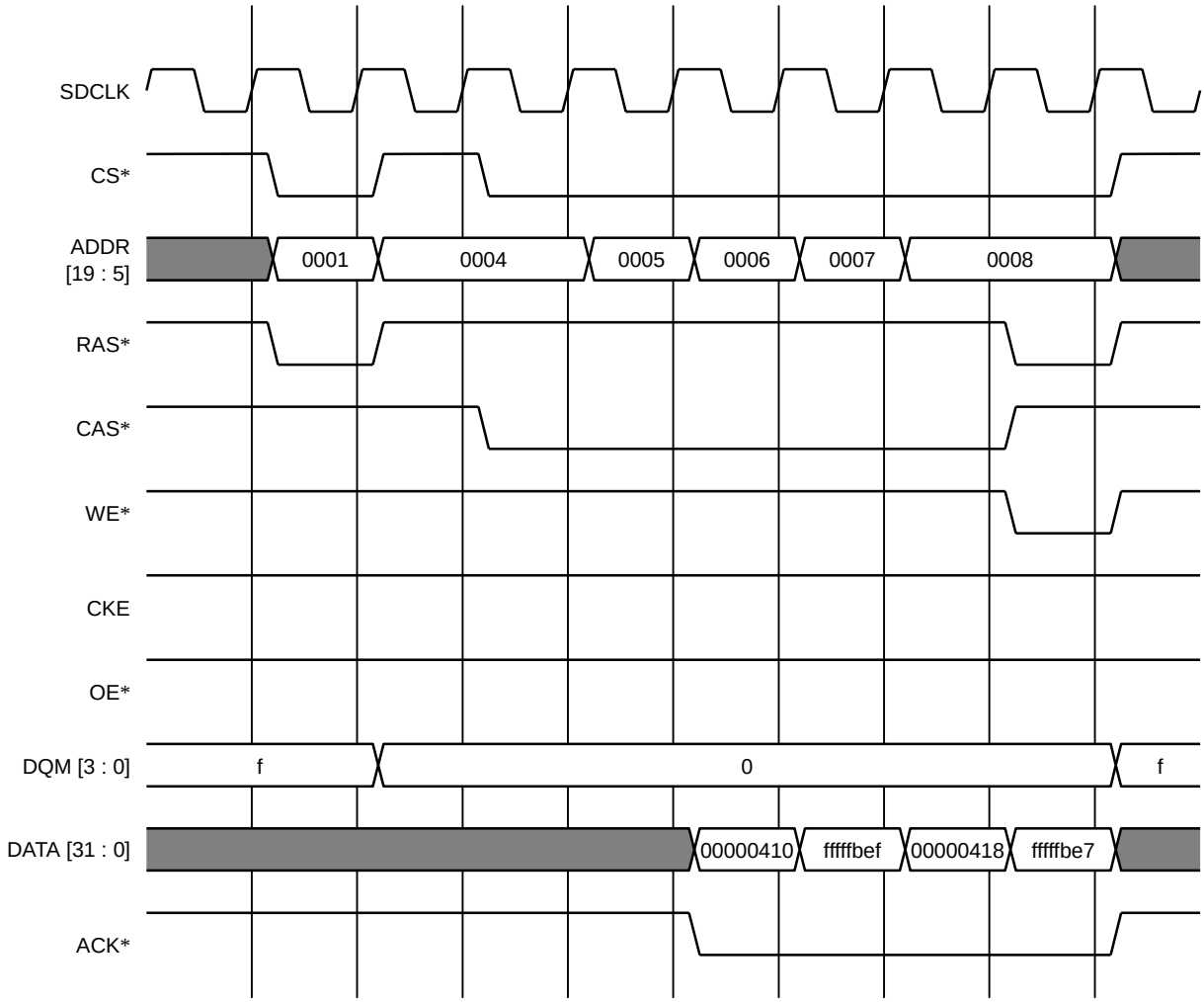


Figure 8.6.4 Burst Read from 32-bit SDRAM (4 Words, t_{CLK} = 15, t_{RCD} = 2, t_{CASL} = 2)

8.6.4 SDRAM Burst Write Operation in 32-bit Bus Mode

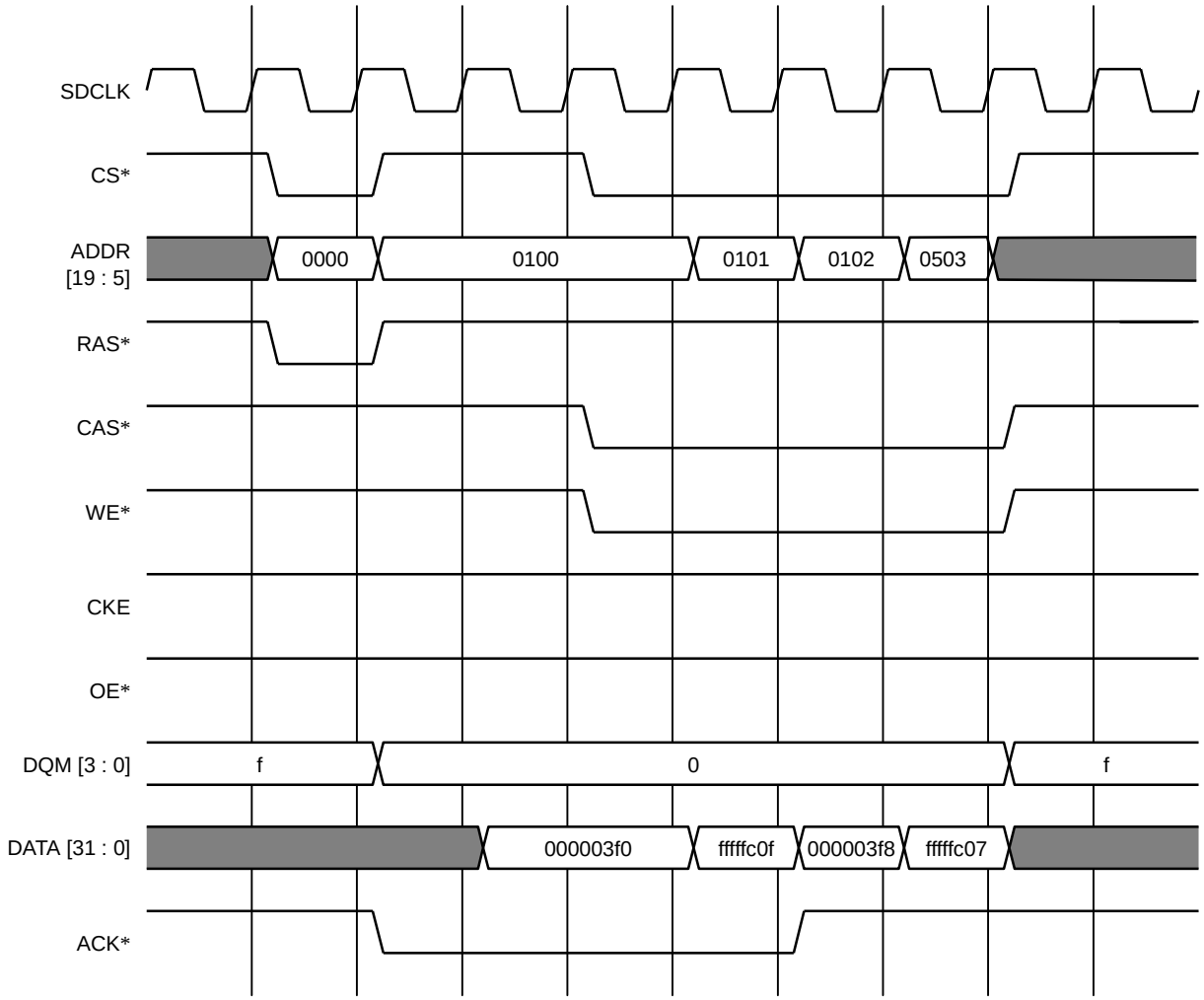


Figure 8.6.5 Burst Write to 32-bit SDRAM (4 Words, t_{CLK} = 15, t_{RCD} = 2, t_{RAS} = 4)

8.6.5 SDRAM Read in 32-bit Bus Mode (Crossing Page Boundary)

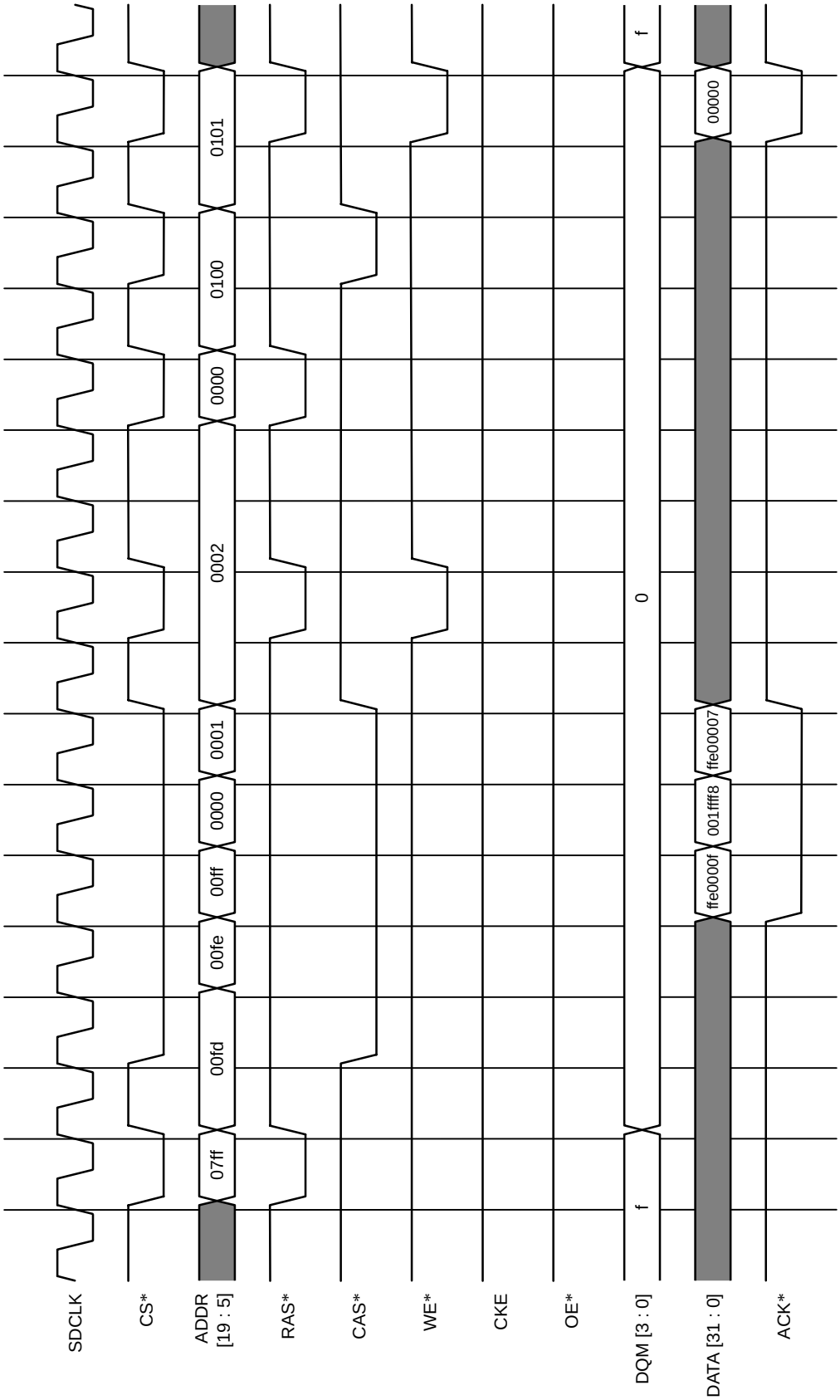


Figure 8.6.6 Read From 32-bit SDRAM (Crossing Page Boundary)
(4th Word Crossing Page Boundary, t_{CLK} = 15, t_{RCD} = 2, t_{CASL} = 2)

8.6.6 SDRAM Write in 32-bit Bus Mode (Crossing Page Boundary)

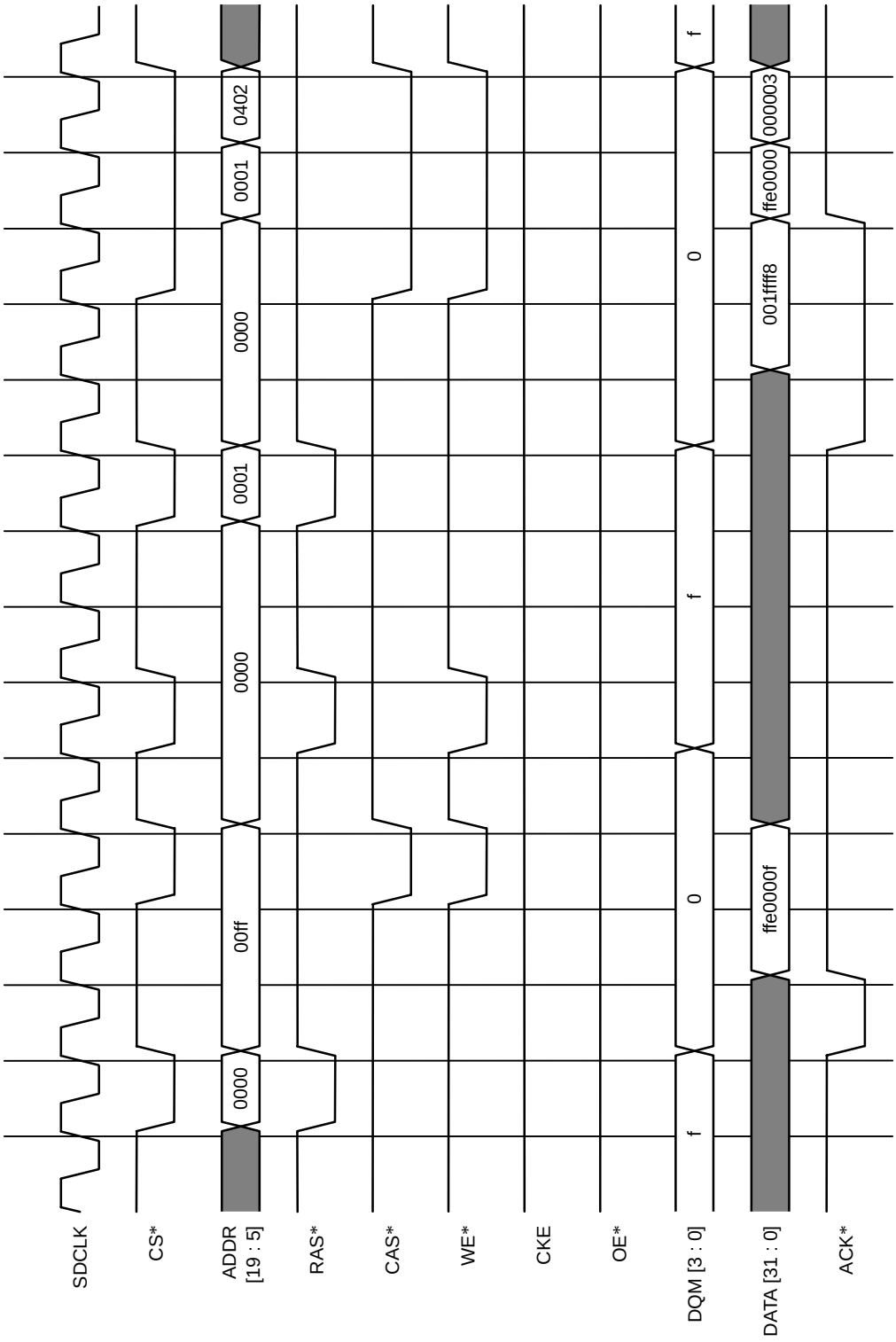


Figure 8.6.7 Write to 32-bit SDRAM (Crossing Page Boundary)
(2nd Word Crossing Page Boundary, t_{CLK} = 15, t_{RCD} = 2, t_{RAS} = 4)

8.6.7 SDRAM Slow Burst Write Operation in 32-bit Bus Mode

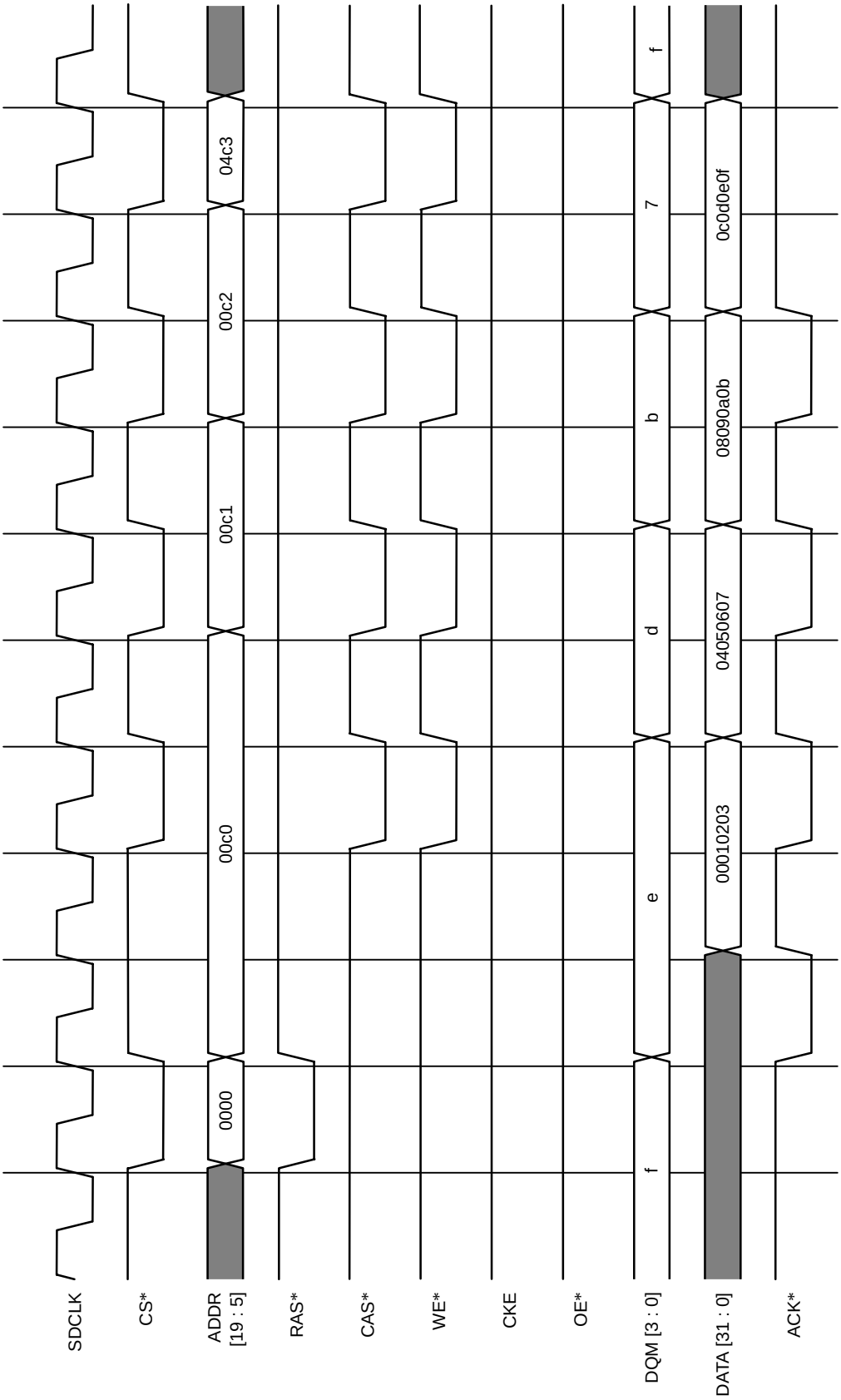


Figure 8.6.8 Slow Burst Write to 32-bit SDRAM (4 Words, t_{CLK} = 15, t_{RCD} = 2, t_{RAS} = 4)

8.6.8 SDRAM Single Read Operation in 16-bit Bus Mode

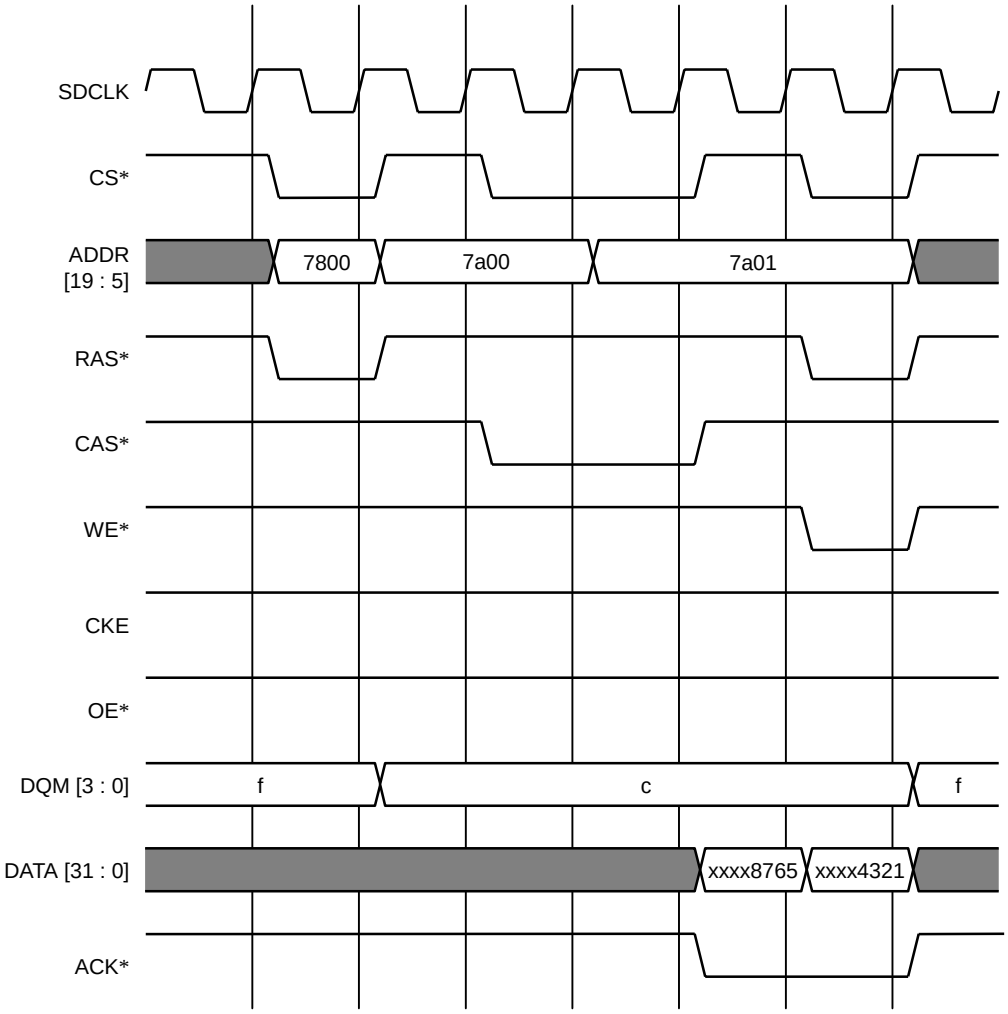


Figure 8.6.9 Single Read from 16-bit SDRAM (t_{CLK} = 15, t_{RCD} = 2, t_{CASL} = 2)

8.6.9 SDRAM Single Write Operation in 16-bit Bus Mode

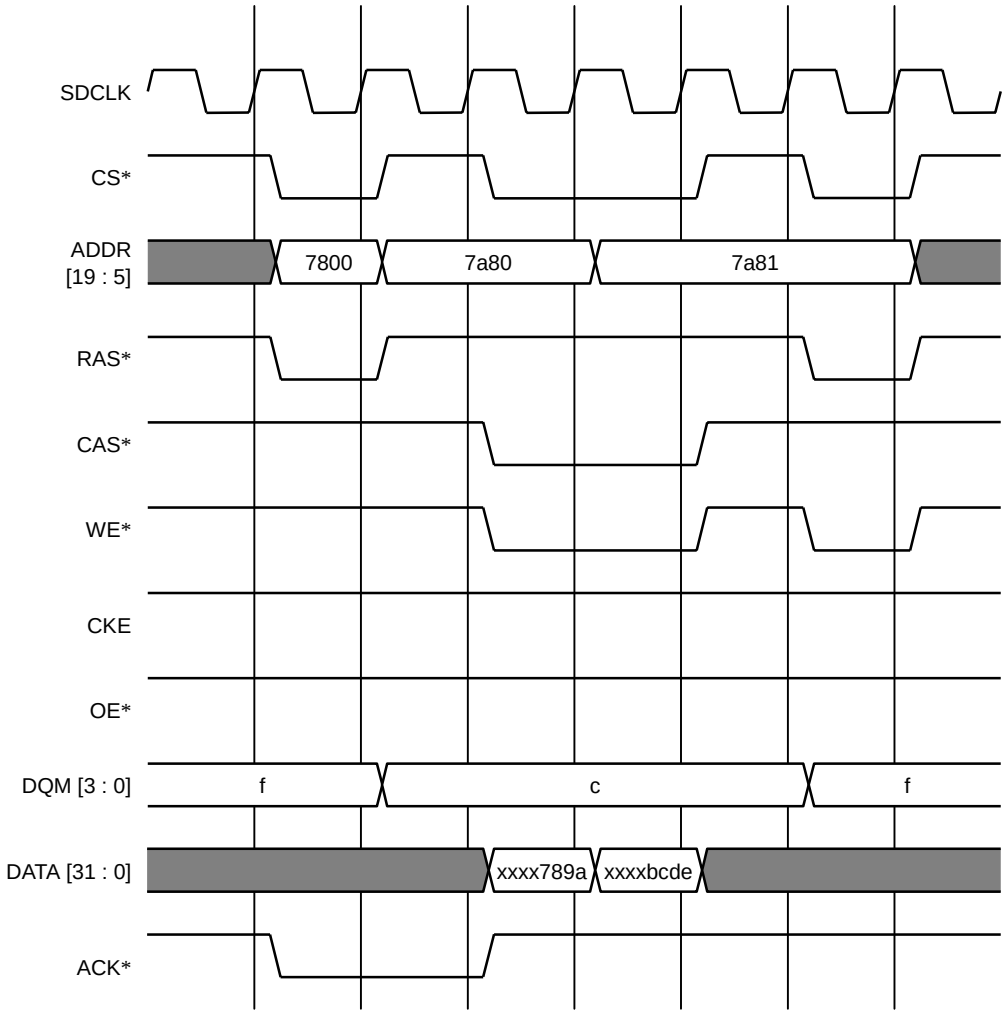


Figure 8.6.10 Single Write to 16-bit SDRAM (t_{CLK} = 15, t_{RCD} = 2, t_{RAS} = 4)

8.6.10 DIMM Flash Memory Single Read Operation in 32-bit Bus Mode

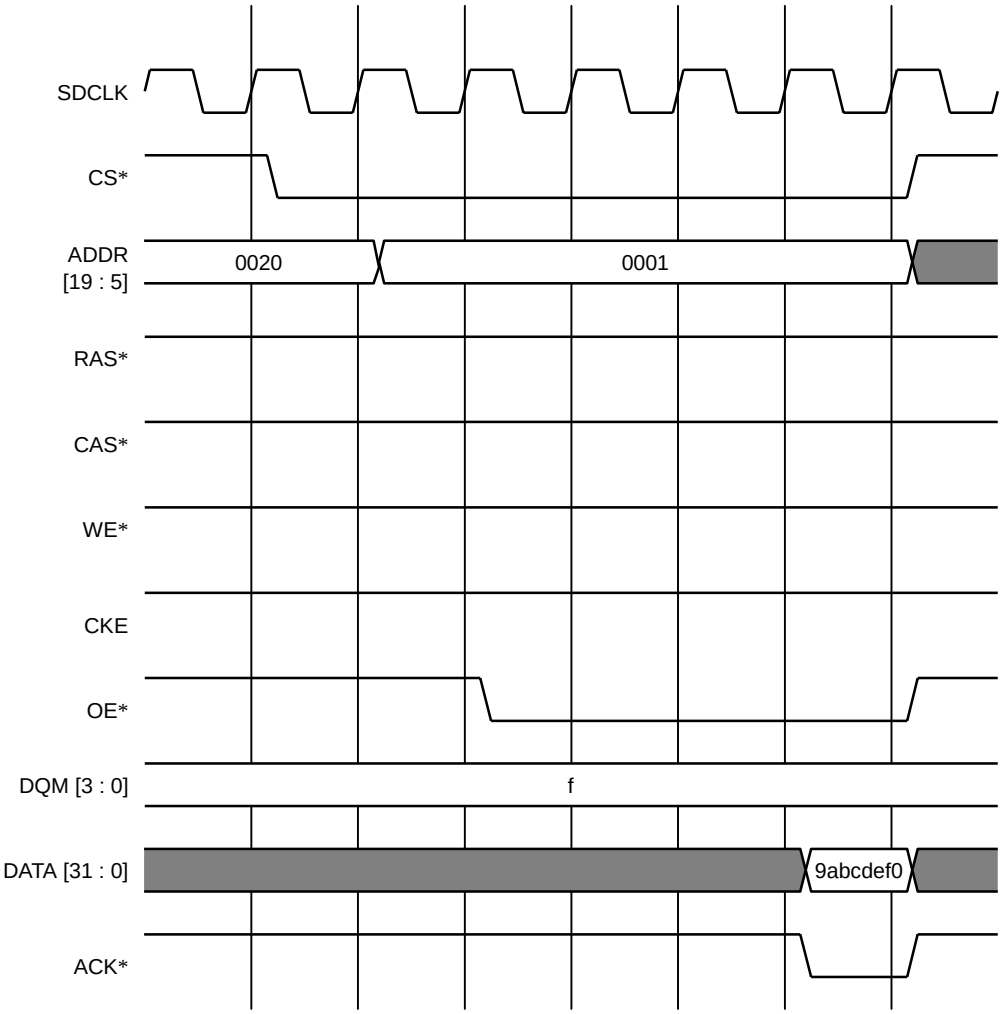


Figure 8.6.11 Single Read from 32-bit DIMM Flash Memory (t_{CLK} = 15, Wait = 4)

8.6.11 DIMM Flash Memory Single Write Operation in 32-bit Bus Mode

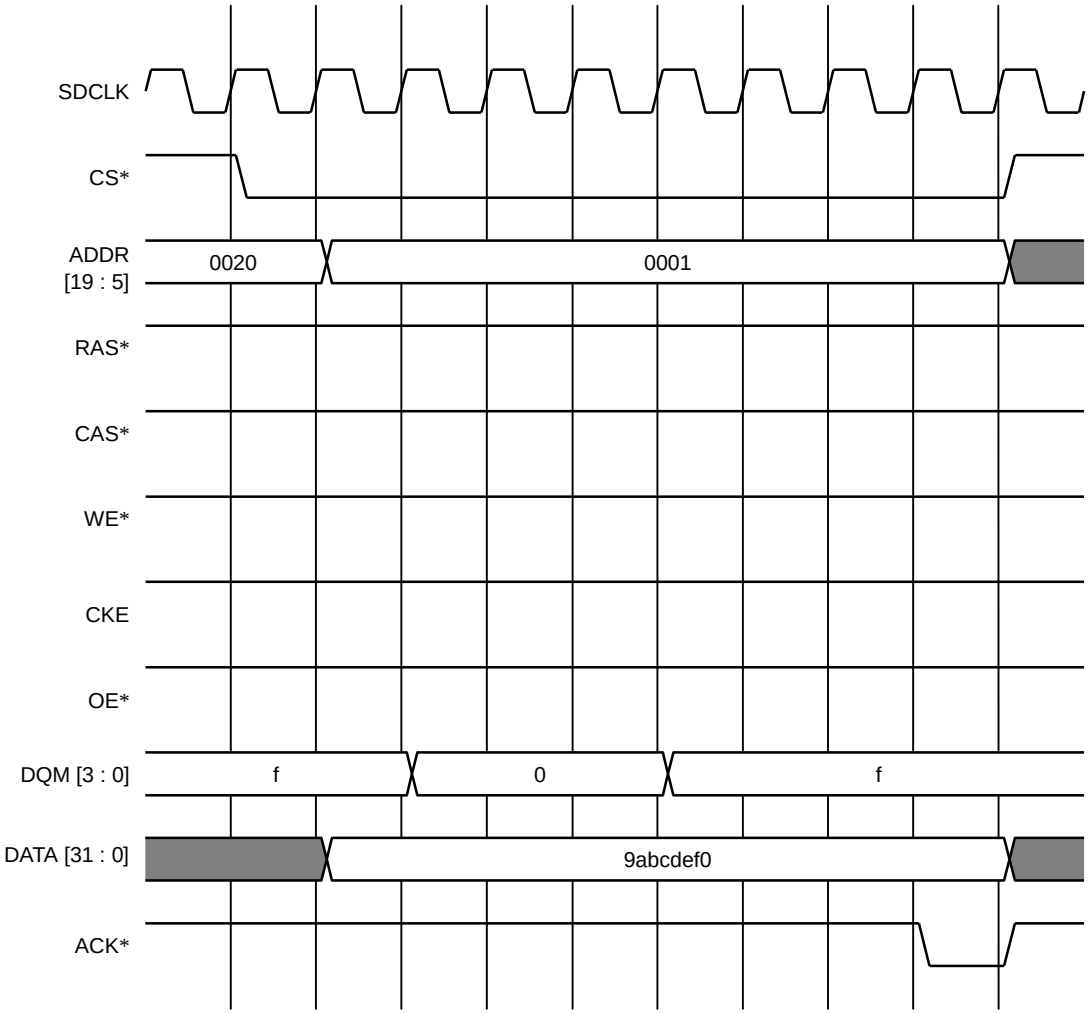


Figure 8.6.12 Single Write to 32-bit DIMM Flash Memory (t_{CLK} = 15, Wait = 3)

8.6.12 SMROM Single Read Operation in 32-bit Bus Mode

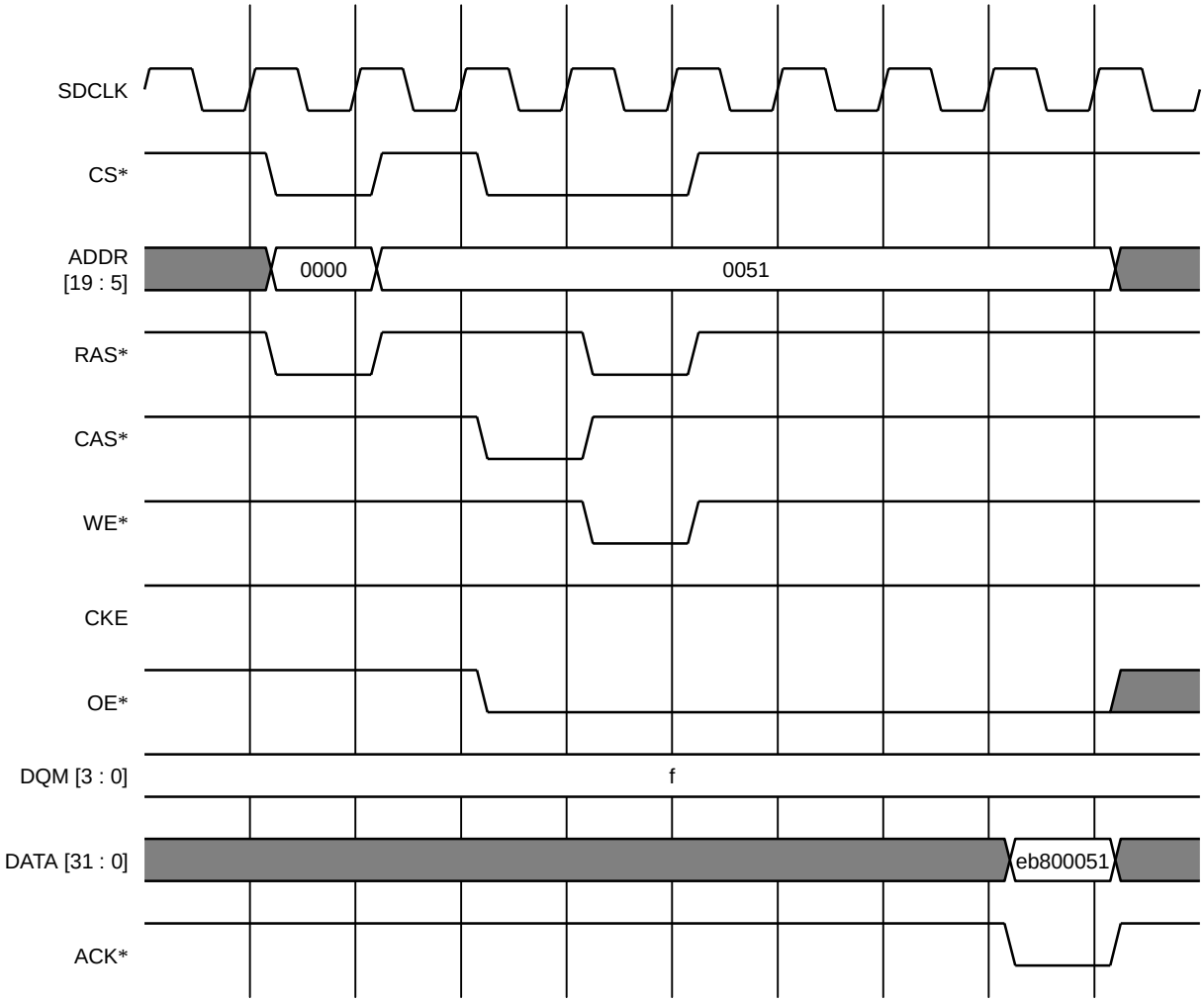


Figure 8.6.13 Single Read from 32-bit SMROM (t_{CLK} = 15, t_{RCD} = 2, t_{CASL} = 5)

8.6.13 SMROM Burst Read Operation in 32-bit Bus Mode

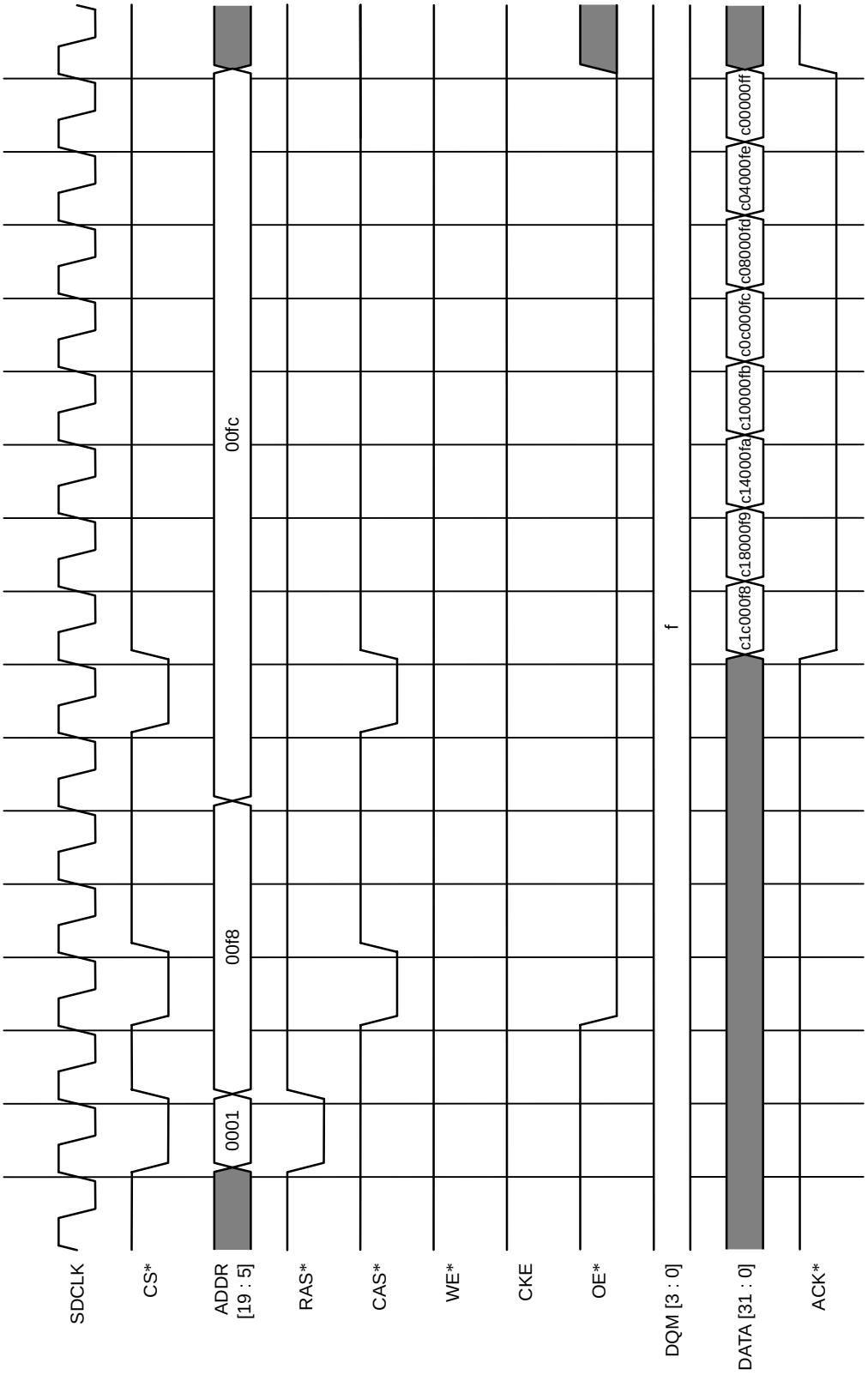


Figure 8.6.14 Burst Read from 32-bit SMROM (8 Words, t_{CLK} = 15, t_{RCD} = 2, t_{CASL} = 5)

8.6.14 Low Power and Power-down Mode

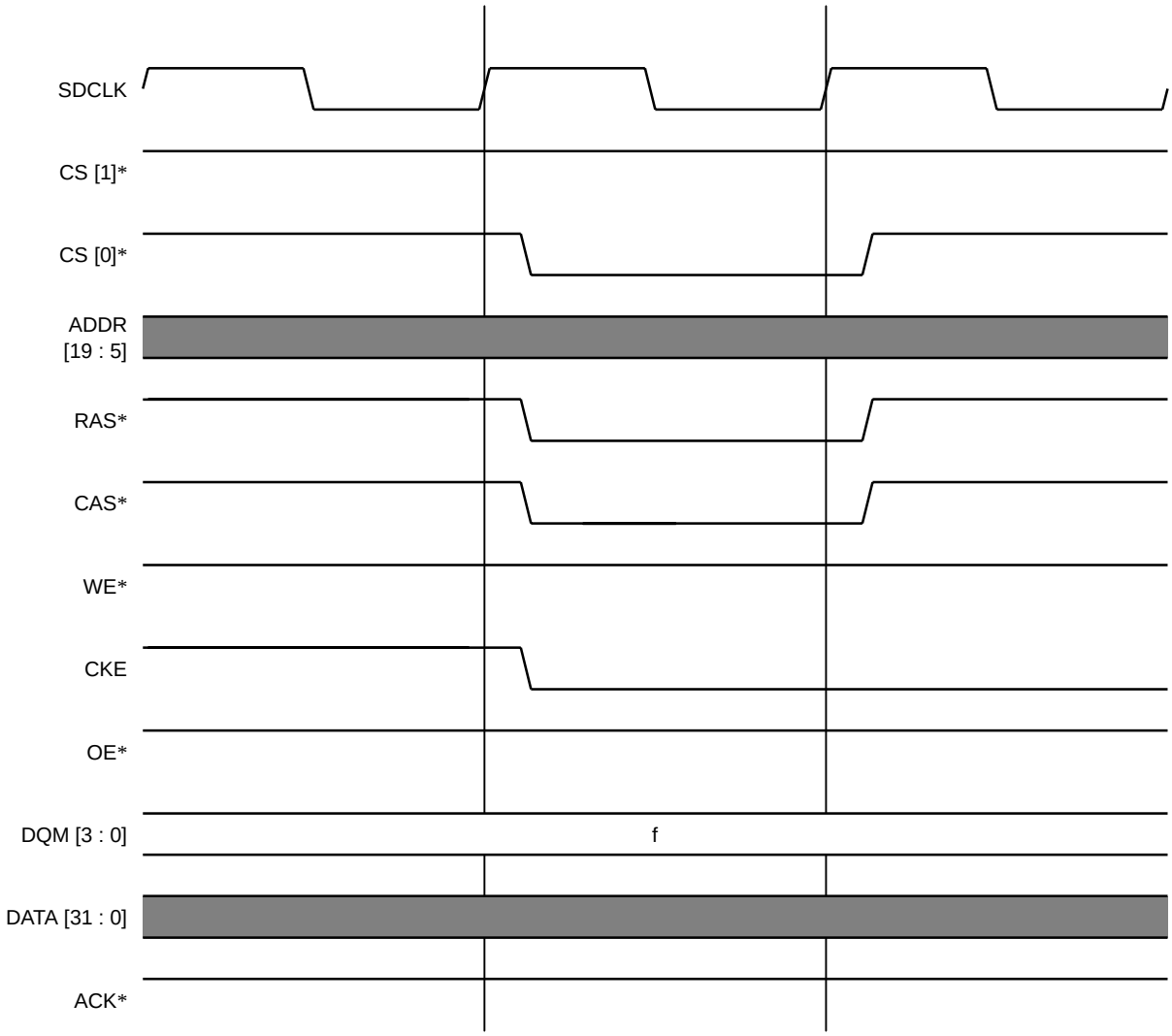


Figure 8.6.15 Enter Low Power Mode (t_{CLK} = 15, Ch. 0 = SDRAM, Ch. 1 = SMROM)

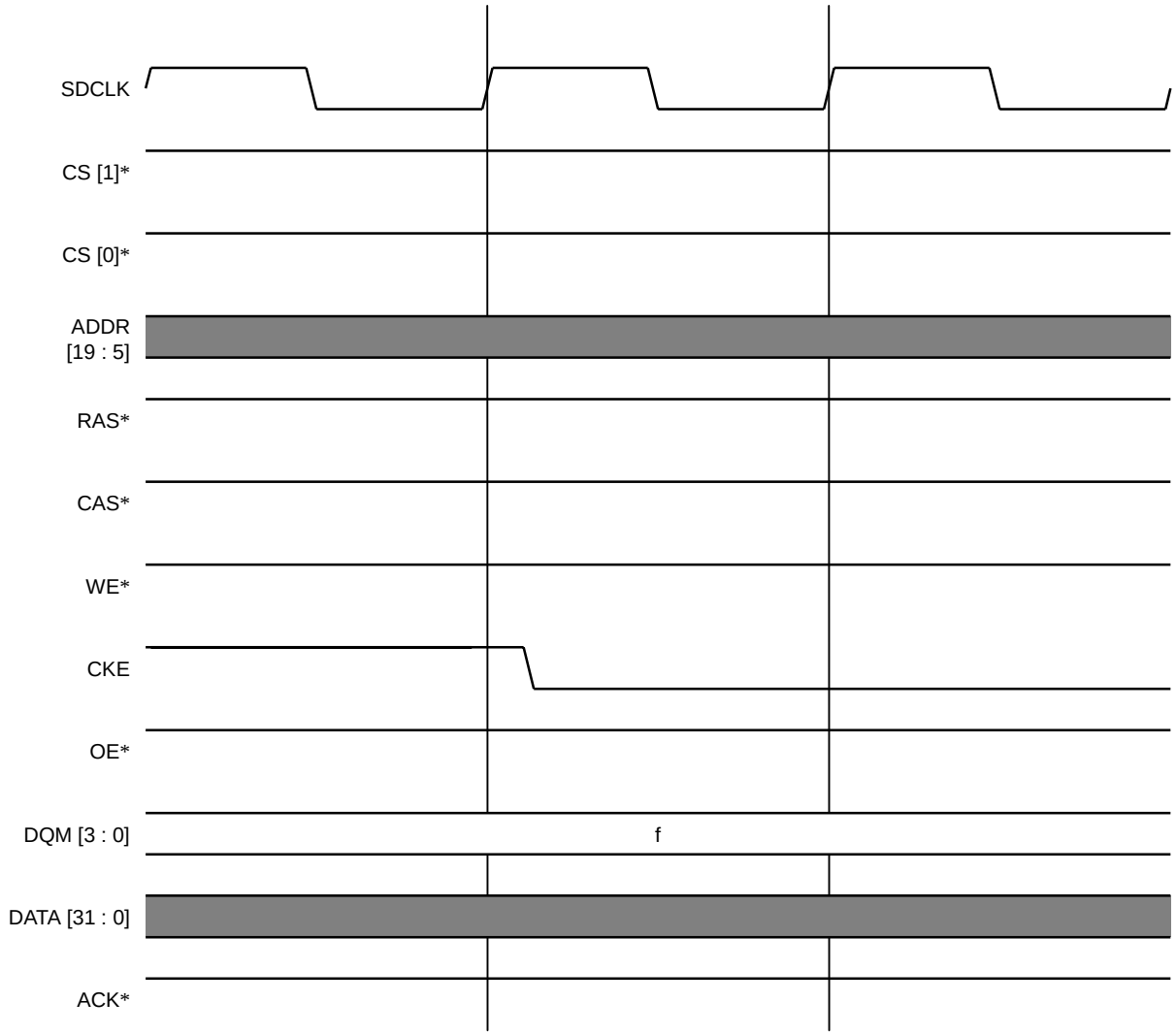


Figure 8.6.16 Enter Power-down Mode (t_{CLK} = 15, Ch. 0 = SDRAM, Ch. 1 = SMROM)

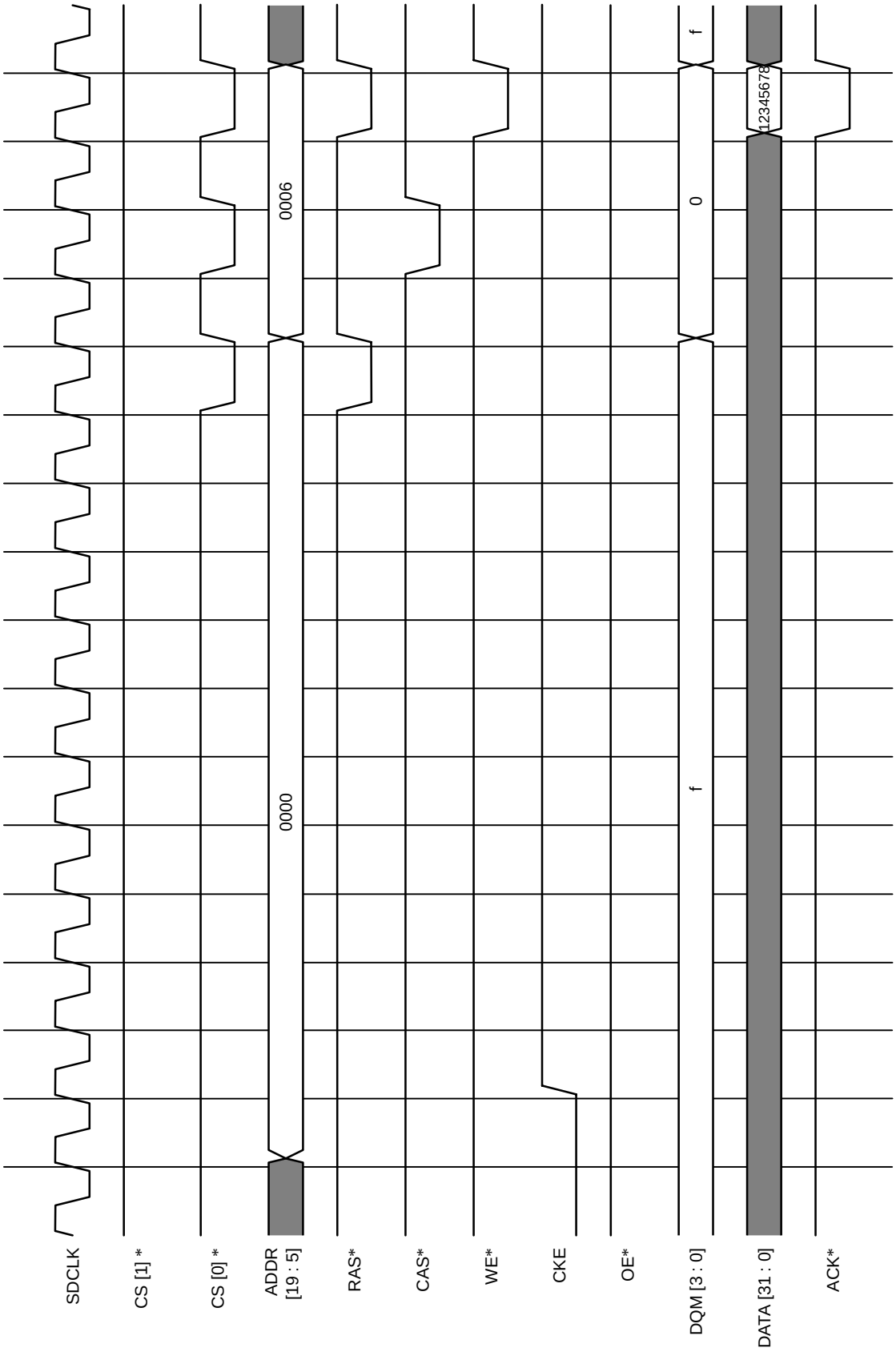


Figure 8.6.17 Auto-Exit from Power-down Modes (t_{CLK} = 15, Ch. 0 = SDRAM, Ch. 1 = SMROM)

8.6.15 SGRAM in 32-bit Bus Mode

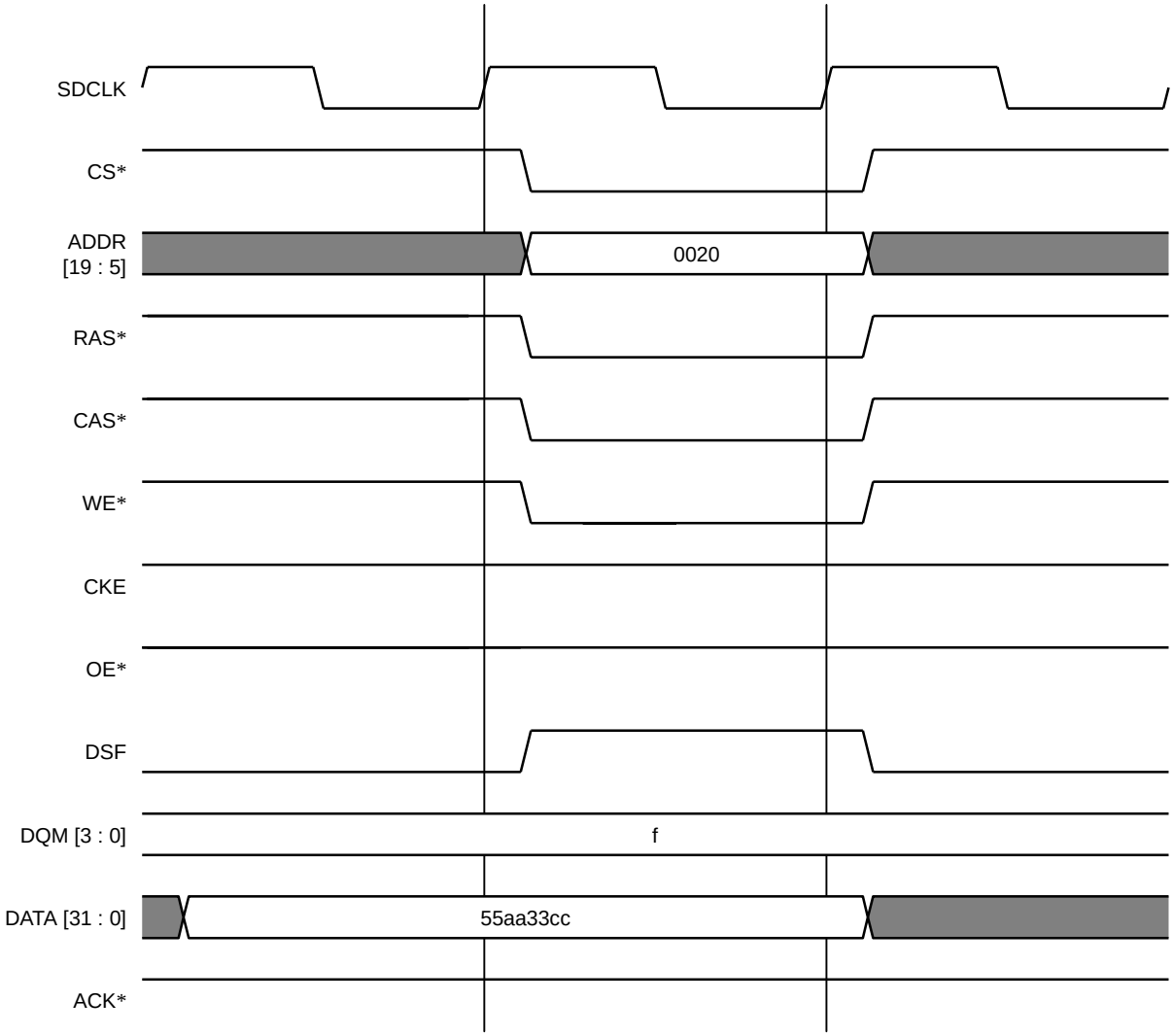


Figure 8.6.18 SMRS of Mask Register in 32-bit SGRAM

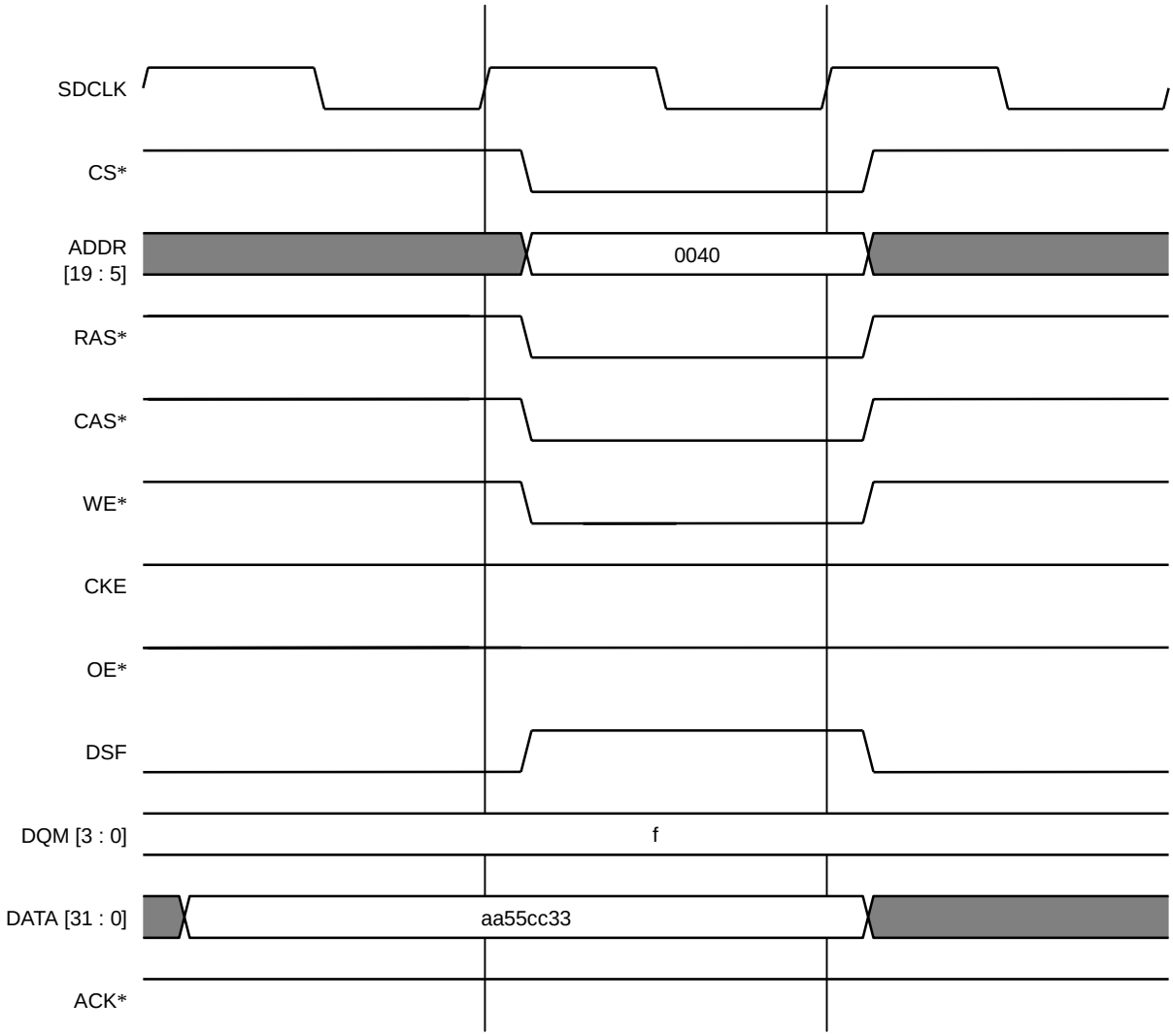


Figure 8.6.19 SMRS of Color Register in 32-bit SGRAM

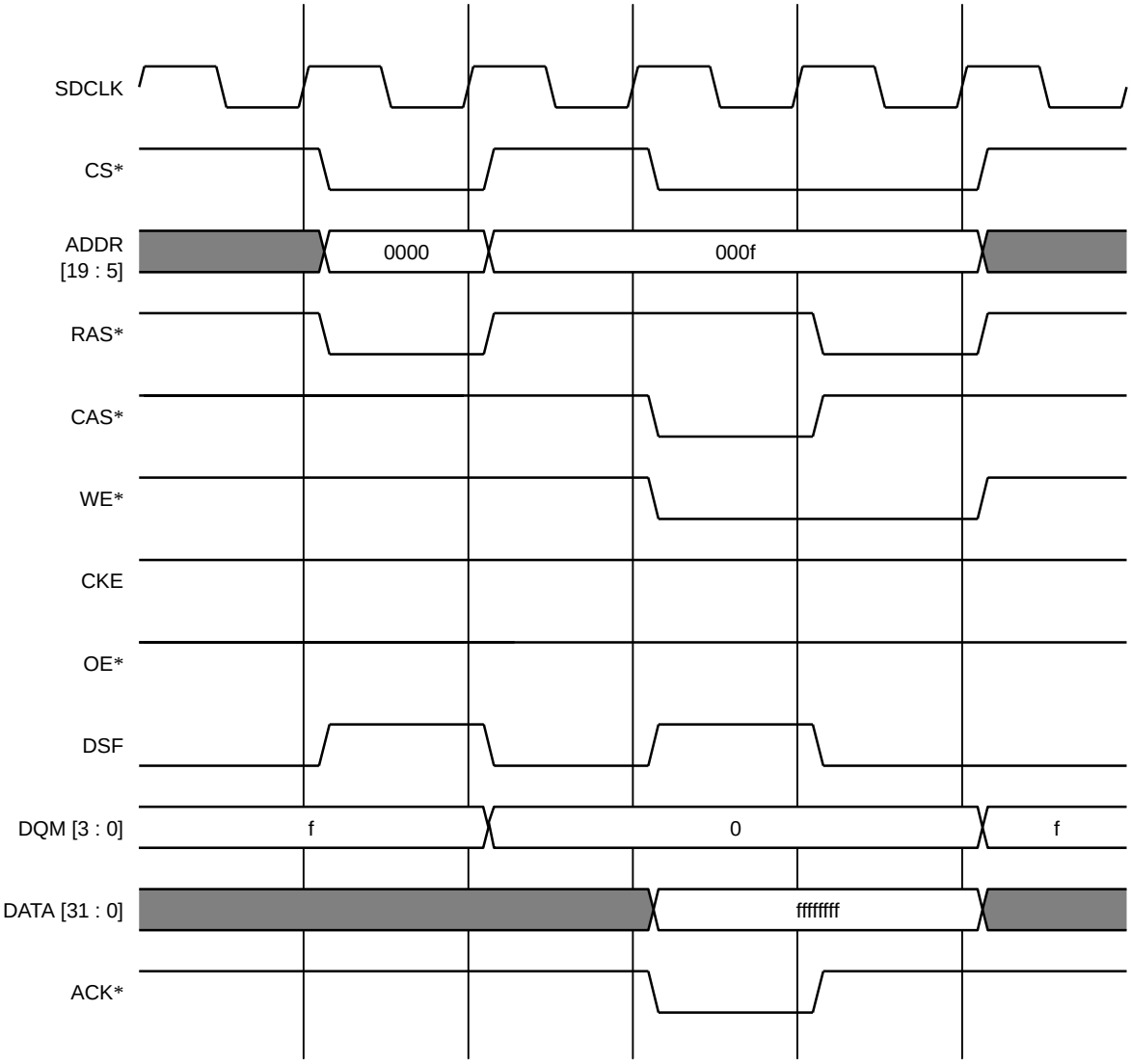


Figure 8.6.20 Simultaneous WPB and BW to 32-bit SGRAM (Non-burst, t_{RCD} = 2, t_{RAS} = 3, t_{BPL} = 1)

8.6.16 External DMA Operation (Big Endian)

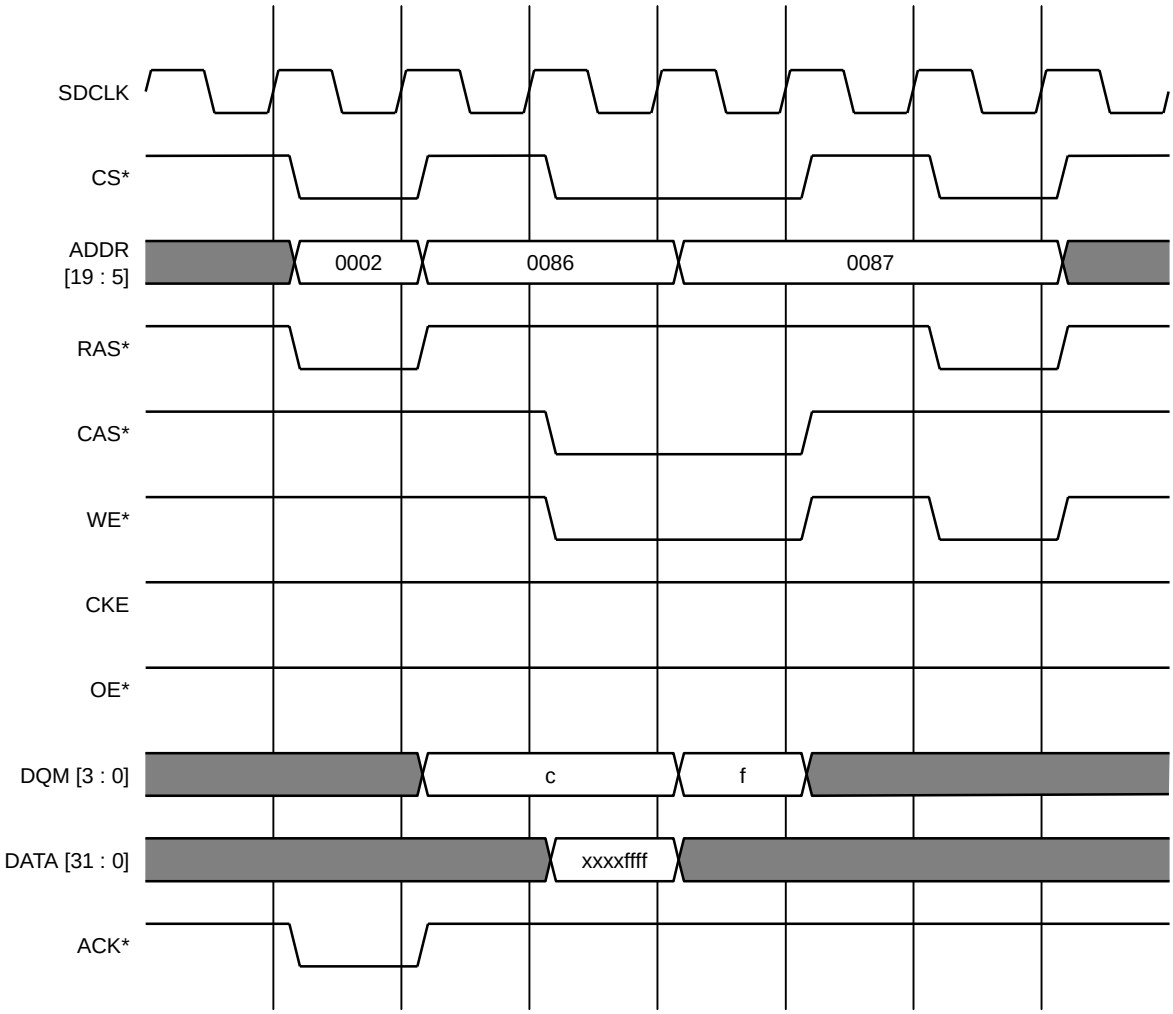


Figure 8.6.21 Big Endian External DMA Transfer: Single Address from 16-bit I/O to 16-bit SDRAM (GBE[3:0]* = 3: Even Address Write)

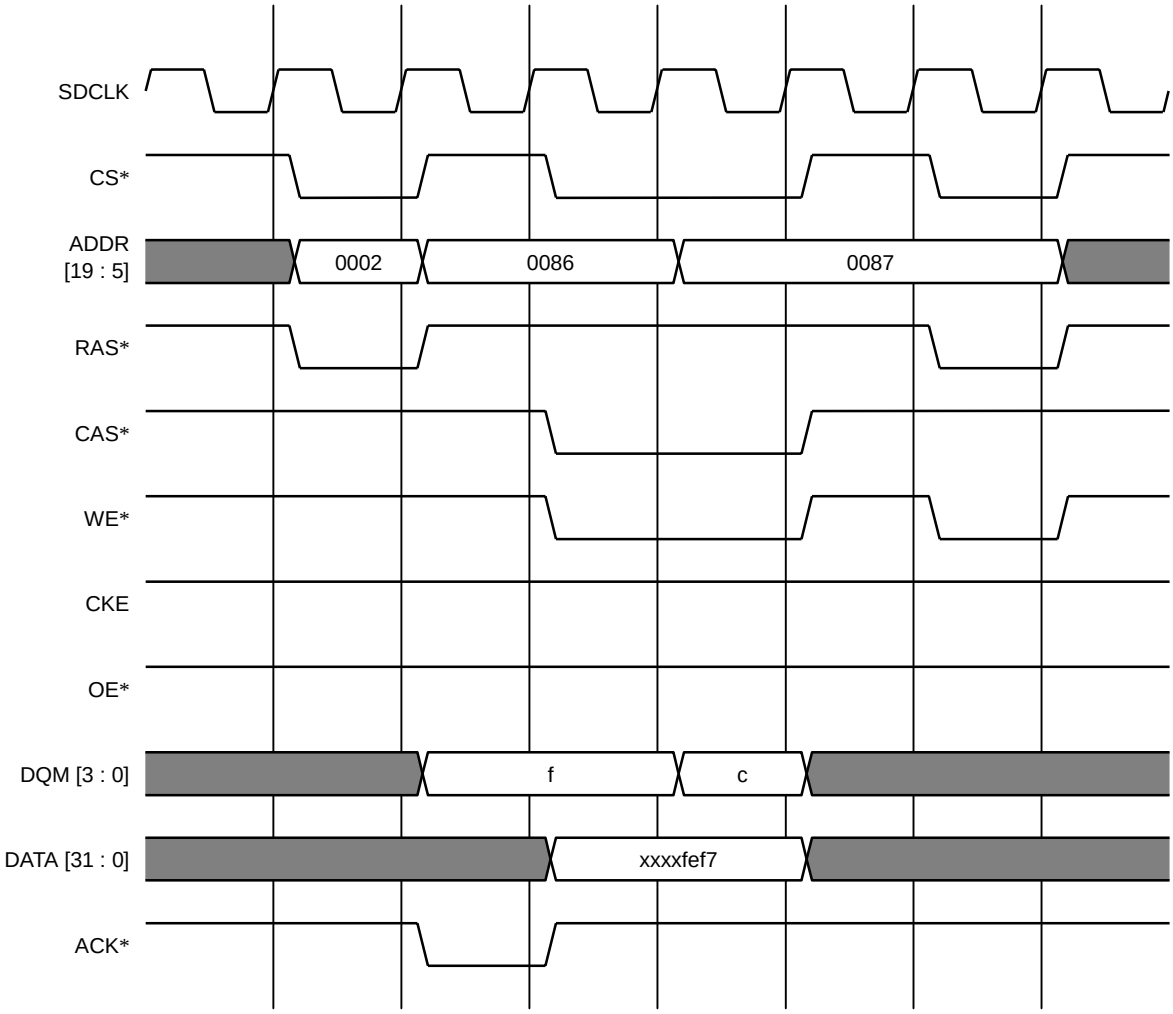


Figure 8.6.22 Big Endian External DMA Transfer: Single Address from 16-bit I/O to 16-bit SDRAM (GBE[3:0]* = C: Odd Address Write)

8.6.17 External DMA Operation (Little Endian)

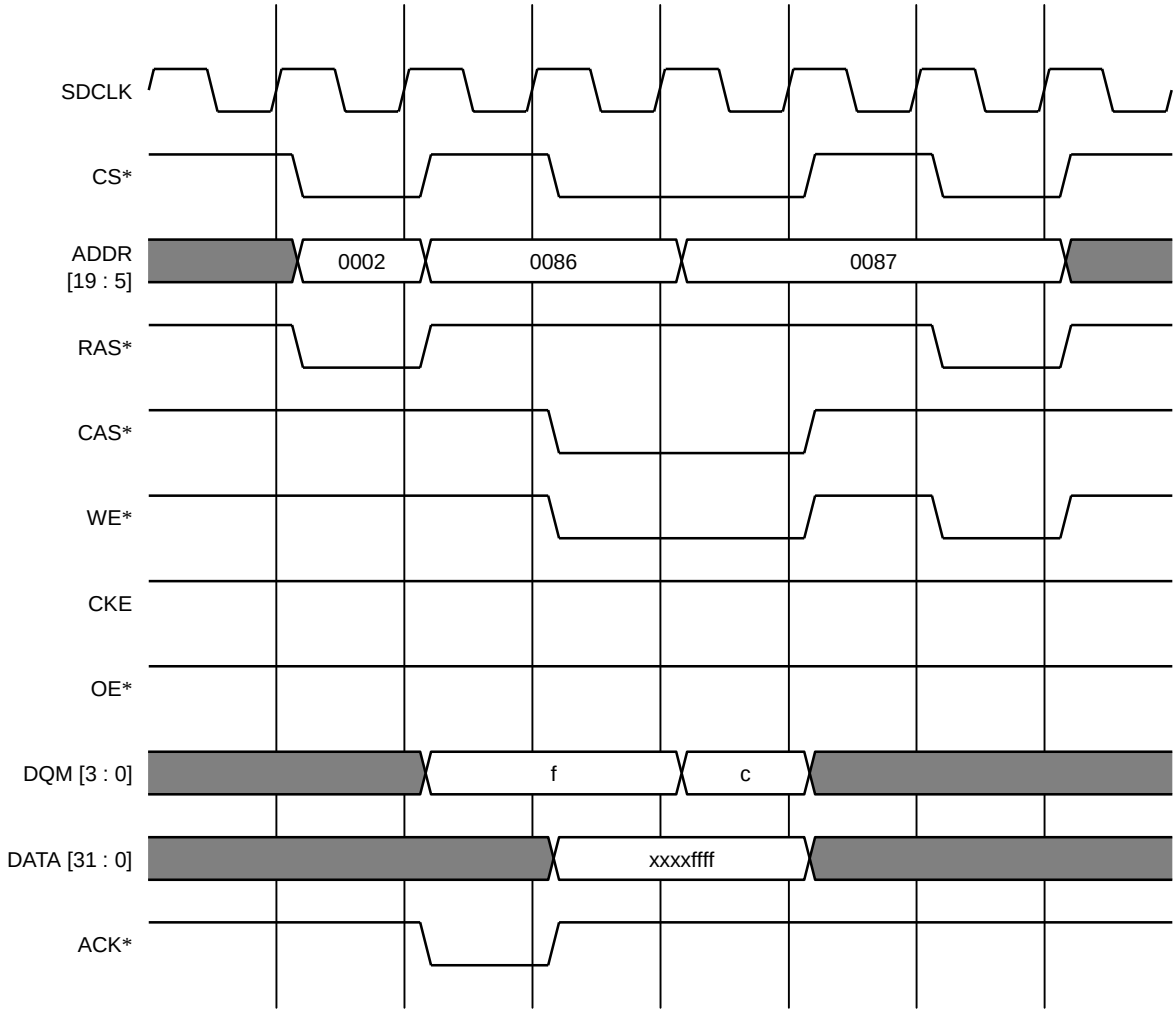


Figure 8.6.23 Little Endian External DMA Transfer: Single Address from 16-bit I/O to 16-bit SDRAM (GBE[3:0]* = 3: Odd Address Write)

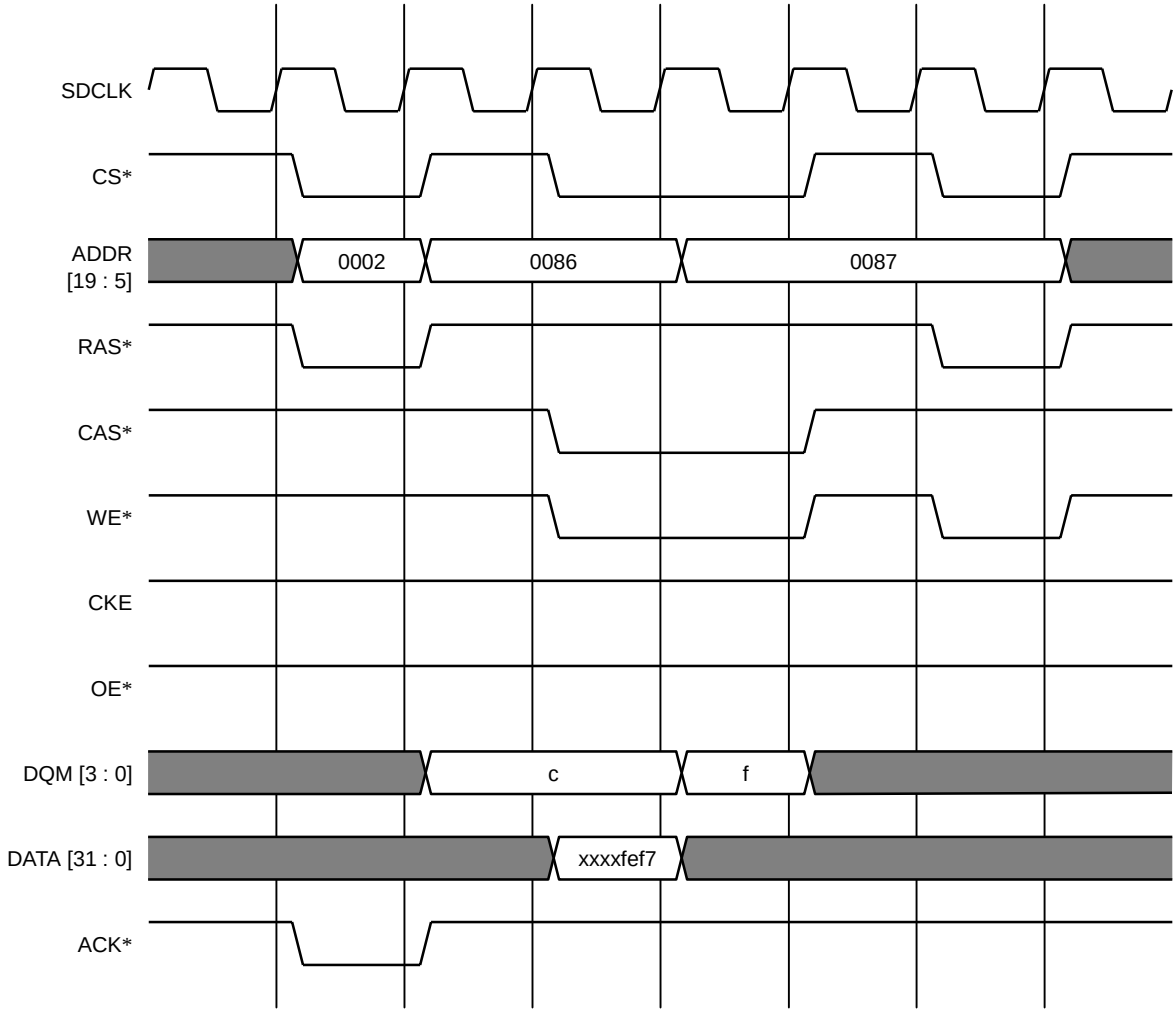


Figure 8.6.24 Little Endian External DMA Transfer: Single Address from 16-bit I/O to 16-bit SDRAM (GBE[3:0]* = C: Even Address Write)

8.7 Examples of Using SDRAM

Figure 8.7.1 shows an example of using SDRAM (× 16 bits). Figure 8.7.2 shows an example of using 100-pin DIMM SDRAM.

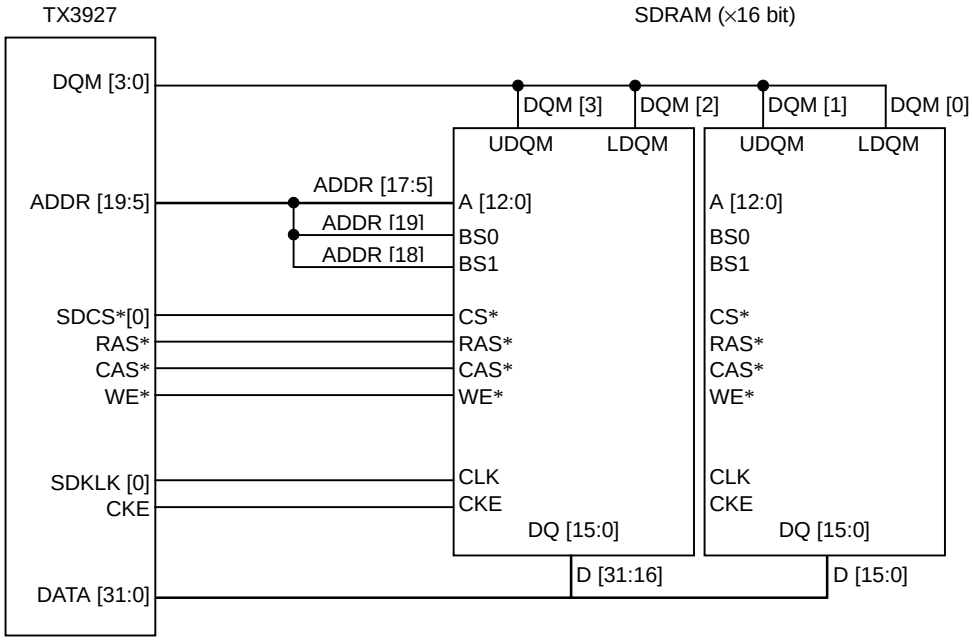


Figure 8.7.1 Example of Using SDRAM (× 16 bits) (32-bit Data Bus)

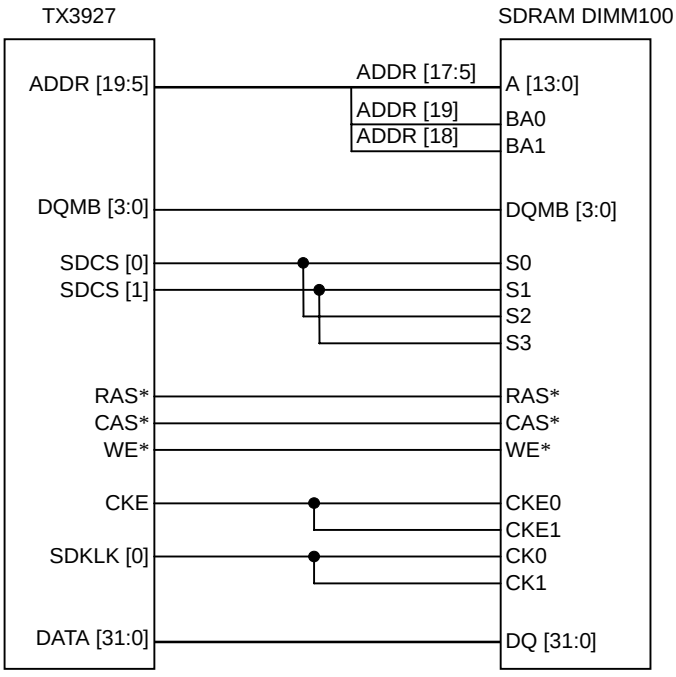


Figure 8.7.2 Example of Using SDRAM (100-pin DIMM)

9. External Bus Controller

9.1 Features

The external bus controller (ROMC) generates all the signals and timings required to access ROMs, SRAMs, and I/O peripherals.

Its features include:

- Operates at up to 66 MHz.
- 8 channels - each independently configurable with 32-bit channel control registers
- Supports accesses to ROM, mask ROM, page mode ROM, EPROM, EEPROM, SRAM, and flash memory.
- Supports accesses to I/O peripherals.
- 16-bit or 32-bit data bus selectable on a per channel basis
- Full- or half-speed bus mode selectable on a per channel basis
- Supports memory sizes from 1 Mbyte to 1 Gbyte in 32-bit bus mode and 1 Mbyte to 512 Mbytes in 16-bit bus mode.
- Multiplexed address inputs (ADDR[19:2]). The ACE* output is provided to latch the upper address (bits 29 to 20) externally.
- G-Bus burst lengths of 4, 8, 16, and 32 words
- Page sizes of 4, 8, and 16 words in page mode
- Programmable setup and hold times for the address, chip enable, write enable, and output enable signals.
- Supports external acknowledge (ACK*) and external Ready signals.
- Supports a boot memory device on Channel 0.

  BOOT16: Selects the data bus width (16-bit or 32-bit).

  BOOTAI: Selects ACK* output or ACK* input mode.

  BOOTBC: Selects whether the BWE* pin is used as byte enable or byte write enable.

  BOOTME[1:0]: Master enable and boot speed
- Supports global options.

  CHANHS: Selects the SYSCLK speed (half speed or full speed).

  ACEHOLD: Selects the address hold time for the ACE* signal (whether to change the address coincident with ACE* or after a delay of one clock cycle).
- Programmable timing per channel
- Programmable wait time per channel (0 to 63 cycles)

9.2 Block Diagram

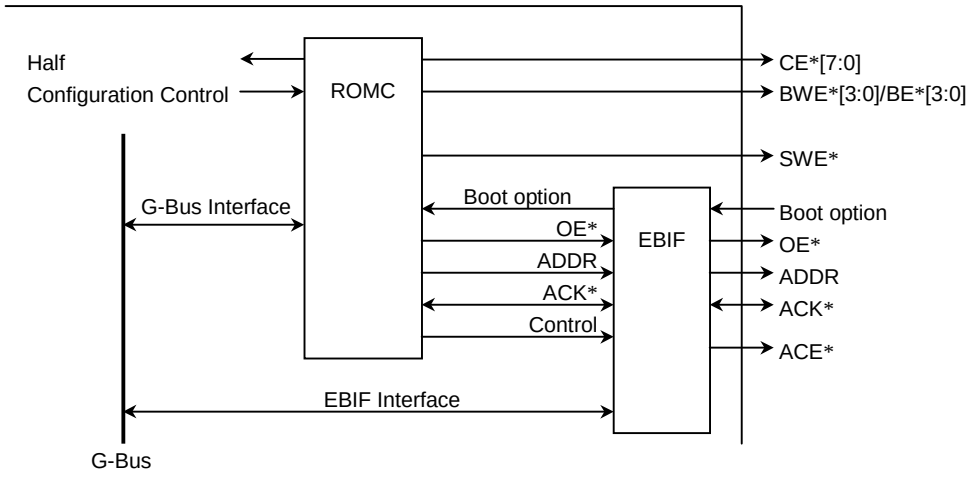


Figure 9.2.1 ROMC Connections within the TX3927

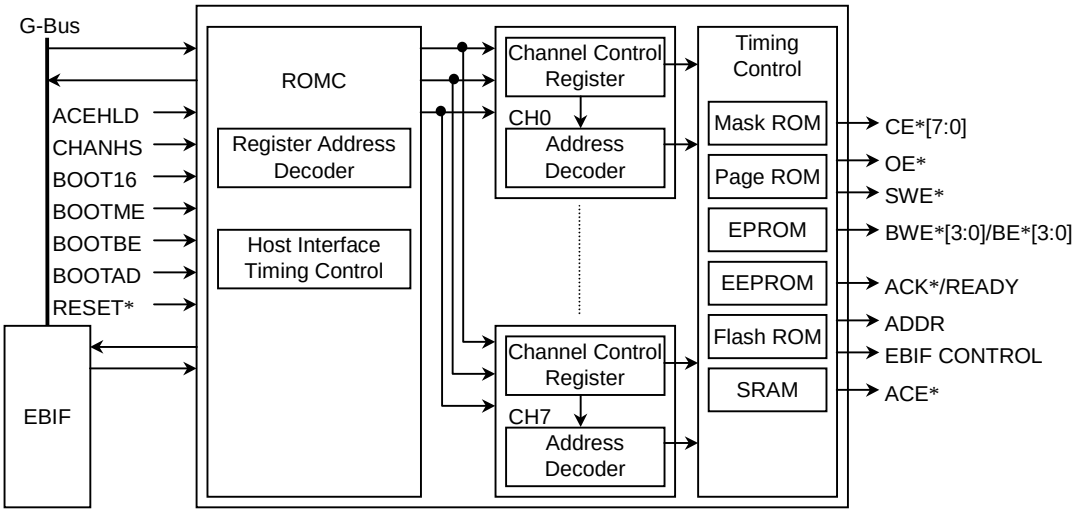


Figure 9.2.2 ROMC Block Diagram

9.3 Registers

9.3.1 Register Map

The base address of the ROMC is 0xFFFE_9000. All registers of the ROMC can only be word-accessed. Any other type of access will produce unexpected results.

For the bits not defined, write the values shown in the figures.

Table 9.3.1 ROM Controller Registers

Address	Register Mnemonic	Register Name
0xFFFE_9000	RCCR0	ROM Channel Control Register 0
0xFFFE_9004	RCCR1	ROM Channel Control Register 1
0xFFFE_9008	RCCR2	ROM Channel Control Register 2
0xFFFE_900C	RCCR3	ROM Channel Control Register 3
0xFFFE_9010	RCCR4	ROM Channel Control Register 4
0xFFFE_9014	RCCR5	ROM Channel Control Register 5
0xFFFE_9018	RCCR6	ROM Channel Control Register 6
0xFFFE_901C	RCCR7	ROM Channel Control Register 7

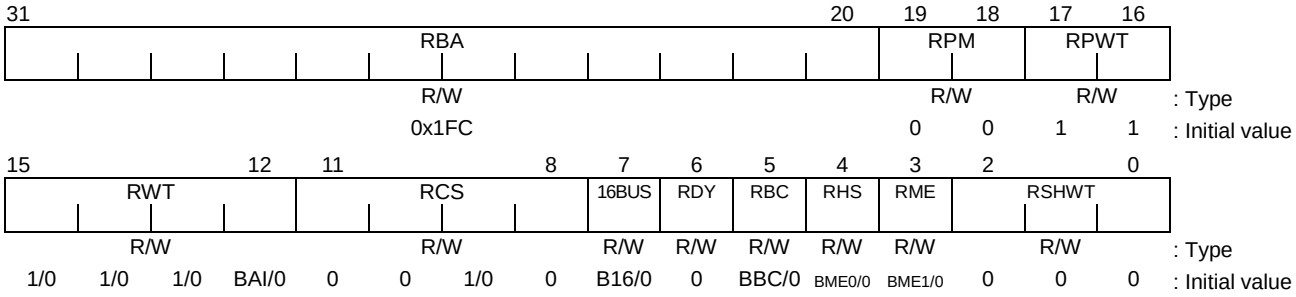
The ROMC has eight ROM Channel Control Registers. The eight registers only differ in the channel number and address; they all have the same functions, except that the initial values for Channel 0 depends on whether half-speed ROM or full-speed ROM mode is selected.

Channels 2 to 7 share the chip select signals with the SDRAM Controller. SDRAM and ROM channels with an identical number may not be enabled at the same time.

9.3.2 ROM Channel Control Registers (RCCR0-RCCR7)

0xFFFE_9000 (ch.0)
0xFFFE_9004 (ch.1)
0xFFFE_9008 (ch.2)
0xFFFE_900C (ch.3)
0xFFFE_9010 (ch.4)
0xFFFE_9014 (ch.5)
0xFFFE_9018 (ch.6)
0xFFFE_901C (ch.7)

Channel 0 can be used to control a boot memory device. Therefore, the initial values for ROM Channel Control Register 0 are set by the values of the boot signals.



Bits	Mnemonic	Field Name	Description
31:20	RBA	Base Address	ROM Control Base Address (initial value: 0x1FC) Specifies the physical base address of the channel.
19:18	RPM	Page Mode Page Size	ROM Control Page Mode Page Size (initial value: 00) Specifies the page size for word burst page mode. 00: Not configured for page mode 01: 4-word burst page mode 10: 8-word burst page mode 11: 16-word burst page mode
17:16	RPWT	Page Mode Wait Time	ROM Control Page Mode Wait Time (initial value: 11) In page mode, specifies the number of wait states to be taken by non-first transfers of a burst. 00: 0 wait states 10: 2 wait states 01: 1 wait state 11: 3 wait states In non-page mode, RPWT is used in conjunction with RWT to specify the number of wait states (the value of BAI) in the range of 0 to 62. (Refer to the description of RWT.)
15:12	RWT	Normal Mode Wait Time	ROM Control Normal Mode Wait Time (initial value: 111(BAI) for Channel 0 and 0000 for Channels 2 to 7) In page mode, specifies the number of wait states to be taken by all single transfers or the first transfer of a burst. 0000: 0 wait states 0100: 4 wait states 1000: 8 wait states 1100: 12 wait states 0001: 1 wait state 0101: 5 wait states 1001: 9 wait states 1101: 13 wait states 0010: 2 wait states 0110: 6 wait states 1010: 10 wait states 1110: 14 wait states 0011: 3 wait states 0111: 7 wait states 1011: 11 wait states 1111: 15 wait states In non-page mode, RWT is used in conjunction with RPWT to specify the number of wait states in the range of 0 to 62. RPWT [1:0] : RWT [3:0] 000000: 0 wait states 010000: 16 wait states 110000: 48 wait states 000001: 1 wait state 010001: 17 wait states 110001: 49 wait states : : : 001110: 14 wait states 011110: 30 wait states 111110: 62 wait states 001111: 15 wait states 011111: 31 wait states 111111: External ACK* mode Note 1: The LSB for Channel 0 is set with the inverted value of boot signal BAI* (ADDR[7] pin). Note 2: When RPM=00 and RDY=0, setting RPWT0:RWT0 to 0x3f causes the ROMC to enter ACK* input mode, instead of the longest wait time in ACK* output mode being selected. Note 3: In READY mode, RWT[0] is used to select ACK*/READY Dynamic or Static mode. In that case, the number of wait states is always even.

Figure 9.3.1 ROM Channel Control Registers (1/2)

Bits	Mnemonic	Field Name	Description
11:8	RCS	Channel Size	ROM Control Channel Size (initial value: 0010 for Channel 0 and 0000 for Channels 2 to 7) Specifies the memory size the channel can access, beginning with the base a assress set in the RBA field. 0000: 1 Mbyte 0101: 32 Mbytes *1010: 1 Gbyte 0001: 2 Mbytes 0110: 64 Mbytes 1011-1111: Reserved 0010: 4 Mbytes 0111: 128 Mbytes 0011: 8 Mbytes 1000: 256 Mbytes 0100: 16 Mbytes 1001: 512 Mbytes * When the memory bus width is 16 bits, the maximum memory size for a channel is 512 Mbytes, not 1G byte.
7	16BUS	Bus Width	ROM Control 16-Bit Width Bus Width (initial value: B16 value for Channel 0 and 0 for Channels 2 to 7) Controls the width of memory accesses for the channel. 0: 32-bit bus 1: 16-bit bus Note: This bit for channel 0 is set with the inverted value of boot signal BOOT16* (ADDR[13] pin).
6	RDY	Ready Input Active	ROM Control Ready Input Active (initial value: 0) Selects the memory access mode: ACK or READY. 0: Disables READY mode. (ACK mode) 1: Enables READY mode. (READY mode) In READY mode, the RPM0 field must be set to 0.
5	RBC	Byte Control	ROM Byte Control (initial value: BBC value for Channel 0 and 0 for Channels 2 to 7) Specifies whether to use the BWE[3:0] signals as byte write enable signals (BWE[3:0]) which become active only for write cycles or as byte enable signals which become active for both read and write cycles. 0: Byte enable (BE[3:0]) 1: Byte write enable (BWE[3:0]) Note: This bit for Channel 0 is set with the value of boot signal BBC (ADDR[6] pin).
4	RHS	Half Speed Bus	ROM Control Half-Speed Bus (initial value: BME0 value for Channel 0 and 0 for Channels 2 to 7) Controls the frequency at which the bus runs with respect to the G-Bus clock. 0: Full speed 1: Half speed Note: This bit for Channel 0 is set with the inverted value of boot signal BME[0] (ADDR[8] pin).
3	RME	Master Enable	ROM Control Master Enable (initial value: BME1 value for Channel 0 and 0 for Channels 2 to 7) Enables or disables the channel. 0: Disable the channel. 1: Enable the channel. Note: This bit for Channel 0 is set with the inverted value of boot signal BME[1] (ADDR[9] pin).
2:0	RSHWT	Setup/Hold Wait Time	ROM Control Setup/Hold Wait Time (initial value: 000) Specifies the chip-enable turn-on delay, which is when the chip enable signal should be asserted relative to the address, and the write-enable/output-enable turn-on delay, which is when the write enable or output enable signal should be asserted relative to the chip enable signal. *000: Disabled 100: 4 wait states 001: 1 wait states 101: 5 wait states 010: 2 wait states 110: 6 wait states 011: 3 wait states 111: 7 wait states * This bit should be set to 0 for burst accesses and page mode.

Figure 9.3.2 ROM Channel Control Registers (2/2)

9.4 Operation

9.4.1 Bootup Options

Channel 0 can be used to control a boot memory device and can be configured through boot options during reset initialization of the TMPR3927. The following provides a description of the boot options. For a complete description of boot-mode settings via external pins (boot pins), refer to "3.4 Initial Setting Signals."

BOOTME

Enables or disables Channel 0 and specifies the bus speed (half/full speed). The values presented on boot signals BME[1:0] (ADDR[9:8] pins) are set in the RME and RHS bits of ROM Channel Control Register 0.

00: Disables Channel 0 as the boot channel. (When BME[1:0]=00)

01: Disables Channel 0 as the boot channel. (When BME[1:0]=01)

10: Enables Channel 0 as the boot channel in half-speed mode
(The SYSCLK frequency is half that of the G-Bus frequency.)
(When BME[1:0]=10)

11: Enables Channel 0 as the boot channel in full-speed mode
(The SYSCLK frequency is equal to the G-Bus frequency.)
(When BME[1:0]=11)

BOOT16

Control the width of memory accesses for Channel 0. The inverted value of boot signal BOOT16* (ADDR[13] pin) is set in the 16Bus bit of ROM Channel Control Register 0.

0: 32-bit wide upon bootup. (When BOOT16*=1)

1: 16-bit wide upon bootup. (When BOOT16*=0)

BOOTAI

Specifies whether the ACK* signal for Channel 0 is internally or externally generated. The inverted value of boot signal BAI* (ADDR[7] pin) is set in bit 12 of ROM Channel Control Register 0.

0: Internal ACK mode upon bootup. (When BAI*=1)

1: External ACK mode upon bootup. (When BAI*=0)

BOOTBC

Controls whether the BWE[3:0] signals are used as byte enable signals (BE[3:0]) or byte write enable signals (BWE[3:0]) when accessing a memory device on Channel 0. The value of boot signal BBC (ADDR[6] pin) is set in the RBC bit of ROM Channel Control Register 0.

0: Used as byte enable signals. (When BBC=0)

1: Used as byte write enable signals. (When BBC=1)

9.4.2 Global Options

In addition to those described in "9.4.1 Bootup Options," the ROMC provides the two options.

CHANHS

Specifies the SYSCLK output frequency. The inverted value of boot signal CHANHS* (ADDR[15] pin) is set in the RHS bit of the Chip Configuration Register (CCFG).

0: Full speed (equal to the G-Bus frequency).

1: Half speed (half the G-Bus frequency). The half-speed option must be selected if at least one channel is run at half speed.

ACEHOLD

Specifies the address hold time relative to the ACE* signal. This option is set by the ACEHOLD bit of the Chip Configuration Register (CCFG).

0: Addresses are made available, coincident with the ACE* signal.

1: Addresses are made available one clock cycle after deassertion of the ACE* signal.

9.4.3 ROM Channel Control Registers

The ROM Channel Control Registers must be accessed using 32-bit reads and writes. There is no priority among the channels. Care must be taken to ensure that more than one channel is not programmed for to the same address region.

9.4.4 Clock Options

Each channel can be independently programmed to operate with a full-speed or half-speed clock. When Channel 0 is used to control boot memory, SYSCLK is always generated at the frequency specified for Channel 0, regardless of the value of CHANHS at boot time.

If both full- and half-speed clocks are necessary, set SYSCLK to half speed and use one of the SDCLK outputs as a full-speed clock.

The following table shows how boot signals BME[1:0] and CHANHS affect the SYSCLK frequency.

BME[1]	BME[0]	CHANHS	SYSCLK	ROMC Channel 0	
1	0	0	1/2	1/2	Use ROMC Channel 0 for boot memory.
1	0	1	1/2	1/2	
1	1	0	1/2	Full	
1	1	1	Full	Full	
0	0	0	1/2	—	Use SDRAMC Channel 0 for boot memory.
0	0	1	Full	—	
0	1	0	1/2	—	
0	1	1	Full	—	

9.4.5 Base Address and Channel Size

The base address of a channel must be aligned on a boundary that is an integer multiple of the selected size. For example, a 64-Mbyte channel size must begin on a 64-Mbyte boundary. When the size of a channel is programmed to 64 Mbytes, the lower six bits of the base address (RBA) field, or bits 25 to 20 of its ROM Channel Control Register, are ignored. (The minimum resolution of channel size is 1 Mbyte.) If the base address (RBA) filed is inadvertently programmed as 0x150 with a size of 64 Mbytes, then the actual base address becomes 0x140, as shown below. This results in the assignment of addresses 0x9400_0000 to 0x97ff_ffff (or 0xb400_0000 to 0xb7ff_ffff) to this channel.

RBA Value 000101010000

Ignored when the channel size is programmed to 64 Mbytes

9.4.6 Operating Modes

The ROMC provides two major operating modes, ACK*/READY static and ACK*/READY Dynamic. The function of the ACK*/READY pin depends on the mode of operation selected for the ROMC channels. In Static mode, the ACK*/READY pin is always an input. In Dynamic mode, the ACK*READY pin is dynamically configured as input or output, depending on which channel is accessed and how that channel is programmed. There are four submodes, Normal, Page, External ACK*, and READY, each separately selectable on a per channel basis.

9.4.6.1 ACK*/READY Dynamic Mode

The ROMC is configured for this mode when no channel is programmed with RDY=1 and RWT[0]=1.

In this mode, the ACK*/READY pin is automatically used either as input or output on a per channel basis. When a channel is programmed for Normal or Page submode, the ACK*/READY pin acts as an output (and uses the internally generated ACK*) whenever that channel is accessed. When a channel is programmed for External ACK* or READY submode, the ACK*/READY pin acts as an input whenever that channel is accessed. Refer to the timing diagrams to prevent signal conflicts when the pin changes its direction.

9.4.6.2 ACK*/READY Static Mode

The ROMC is configured for this mode when any one of the channels is programmed with RDY=1 and RWT[0]=1.

In this mode, the ACK*/READY pin is always an input. Thus, even when a channel is programmed for Normal or Page submode, the internal ACK* signal is not made visible externally. For this reason, in cases where any channel usage causes the ROMC to be configured for ACK*/READY Static mode, the ACK*/READY pin must be connected to a device with an open-drain node.

9.4.6.3 Normal Submode

A channel is configured for this mode when the associated ROM Channel Control Register is programmed as follow:

RPM = 00
RDY = 0
RPWT:RWT != 0x3f

In this mode, the ACK*/READY pin is used as the ACK* output. The RPWT and RWT fields together specify the number of wait states to be taken by each transfer. Since RPWT:RWT=0x3f results in External ACK* mode, the number of wait states on Normal-mode transfers can be between 0 and 62, inclusive.

9.4.6.4 External ACK* Submode

A channel is configured for this mode when the associated ROM Channel Control Register is programmed as follows:

RPM = 00
RDY = 0
RPWT:RWT = 0x3f

In this mode, the ACK*/READY pin is used as an ACK* input, and the external device must issue the ACK* signal to terminate a memory cycle. The ROMC internally synchronize the ACK* signal. For details, refer to "9.4.9.2 ACK* Input Timing."

9.4.6.5 Page Submode

A channel is configured for this mode when the associated ROM Channel Control Register is programmed as follws:

RPM != 00
RDY = 0

In this mode, the ACK*/READY pin is used as an ACK* output. The RPWT and RWT fields specify the number of wait states to be taken. In particular, this mode conforms to the requirements for page-mode ROMs. The 4-bit RWT field specifies the number of wait states (0 to 15) to be taken by all single-transfers and the first transfer of a burst. The 2-bit RPWT field specifies the number of wait states (0 to 3) to be taken by accesses beyond the first during a burst transfer.

The RPM field controls the page-mode burst size. If it is less than the CPU burst size, multiple page-mode burst cycles will be used. In this case, the wait states specified in the 4-bit RWT field are inserted for each burst cycle. In Page mode, the number of wait states specified by the RWT field should be greater than or equal to that specified by the RPWT field.

9.4.6.6 READY Submode

A channel is configured for this mode when the associated ROM Channel Contorl Reigster is programmed as follows:

RPM = 00
RDY = 1

In this mode, the ACK*/READY pin is used as a READY input, and the external device must issue the READY signal to terminate a memory cycle. The ROMC internally synchronize the READY signal. For details, refer to "9.4.10 READY Input Timing."

The TX3927 examines the READY input after the wait period programmed in RPWT:RWT expires. RWT[0] is used to select either ACK*/READY Static or Dynamic mode; hence it does not participate in determining the number of wait states. Therefore, possible wait state counts for READY mode are 0, 2, 4, 6, ... 62.

Burst accesses are dis allowed in READY mode.

9.4.7 16-Bit Data Bus Operation

A channel configured for a 16-bit data bus can access a byte or halfword data item in a single-transfer cycle. The transfer of a word data item requires two bus cycles. A burst request to a 16-bit channel always causes two 16-bit accesses to be performed, regardless of the requested data size (byte, halfword or any combinations of non-word data size)

The maximum memory size that can be assigned to a channel in 16-bit bus mode is 512MB, not 1GB.

9.4.8 SHWT Option

The SHWT option is selected when the RSHWT field is set to a non-zero value. This option uniformly extends the setup and hold times listed below to support slow I/O devices.

Setup: ADDR to CE, CE to OE, and CE to BWE/BE.

Hold: CE to ADDR, OE to CE, and BWE/BE to CE.

When this option is enabled for a channel, that channel will have an equal, requirement for all of the above setup/hold times, each setup or hold property cannot be programmed separately.

The SHWT option cannot be used in Page mode. All the other modes support the SHWT option, but it makes burst accesses unusable.

The TX3927 executes burst accesses for the following cases. The RSHWT field should be set to "0" if any burst accesses can occur for a relevant memory channel.

(1) TX3927 instruction fetches

(2) Cache write-back when the data cache employs a write-back mode

(3) DMA access with a transfer size (XFSZ) of 4 or more words, unless the DBINH or SBINH field of the Channel Control Register (CCR) is programmed prohibit burst accesses for the relevant memory

(4) The following accesses from an external PCI master when the TBL_OFIFO or TBL_IFIFO field of the PCI Target Burst Length Register (TBL) is programmed with a size larger than one doubleword.
a) Read (including a single read)
b) Burst write exceeding the specified size

9.4.9 ACK*/READY Signal Timing

9.4.9.1 ACK* Output Timing

The external device can use the ACK* signal as a timing reference for reads and writes when it is an output.

During a read cycle, data is always latched on the rising edge of the clock after the assertion of ACK*.

During a write cycle, data will remain valid for two clock cycles after ACK* is recognized as asserted on the rising edge of the clock (i.e., one clock cycle after SWE* or BWE* is deasserted.

9.4.9.2 ACK* Input Timing

The ROMC internally synchronizes the ACK* signal when it is an input. Due to internal state machine restrictions, ACK* cannot be recognized in back-to-back cycles. Once ACK* is recognized, the earliest it can be recognized again is two clock cycles later for a read and four clock cycles later for a write.

The timing at which the ROMC starts sampling ACK* differs between read and write cycles, as shown below. ACK* is continuously sampled on the rising edge of the clock thereafter.

- Read cycle

  The first rising edge of the clock after OE* is asserted low

- Write cycle

  The rising edge of the clock where SWE* is asserted low

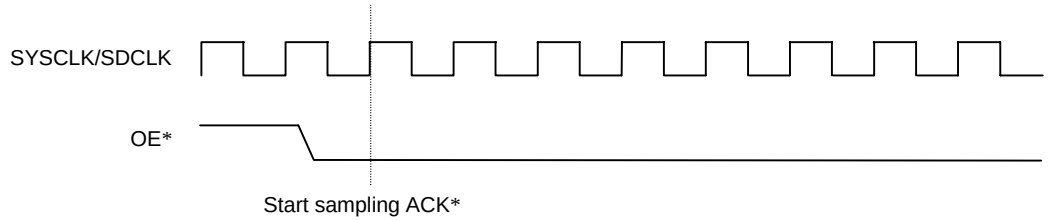


Figure 9.4.1 Timing for Starting Sampling ACK* During a Read Cycle

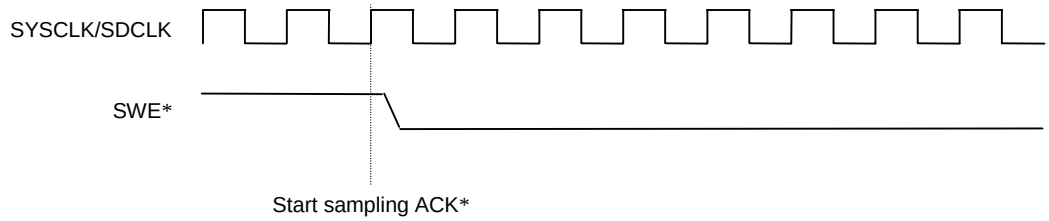


Figure 9.4.2 Timing for Starting Sampling ACK* During a Write Cycle

During a read cycle, data will be latched two clock cycles after the ACK* signal is recognized.

During a write cycle, data will remain valid for four clock cycles after ACK* is recognized as asserted. (i.e., one clock cycle after SWE* or BWE* is deasserted).

9.4.10 READY Input Timing

When the ACK*/READY pin is configured as a READY input, the description of the ACK* input timing in "9.4.9.2 ACK* Input Timing" also applies to the READY input, with two exceptions. First, READY is active high while ACK* is active low. Secondly, in READY mode, wait states can be inserted as specified by RPWT:RWT, so that the READY input is examined after the wait period expires. For details, refer to "9.4.6.6 READY Submode."

The timing at which the ROMC starts sampling READY differs between read and write cycles, as shown below. READY is continuously sampled on the rising edge of the clock thereafter.

- Read cycle
  (1) When the number of wait states is 0 (RPWT:RWT = 0)
  The first rising edge of the clock after OE* is asserted low
  (2) When the number of wait states is n, which is a non-zero value (RPWT:RWT != 0)
  The nth rising edge of the clock after OE* is asserted low

- Write cycle
  (1) When the number of wait states is 0 (RPWT:RWT = 0)
  The rising edge of the clock where SWE* is asserted low
  (2) When the number of wait states is n, which is a non-zero value (RPWT:RWT != 0)
  The (n-1)th rising edge of the clock after SWE* is asserted low

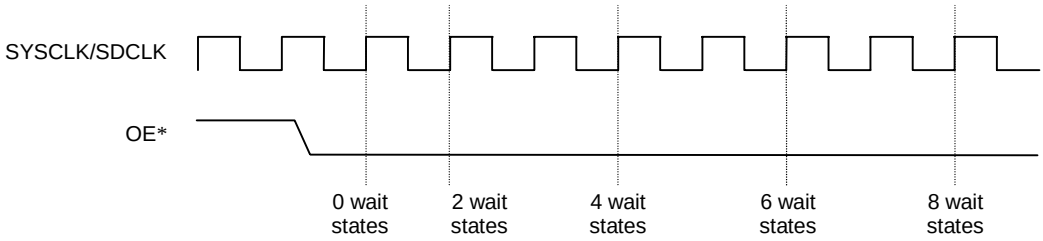


Figure 9.4.3 Timing for Starting the Sampling READY During a Read Cycle

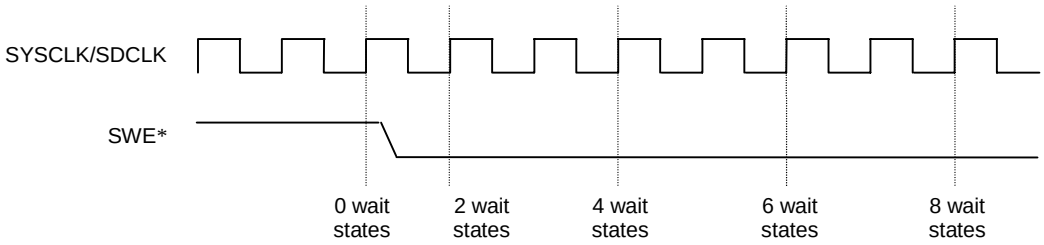


Figure 9.4.4 Timing for Starting Sampling of READY During a Write Cycle

9.4.10.1 ACK*/READY Turnaround Timing

In ACK*/READY Static mode, the ACK*/READY pin is always an input. In ACK*/READY Dynamic mode, the ACK*/READY pin is usually an output except when a channel is configured for a submode that requires the pin to be an input. The ACK* pin goes into the high-impedance state in the same clock cycle when the CE* signal is asserted. The ACK* signal is actively driven again one clock cycle after CE* is deasserted.

9.4.11 Addressing

Using a 32-bit address internally, the TX3927 address space encompasses 4 Gbytes. Externally, the TX3927 provides the multiplexed address output pins named ADDR[19:2].

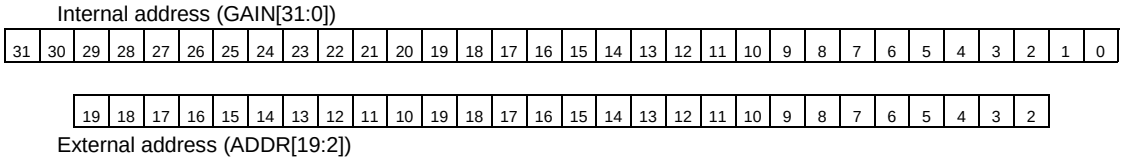
In 32-bit bus mode, internal address bits [19:2] are directly routed to the ADDR[19:2] pins. The upper address bits [29:20] are multiplexed over the ADDR[19:10] pins. In 32-bit bus mode the maximum memory size is 1GB.

In 16-bit bus mode, internal address bit [1] is generated, based on the internal byte enable and endianness, and routed to the ADDR[2] pin. Internal address bits [18:2] are directly routed to the ADDR[19:3] pins. The upper internal address bits [28:19] are multiplexed over the ADDR[19:10] pins. In 16-bit bus mode, the maximum memory size is 512MB.

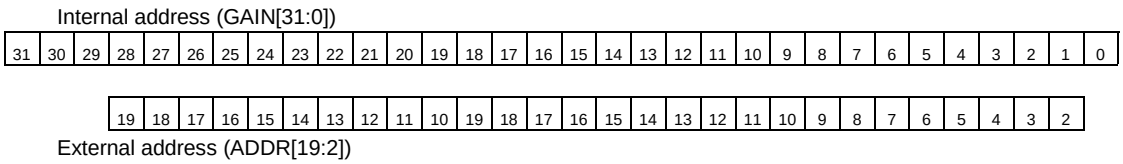
Refer to "9.4.12 ACE* Operation" for mode on address multiplexing.

The following figures show the correspondence between the internal address bits (GAIN[31:0]) and the externalized address pins (ADDR[19:2]):

1) 32-bit bus mode



2) 16-bit bus mode



9.4.12 ACE* Operation

The ACE* signal makes it possible to demultiplex the address on ADDR[19:2]. To generated a full address, an external latch must be used to latch the first (upper) ten bits of the address signals using the active-low ACE* (Address Clock Enable) signal together with the rising edge of the system clock (SYSCLK). Alternatively, ACE* can be used directly as a latch enable signal.

After reset, the ACE* signal always goes active during the first clock cycle. In subsequent cycles, the first ten bits of the new address are compared to those of the preceding address. If they do not match, the upper address is output and ACE* is activated. In 32-bit bus mode, ACE* will never be active if the RCS field is programmed for a channel size of 1 MB since the upper ten bits of the address do not participate in address decoding. If the RCS field is programmed for 1-MB channel size in 16-bit bus mode, ACE* will go active only when ADDR[20] is not used for address decoding for a chip select.

By default, the address is held stable one clock after the rising edge of ACE*. This hold time can be eliminated by clearing the ACEHOLD bit of the Chip Configuration Register (CCFG). The hold-time bit applies to all the channels.

9.5 Timing Diagrams

Note the following, when referring to the timing diagrams in this section:

1. SYSCLK/SDCLK in the timing diagrams indicates the state of SYSCLK or SDCLK[4:0]. SDCLK[4:0] always operate at full speed, while SYSCLK can be used in either full-speed or half-speed mode. All relevant signals are synchronous to the half or full speed clock, depending on the channel configuration.

2. All timing diagrams show both the BWE* and BE* signals. Either one of those is mode available at any given time from the BWE* pin, depending on the setting of the ROM Channel Control Register.

3. All burst cycles shown start on page boundaries. This is true except when CWF (Critical Word First) is enabled. The address order may be different in 4-word critical-word-first bursts.

4. Wait states are indicated as SWx in the diagrams. Setup and hold states programmed by SHWT are indicated as ASx (setup from address valid to CE falling), CSx (setup from CE falling to OE/SWE falling), AHx (hold from CE rising to address change), and CHx (hold from OE/SWE rising to CE rising). Synchronization states for external input signals are indicated as ESx. Address clock enable states are indicated as ACEx. All other states are indicated as Sx. In cases where two states overlap the non-Sx state is indicated.

5. The ACK* pin can be an input or output in Dynamic mode. When switching from an output to input, it enters the high-impedance state coincident with the assertion of CE*. The middle-level lines in the diagrams represent the high-impedance state. When switching ACK* from an input to output, the TX3927 starts driving ACK* one clock after CE* is deasserted. It should be noted that in the case of a 32-bit access to a 16-bit memory in External ACK* (READY) mode, CE* is deasserted between two half-word accesses.

6. External ACK* (READY) signal is shown as active for one clock cycle. External ACK* (READY) may, however, remain active for more than one clock cycle as long as the following requirements are satisfied:

   - In the case of External ACK* and READY single-transfer accesses, the ACK* pin may remain active until the end of the cycle where CE* is deasserted.

   - Depending on the mode of the ACK* pin (Dynamic or Static), the external device must either drive it high or tri-state it to meet the requirements described in Note 5 above.

   - In the case of External ACK* burst cycles, the external device may keep the signal active for up to 3 clock cycles for a read and up to 5 clock cycles for a write. If the signal remains active longer, the TX3927 recognizes it as a next valid ACK*.

     Note: The ACK* (READY) pin has an internal pull-up resistor. However, because the resistance is large, an external resistor can also be attached.

9.5.1 ACE* Signal Operation

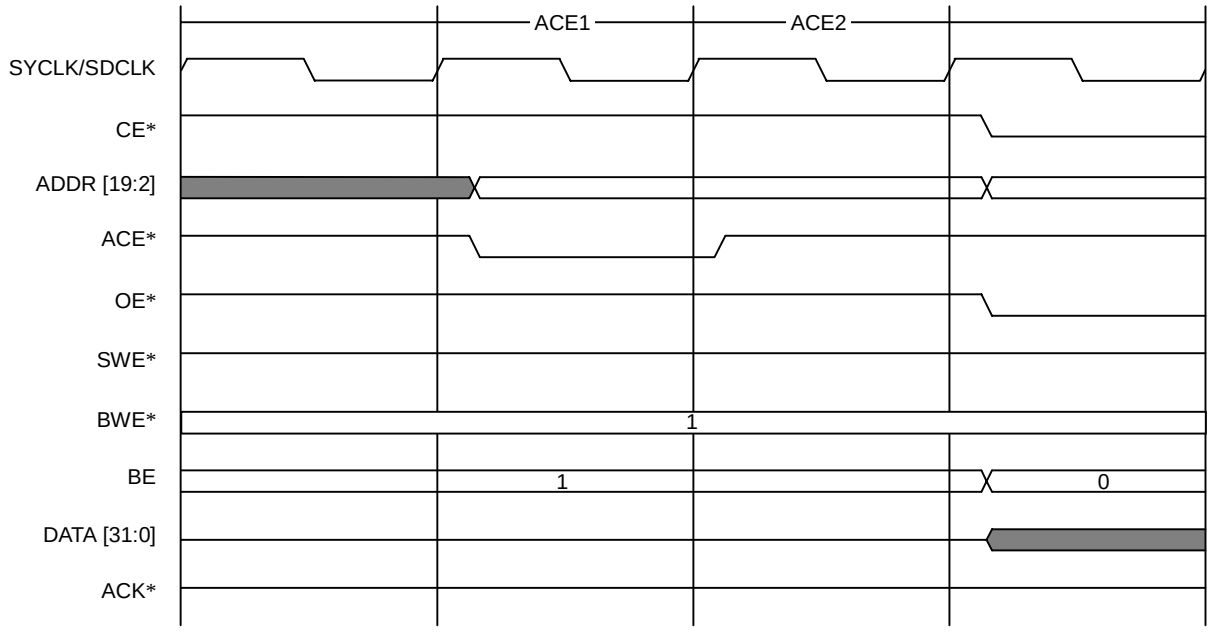


Figure 9.5.1 ACE* with Hold Time

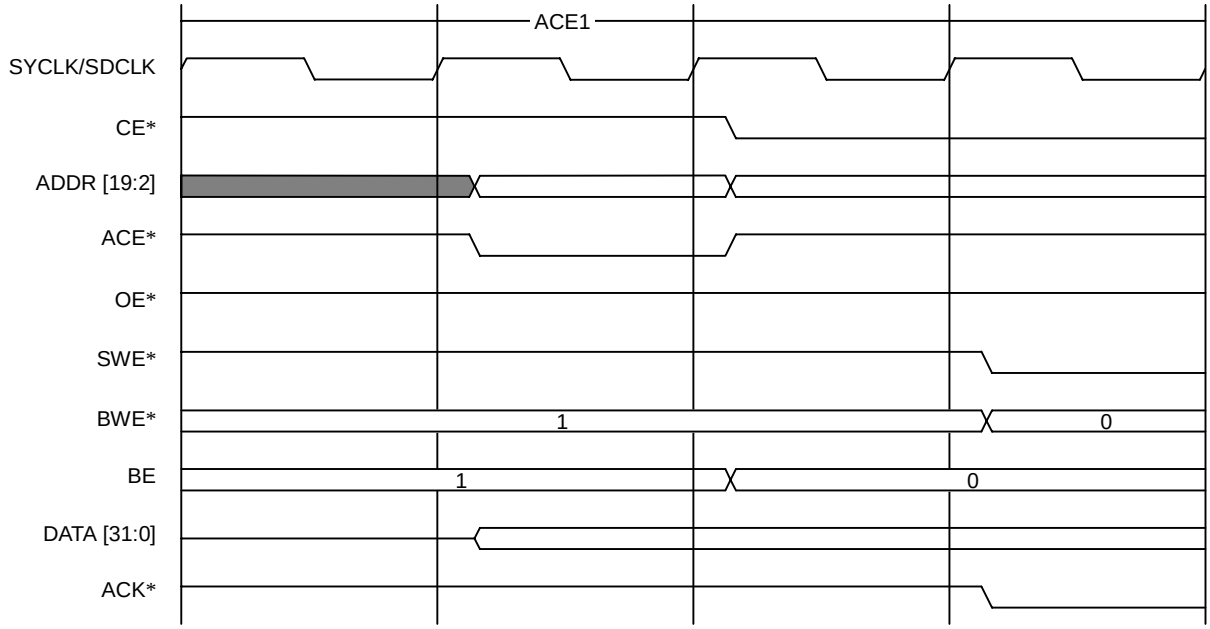


Figure 9.5.2 ACE* without Hold Time

9.5.2 Normal Mode 32-bit Write Operation

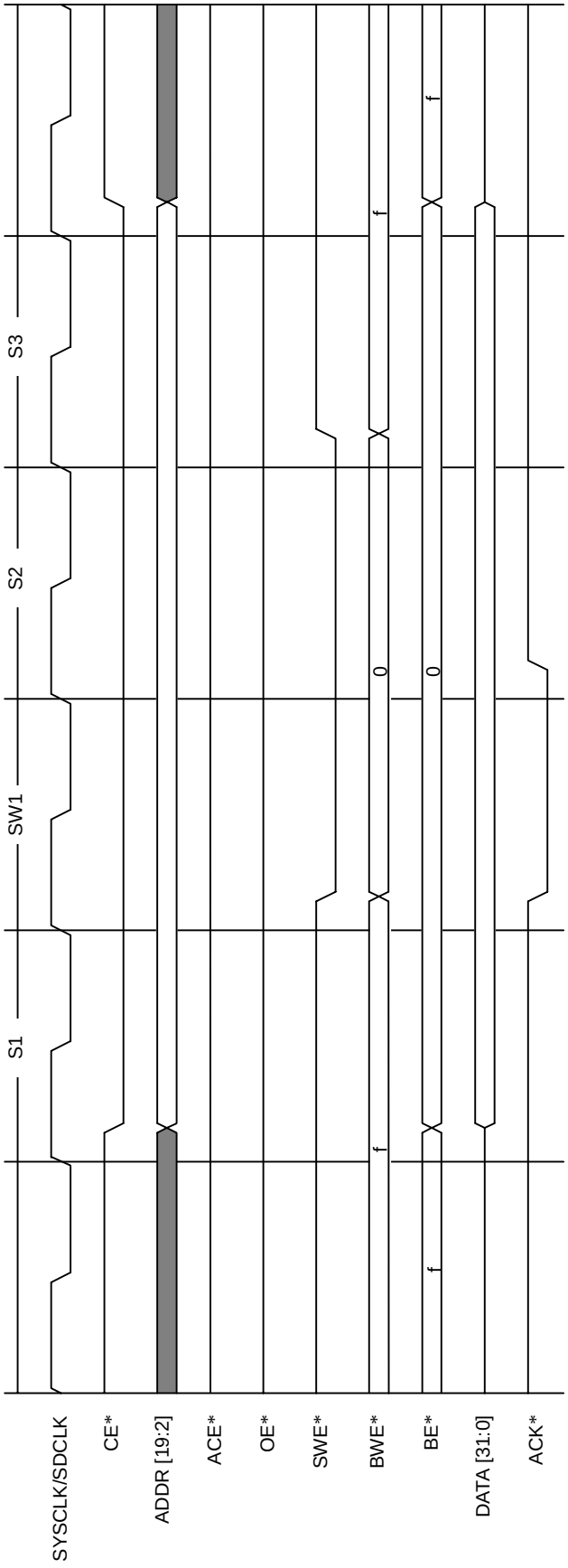


Figure 9.5.3 Normal Mode 32-bit Bus Operation (32-bit Single Write, 1 Wait State)

9.5.3 Normal Mode 32-bit Operation

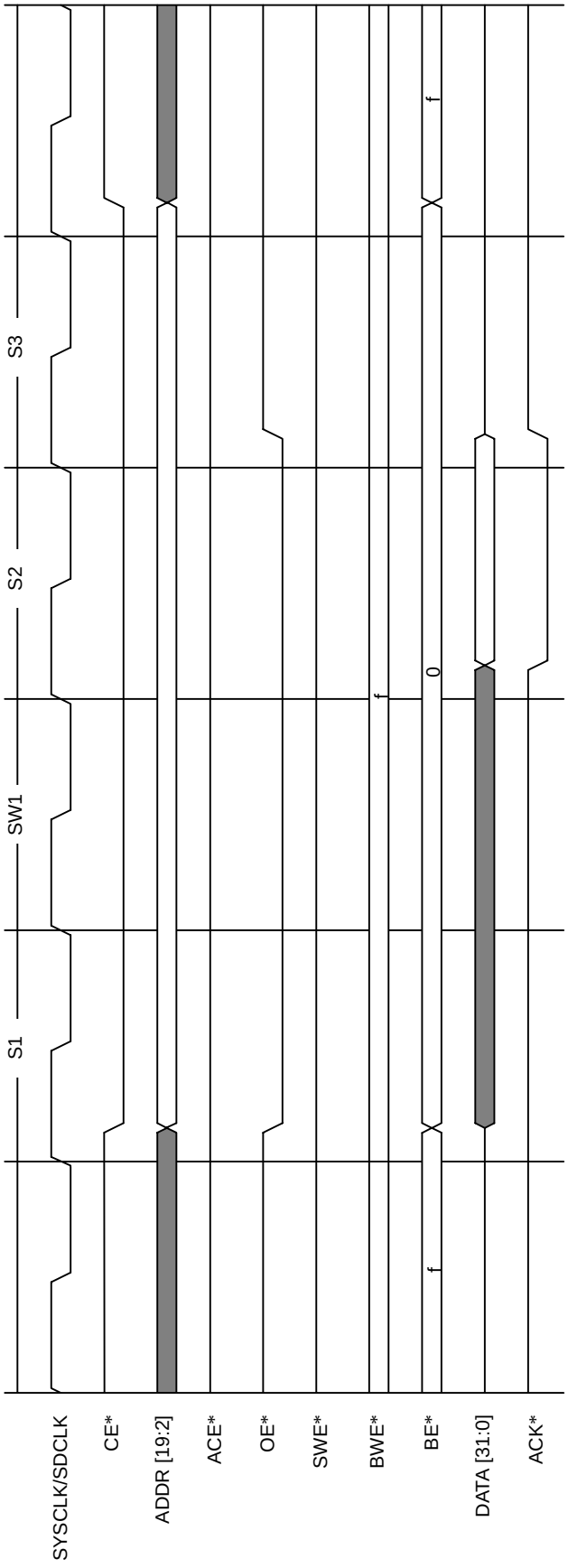


Figure 9.5.4 Normal Mode 32-bit Bus Operation (32-bit Single Read, 1 Wait State)

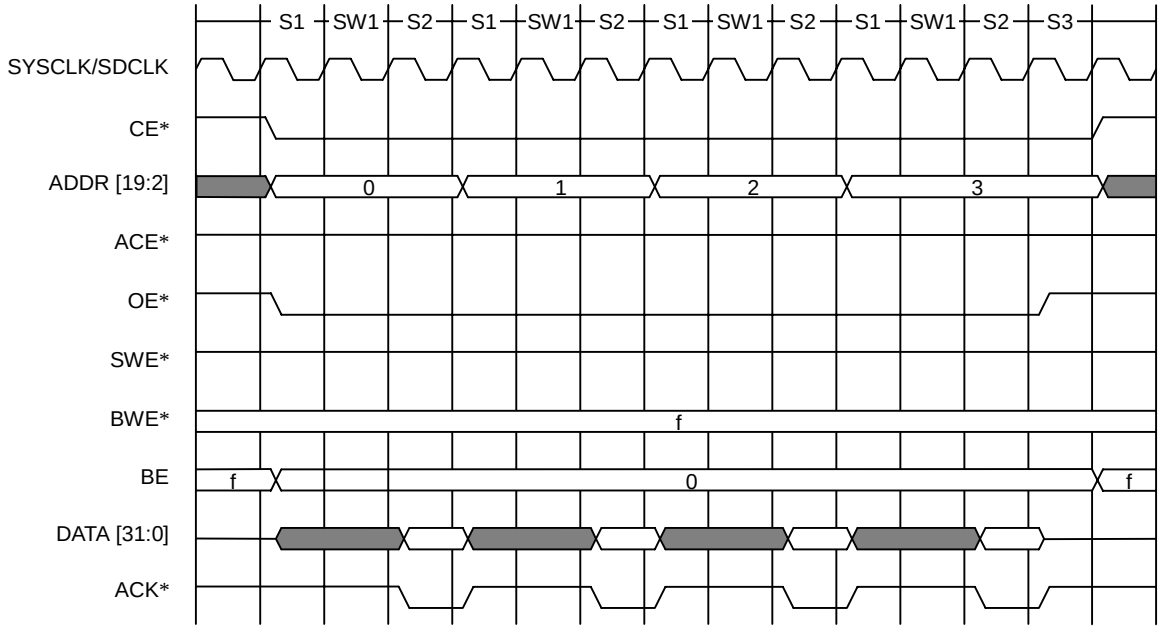


Figure 9.5.5 Normal Mode 32-bit Bus Operation (4-word Burst Read, 1 Wait State)

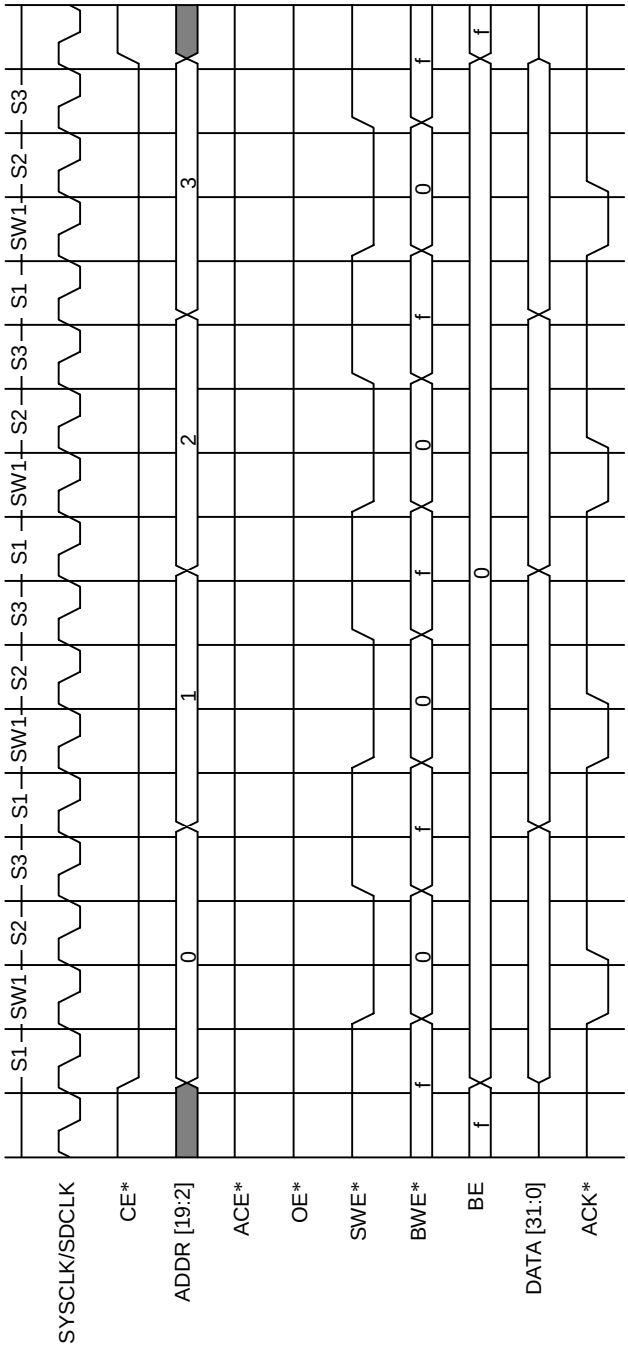


Figure 9.5.6 Normal Mode 32-bit Bus Operation (4-word Burst Write, 1 Wait State)

9.5.4 Normal Mode 16-bit Bus Operation

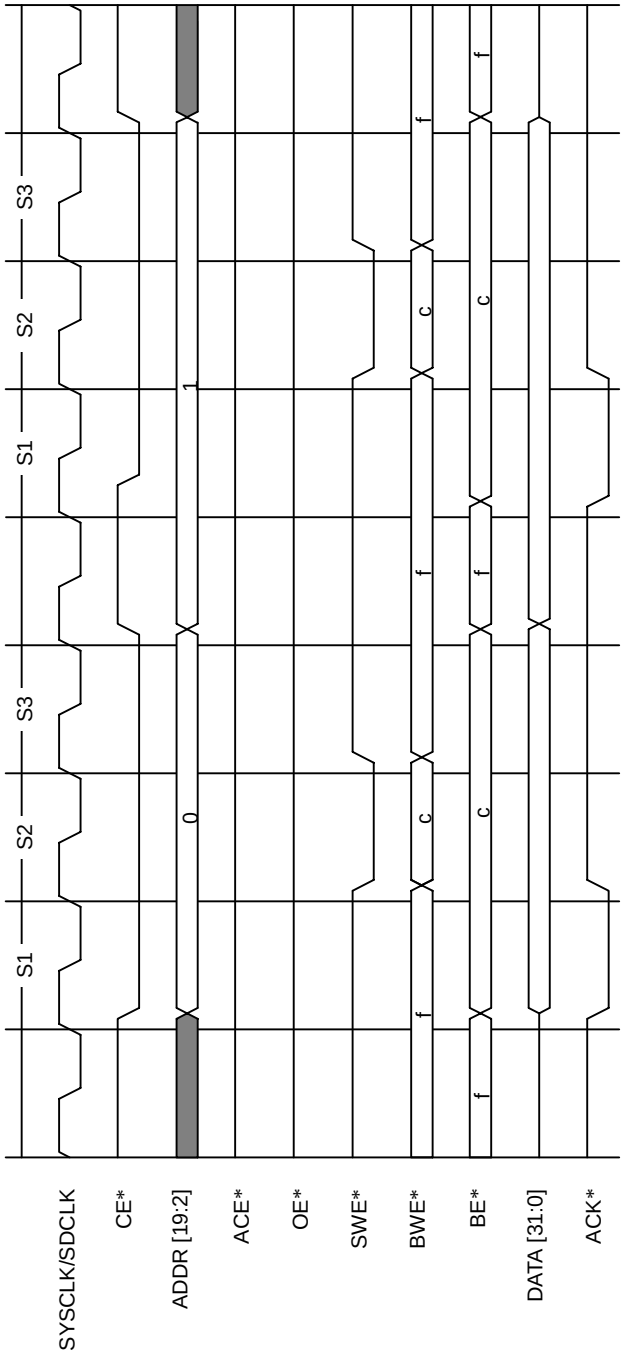


Figure 9.5.7 Normal Mode 16-bit Bus Operation (32-bit Single Write, No Wait State)

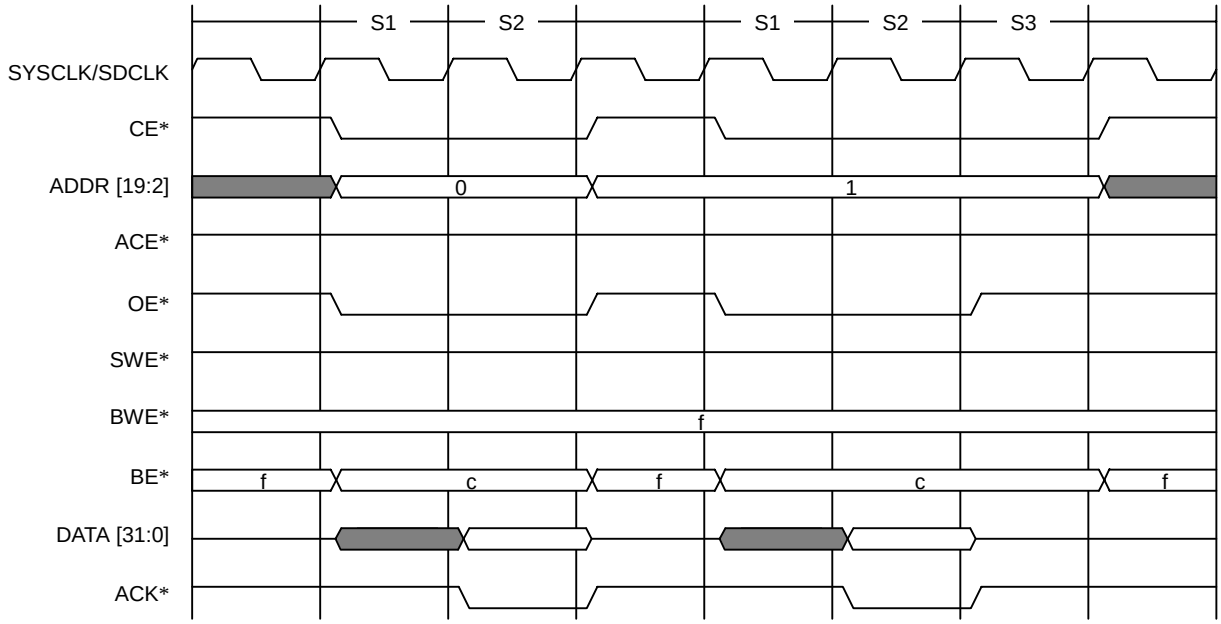


Figure 9.5.8 Normal Mode 16-bit Bus Operation (32-bit Single Read, No Wait State)

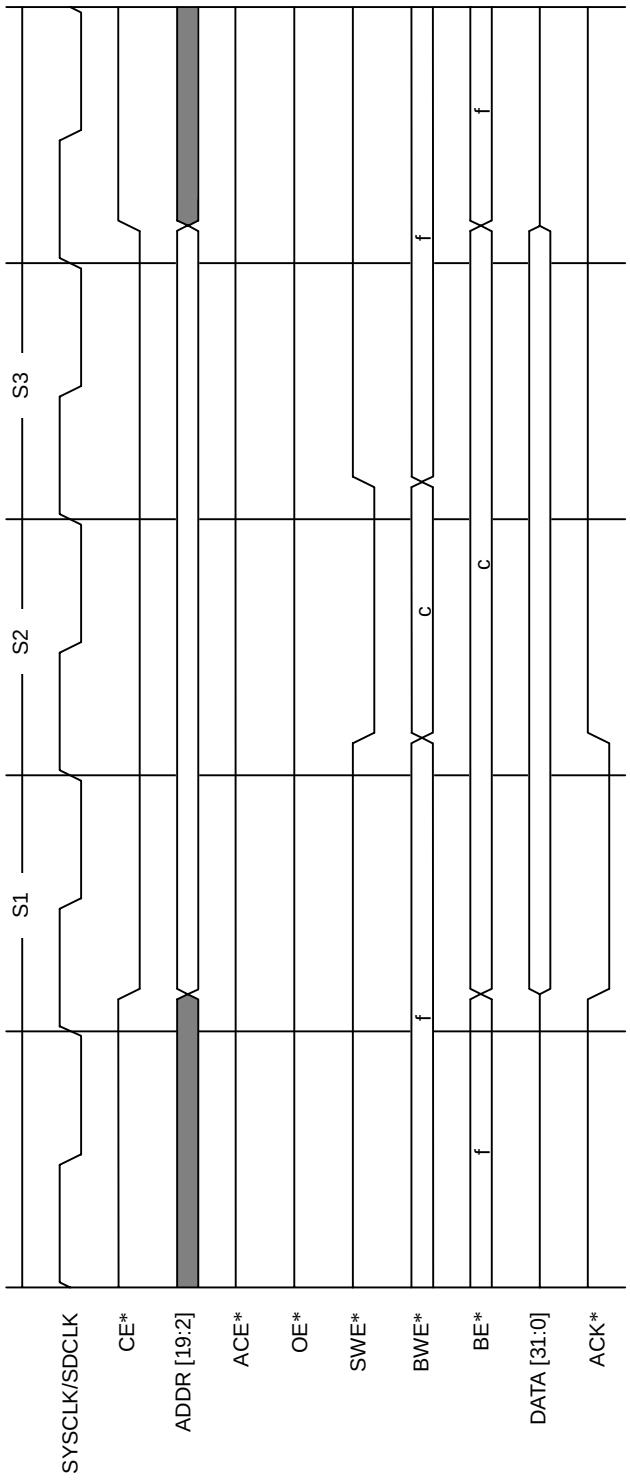


Figure 9.5.9 Normal Mode 16-bit Bus Operation (16-bit Single Write, No Wait State)

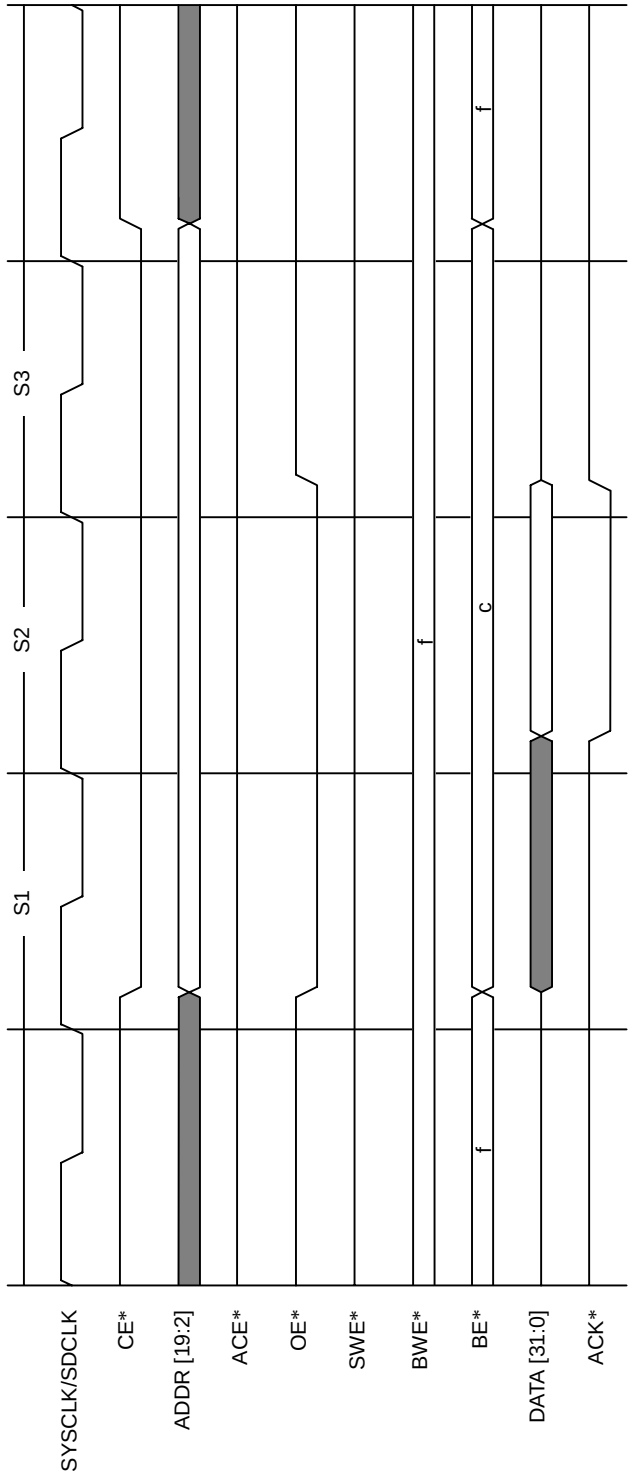


Figure 9.5.10 Normal Mode 16-bit Bus Operation (16-bit Single Read, No Wait State)

9.5.5 Normal Mode 16-bit Burst Operation

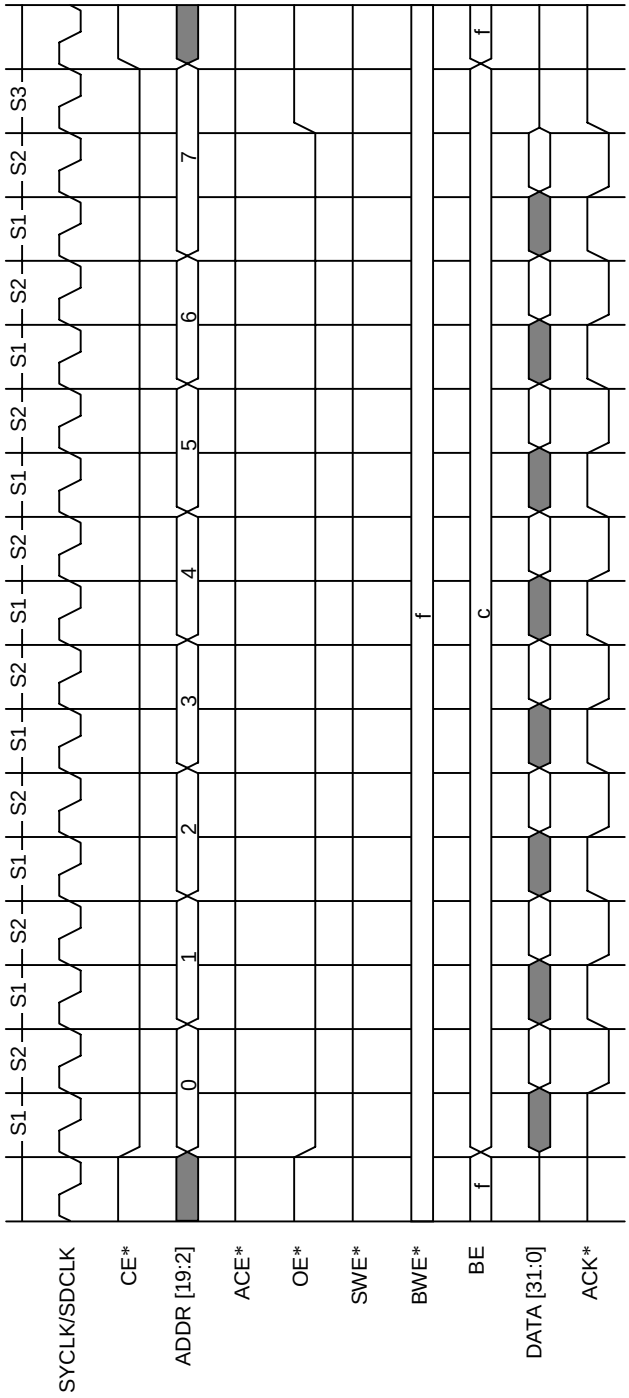


Figure 9.5.11 Normal Mode 16-bit Bus Operation (4-word Burst Read, No Wait State)

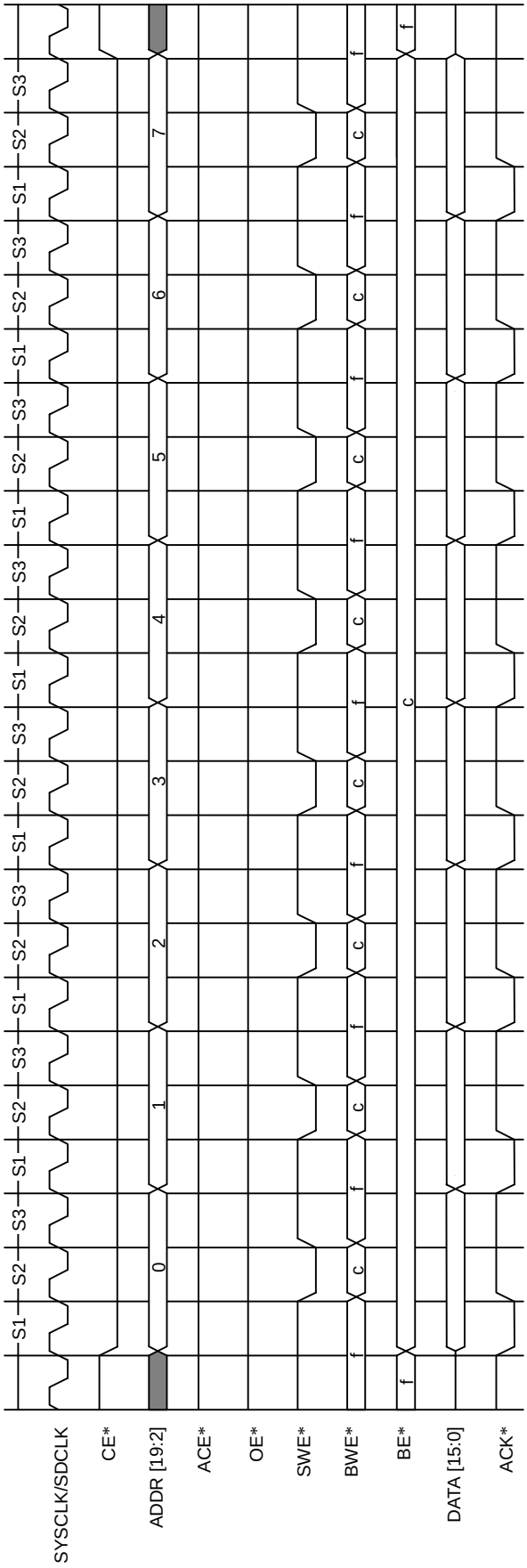


Figure 9.5.12 Normal Mode 16-bit Bus Operation (4-word Burst Write, No Wait Stets)

9.5.6 Page Mode 32-bit Burst Operation

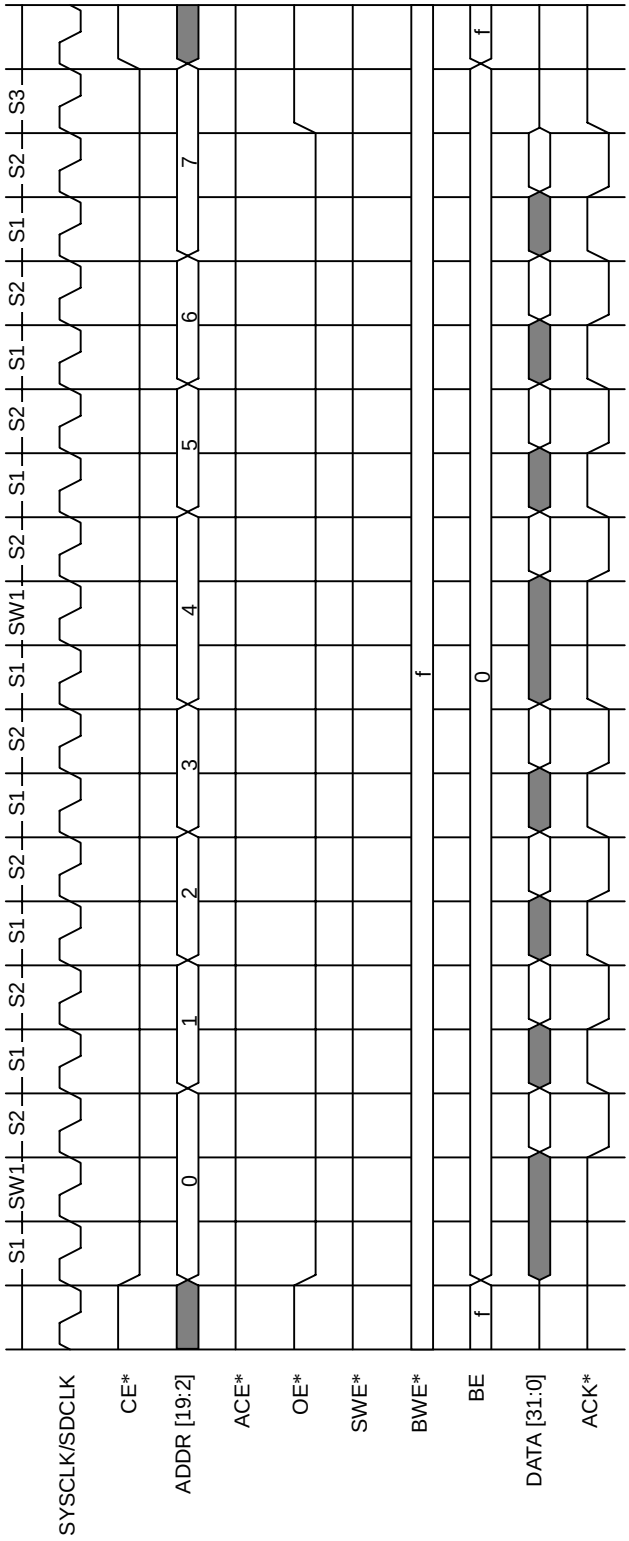


Figure 9.5.13 Page Mode 32-bit Bus Operation (8-word Burst Read, 1 Wait State, No Page Wait State)

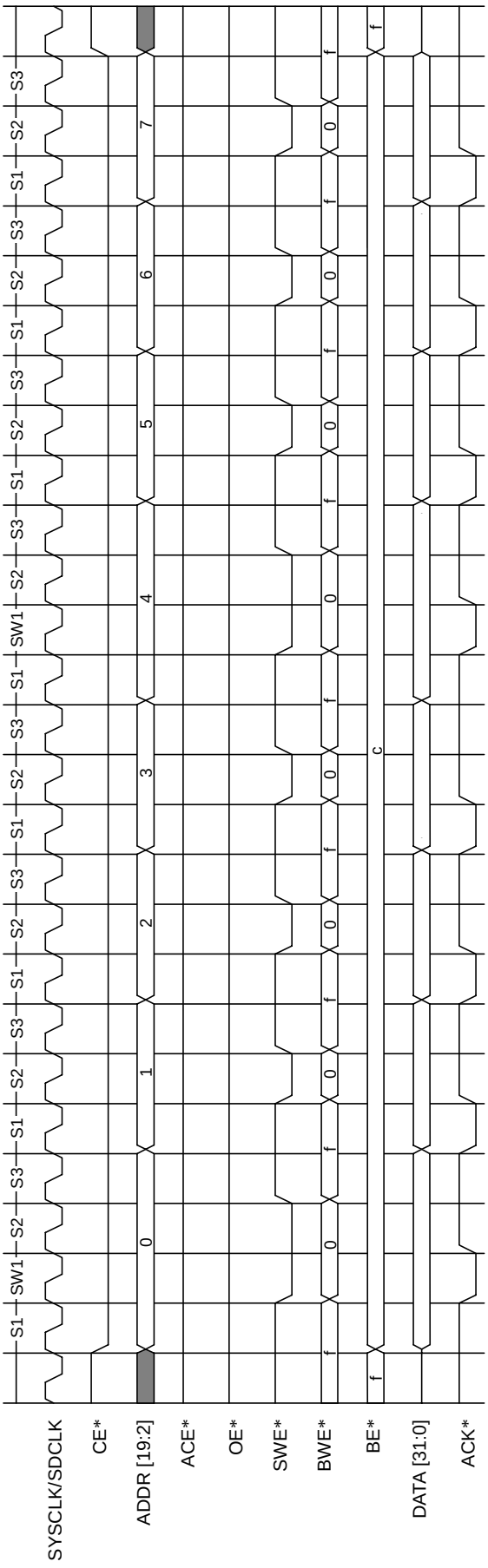
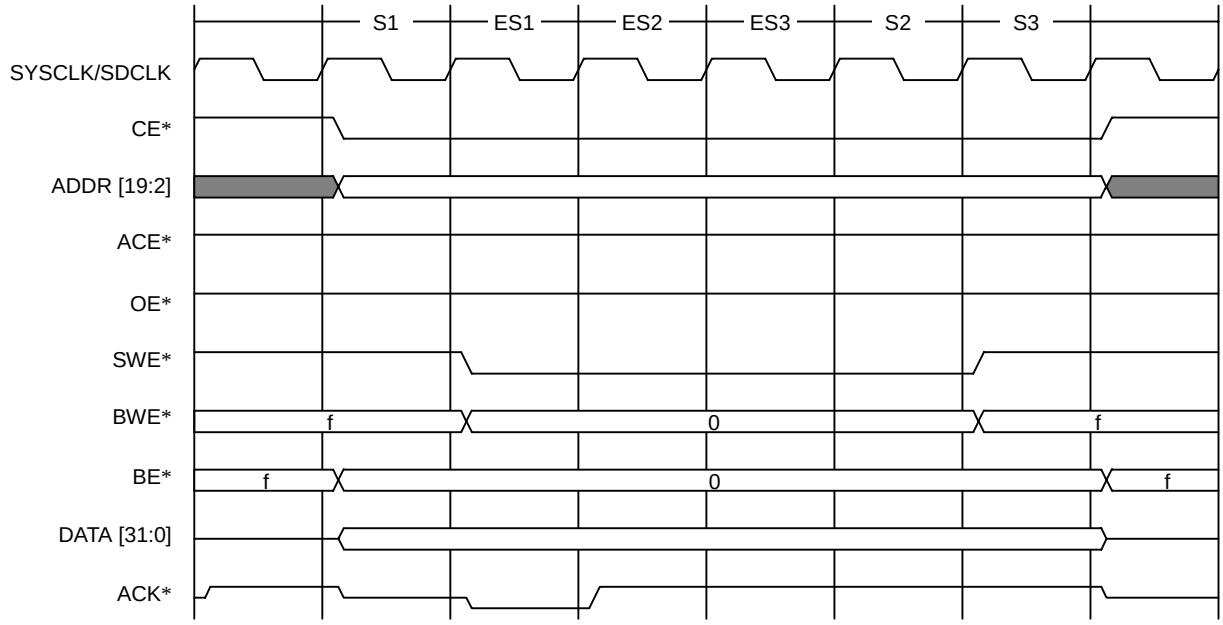


Figure 9.5.14 Page Mode 32-bit Bus Operation (8-word Burst Write, 1 Wait State, No Page Wait State)

9.5.7 External ACK* Mode 32-bit Operation



Note 1: The TX3927 places the ACK* pin in the high-impedance state in S1.

Note 2: The external device must assert ACK* (low) before the end of ES1. In the above diagram ACK* is asserted in ES1 to clarify when the drive is switched. If ACK* is asserted before the end of S1, S1 is immediately followed by ES2, with no ES1 state (no wait states inserted).

Note 3: The external device must deasserts ACK* (high) in ES2. If assertion of ACK* is delayed, extra wait states are inserted. The ACK* signal is allowed to remain low for more than one clock in certain situations. For details, refer to "9.5 Timing Diagrams" and "9.4.9.2 ACK* Input Timing."

Figure 9.5.15 External ACK* Mode 32-bit Bus Operation (32-bit Single Write, 1 Wait State)

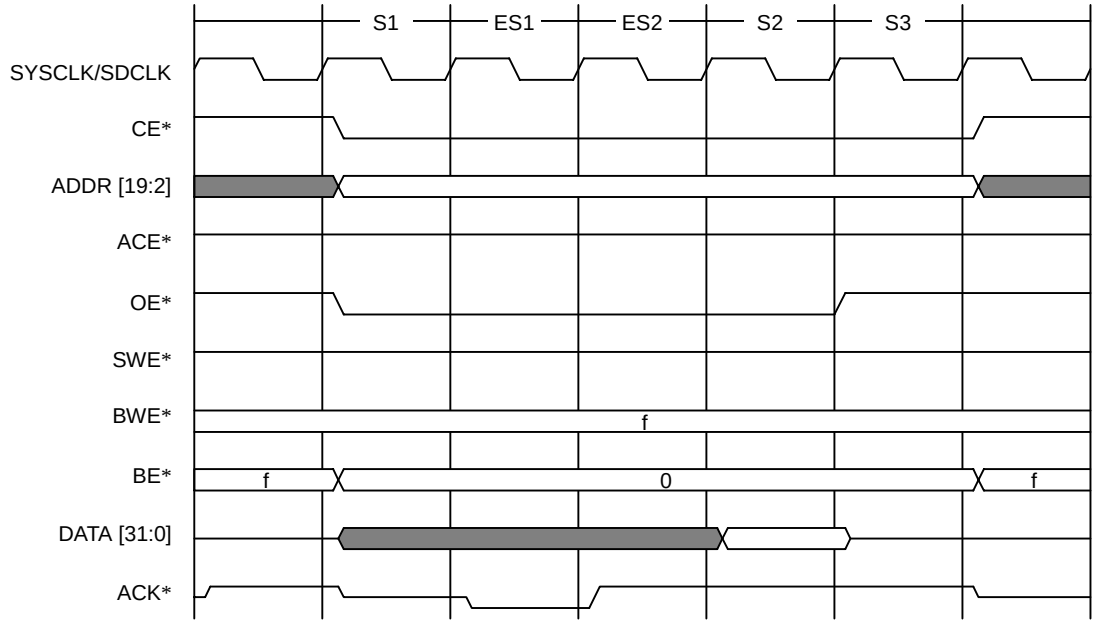


Figure 9.5.16 External ACK* Mode 32-bit Bus Operation (32-bit Single Read, 1 Wait State)

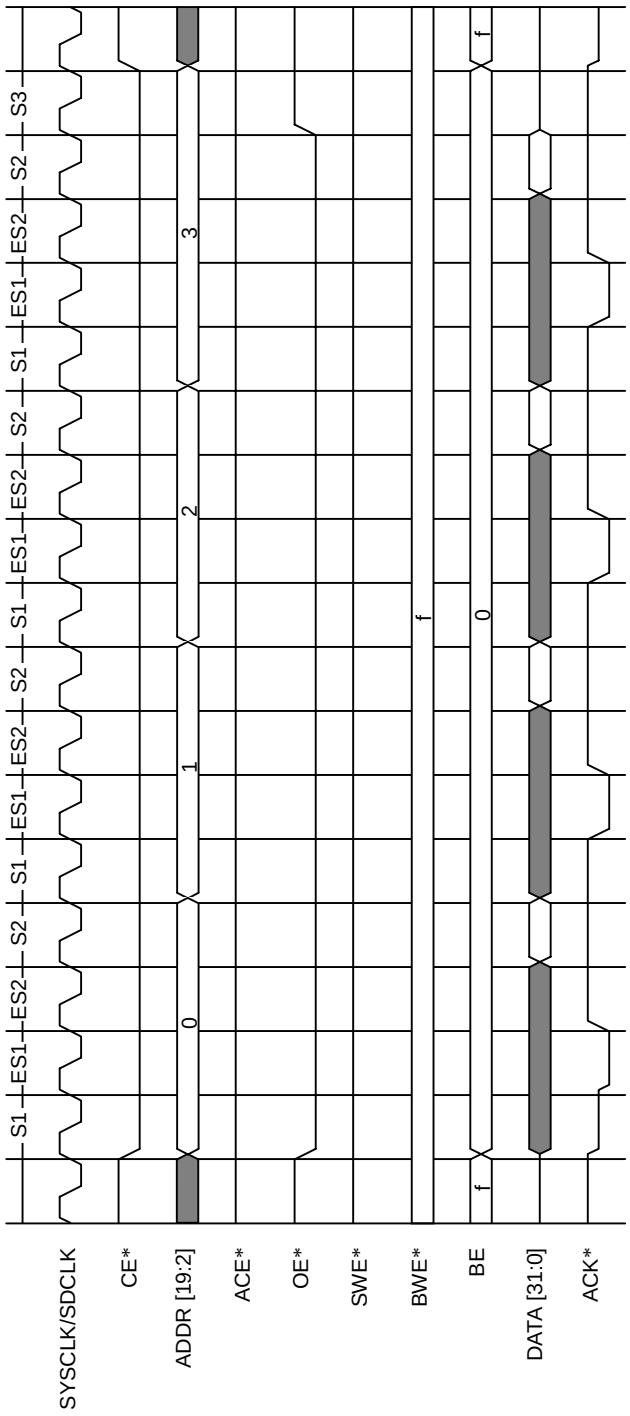


Figure 9.5.17 External ACK* Mode 32-bit Bus Operation (4-word Burst Read, 1 Wait State)

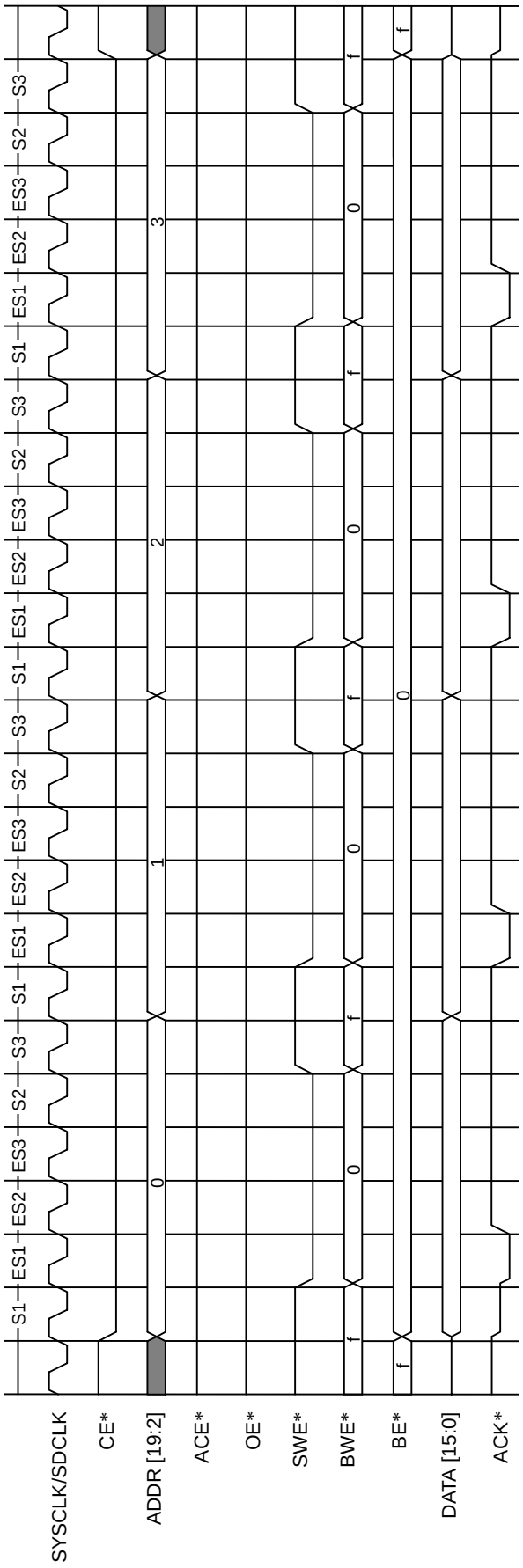


Figure 9.5.18 External ACK* Mode 32-bit Bus Operation (4-word Burst Write, 1 Wait State)

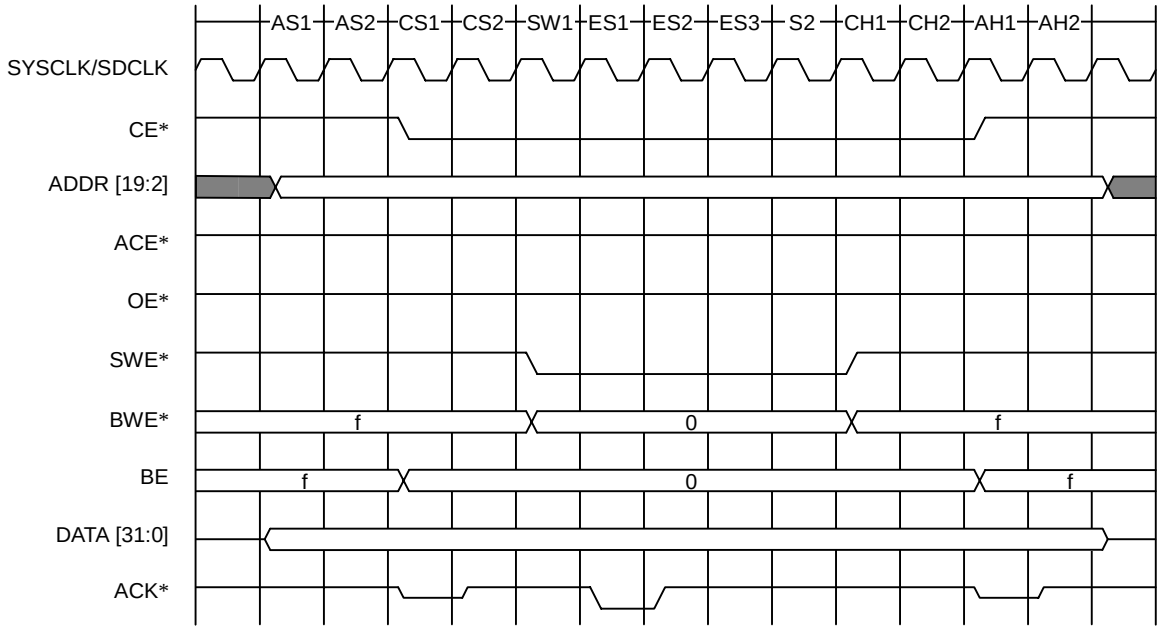


Figure 9.5.19 External ACK* Mode 32-bit Bus Operation (32-bit Single Write, 2 Wait States, SHWT=2)

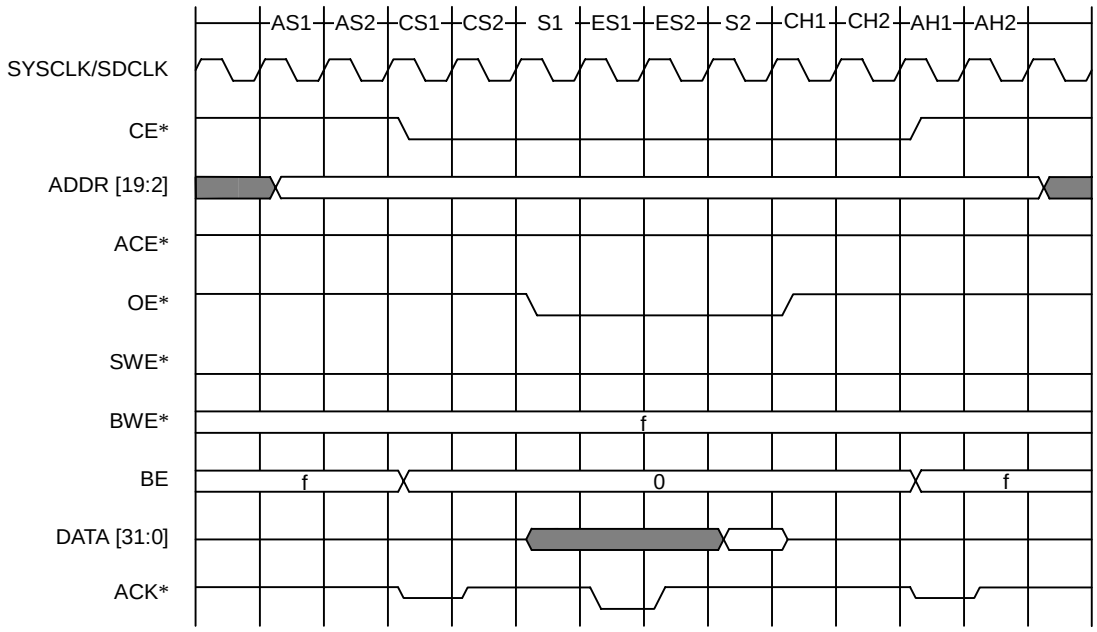


Figure 9.5.20 External ACK* Mode 32-bit Bus Operation (32-bit Single Read, 1 Wait State, SHWT=2)

9.5.8 External ACK* Mode 16-bit Operation

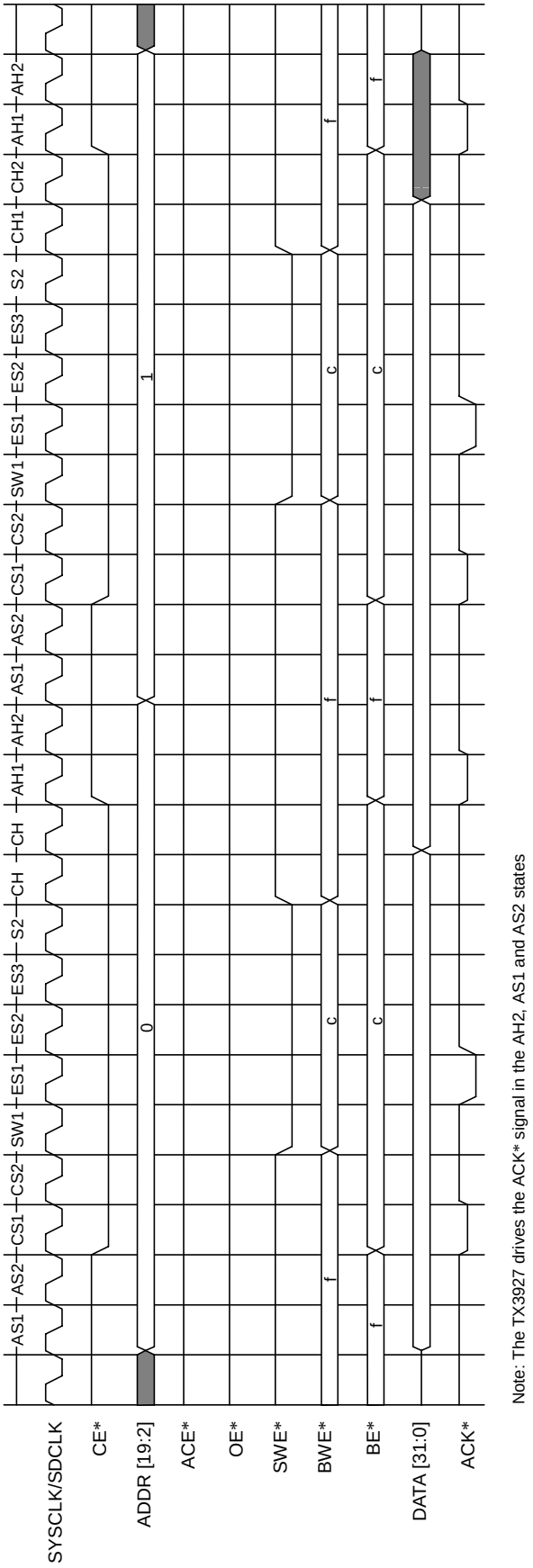


Figure 9.5.21 External ACK* Mode 16-bit Bus Operation (32-bit Single Write, 2 Wait States, SHWT=2)

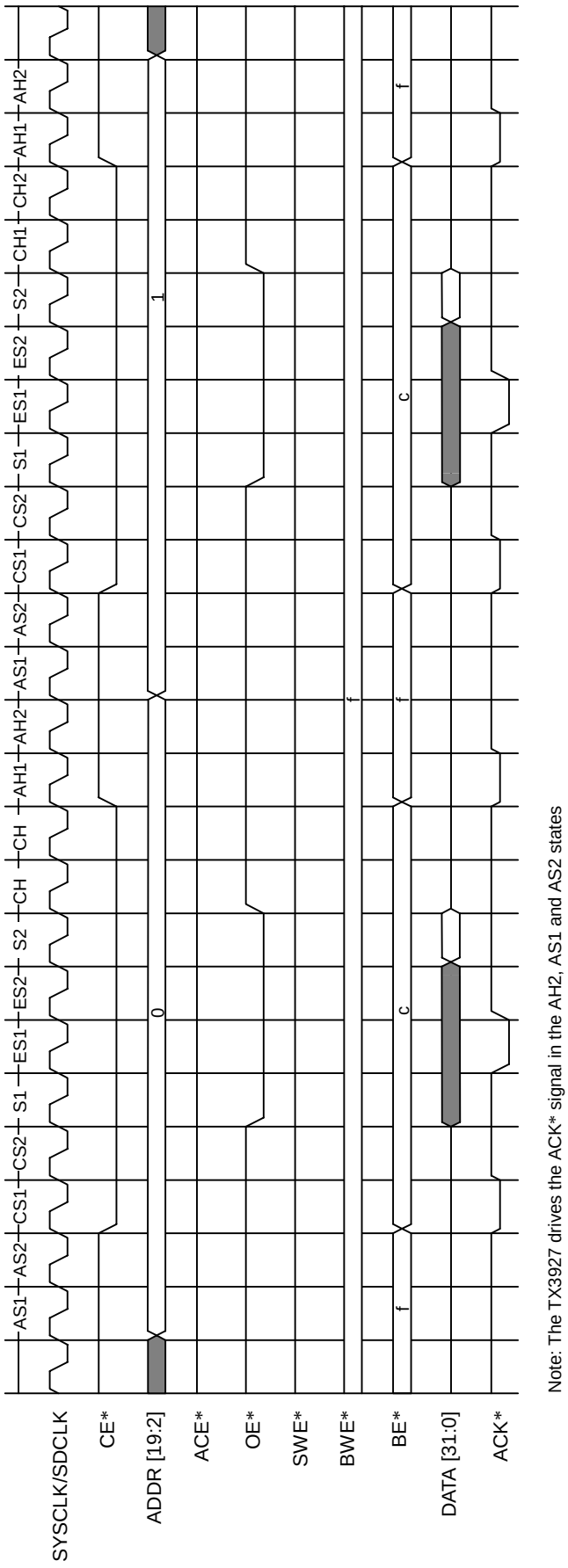


Figure 9.5.22 External ACK* Mode 16-bit Bus Operation (32-bit Single Read 1 Wait State, SHWT=2)

9.5.9 READY Mode 32-bit Operation

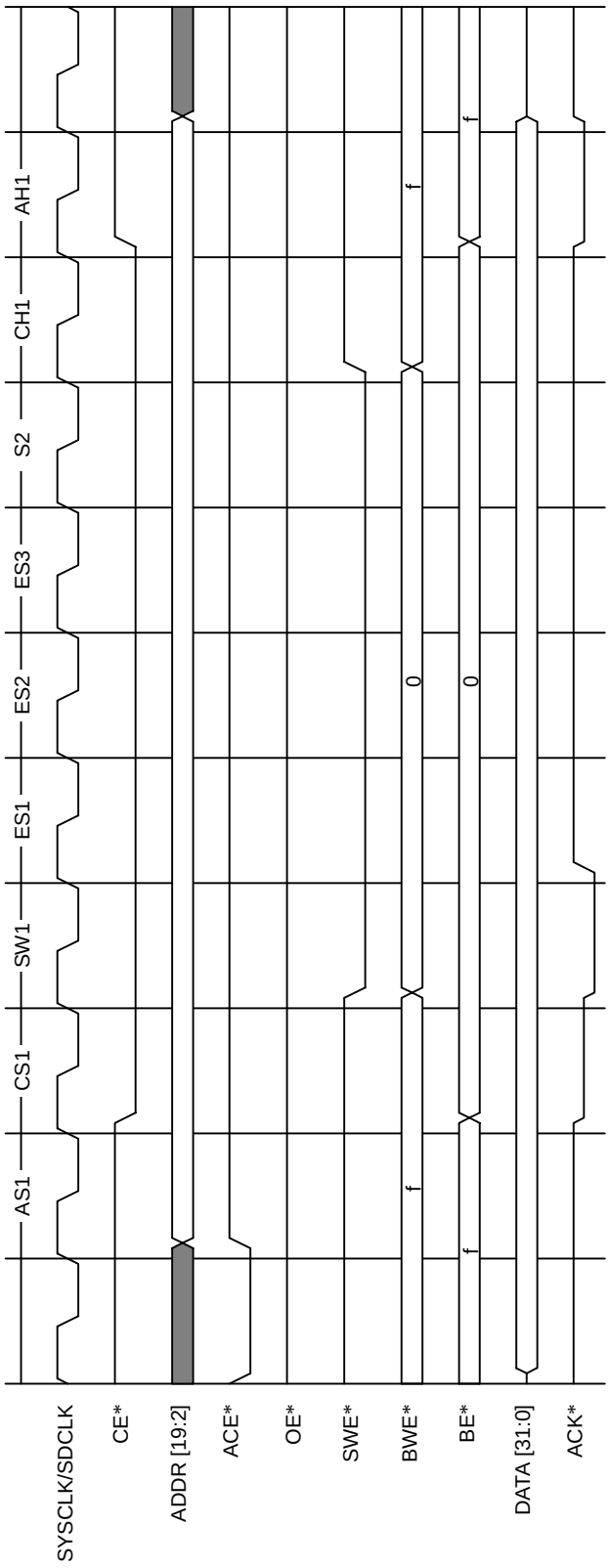


Figure 9.5.23 READY Mode 32-bit Bus Operation (32-bit Single Write, 2 Wait States, SHWT=1)

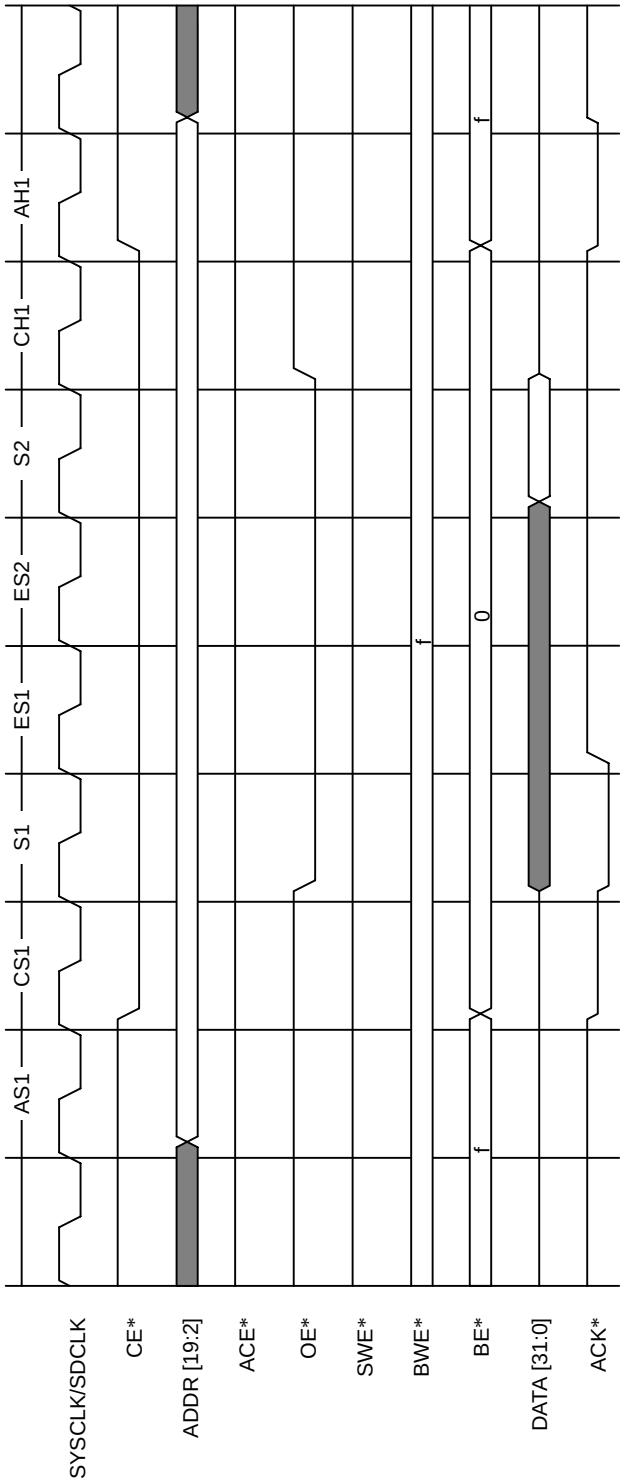


Figure 9.5.24 READY Mode 32-bit Bus Operation (32-bit Single Read, 1 Wait State, SHWT=1)

9.6 Examples of Using Flash ROM and SRAM

Figure 9.6.1 and Figure 9.6.2 show examples of using flash ROM. Figure 9.6.3 shows an example of using SRAM.

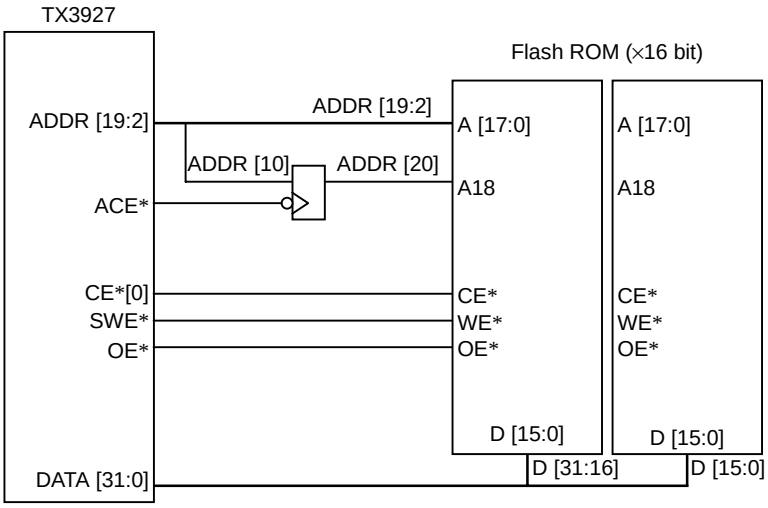


Figure 9.6.1 Example of Using Flash ROM (×16 bits) (32-bit Data Bus)

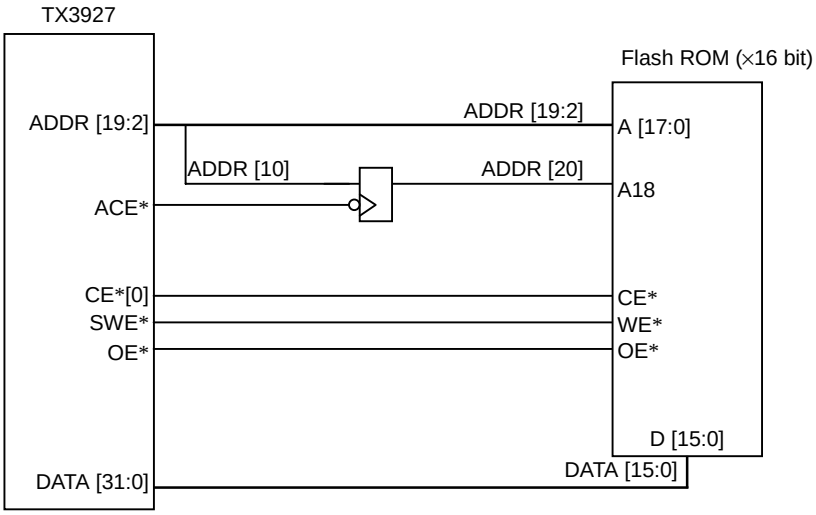


Figure 9.6.2 Example of Using Flash ROM (×16 bits) (16-bit Data Bus)

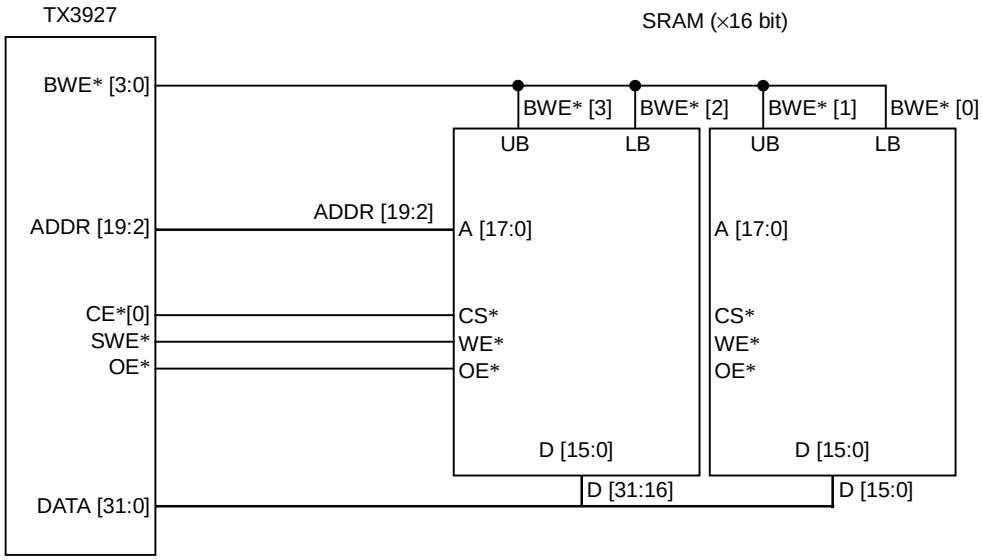


Figure 9.6.3 Example of Using SRAM (×16 bits) (32-bit Data Bus)

10. DMA Controller

10.1 Features

The TX3927 DMA Controller (DMAC) handles data transfers between memory and I/O peripherals and from memory to memory.

The DMA Controller features the following:

- 4 DMA channels
- Supports linked-list command chaining on all channels.
- Single- and dual-address transfers
- Memory-to-memory transfers
  Supports burst reads and writes of up to 8 words (provided data is word-aligned).
- Channel priorities are individually determined for channels configured for snooping and non-snooping operations.
- 24-bit signed address increment registers for both source and destination addresses
- 24-bit byte transfer counters per channel
- Supports 8-, 16-, and 32-bit I/O peripherals.
  16-bit peripherals can use both single- and dual-address transfers.
  8-bit peripherals can use only dual-address transfers.
- The DMA acknowledge pins are available for use as the read/write transfer strobe for off-chip peripherals using single-address transfers.
  For single-address transfers, the I/O peripheral width must be equal to the memory width.

10.2 Block Diagram

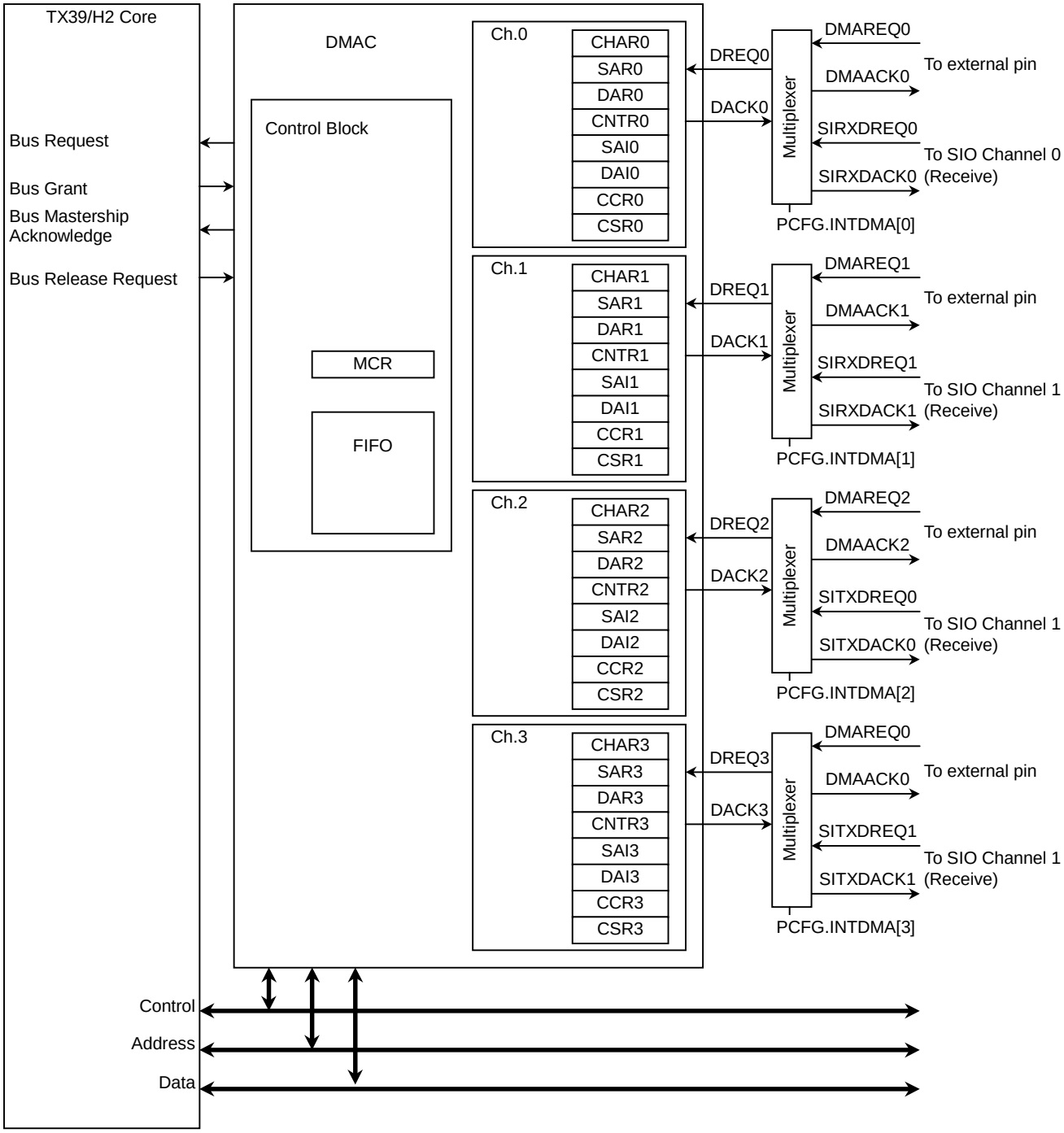


Figure 10.2.1 TX3927 DMAC Block Diagram

10.3 Registers

10.3.1 Register Map

The base address of the DMAC registers is 0xFFFE_B000. All registers in the DMAC can only be word-accessed. Any other type of access will produce an undefined result.

For the bits not defined in this section, the values shown in the figures must be written.

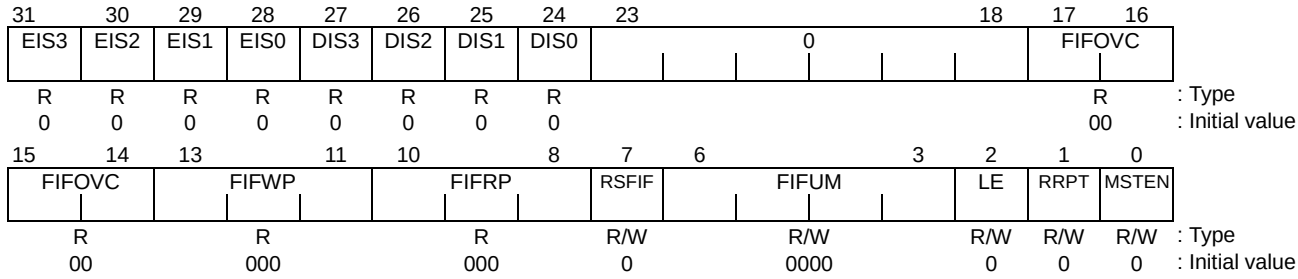
Table 10.3.1 DMA Controller Registers

Address	Register Mnemonic	Register name
0xFFFE_B0A4	MCR	Master Control Register
0xFFFE_B0A0	TDHR	Temporary Data Holding Register *
0xFFFE_B09C	DBR7	Data Buffer Register 7*
0xFFFE_B098	DBR6	Data Buffer Register 6*
0xFFFE_B094	DBR5	Data Buffer Register 5*
0xFFFE_B090	DBR4	Data Buffer Register 4*
0xFFFE_B08C	DBR3	Data Buffer Register 3*
0xFFFE_B088	DBR2	Data Buffer Register 2*
0xFFFE_B084	DBR1	Data Buffer Register 1*
0xFFFE_B080	DBR0	Data Buffer Register 0*
0xFFFE_B07C	CSR3	Channel Status Register 3
0xFFFE_B078	CCR3	Channel Control Register 3
0xFFFE_B074	DAI3	Destination Address Increment Register Channel 3
0xFFFE_B070	SAI3	Source Address Increment Register Channel 3
0xFFFE_B06C	CNTR3	Count Register 3
0xFFFE_B068	DAR3	Destination Address Register 3
0xFFFE_B064	SAR3	Source Address Register 3
0xFFFE_B060	CHAR3	Chained Address Register 3
0xFFFE_B05C	CSR2	Channel Status Register 2
0xFFFE_B058	CCR2	Channel Control Register 2
0xFFFE_B054	DAI2	Destination Address Increment Register 2
0xFFFE_B050	SAI2	Source Address Increment Register 2
0xFFFE_B04C	CNTR2	Count Register 2
0xFFFE_B048	DAR2	Destination Address Register 2
0xFFFE_B044	SAR2	Source Address Register 2
0xFFFE_B040	CHTR2	Chained Address Register 2
0xFFFE_B03C	CSR1	Channel Status Register 1
0xFFFE_B038	CCR1	Channel Control Register 1
0xFFFE_B034	DAI1	Destination Address Increment Register 1
0xFFFE_B030	SAI1	Source Address Increment Register 1
0xFFFE_B02C	CNTR1	Count Register 1
0xFFFE_B028	DAR1	Destination Address Register 1
0xFFFE_B024	SAR1	Source Address Register 1
0xFFFE_B020	CHTR1	Chained Address Register 1
0xFFFE_B01C	CSR0	Channel Status Register 0
0xFFFE_B018	CCR0	Channel Control Register 0
0xFFFE_B014	DAI0	Destination Address Increment Register 0
0xFFFE_B010	SAI0	Source Address Increment Register 0
0xFFFE_B00C	CNTR0	Count Register 0
0xFFFE_B008	DAR0	Destination Address Register 0
0xFFFE_B004	SAR0	Source Address Register 0
0xFFFE_B000	CHAR0	Chained Address Register 0

Note: The temporary data holding register and the data buffer registers are read-only registers used for debugging purposes. If a DMA transfer is aborted or halted, the contents of these registers can be read via software to execute the subsequent transfer operation.

10.3.2 Master Control Register (MCR) 0xFFFE_B0A4

This register contains control and status bits for all DMAC channels.



Bits	Mnemonic	Field Name	Description
31	EIS3	Error Interrupt Status for Channel 3	Error Interrupt Status for Channel 3 (initial value: 0x0) Indicates whether an error interrupt condition has occurred on Channel 3. This bit is the logical AND of the Interrupt Enable on Error bit of the Channel Control Register 3 (CCR3.INTENE) and the Abnormal Chain Completion bit of the Channel Status Register 3 (CSR3.ABCHC). 1: An error interrupt condition has occurred. 0: An error interrupt condition has not occurred.
30	EIS2	Error Interrupt Status for Channel 2	Error Interrupt Status for Channel 2 (initial value: 0x0) Indicates whether an error interrupt condition has occurred on Channel 2. This bit is the logical AND of the Interrupt Enable on Error bit of the Channel Control Register 2 (CCR2.INTENE) and the Abnormal Chain Completion bit of the Channel Status Register 2 (CSR2.ABCHC). 1: An error interrupt condition has occurred. 0: An error interrupt condition has not occurred.
29	EIS1	Error Interrupt Status for Channel 1	Error Interrupt Status for Channel 1 (initial value: 0x0) Indicates whether an error interrupt condition has occurred on Channel 1. This bit is the logical AND of the Interrupt Enable on Error bit of the Channel Control Register 1 (CCR1.INTENE) and the Abnormal Chain Completion bit of the Channel Status Register 1 (CSR1.ABCHC). 1: An error interrupt condition has occurred. 0: An error interrupt condition has not occurred.
28	EIS0	Error Interrupt Status for Channel 0	Error Interrupt Status for Channel 0 (initial value: 0x0) Indicates whether an error interrupt condition has occurred on Channel 0. This bit is the logical AND of the Interrupt Enable on Error bit of the Channel Control Register 0 (CCR0.INTENE) and the Abnormal Chain Completion bit of the Channel Status Register 0 (CSR0.ABCHC). 1: An error interrupt condition has occurred. 0: An error interrupt condition has not occurred.
27	DIS3	Done Interrupt Status for Channel 3	Done Interrupt Status for Channel 3 (initial value: 0x0) Indicates whether a transfer-done interrupt condition has occurred on Channel 3. The value of this bit is determined from the interrupt enable bits of the Channel Control Register 3 (CCR3.INTENC and CCR3.INTENT) and the normal completion bits of the Channel Status Register 3 (CSR3.NCHNC and CSR3.NTRNFC) as follows: (CCR3.INTENC & CSR3.NCHNC) | (CCR3.INTENT & CSR3.NTRNFC) 1: A transfer-done interrupt condition has occurred. 0: A transfer-done interrupt condition has not occurred.
26	DIS2	Done Interrupt Status for Channel 2	Done Interrupt Status for Channel 2 (initial value: 0x0) Indicates whether a transfer-done interrupt condition has occurred on Channel 2. The value of this bit is determined from the interrupt enable bits of the Channel Control Register 2 (CCR2.INTENC and CCR2.INTENT) and the normal completion bits of the Channel Status Register 2 (CSR2.NCHNC and CSR2.NTRNFC) as follows: (CCR2.INTENC & CSR2.NCHNC) | (CCR2.INTENT & CSR2.NTRNFC) 1: A transfer-done interrupt condition has occurred. 0: A transfer-done interrupt condition has not occurred.

Figure 10.3.1 Master Control Register (1/2)

Bits	Mnemonic	Field Name	Description
25	DIS1	Done Interrupt Status for Channel 1	Done Interrupt Status for Channel 1 (initial value: 0x0) Indicates whether a transfer-done interrupt condition has occurred on Channel 1. The value of this bit is determined from the interrupt enable bits of the Channel Control Register 1 (CCR1.INTENC and CCR1.INTENT) and the normal completion bits of the Channel Status Register 1 (CSR1.NCHNC and CSR1.NTRNFC) as follows: (CCR1.INTENC & CSR1.NCHNC) | (CCR1.INTENT & CSR1.NTRNFC) 1: A transfer-done interrupt condition has occurred. 0: A transfer-done interrupt condition has not occurred.
24	DIS0	Done Interrupt Status for Channel 0	Done Interrupt Status for Channel 0 (initial value: 0x0) Indicates whether a transfer-done interrupt condition has occurred on Channel 0. The value of this bit is determined from the interrupt enable bits of the Channel Control Register 0 (CCR0.INTENC and CCR0.INTENT) and the normal completion bits of the Channel Status Register 0 (CSR0.NCHNC and CSR0.NTRNFC) as follows: (CCR0.INTENC & CSR0.NCHNC) | (CCR0.INTENT & CSR0.NTRNFC) 1: A transfer-done interrupt condition has occurred. 0: A transfer-done interrupt condition has not occurred.
17:14	FIFOVC	FIFO Valid Entry Count	FIFO Valid Entry Count (initial value: 0x0) These read-only bits contain the number of data items present in the FIFO. This information is provided for a software procedure to recover any data in the FIFO in the event a transfer is halted by clearing the Transfer Active (XTACT) bit of the Channel Control Register.
13:11	FIFWP	FIFO Write Pointer	FIFO Write Pointer (initial value: 000) These read-only bits point to the next FIFO location to be written to. The write pointer is provided only to maintain symmetry with the read pointer, and is not needed for software recovery in the event a transfer is halted by clearing the Transfer Active (XTACT) bit of the Channel Control Register cleared.
10:8	FIFRP	FIFO Read Pointer	FIFO Read Pointer (initial value: 000) These read-only bits point to the next FIFO location to be read out. This information is provided for software recovery in the event a transfer is halted by clearing the Transfer Active bit.
7	RSFIF	Reset FIFO	Reset FIFO (initial value: 0) Resets the FIFO. Setting this bit to 1 initializes the FIFO read pointer, write pointer and valid entry count to zero.
6:3	FIFUM	FIFO Use Mask	FIFO Use Mask (initial value: 0000) These four bits control the usability of the 8-word FIFO for memory-to-memory transfers. Bit 3 corresponds to Channel 0, bit 4 to Channel 1, bit 5 to Channel 2, and bit 6 to Channel 3. Multiple channels can share the FIFO if none of them needs to retain the FIFO data between DMA bus cycles. There is no way to detect a violation by hardware; in case of a violation, correct operation is not guaranteed. If a channel is configured for dual-address mode with 4- or 8-word transfer size, the corresponding FIFUM bit must be set to 1.
2	LE	Little Endian	Little Endian (initial value: 0) Controls the endian mode of the DMAC. It must be the same as that specified with the ENDIAN boot signal. 1: The DMAC operates in little-endian mode. 0: The DMAC operates in big-endian mode.
1	RRPT	Round Robin Priority	Round Robin Priority (initial value: 0) Sets the priority mode. 1: Round robin priority. The channel which was most recently serviced inherits the lowest priority. After the highest-priority channel is serviced, priority is passed to the next highest-priority channel. 0: Fixed priority. Channel 0 has the highest priority, then 1, 2 and Channel 3 has the lowest priority. Note: The priority is set up separately for a group of channels enabled for snooping and for a group of channels with no snooping.
0	MSTEN	MSTEN Master Enable	Master Enable (initial value: 0) Enables the DMAC. 1: Enabled (Dreqs is recognized) 0: Disabled (Dreqs is not recognized) Note: When the DMAC is disabled, all its internal logic is reset, including the bus interface logic and the state machine.

Figure 10.3.1 Master Control Register (2/2)

10.3.3 Channel Control Registers (CCRn)

0xFFFE_B018 (ch. 0),
0xFFFE_B038 (ch. 1),
0xFFFE_B058 (ch. 2),
0xFFFE_B078 (ch. 3),

31	27	26	25	24	23	22	21	20	19	18	17	16	
0		DBINH	SBINH	CHRST	RVBYTE	AckPOL	REQPL	EGREQ	CHDN	DNCTL		EXTRQ	
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	: Type
		0	0	1	0	0	0	0	0	00		0	: Initial value

15	13	12	11	10	9	8	7	6	5	4	2	1	0	
INTRQD		INTENE	INTENC	INTENT	CHNEN	XTACT	SNOP	DSTINC	SRCINC	XFSZ		MEMIO	ONEAD	
R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	: Type
000		0	0	0	0	0	0	0	0	000		0	0	: Initial value

Bits	Mnemonic	Field Name	Description
26	DBINH	Destination Burst Inhibit	Destination Burst Inhibit (initial value: 0) Controls whether to use burst transfers or single transfers for destination write cycles in dual-address transfers when the transfer size (XFSZ) field is programmed for burst transfers. This bit does not affect single-address transfers or chaining operations. 1: Channel does not burst to memory. 0: Channel will burst to memory. To use the burst capability, the destination address alignment and the byte transfer count must meet certain requirements.
25	SBINH	Source Burst Inhibit	Source Burst Inhibit (initial value: 0) Controls whether to use burst transfers or single transfers for source read cycles in dual-address transfers when the transfer size (XFSZ) field is programmed for burst transfers. This bit does not affect single-address transfers or chaining operations. 1: Channel does not burst from memory. 0: Channel will burst from memory. To use the burst capability, the source address alignment and the byte transfer count must meet certain requirements.
24	CHRST	Channel Reset	Channel Reset (initial value: 1) Initializes the channel. 1: Clears all the status bits, the Transfer Active bit and the Chain Enable bit as well as resets all internal logic and states for the channel. This bit must be cleared by software prior to executing DMA transfers. 0: Enables the channel for DMA operation.
23	RVBYTE	Reverse Bytes	Reverse Bytes (initial value: 0) Controls whether to reverse the byte order or not. 1: Reverses the byte order for dual-address transfers of 32 bits or more. 0: Does not reverse the byte order.
22	ACKPOL	Acknowledge Polarity	Acknowledge Polarity (initial value: 0) Selects the polarity of the DMA acknowledge signal (DMAACKn). 1: Active-high 0: Active-low
21	REQPL	REQPL Request Polarity	Request Polarity (initial value: 0) Selects the polarity of the DMA request signal (DMAREQn). 1: Active-high 0: Active-low
20	EGREQ	Edge Request	Edge Request (initial value: 0) Controls the edge/level sensitivity for the DMA request signal (DMAREQn). 1: The external DMA request signal is edge-triggered. The channel starts a DMA transfer each time an active edge is applied to the DMAREQn pin. In edge-triggered mode, before next DMA transfer is requested, the DMAREQn pin must be deasserted. 0: The external DMA request signal is level-sensitive.

Figure 10.3.2 Channel Control Registers (1/3)

Bits	Mnemonic	Field Name	Description
19	CHDN	Chain Done	Chain Done (initial value: 0) 1: DMADONE* controls chained transfers instead of one DMA process. As an output, DMADONE* will be asserted when DMA data chaining completes. As an input, assertion of DMADONE* will cause the active channel to stop the chaining. 0: DMADONE* controls one DMA process instead of chained transfers. As an outputs DMADONE* will be asserted upon completion of a DMA process. As an input, assertion of DMADONE* will cause the channel to stop the entire DMA transaction (the current transfer and the entire chaining operation).
18:17	DNCTL	Done Control	Done Control (initial value: 00) Enables or disables the validity of the DMADONE* signal. 00: DMADONE* is used neither as an output nor as an input. 01: DMADONE* is not used as an output. As an input, DMADONE* will cause the current DMA process or chaining operation to terminate. 10: As an output, DMADONE* is asserted upon completion of a DMA process or chaining operation. DMADONE* is not used (i.e., ignored) as an input. 11: When the channel is active, DMADONE* is used as an open-drain output. It is asserted upon completion of a DMA process or chaining operation. As an input, DMADONE* will cause the current DMA process or chaining operation to terminate. Note: Care must be taken when using open-drain mode because an external pull-up resistor will cause the transition time of the output driver to increase.
16	EXTRQ	External Request	External Request (initial value: 0) Selects the transfer request mode. 1: Assertion of the external DMA request signal requests DMA service. 0: Internally generated request transfers. This mode is used for memory-to-memory transfers.
15:13	INTRQD	Internal Request Delay	Internal Request Delay (initial value: 000) Limits the amount of bus utilization by setting an interval between two internally generated transfer requests. 000: Minimum interval (Even in this case, the bus is once released and then reacquired.) 001: 16 clock cycles 010: 32 clock cycles 011: 64 clock cycles 100: 128 clock cycles 101: 256 clock cycles 110: 512 clock cycles 111: 1024 clock cycles
12	INTENE	Interrupt Enable on Error	Interrupt Enable on Error (initial value: 0) 1: Enables DMA interrupts for errors. 0: Disable DMA interrupts for errors.
11	INTENC	Interrupt Enable on Chain Done	Interrupt Enable on Chain Done (initial value: 0) 1: Enables DMA interrupts for chaining. 0: Disables DMA interrupts for chaining.
10	INTENT	Interrupt Enable on Transfer Done	Interrupt Enable on Transfer Done (initial value: 0) 1: Enables DMA interrupts for end-of-transfer. 0: Disables DMA interrupts for end-of-transfer.
9	CHNEN	Chain Enable	Chain Enable (initial value: 0) Enables DMA data chaining. 1: Immediately upon completion of the transfers, the channel's registers are reloaded from the address specified in the Chained Address Register. Transfers then continue uninterrupted. 0: The channel does not chain.
8	XTACT	Transfer Active	Transfer a Active (initial value: 0) 1: Starts DMA transfer on the channel. Prior to executing a DMA transfer, set appropriate parameters in the register and then write a 1 to this bit. Writing a 0 to this bit causes the transfer to halt gracefully; then resetting it to 1 restarts the transfer. This bit is automatically cleared when the transfer completes normally or halts due to an error.

Figure 10.3.2 Channel Control Registers (2/3)

Bits	Mnemonic	Field Name	Description
7	SNOP	Snoop	Snoop (initial value: 0) Controls whether to request bus mastership involving snoop operations. 1: Uses snooping bus operations (GSREQ). Snooping is enabled during DMA write cycles to maintain coherency between the data cache and main memory. 0: Uses non-snooping bus operations (GHPGREQ). Note: This bit must be cleared when the data cache uses write-back mode.
6	DSTINC	Mixed Destination Increment	Mixed Destination Increment (initial value: 0) Controls how to increment the destination address. 1: After each transfer increments the destination address by the value of the Destination Address Increment Register at odd word boundaries and by 4 bytes at even word boundaries. 0: After each transfer always increments the destination address by the value of the Destination Address Increment Register. Note: The destination address is also incremented during burst cycles. However, only memory devices connected to the SDRAM Controller can use non-standard increments (other than 4) for burst transfers. The increment can be between 0 and 255 words (1020 bytes).
5	SRCINC	Mixed Source Increment	Mixed Source Increment (initial value: 0) Controls how to increment the source address. 1: After each transfer, increments the source address by the value of the Source Address Increment Register at odd word boundaries and by 4 bytes at even word boundaries. 0: After each transfer, always increments the source address by the value of the Source Address Increment Register. Note: The source address is also incremented during burst cycles. However, only memory devices connected to the SDRAM Controller can use non-standard increments (other than 4) for burst transfers. The increment can be between 0 and 255 words (1020 bytes).
4:2	XFSZ	Transfer Size	Transfer Size (initial value: 000) Specifies the data transfer size for a DMA request. 000: 8 bits. Valid for dual-address transfers. 001: 16 bits. Valid for dual- and single-address transfers between memory and I/O peripherals. 010: 1 word. Valid for dual- and single-address transfers between memory and I/O peripherals. 011: Reserved 100: 4 words. Valid for dual-address transfers when the MCR FIFUM bit for this channel is set, and for single-address transfers between memory and I/O peripherals. 101: 8 words. Valid for dual-address transfers when the MCR FIFUM bit for this channel is set, and for single-address transfers between memory and I/O peripherals. 110: 16 words. Valid for single-address transfers between memory and I/O peripherals. 111: 32 words. Valid for single-address transfers between memory and I/O peripherals.
1	MEMIO	Memory to I/O	Memory to I/O (initial value: 0) Selects the direction of single-address transfers. (This bit does not affect dual-address transfers.) 1: Memory to I/O 0: I/O to memory
0	ONEAD	One Address	One Address (initial value: 0) Controls whether to run single-address or dual-address transfers. 1: Single-address transfers 0: Dual-address transfers

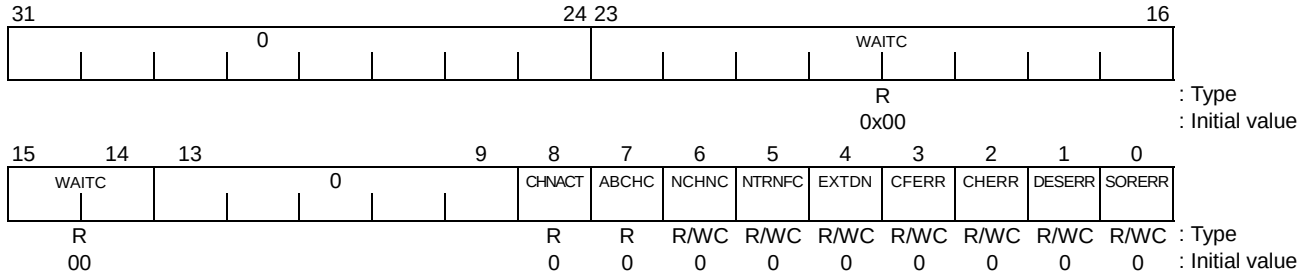
Figure 10.3.2 Channel Control Registers (3/3)

Note: Bit 7 (SNOP) must be cleared when the data cache uses write-back mode.

10.3.4 Channel Status Registers (CSRn)

0xFFFE_B01C (ch. 0),
0xFFFE_B03C (ch. 1),
0xFFFE_B05C (ch. 2),
0xFFFE_B07C (ch. 3)

Reading a Channel Status Register has no effect on the value of any of its bits. Writing a 0 to any of the bits also has no effect on the bit. Writing a 1 to a bit will clear that bit except for the Channel Active (CHNACT) bit, the Abnormal Chain Completion (ABCHC) bit and the Internal Wait Counter (WAITC) field. The Channel Active (CHNACT) bit is read-only and contains a copy of the XTACT bit of the corresponding Channel Control Register (CCRn). The Abnormal Chain Completion (ABCHC) bit is the logical OR of all the error bits (CFERR, CHERR, DESERR and SORERR). To clear this bit, all the error bits must be cleared.



Bits	Mnemonic	Field Name	Description
23:14	WAITC	Internal Wait Counter	Internal Wait Counter (initial value: 0x000) This read-only field allows software to track the elapsed time for the interval taken between DMA bus cycles.
8	CHNACT	CHNACT Channel Active	Channel Active (initial value: 0) This bit contains a copy of the XTACT bit of the Channel Control Register (CCRn).
7	ABCHC	Abnormal Chain Completion	Abnormal Chain Completion (initial value: 0) Indicates an occurrence of an error. This bit contains the logical OR of all the error bits (CFERR, CHERR, DESERR and SORERR). 1: The chaining operation has terminated with an error. 0: The chaining operation has not encountered an error since the error bits were last cleared.
6	NCHNC	Normal Chain Completion	Normal Chain Completion (initial value: 0) 1: The chaining operation has completed normally. 0: The chaining operation has not completed normally since this bit was last cleared.
5	NTRNFC	Normal Transfer Completion	Normal Transfer Completion (initial value: 0) Indicates whether a DMA transfer has been completed normally by either a software command or the DMADONE* hardware signal. 1: The DMA transfer has completed normally. 0: The DMA transfer has not completed since this bit was last cleared.
4	EXTDN	External Done Asserted	External Done Asserted (initial value: 0) Indicates whether the DMADONE* signal has been asserted externally. 1: The external DMADONE* signal has been asserted. 0: The external DMADONE* signal has not been asserted
3	CFERR	Configuration Error	Configuration Error (initial value: 0) Indicates whether a configuration error has occurred. A configuration error results when channel settings are inconsistent with the current command. 1: A configuration error is present. 0: No configuration error exists.
2	CHERR	Chain Bus Error	Chain Bus Error (initial value: 0) Indicates whether a bus error has occurred due to no response from memory during a chaining operation. 1: A bus error has occurred. 0: No bus error has occurred.

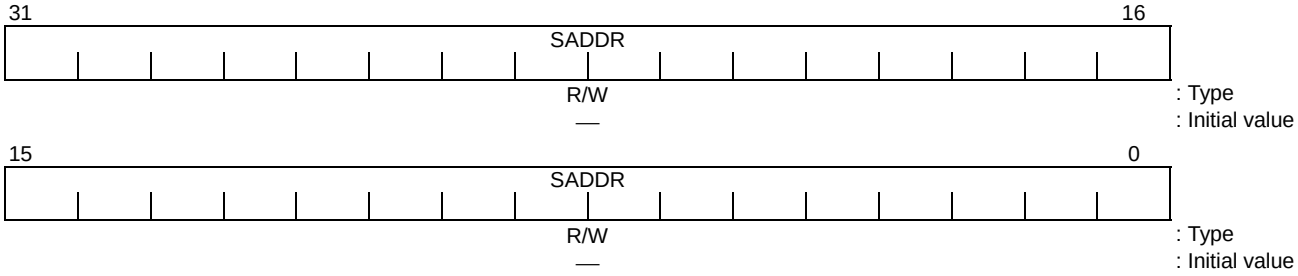
Figure 10.3.3 Channel Status Registers (1/2)

Bits	Mnemonic	Field Name	Description
1	DESERR	Destination Bus Error	Destination Bus Error (initial value: 0) Indicates whether a bus error has occurred during a destination write bus cycle. 1: A bus error has occurred. 0: No bus error has occurred.
0	SORERR	Source Bus Error	Source Bus Error (initial value: 0) Indicates whether a bus error has occurred during a source read or write bus cycle. 1: A bus error has occurred. 0: No bus error has occurred.

Figure 10.3.3 Channel Status Registers (2/2)

10.3.5 Source Address Registers (SARn)

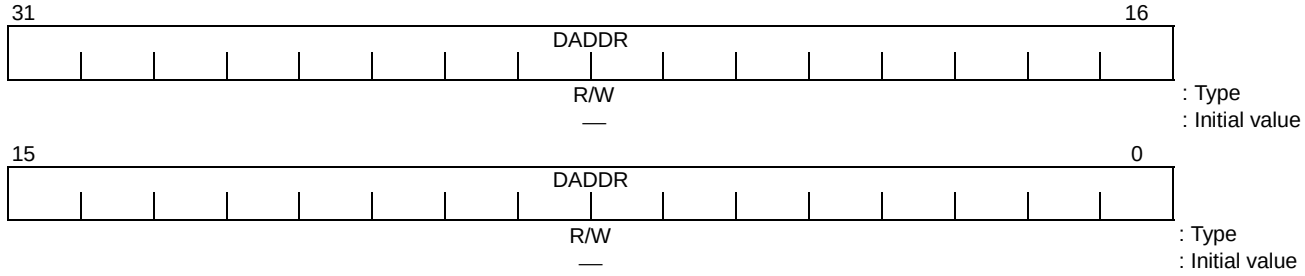
0xFFFE_B004 (ch. 0),
0xFFFE_B024 (ch. 1),
0xFFFE_B044 (ch. 2),
0xFFFE_B064 (ch. 3)



Bits	Mnemonic	Field Name	Description
31:0	SADDR	Source Address	Source Address (initial value: undefined) The Source Address Register is loaded with the physical address of the source for dual-address transfers. For single-address transfers, this register is loaded with the memory address, whether the transfer is from memory to I/O or from I/O to memory. The address must be aligned, depending on the transfer size. For example, if the transfer size is a 32-bit word, the low-order two bits of the address must be 0.

Figure 10.3.4 Source Address Registers

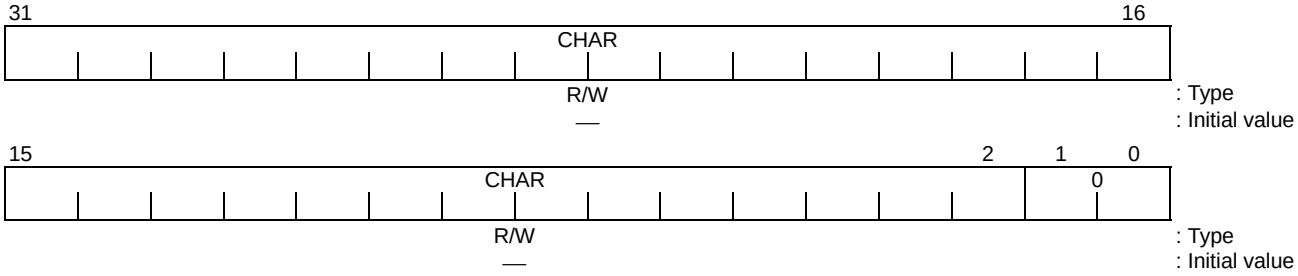
10.3.6 Destination Address Registers (DARn) 0xFFFE_B008 (ch. 0), 0xFFFE_B028 (ch. 1), 0xFFFE_B048 (ch. 2), 0xFFFE_B068 (ch. 3)



Bits	Mnemonic	Field Name	Description
31:0	DADDR	Destination Address	Destination Address Registers (initial value: undefined) The Destination Address Register is loaded with the physical address of the destination for dual-address transfers. It is not used for single-address transfers. The address must be aligned, depending on the transfer size. For example, if the transfer size is a 32-bit word, the low-order two bits of the address must be 0.

Figure 10.3.5 Destination Address Registers

10.3.7 Chained Address Registers (CHARn) 0xFFFE_B000 (ch. 0), 0xFFFE_B020 (ch. 1), 0xFFFE_B040 (ch. 2), 0xFFFE_B060 (ch. 3)



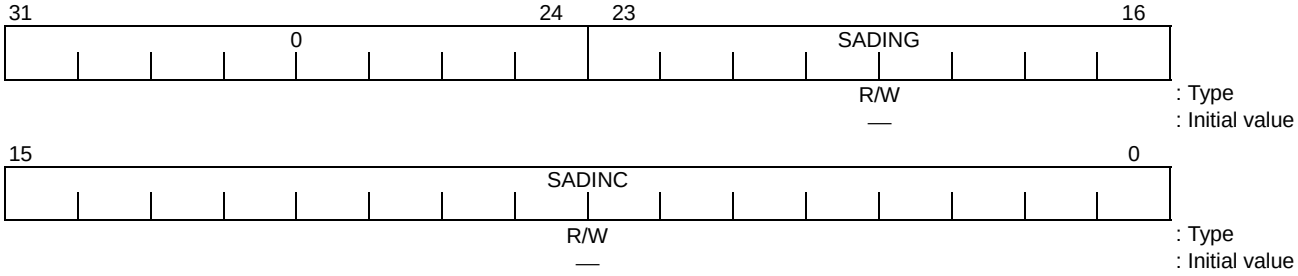
Bits	Mnemonic	Field Name	Description
31:2	CHAR	Chained Address	Chained Address This register is loaded with the physical starting address of the data block from which the DMAC registers will be reloaded. When the current transfer completes, if the CHNEN bit in the CCRn register is set, eight words are read from this address and loaded into the CHARn, SARn, DARn, CNTRn, SAIn, DAIn, CCRn and CSRn registers for the channel. If the new CCRn.XTACT value is 1, the next transfer starts immediately.

Note: The low-order two bits of the Chain Address Register (CHARn) must be 0; otherwise, a configuration error occurs, if the Master Enable bit (MCR.MSTEN) is 1, the Channel Reset bit (CCRn.CHRST) is 0, and the Transfer Active bit (CCRn.XTACT) is 1.

If the chained address is aligned on an odd 4-word boundary, a chain refill is distributed over two consecutive 4-word bursts. If the chained address is aligned on a modulo-8 word boundary, a chain refill is accomplished in a 8-word burst. Even if the chain address is not aligned on a modulo-4 word boundary, the DMAC recognizes the situation and achieves a chain refill as a sequence of four single reads from misaligned addresses and one 4-word burst read from a 4-word-aligned address.

Figure 10.3.6 Chain Address Registers

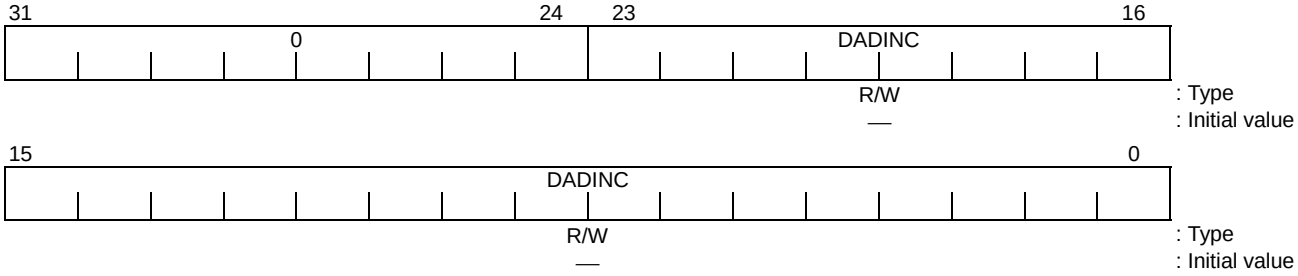
10.3.8 Source Address Increment Registers (SAIn) 0xFFFE_B010 (ch. 0), 0xFFFE_B030 (ch. 1), 0xFFFE_B050 (ch. 2), 0xFFFE_B070 (ch. 3)



Bits	Mnemonic	Field Name	Description
23:0	SADINC	Source Address Increment	Source Address Increment (initial value: undefined) This register is loaded with a 24-bit signed byte count to be added to the source address after each transfer. The increment must be an integer multiple of the transfer size. If the Mixed Source Increment (SRCINC) bit in the CCR register is set, this value is added to addresses at odd word boundaries; the fixed increment of four is added to addresses at even word boundaries.

Figure 10.3.7 Source Address Increment Registers

10.3.9 Destination Address Increment Registers (DAIn) 0xFFFE_B014 (ch. 0), 0xFFFE_B034 (ch. 1), 0xFFFE_B054 (ch. 2), 0xFFFE_B074 (ch. 3)

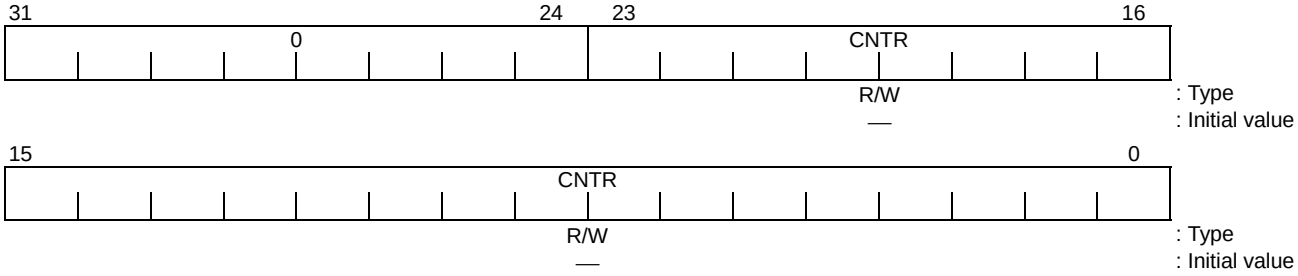


Bits	Mnemonic	Field Name	Description
23:0	DADINC	Destination Address Increment	Destination Address Increment (initial value: undefined) This register is loaded with 24-bit signed byte count to be added to the destination address after each transfer. The increment must be an integer multiple of the transfer size. If the Mixed Destination Increment (DSTINC) bit in the CCR register is set, this value is added to addresses at odd word boundaries; the fixed increment of four is added to addresses at even word boundaries.

Figure 10.3.8 Destination Address Increment Registers

10.3.10 Count Registers (CNTRn)

0xFFFE_B00C (ch. 0),
0xFFFE_B02C (ch. 1),
0xFFFE_B04C (ch. 2),
0xFFFE_B06C (ch. 3)



Bits	Mnemonic	Field Name	Description
23:0	CNTR	Count	Count Register (initial value: undefined) Contains the number of bytes left in the DMA transaction for respective channels. The count is a 24-bit unsigned value, decremented for each successful transfer. It must be an integer multiple of the transfer size.

Figure 10.3.9 Count Registers

10.4 Operation

The TX3927 DMA Controller (DMAC) consists of four independent, programmable channels. Each channel has a set of eight 32-bit registers called CHARn, SARn, DARn, CNTRn, SAIn, DAIn, CCRn and CSRn. Additionally, the Master Control Register (MCR) contains control bits that affect the operation of all the DMA channels. The DMA Controller also has a temporary data holding register and an 8-word data buffer used in dual-address transfers. All the registers can only be accessed as full 32-bit words.

The DMAC supports dual- and single-address transfers. In dual-address mode, any data transfer takes place in two DMA bus cycles. In this mode, the data is read from the source address and placed in the internal buffer. The data is then written to the destination address.

The single-address transfer mode consists of one DMA bus cycle, which allows data to be transferred directly from the source address to the destination address without going through the DMAC.

- Data flow
  (1) Dual-address mode
    - First bus cycle: Source address to DMAC
    - Second bus cycle: DMAC to destination address

  (2) Single-address mode
    - First bus cycle: Source address to destination address

Each channel supports chained DMA operations. DMA data chaining allows the DMAC to reload a channel's registers from memory upon completion of the transfer, so the next transfer can then continue uninterrupted.

10.4.1 Dual-Address Transfers

Dual-address transfers require at least one read bus cycle and one write bus cycle to execute.

If a channel is programmed with a transfer size of 8, 16 or 32 bits, one data item is transferred once for each bus transaction. The data being transferred is read into the temporary data buffer before being written to the destination. Note that, however, the data buffer might not retain useful information between two DMA requests.

The source and destination addresses are incremented by the signed value specified in the address increment registers (SAIn and DAIn). Each time a channel transfers the amount of data specified in the XFSZ field of the Channel Control Register (CCRn), the Count Register (CNTRn) is decremented by the number of bytes transferred. The transfer continues until the Count Register reaches 0. If the Chain Enable bit (CCRn.CHNEN) is set, the channel will chain when the Count Register has decremented to zero.

Both source and destination addresses have a mixed increment option. If it is enabled, the address is incremented by the value specified in the address increment registers (SAIn or DAIn) at odd word boundaries and incremented by four at even word boundaries. (The value of the A2 bit determines whether the current address is on an odd or even boundary.) The mixed address increment must not be used with a transfer size of less than 32 bits.

Dual-address transfers support of four- and eight-word bursts. To use the burst capability, the source and destination addresses, the address increments, and the byte transfer count must be word-aligned. If any of them is not word-aligned, a configuration error results, and the DMA operation is aborted. In

dual-address burst transfer mode, the data is temporarily read into the internal 8-word buffer. This buffer can be shared between multiple active channels, provided that each channel takes care of the data in the buffer during the current bus transaction. If a channel leaves any data in the buffer, it might get overwritten by new data read from another channel during the next bus transaction. (Refer to "10.4.10 Notes on Using the DMAC FIFO.")

For dual-address transfers with a transfer size of four or eight words, the addresses of data being transferred may not be aligned on a modulo-4 or modulo-8 word boundary. If the addresses are misaligned, the DMAC will recognize the situation and read the data in groups of words using single transfers until the contents of the Source Address Register is on a modulo-4 word boundary, or the Count Register decrements to zero, or the amount of data equal to the transfer size is stored in the buffer. Single-transfers also occur on the destination side if the destination address is not properly aligned. Such single reads and writes can take place at the beginning and end of a DMA transfer when dual-address bursting is enabled. Once the address is on a word boundary, the DMAC performs a burst transfer.

When the source address is modulo-4 word aligned and there are at least four words left to read, the DMAC executes a burst read, placing them into the buffer. The burst size will be eight words if the transfer size is programmed as eight words, AND the source address is modulo-8 word aligned, AND there are at least eight words left to transfer, AND the buffer is empty. If any of these conditions is not met, the burst size will be four words. For the last data that is misaligned or short of the burst size, single transfers will be used. The DMAC will never read or write an unintended word; it always honors the programmer's specifications. The DMAC will perform bursts only when the address is aligned to the burst size.

In dual-address transfer mode, burst reads take place when all of the following conditions are satisfied.

(1) 8-word burst reads

- The transfer size is eight words.
- The source address is aligned on a modulo-8 word boundary.
- There are at least eight words left to read.
- The FIFO is enabled.
- The FIFO is empty.

(2) 4-word burst reads

- The transfer size is four or eight words.
- The source address is aligned on a modulo-4 word boundary.
- There are at least four words left to read.
- The FIFO is enabled.
- The FIFO has empty space for four words.

When any of the following conditions are detected, the DMAC uses 4-word burst reads even when the transfer size is programmed as eight words:

- The source address is aligned on an odd 4-word boundary.
- There are four or more words of data left to read, but less than eight words.
- The FIFO has empty space only for four words.

The write phase of a dual-address transfer follows the same basic steps as the read phase. The data is written to the destination after all data is read into the buffer from the source. The transfer count is decremented at the end of each read operation; so it will be zero when the last write operation is about to

start. When the count is zero, all words in the buffer are written out to memory, using either burst or single writes. When the count is not zero, the DMAC performs single writes until the destination address is on a modulo-4 word boundary.

If four or more words remain in the buffer, the DMAC runs a 4-word burst write to the memory. Otherwise, the bus operation ends. As is the case with the read phase, the DMAC uses burst-writes except on misaligned addresses at the beginning and end of a transfer. Unlike reads, however, the DMAC can perform two 4-word burst writes back-to-back in a bus transaction. The DMAC performs a 8-word burst write if the transfer size is eight words, AND the destination address is modulo-8 word aligned, AND there are eight words of data in the FIFO, AND the FIFO is enabled for the channel (the MCR.FIFUM bit is set to 1). The DMAC performs two 4-word burst writes if the destination address is aligned on an odd 4-word boundary (and all of the other conditions above are met).

It should be noted that multiple channels can share the 8-word buffer only when their initial source and destination addresses have the same alignment with respect to the transfer size. In other words, (source address MOD transfer size) and (destination address MOD transfer size) must be initially equal.

If a bus error occurs during a dual-address burst transfer, the DMAC terminates the bus operation and clears the buffer.

10.4.2 Chaining Operations

The DMAC uses a linked-list method for DMA data chaining. Data chaining can be used by a channel to automatically initiate the next DMA transfer immediately upon completion of the current transfer. If the Chain Enable bit of the Channel Register (CCRn.CHNEN) is set when the transfer count reaches 0 (or the external DMA-done signal, DMADONE*, is asserted), the DMAC automatically reads eight words from the address pointed to by the Chained Address Register and reloads the CHARn, SARn, DARn, CNTRn, SAIn, DAIn, CCRn and CSRn registers for the channel in this order.

Transfers then continue interrupted, as specified by the new parameters, provided that the Transfer Active bit (CCRn.XTACT) is set. This process continues until the Chain Enable bit (CCRn.CHNEN) or the Transfer Active bit (CCRn.XTACT) is cleared as a result of reloading the channel's register (unless errors occur or DMADONE* is asserted).

For chaining operations, the refilling of the channel's registers use burst operations, in the same way as a dual-address transfer. A register refill always takes place as a 8-word transfer.

If the chained address is modulo-8 word aligned, the DMAC will perform an 8-word burst read to refill all the registers for a channel. If the chained address is modulo-4 word aligned, the DMAC will perform two 4-word bursts consecutively. If the chained address is neither modulo-4 word nor modulo-8 word aligned, the DMAC will perform single reads until the address increments to a modulo-4 word boundary, then a 4-word burst read, and finally single reads until it has read a total of eight words.

10.4.3 Single-Address Transfers

During the single-address read cycle, the DMAC controls the transfer of data from memory to an I/O device. The DMAC reads data from the address specified in the Source Address Register, but does not buffer the transferred data. The off-chip I/O device must use the ACK* signal as a read strobe and captures the data on the rising edge of the clock where ACK* is asserted.

During the single-address write cycle, the DMAC controls the transfer of data from an I/O device to memory. The DMAC writes the data to the address specified in the Source Address Register. The DMAC asserts ACK* to indicate to the off-chip I/O device that a request is now being serviced. The I/O device must place data on the data bus when it receives the DMAACK* signal.

For burst write transfers, the off-chip device must supply the next data within two clock cycles after ACK* is asserted. This process continues until the burst is complete. There is no way to slow down the memory operation for DMA transfers; so the system designer must ensure that any off-chip device that will participate in single-address DMA transfers can keep up with the memory access rate.

Note: The TX3927 does not drive the data bus during single-address mode DMA transfers.

10.4.4 DMA Channel Termination by the External DMADONE* Input

The DMADONE* pin can be configured as an input, output, bidirectional or don't-care pin on a per channel basis. When channel is programmed to use the DMADONE* pin, DMADONE* responds to the DMAACK signal. If DMADONE* is configured as don't-care for all active channels, it can be used for its alternate function, or timer output (TIMER[0]).

When DMADONE* is programmed as an output for a channel, it is asserted low by the TX3927 when the channel completes a DMA transfer or the entire chained transfers, depending on the setting of the Chain Done bit (CHDN) in the Channel Control Register (CCRn).

DMADONE* will be asserted for one clock cycle, coincident with the last clock cycle where DMAACK is asserted. If the Chain Done bit is set, DMADONE* will be asserted when DMAACK is asserted for completion of the chained transfer. In that case, no more DMA transfer takes place until the channel is reprogrammed.

If the Chain Done bit is cleared, DMADONE* will be asserted when DMAACK is asserted at the end of each DMA transfer. If the Chain Enable bit is set, the DMAC will reload the channel’s registers from the address pointed to by the Chain Address Register (CHARn), so the channel will chain.

When DMADONE* is programmed as an input for a channel, the external device can assert it (low) to terminate the entire DMA transfer while DMAACK is asserted. Even if chaining is enabled, the chained transfers stop, regardless of the setting of the Chain Done bit in the Channel Control Register (CCRn.CHDN). It takes three clock cycles for the DMAC to inhibit further DMA operations after the assertion of DMADONE*. The ongoing DMA bus cycle, whether single or burst, is not discontinued.

In single-address transfer mode, each time the DMAC acquires bus mastership, exactly one bus cycle is executed. Thus, it is clear when the DMA transfer terminates. That is, the channel will stop once the current transfer completes, (provided the Count Register has reached 0 with the Chain Done bit cleared). If the Count Register has not decremented to 0, the DMAC regains bus mastership and executes the next single-address transfer.

In non-burst dual-address transfer mode, each time the DMAC is granted bus mastership, one read bus cycle and one write bus cycle are executed. In this mode of operation, when DMADONE* is asserted externally, the channel will stop after both the read and write bus cycles have been completed.

Dual-address burst transfers are not so straightforword because each time the DMAC assumes bus mastership, multiple read and write cycles may be executed. This could give rise to situations where the DMAC is forced to relinquish the bus while valid data still remains in the FIFO buffer. Once the Transfer Active (CCRn.XTACT) bit is cleared in response to the external DMADONE* signal, no more reads will occur, but as long as DMADONE* is asserted during a read bus cycle, the DMAC will properly take care of the corresponding write cycle before terminating the DMA transfer.

However, should the assertion of DMADONE* be delayed until a write cycle, the channel will stop immediately after the current write cycle. Therefore, a DMA transfer requiring multiple write bus cycles will end prematurely while partial data remains in the FIFO buffer. It must be noted that this data can be retrieved later only when it is left intact until the channel regains bus mastership. This is possible only when the FIFO buffer is dedicated for use by that channel. If this is the case, then the channel's registers can be examined via software to determine where to deliver the data remaining in the FIFO buffer.

When DMADONE* is programmed as bidirectional, it is configured as an open-drain output when that channel is active. Therefore, the DMADONE* requires an external pull-up resistor. In all other respects, the pin function is exactly the same as described above. Simultaneous assertion of DMADONE* by external logic and the TX3927 causes the External Done Asserted bit in the Channel Status Register (CSRn.EXTDN) to be set.

10.4.5 Restrictions on Non-standard Increment Values

The TX3927 DMAC provides a great flexibility in defining address increment values to support various kinds of DMA transfers. However, burst DMA transfers have restrictions. When the transfer size is programmed to 8, 16 or 32 bits (or non-burst transfers), the source and destination address increments have no particular restrictions except alignment requirements. For a transfer size of 8 bits, there is no restriction on increment values. For a transfer size of 16 bits, address increment values must be even (i.e., the least-significant bit must be 0). For a transfer size of 32 bits, address increment value must be an integer multiple of four (i.e., the two least-significant bits must be 0). After each DMA transfer, the increment values programmed in the SAIn and DAIn registers are sign-extended and added to the current source and destination addresses respectively. For example, it is possible to read every third word in one memory and write them to every fifth word location in another memory.

The situation is more complicated for burst transfers. For single-address burst operations, the DMAC calculates the address using the value of the Source Address Increment Register for each transfer of a word and places the address on the internal bus. However, the on-chip ROM and SDRAM Controllers automatically increment addresses internally to improve the burst performance. Because the ROM Controller can handle addresses in increments of four during bursts, single-address burst DMA transfers can not be executed properly, if the source address increment is not four.

The SDRAM Controller supports unsigned increment values from 0 to 255 words (0 to 1020 bytes). Therefore, high-speed burst capability of SDRAM devices is available only for a sequence of addresses with an equal row address. This means most address increment values possible with the DMAC are compatible only with non-burst SDRAM operations.

10.4.6 Restrictions on Dual-Address Burst Transfers

Dual-address mode has more restrictions than single-address mode for using burst transfers. Generally in dual-address mode, burst operations must occur with an address increment of four (even when bursting to SDRAM). If the source or destination address increment is not four, single transfers need to be used.

Another restriction on using burst transfers in dual-address mode is that the address must come to be aligned on the burst size at least once as it is incremented. This is not required in single-address mode. This restriction is part of the conditions required for the DMAC to successfully terminate burst transfers. Here too, using the standard increment value (4 bytes) causes no problem.

The TX3927 DMAC contains separate burst inhibit bits for source and destination addresses per channel. If the source or destination burst inhibit bit is set, the DMAC performs single transfers during source read or destination write cycles, respectively, even if 4- or 8-word burst transfers are specified with proper address alignment and increments. This helps to avoid violating the above restrictions when the address increment is not the standard 4 bytes.

When the DMA Channel is programmed to perform dual-address transfers with 4-word bursts, a DMA request (DMAREQ) will never cause more than four words to be read at a time, whereas situations in which five or more words are written from the internal FIFO buffer to the destination exist. This is because five or more words may be buffered in the FIFO during the course of a DMA transfer due to differing alignments between the source and destination addresses. In response to the last DMAREQ request, the DMAC reads the last one to four words into the FIFO and then writes the entire contents of the FIFO (one to seven words) to the destination. The FIFO is eight-word deep; due to programmability limitation of the DMAC, the maximum burst size in dual-address mode is eight words.

10.4.7 DMA Transfers with On-Chip I/O Peripherals

DMA transfers to and from on-chip peripherals can only occur in dual-address mode. Single-address mode is not supported.

When the TX3927 is acting as a PCI initiator, DMA transfers with memory devices on the PCI bus take place via the PCI Controller (PCIC). The initiator interface must be programmed with dual-address mode. When the TX3927 is a PCI target, the PCIC takes over DMA transfers without using the DMAC.

When the TX3927 is the PCI initiator, the DMAC cannot perform burst transfers to the PCIC. Therefore, bits 26 (DBINH) and 25 (SBINH) of the Channel Control Register (CCRn) must be set to 1 to inhibit bursts. For example, when the TX3927 transfers data between SDRAM and a device on the PCI bus as an initiator, burst transfers can occur from the source SDRAM, but not to the destination device; thus the Channel Control Register must be programmed as CCRn.DBINH=1 and CCRn.SBINH=0. Because burst transfers use the FIFO in the DMAC, the FIFO must be enabled for the relevant channel by setting its FIFUM bit of the Master Control Register (MCR).

The TX3927 supports the use of SIO ports as DMA requesters. For details, refer to "13.4.19 DMA Transfer Mode."

10.4.8 Timing for an External DMA Request (DMAREQ)

When the off-chip peripheral requires DMA service, it asserts the external DMA request signal (DMAREQ) for a channel. The DMAREQ signal is internally cycled based on GBUSCLK (which runs at half CPU operating frequency).

External DMA request pins can be configured for either edge or level sensitivity. If programmed as edge-triggered, once the DMAC channel starts a DMA transfer, the next DMA request can be recognized (i.e., one clock cycle before DMAACK is asserted). If programmed as level-sensitive, the DMA channel recognizes the next DMA request two clock cycles before DMAACK is asserted. (Refer to "10.5 Timing Diagrams.")

The request on the level-sensitive DMAREQ pin must remain asserted until the DMA transfer starts.

10.4.9 Configuration Errors

A configuration error results when a DMA channel is activated by bit 8 (Transfer Active) of the Channel Control Register (CCRn) in the following circumstances:

(1) The two least-significant bits of the Chained Address Register (CHARn) are non-zero.

(2) The XFSZ field of the CCRn register specifies a transfer size of 8 bits (000) in single-address mode.

(3) The XFSZ field of the CCRn register specifies a transfer size of 16 or 32 words (11x) in dual-address mode.

(4) The XFSZ field of the CCRn register contains a reserved value (011).

(5) Either of the following conditions is detected when byte order reversing is enabled by bit 23 of the CCRn register:
   1) Single-address mode is selected.
   2) Dual-address mode is selected, with the XFSZ field of the CCRn register programmed for a transfer size of 8 or 16 bits (00x).

(6) Any of the following conditions is detected when the transfer size is 16 bits:
   1) The least-significant bit of the Source Address Register (SARn) is 1.
   2) The least-significant bit of the Source Address Increment Register (SAIn) is 1.
   3) The least-significant bit of the Destination Address Register (DARn) is 1 in dual-address mode.
   4) The least-significant bit of the Destination Address Increment Register (DAIn) is 1 in dual-address mode.

(7) The least-significant bit of the Count Register (CNTRn) is 1 when the transfer size is 16 bits.

(8) Any of the following conditions is detected when the transfer size is 32 bits:
   1) The two least-significant bits of the Source Address Register (SARn) are non-zero.
   2) The two least-significant bits of the Source Address Increment Register (SAIn) are non-zero.
   3) The two least-significant bits of the Destination Address Register (DARn) are non-zero in dual-address mode.
   4) The two least-significant bits of the Destination Address Increment Register (DAIn) non-zero in dual-address mode.

(9) The two least-significant bits of the Count Register (CNTRn) non-zero when the transfer size is 32 bits.

10.4.10 Notes on Using the DMAC FIFO

When the data transfer size is programmed to more than one word, the DMAC attempts to use burst transfers to improve data throughput. For burst transfers to occur, addresses must be aligned on a boundary equal to the data transfer size. If this condition is not satisfied, the DMAC only transfers data up to the boundary in the first DMA transaction. Thus, the number of transferred words will be less than the programmed transfer size.

If the source and destination addresses have disparate word alignments, partial data is temporarily buffered in the FIFO. Because the FIFO can be shared by multiple channels, the data in the FIFO may be overwritten. To prevent this situation, the source and destination addresses must be properly aligned to the same-sized boundary.

Example 1: DMA operations where one word of data is left in the FIFO temporarily

Program settings:

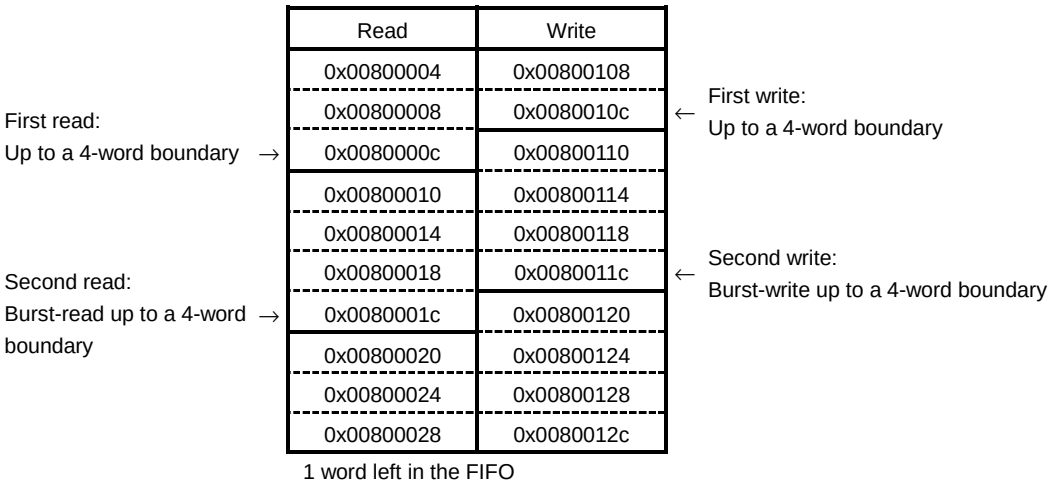
- 4-word transfer size
- SAR: 0x00800004
- DAR: 0x00800108

Operation:

(1) First DMA transfer
   a. Because the source address 0x00800004 is not aligned to a 4-word boundary, the DMAC performs single reads for the first three words, until a 4-word boundary is reached (immediately before 0x00800010).
   b. Because the destination address 0x00800108 is not aligned to a 4-word boundary, the DMAC performs single writes for the first two words; i.e., until a 4-word boundary is reached (immediately before 0x00800110). Consequently, one word of data is left in the FIFO.

(2) Second DMA transfer
   a. Because 0x00800010 is aligned to a 4-word boundary, the DMAC performs a burst read for the next four words.
   b. Because 0x00800110 is aligned to a 4-word boundary, the DMAC performs a 4 words burst write.

(3) Third and subsequent DMA transfers
   Same as (2).



Example 2: DMA operation where three words of data is left in the FIFO temporarily

Program settings:

- 4-word transfer size
- SAR: 0x00800008
- DAR: 0x00800104

Operation:

(1) First DMA transfer

a. Because the source address 0x00800008 is not aligned to a 4-word boundary, the DMAC performs single reads for the first words; i.e., until a 4-word boundary is reached (immediately before 0x00800010).

b. Because the destination address 0x00800104 is not aligned to a 4-word boundary, the DMAC attempts to perform single writes for three words up to a 4-word boundary (immediately before 0x00800110). However, because only two words of data are present in the FIFO, the DMAC writes two words and stops short of a 4-word boundary. Therefore, no data remains in the FIFO.

(2) Second DMA transfer

a. Because 0x00800010 is aligned to a 4-word boundary, the DMAC performs a burst read for the next four words.

b. Because 0x0080010c is not aligned to a 4-word boundary, the DMAC performs a single write for one word until a 4-word boundary is reached (immediately before 0x00800110). Consequently, three words of data is left in the FIFO.

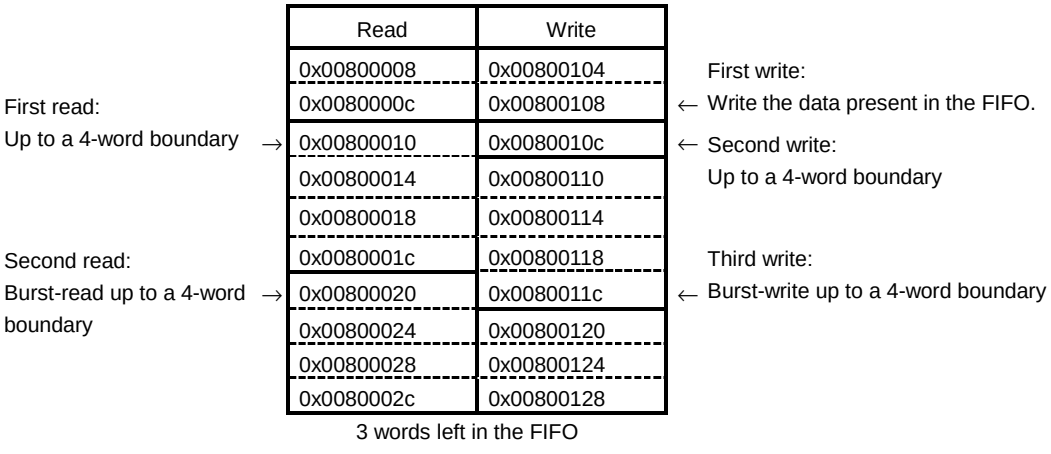
(3) Third DMA transfer

a. Because 0x00800020 is aligned to a 4-word boundary, the DMAC performs a burst read for the next four words.

b. Because 0x00800110 is aligned to a 4-word boundary, the DMAC performs a burst write for the four words.

(4) Third and subsequent DMA transfers

Same as (3).



10.5 Timing Diagrams

The following diagrams assume that both the DMAREQ and DMAACK signals are programmed as active-low.

10.5.1 Single-Address Mode, 32-bit Read Operation (ROM)

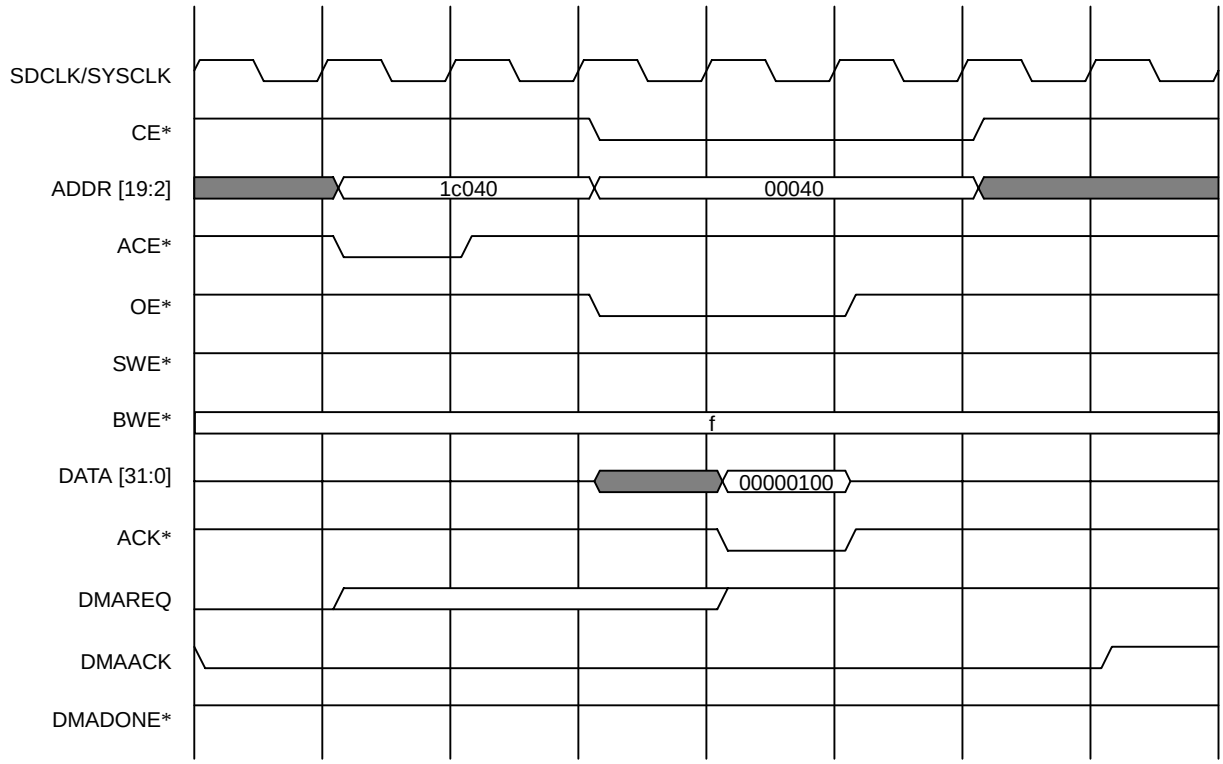


Figure 10.5.1 Single-Address Mode Single-Read Timing (Reading 32-bit Data from a 32-bit ROM)

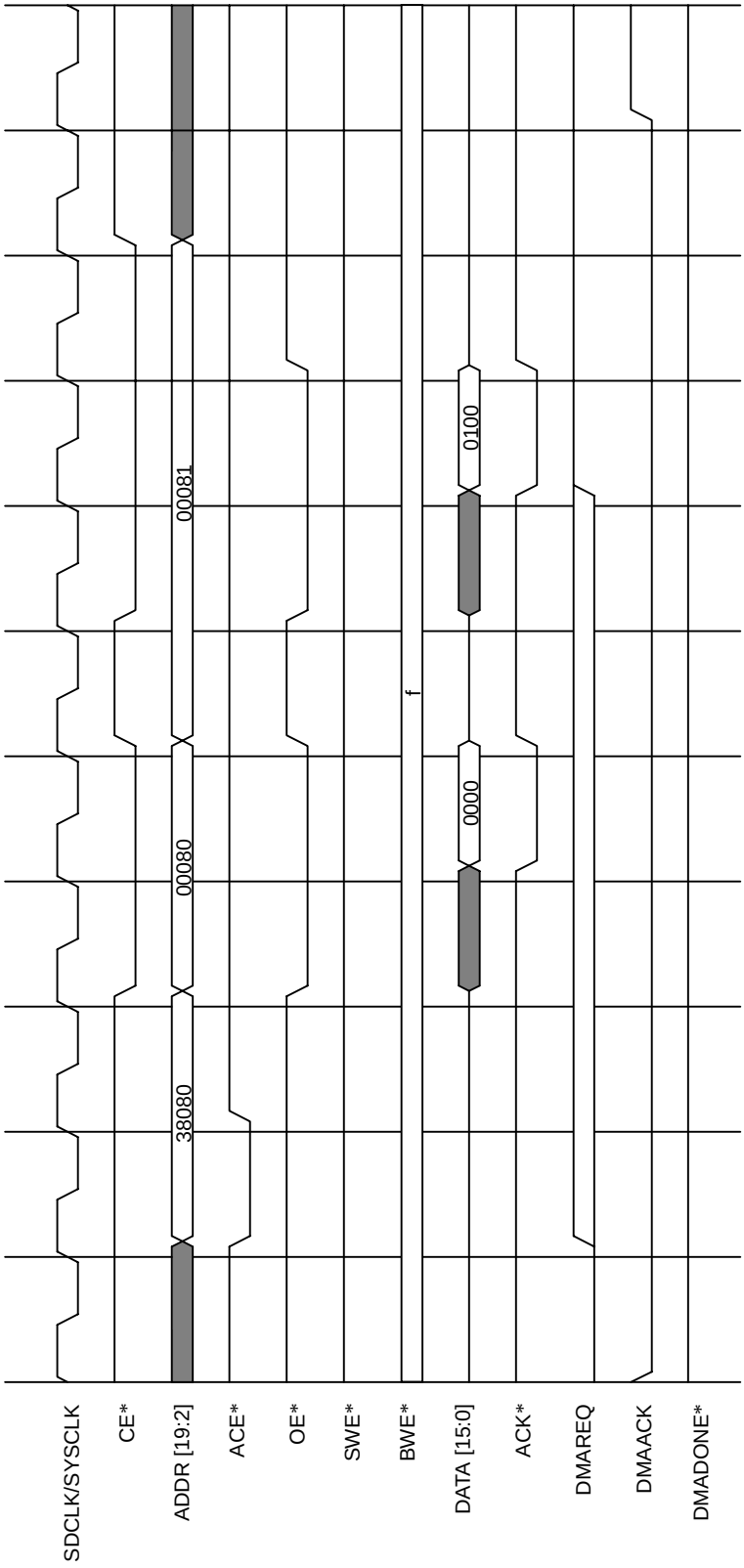


Figure 10.5.2 Single-Address Mode Single-Read Timing (Reading 32-bit Data from a 16-bit ROM)

10.5.2 Single-Address Mode, 32-bit Write Operation (SRAM)

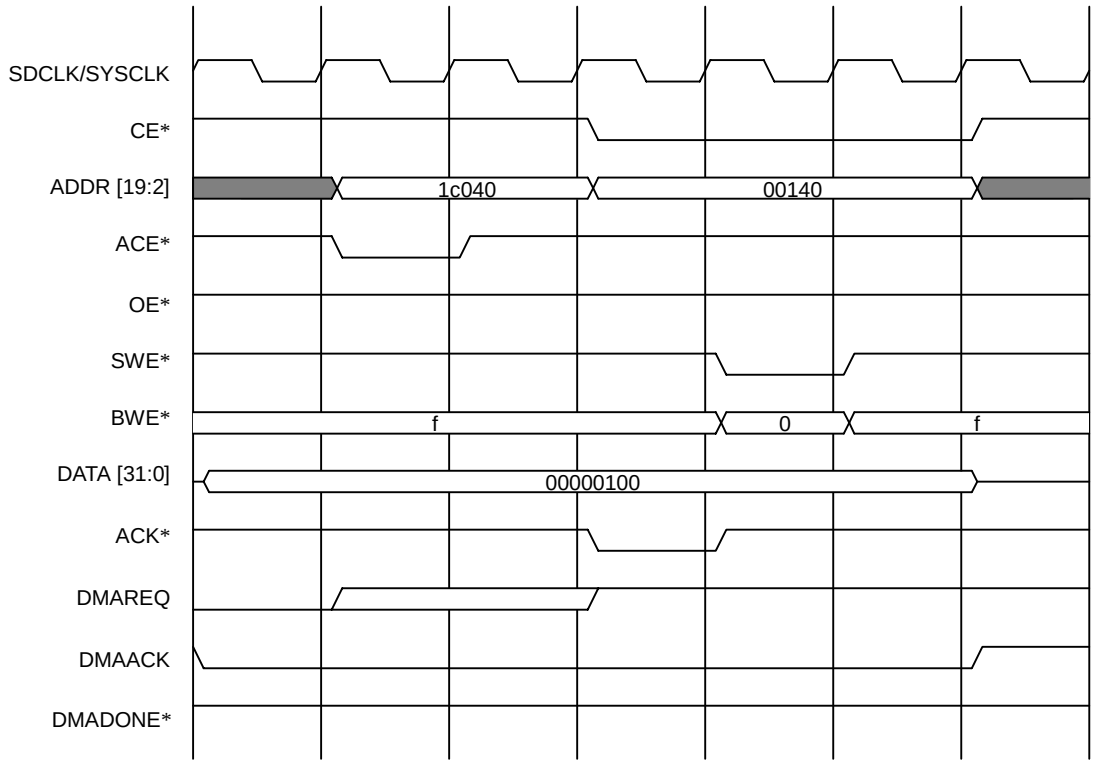


Figure 10.5.3 Single-address Mode Single-Write Timing (Writing 32-bit Data to a 32-bit SRAM)

10.5.3 Single-Address Mode, 32-bit Burst-Read Operation (ROM)

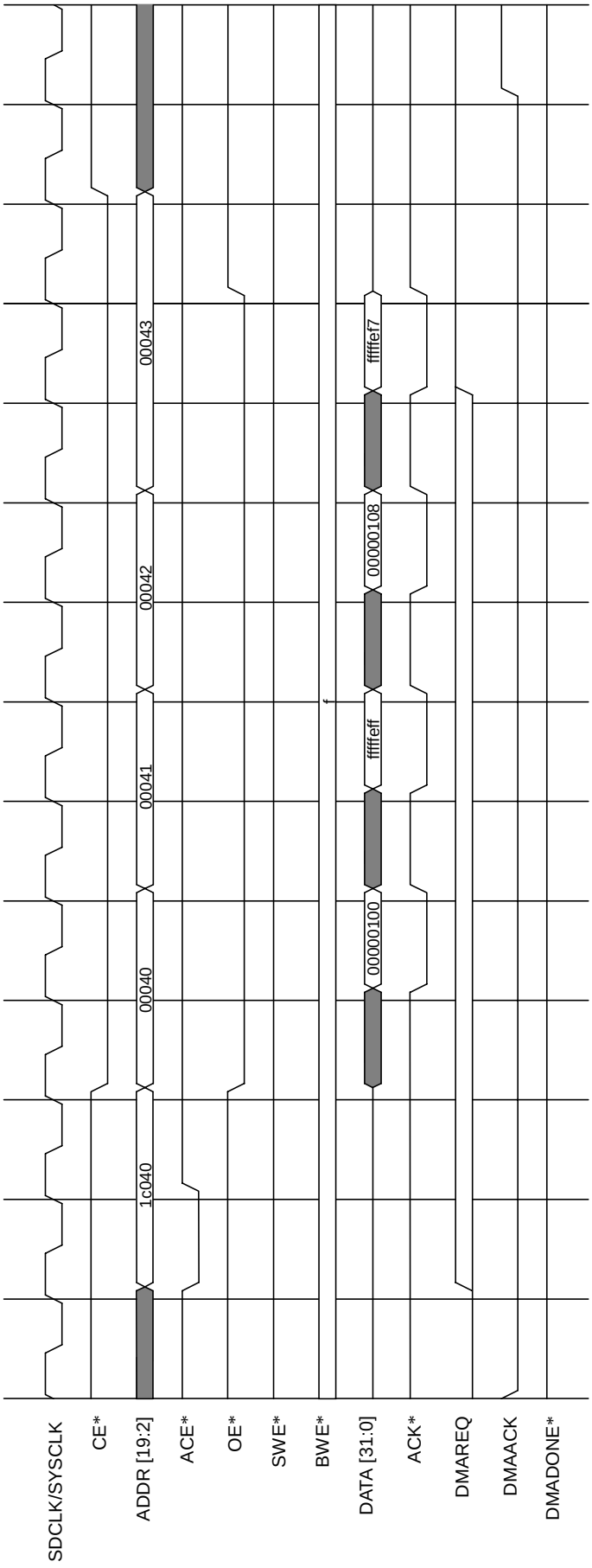


Figure 10.5.4 Single-Address Mode Burst-Read Timing (Burst-Read from a 32-bit ROM)

10.5.4 Single-Address Mode, 32-bit Burst-Write Operation (SRAM)

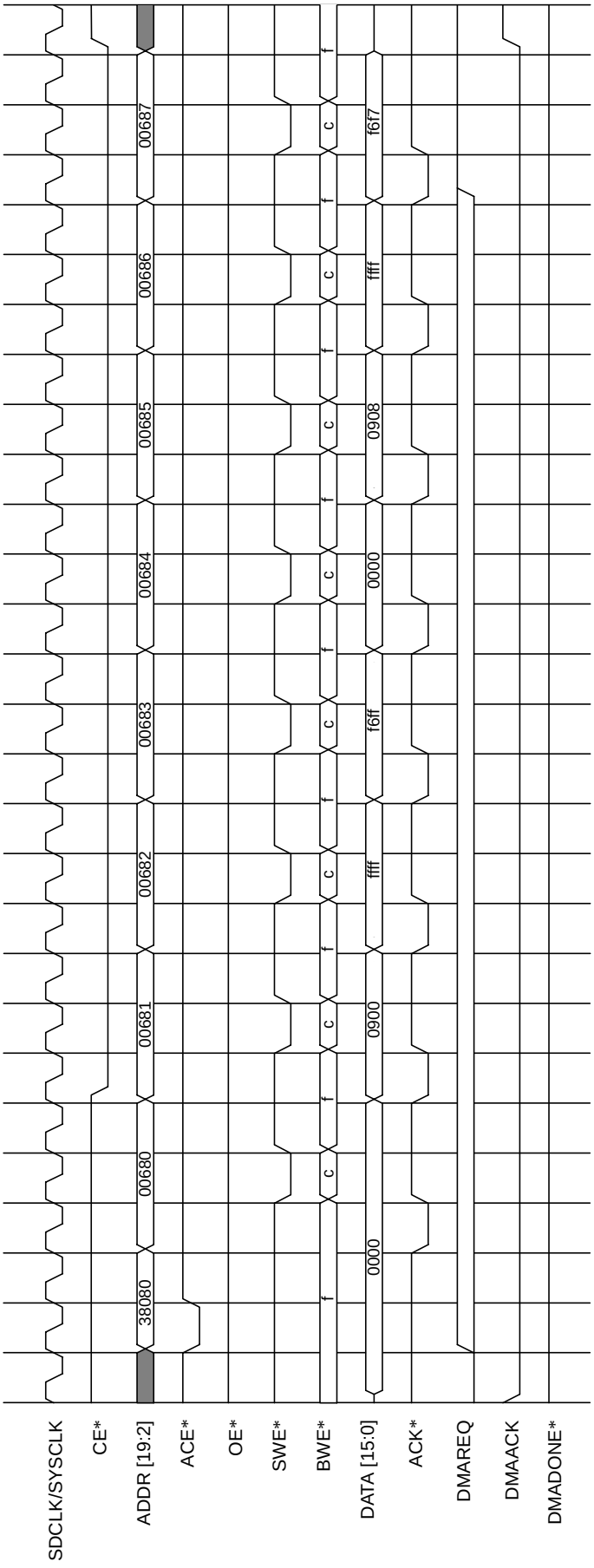


Figure 10.5.5 Single-Address Mode Burst-Write Timing (Burst-Write to a 16-bit SRAM)

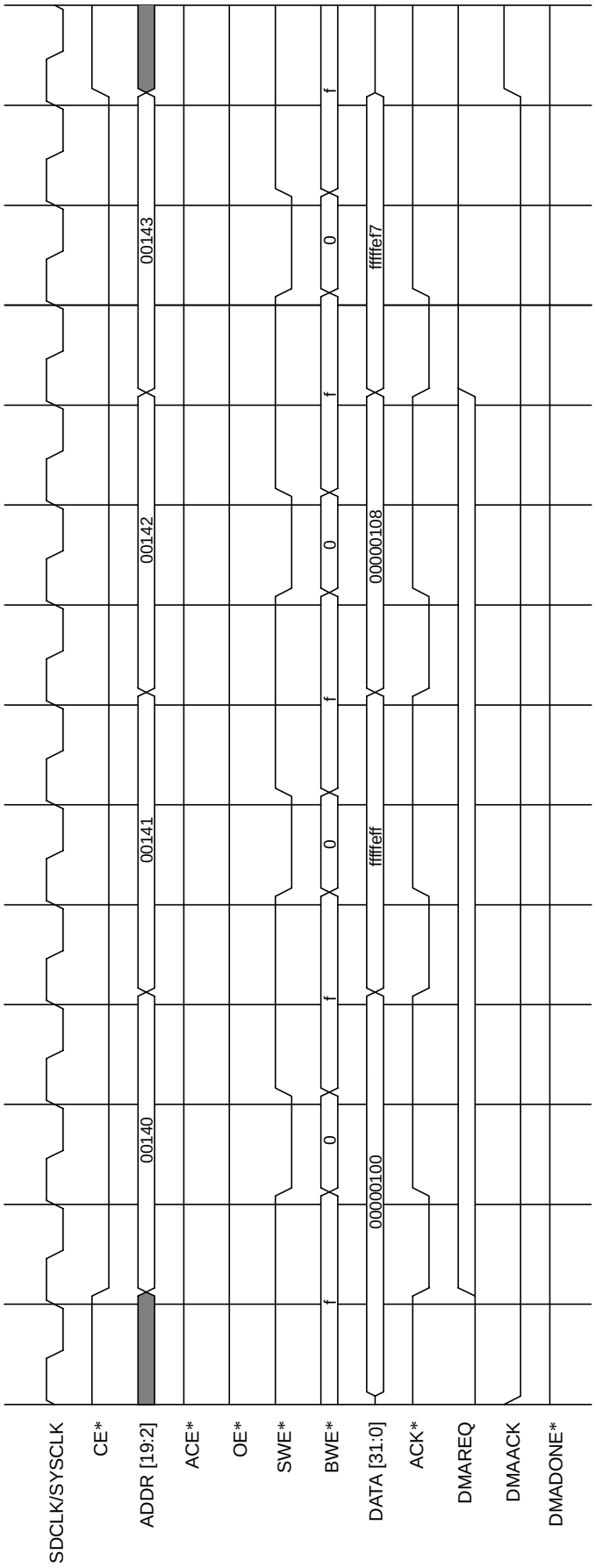


Figure 10.5.6 Single-Address Mode Burst-Write Timing (Writing 32-bit Data to a 32-bit SRAM)

10.5.5 Single-Address Mode, 16-bit Read Operation (ROM)

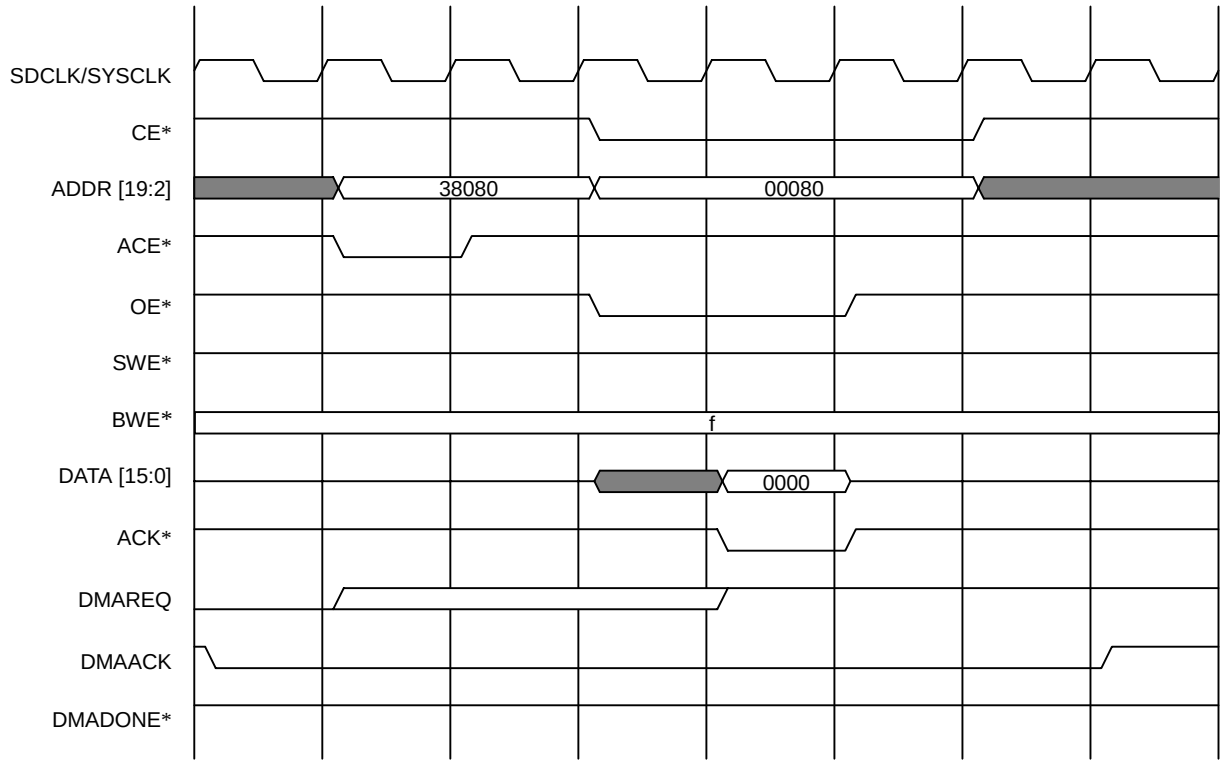


Figure 10.5.7 Single-Address Mode Single-Read Timing (Reading 16-bit Data from a 16-bit ROM)

10.5.6 Single-Address Mode, 16-bit Write Operation (SRAM)

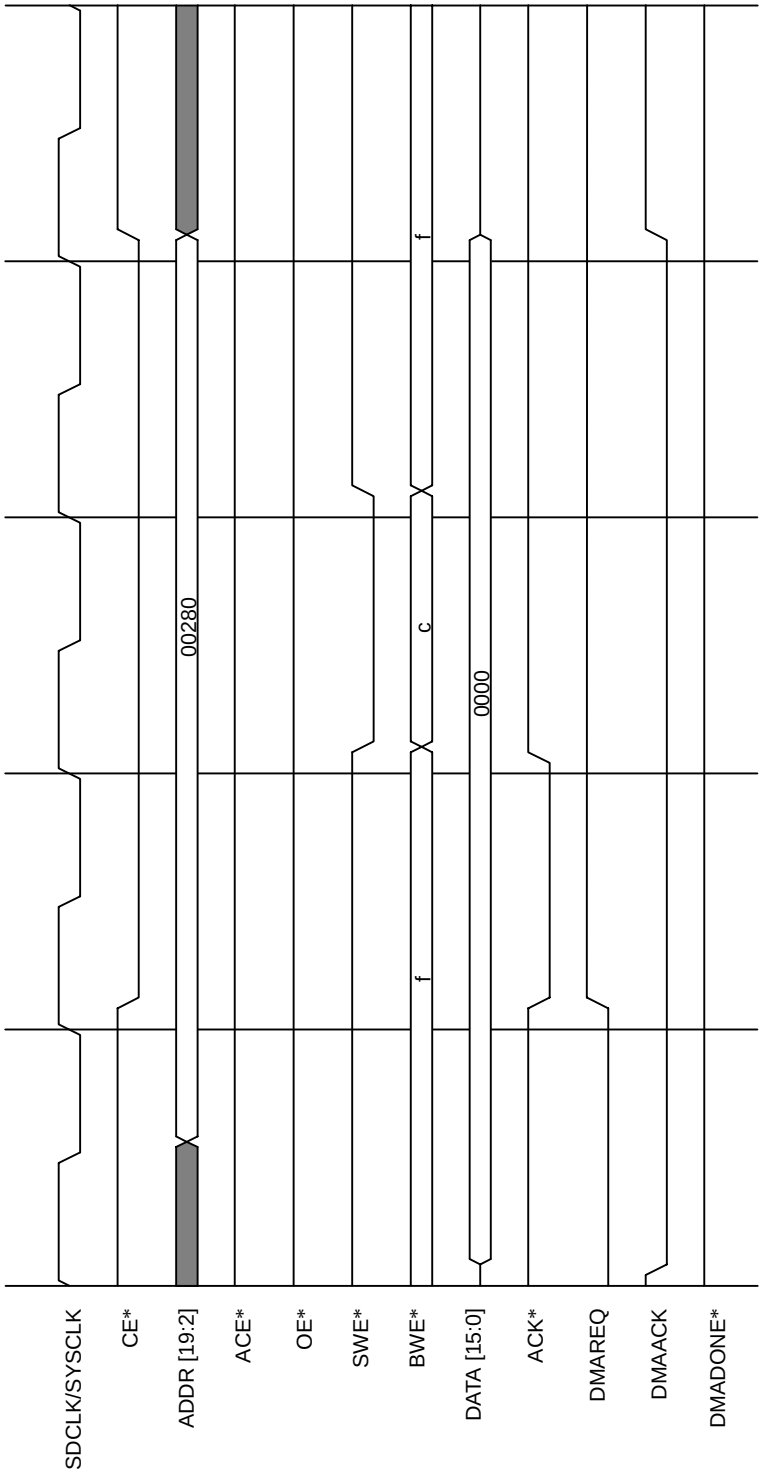


Figure 10.5.8 Single-Address Mode Single-Write Timing (Writing 16-bit Data to a 16-bit SRAM)

10.5.7 Single-Address, Half-speed Mode, 32-bit Read Operation (ROM)

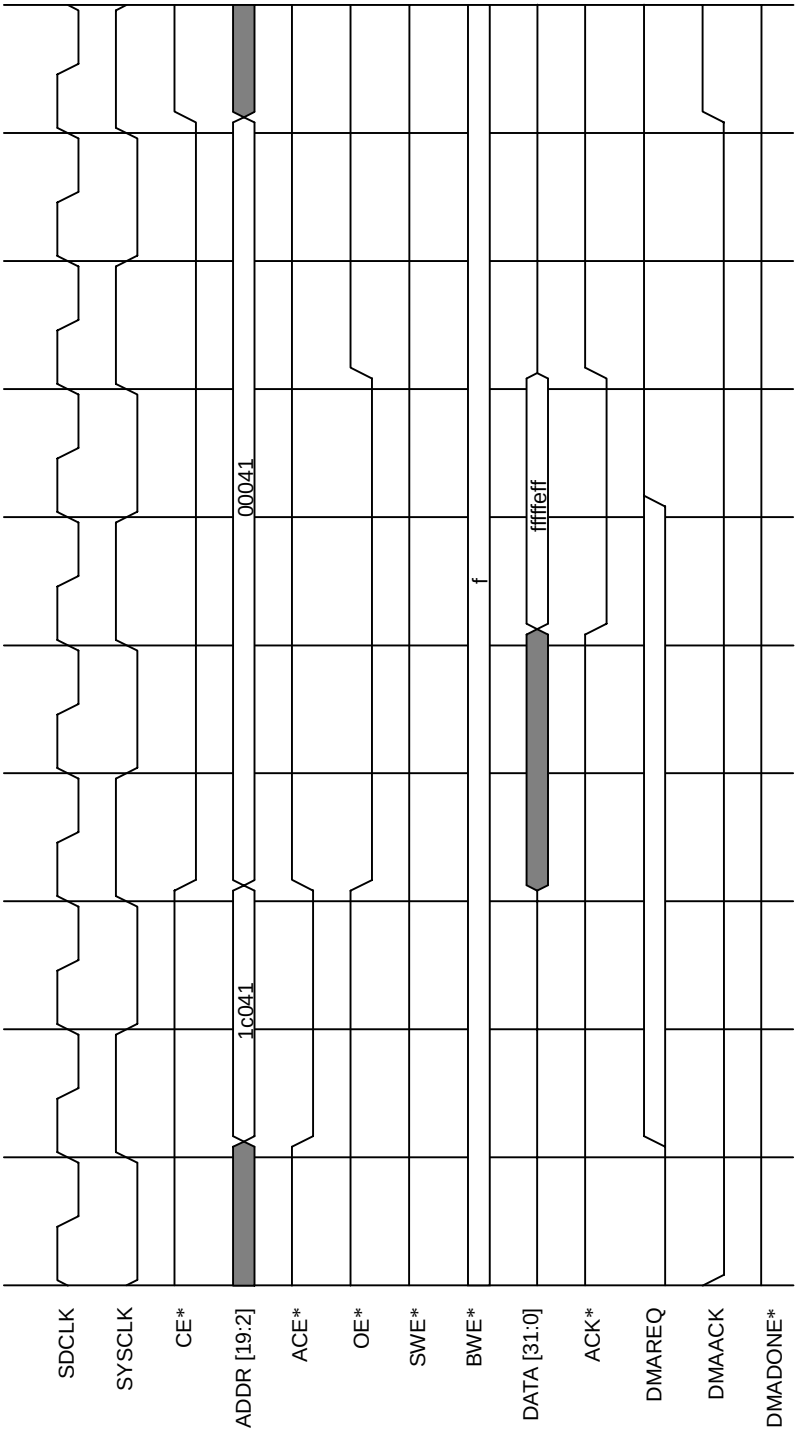


Figure 10.5.9 Single-Address Mode Single-Read Timing
(Reading 32-bit Data from a 32-bit Half-speed ROM)

10.5.8 Single-Address, Half-speed Mode, 32-bit Write Operation (SRAM)

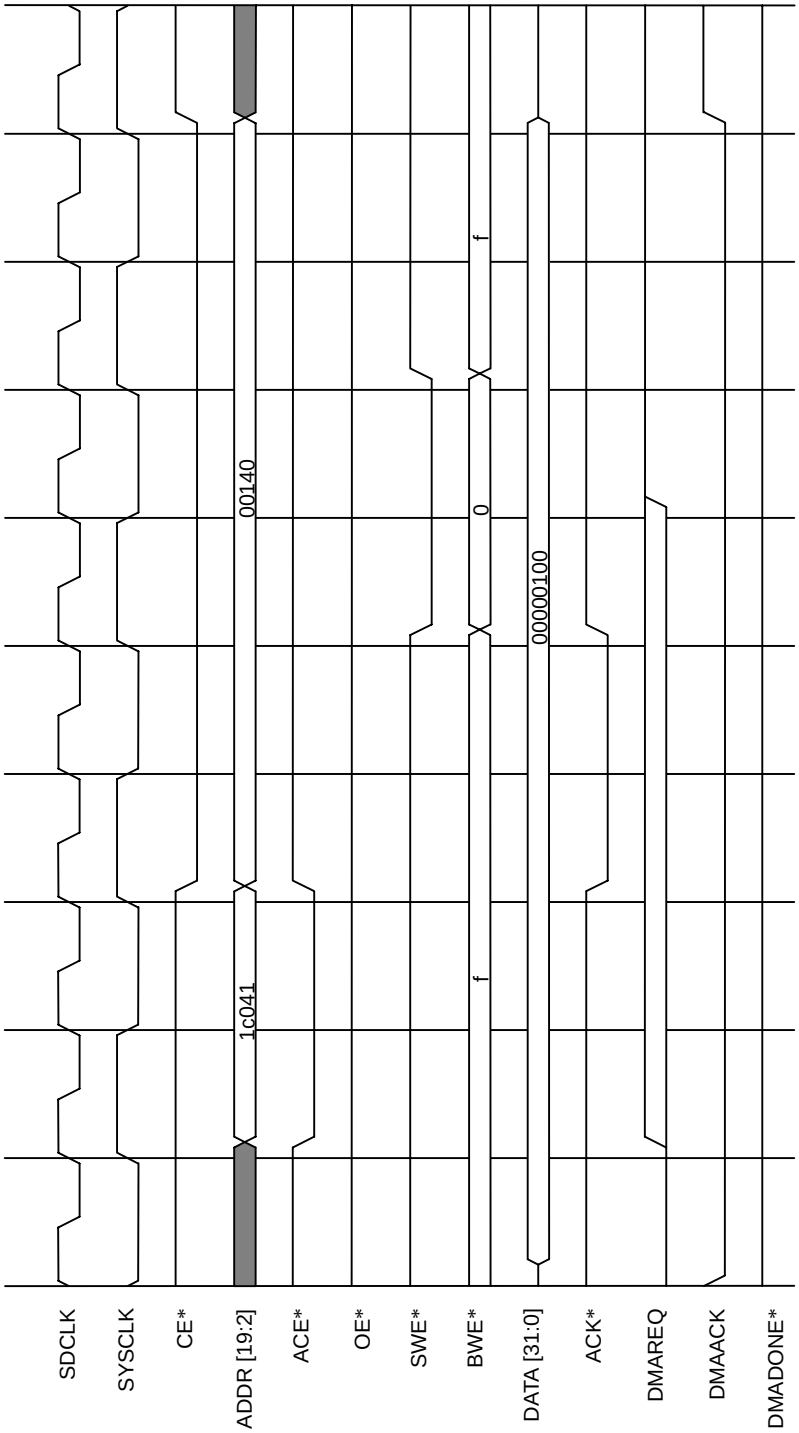


Figure 10.5.10 Single-address Mode Single-Write Timing (Writing 32-bit Data to a 32-bit Half-speed SRAM)

10.5.9 Single-Address Mode, 32-bit Read Operation (SDRAM)

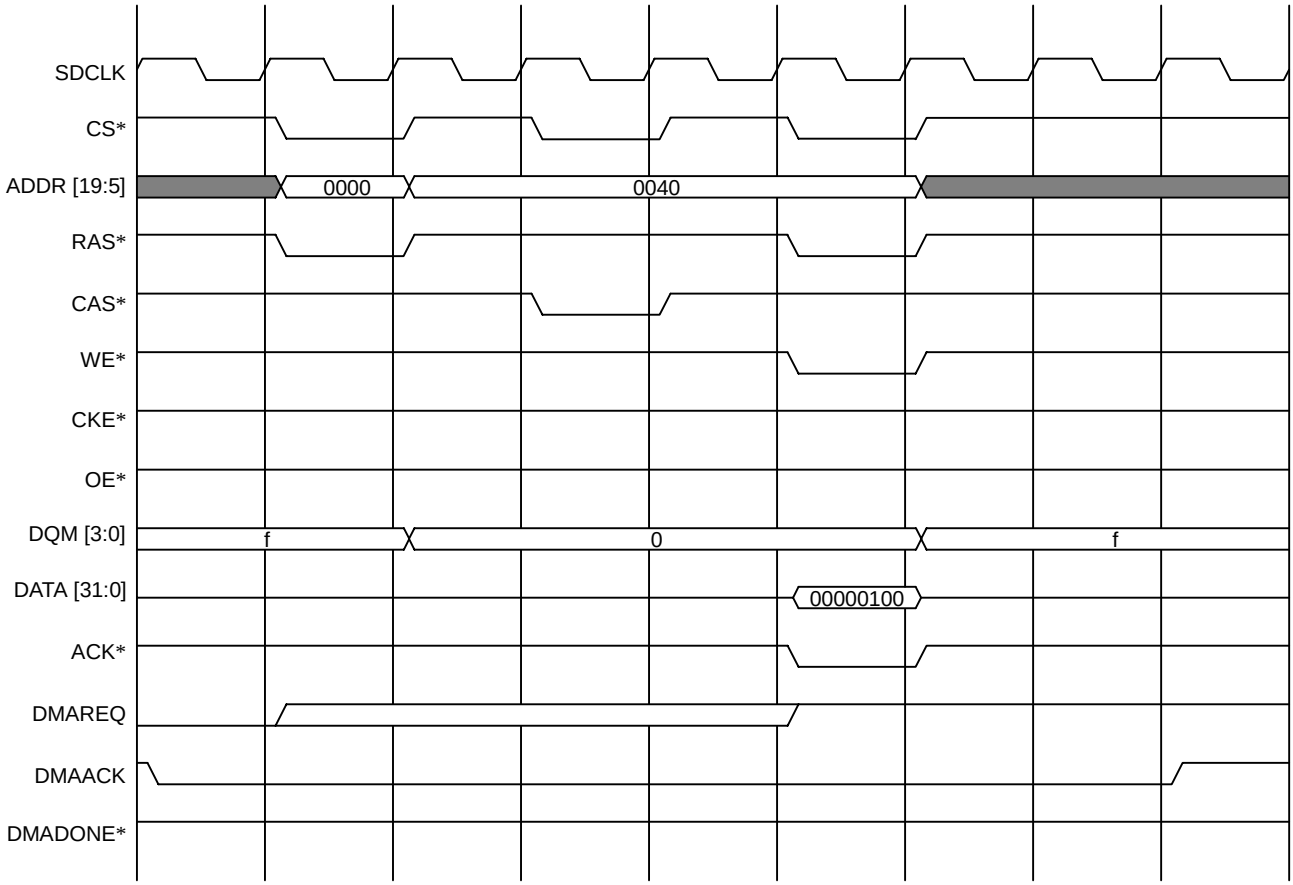


Figure 10.5.11 Single-address Mode Single-Read Timing (Reading 32-bit Data from a 32-bit SDRAM)

10.5.10 Single-Address Mode, 32-bit Write Operation (SDRAM)

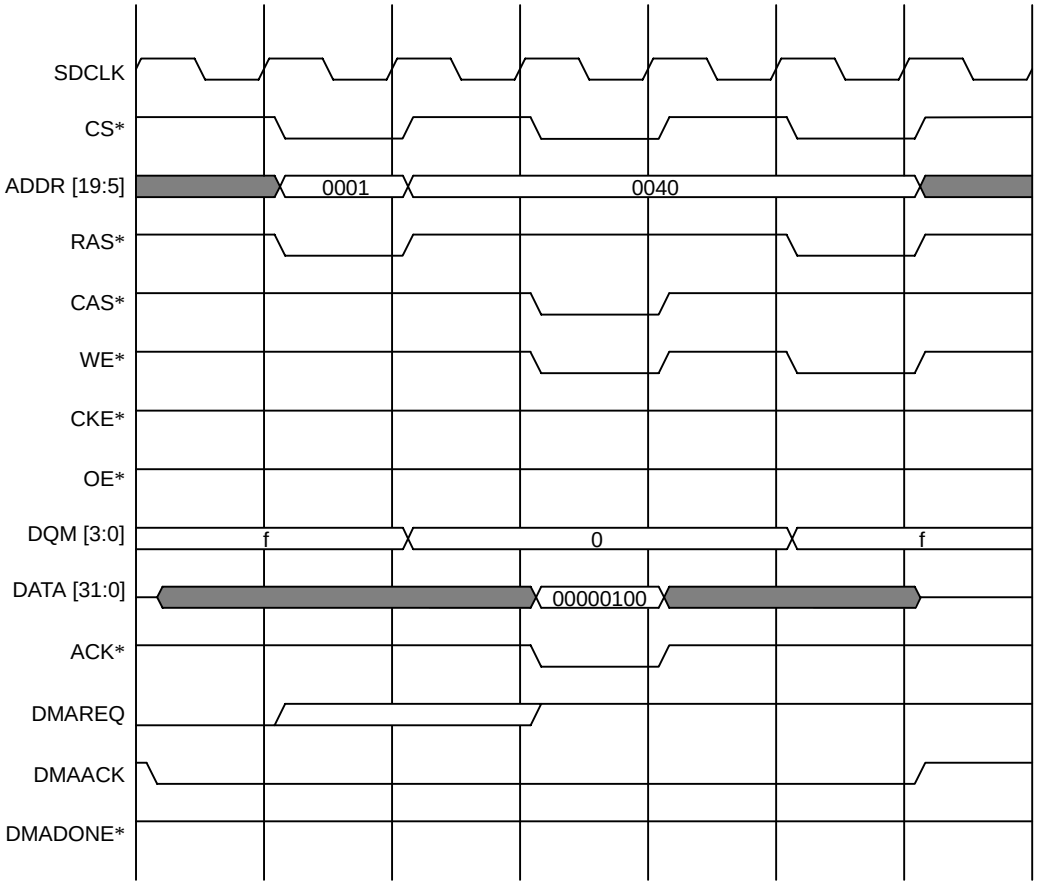


Figure 10.5.12 Single-Address Mode Single-Write Timing (Writing 32-bit Data to a 32-bit SDRAM)

10.5.11 Single-Address Mode, 16-bit Burst-Read Operation (SDRAM)

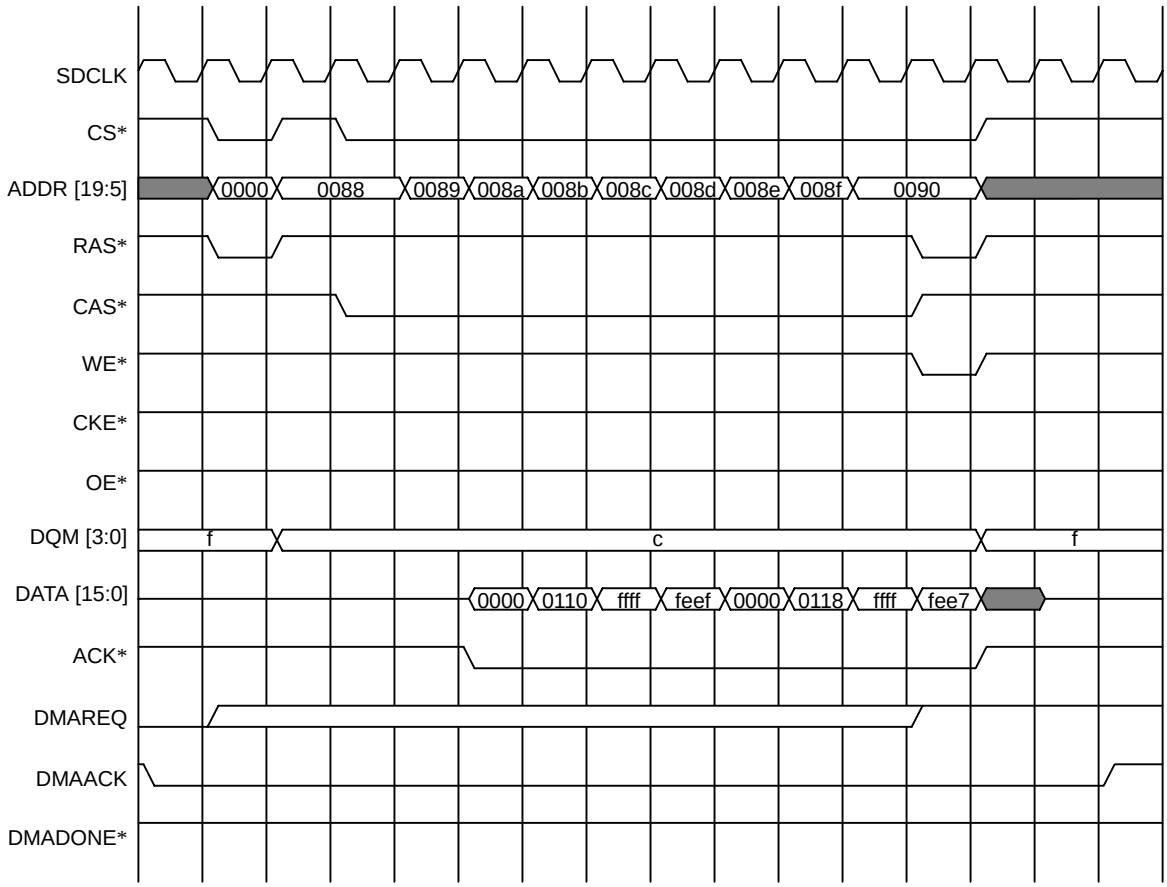


Figure 10.5.13 Single-Address Mode Burst-Read Timing (Burst-Read from a 16-bit SDRAM)

10.5.12 Single-Address Mode, 32-bit Burst-Read Operation (SDRAM)

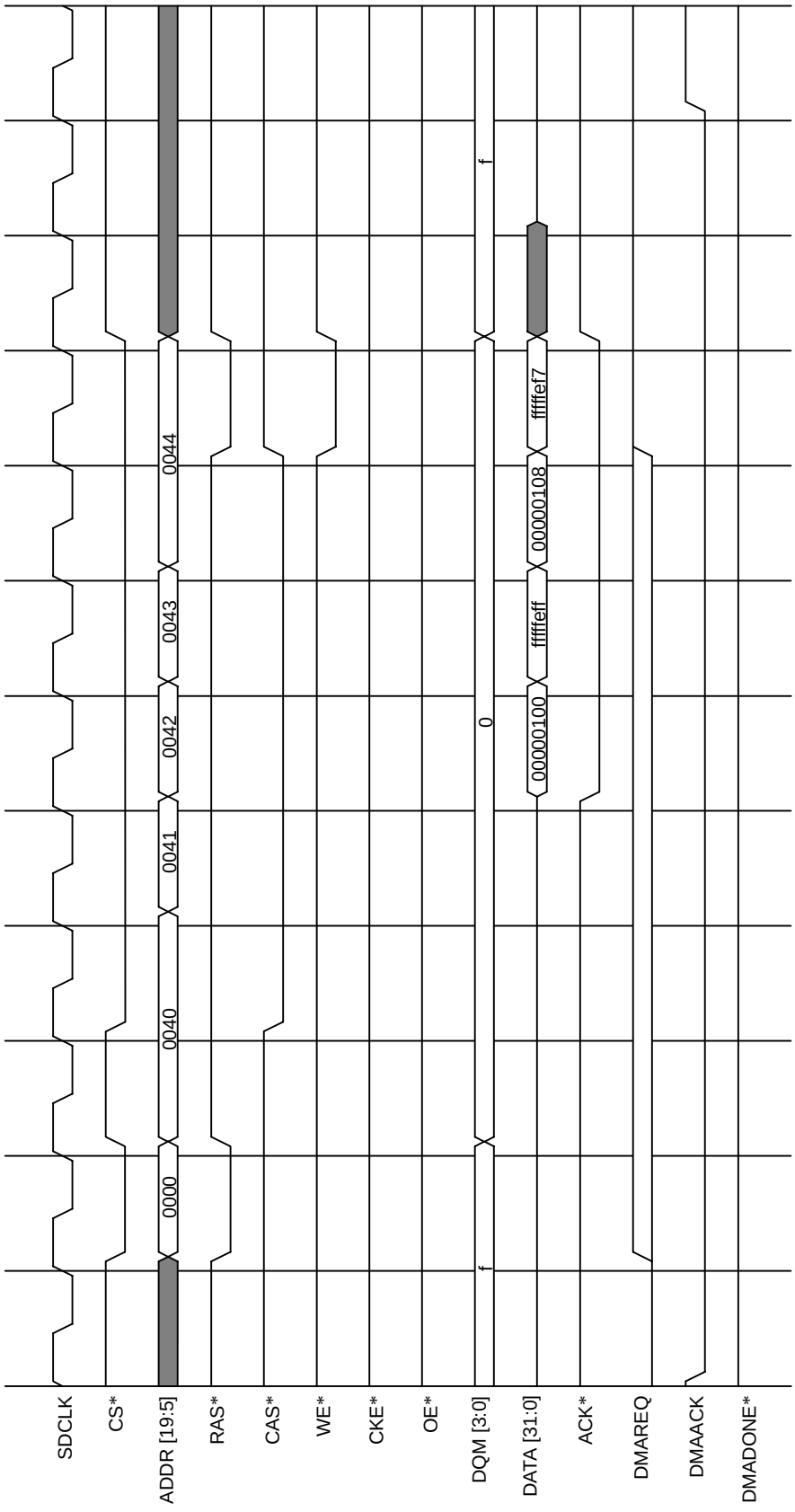


Figure 10.5.14 Single-Address Mode Burst-Read Timing (Burst-Read from a 32-bit SDRAM)

10.5.13 Single-Address Mode, 32-bit Burst-Write Operation (SDRAM)

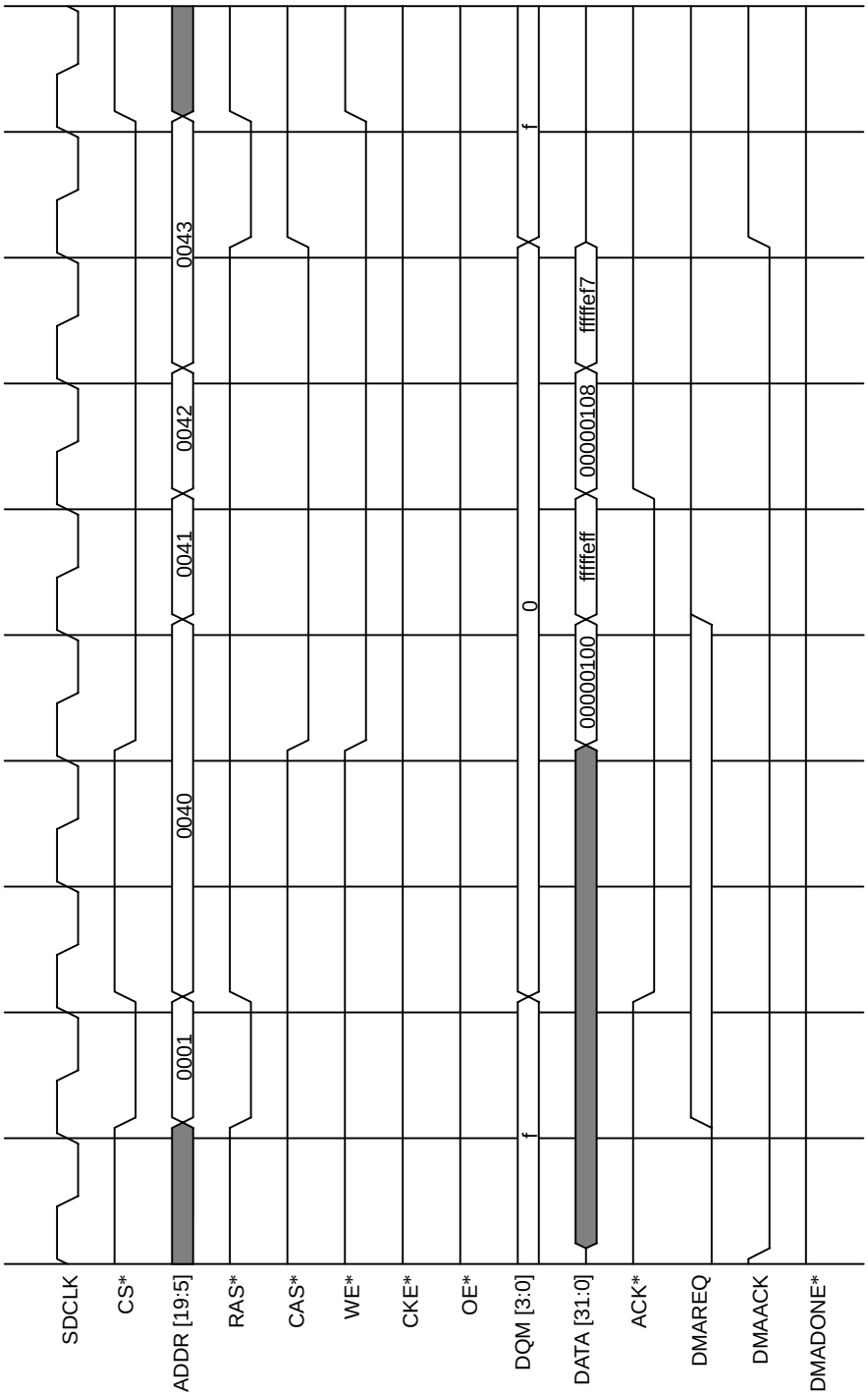


Figure 10.5.15 Single-address Mode Burst-Write Timing (Burst-Write to a 32-bit SDRAM)

10.5.14 Single-Address Mode, 32-bit Last Single-Read Operation (SDRAM)

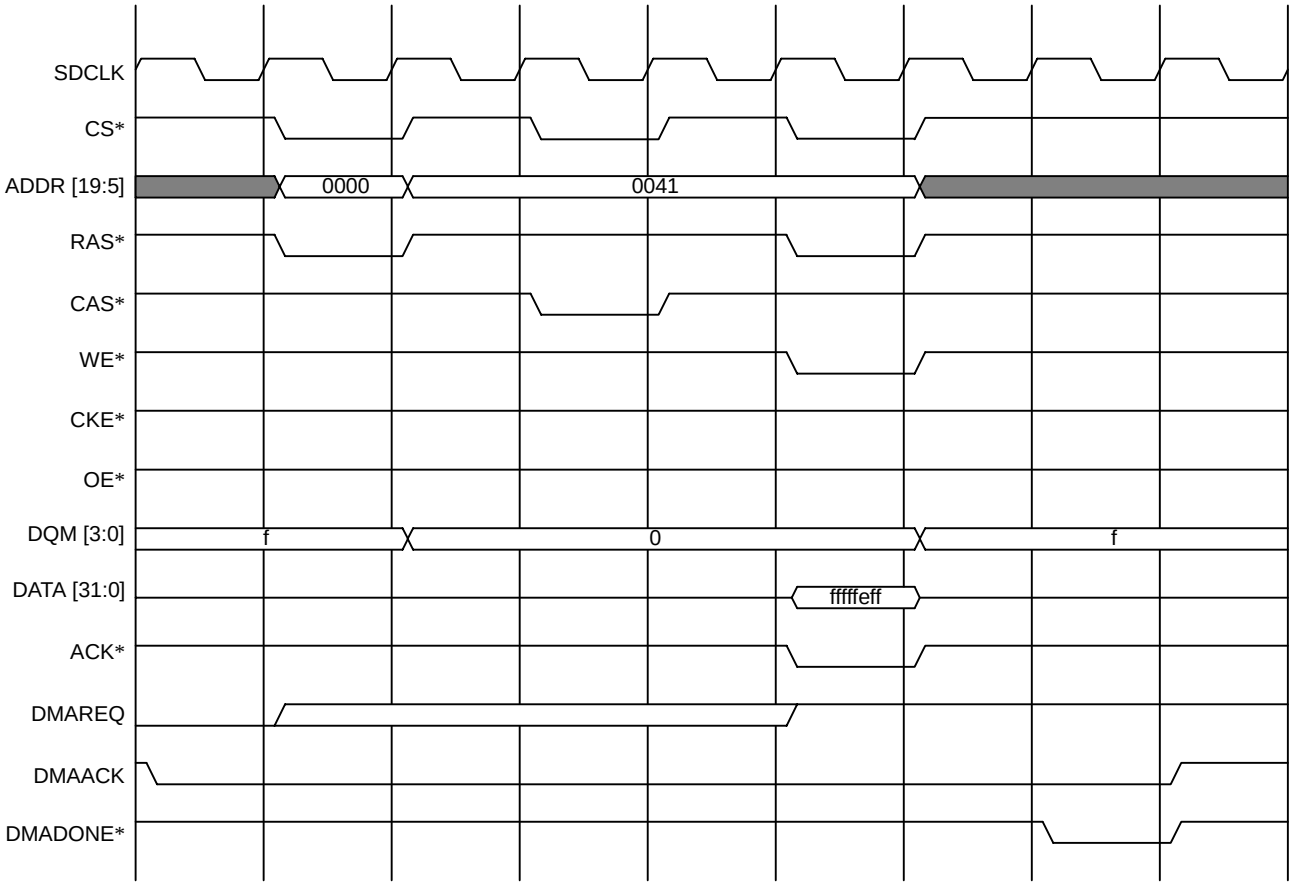


Figure 10.5.16 Single-Address Mode Single-Read Timing (Reading 32-bit Data from a 32-bit SDRAM)

10.5.15 Single-Address Mode, 16-bit Read Operation (SDRAM)

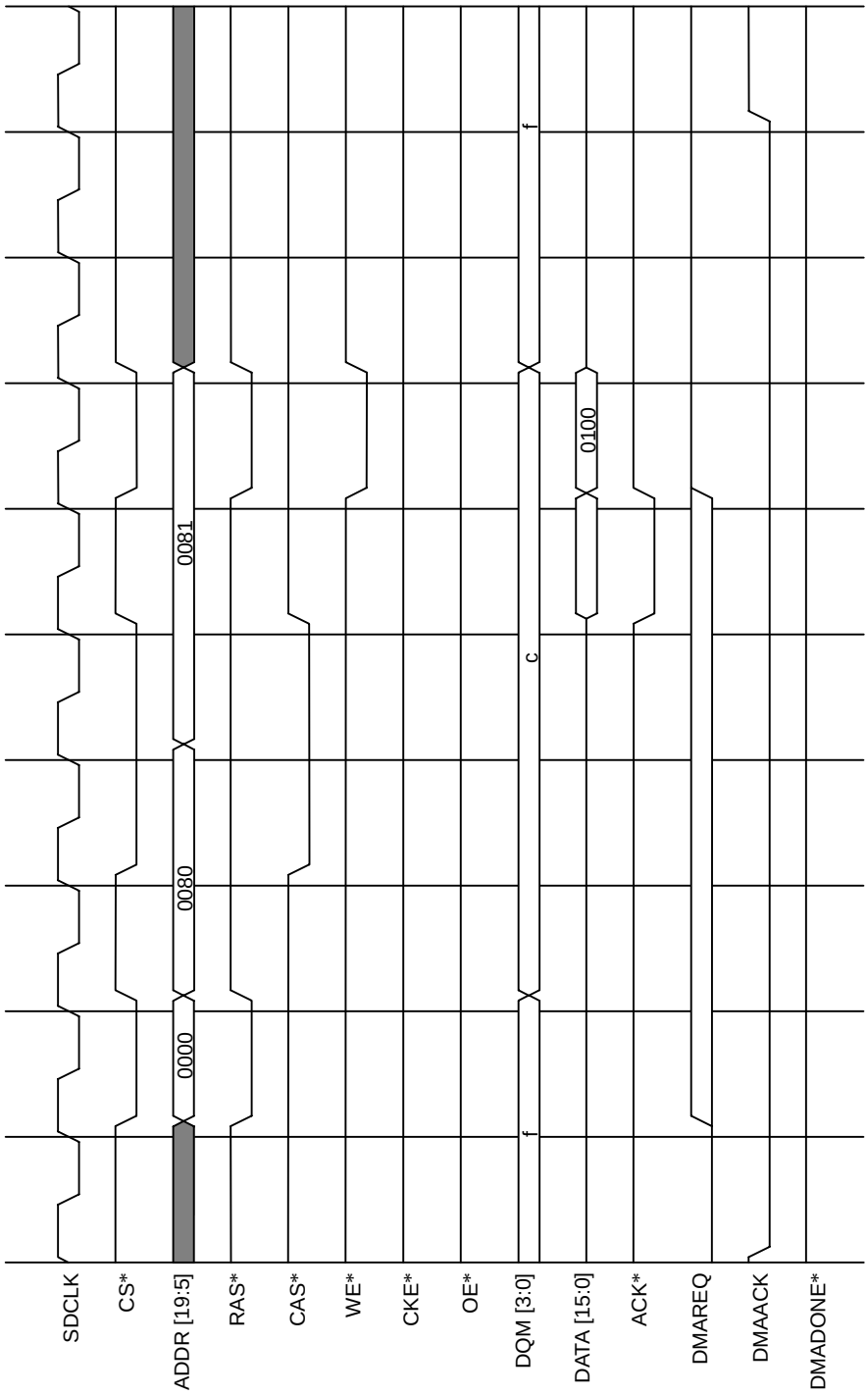


Figure 10.5.17 Single-Address Mode Single-Read Timing (Reading 16-bit Data from a 16-bit SDRAM)

10.5.16 Single-Address Mode, 16-bit Write Operation (SDRAM)

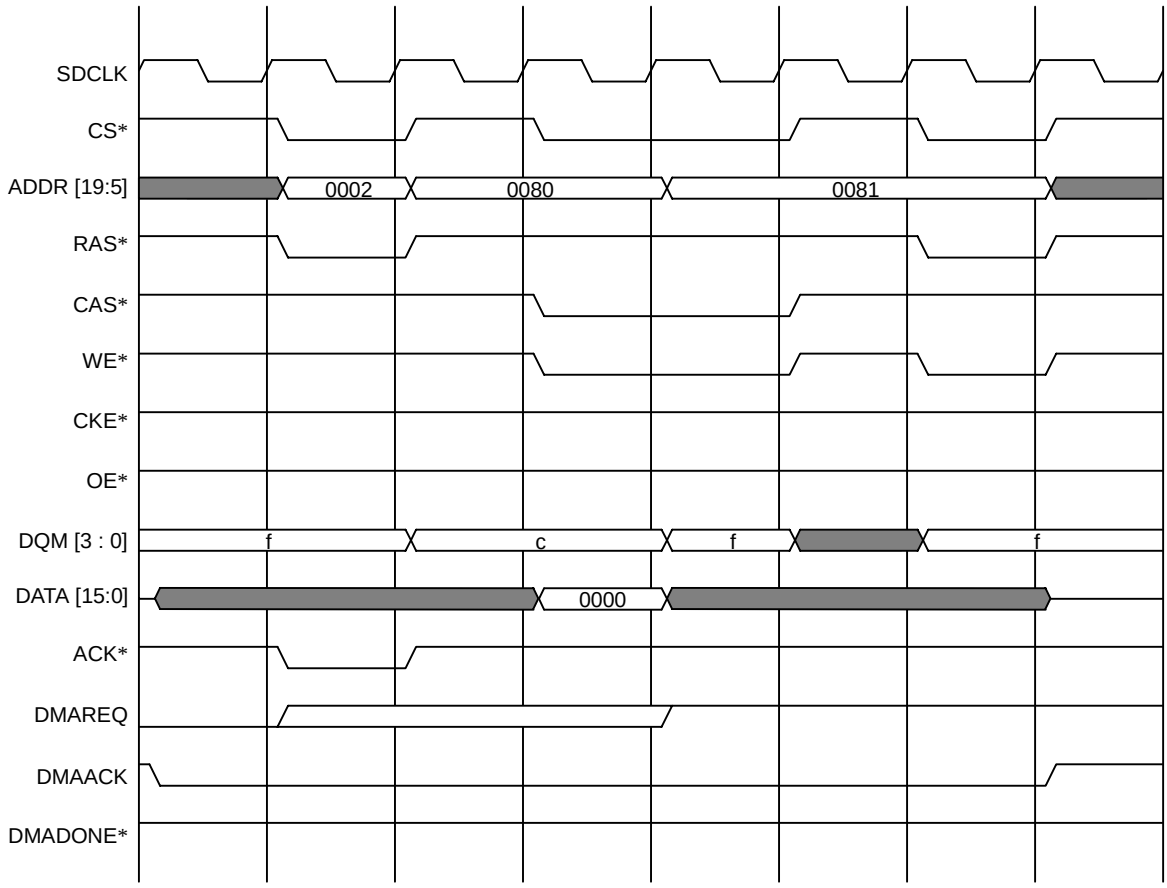


Figure 10.5.18 Single-Address Single-Write Timing (Writing 16-bit Data to a 16-bit SDRAM)

10.5.17 Single-Address Mode, 32-bit Read Operation (SDRAM)

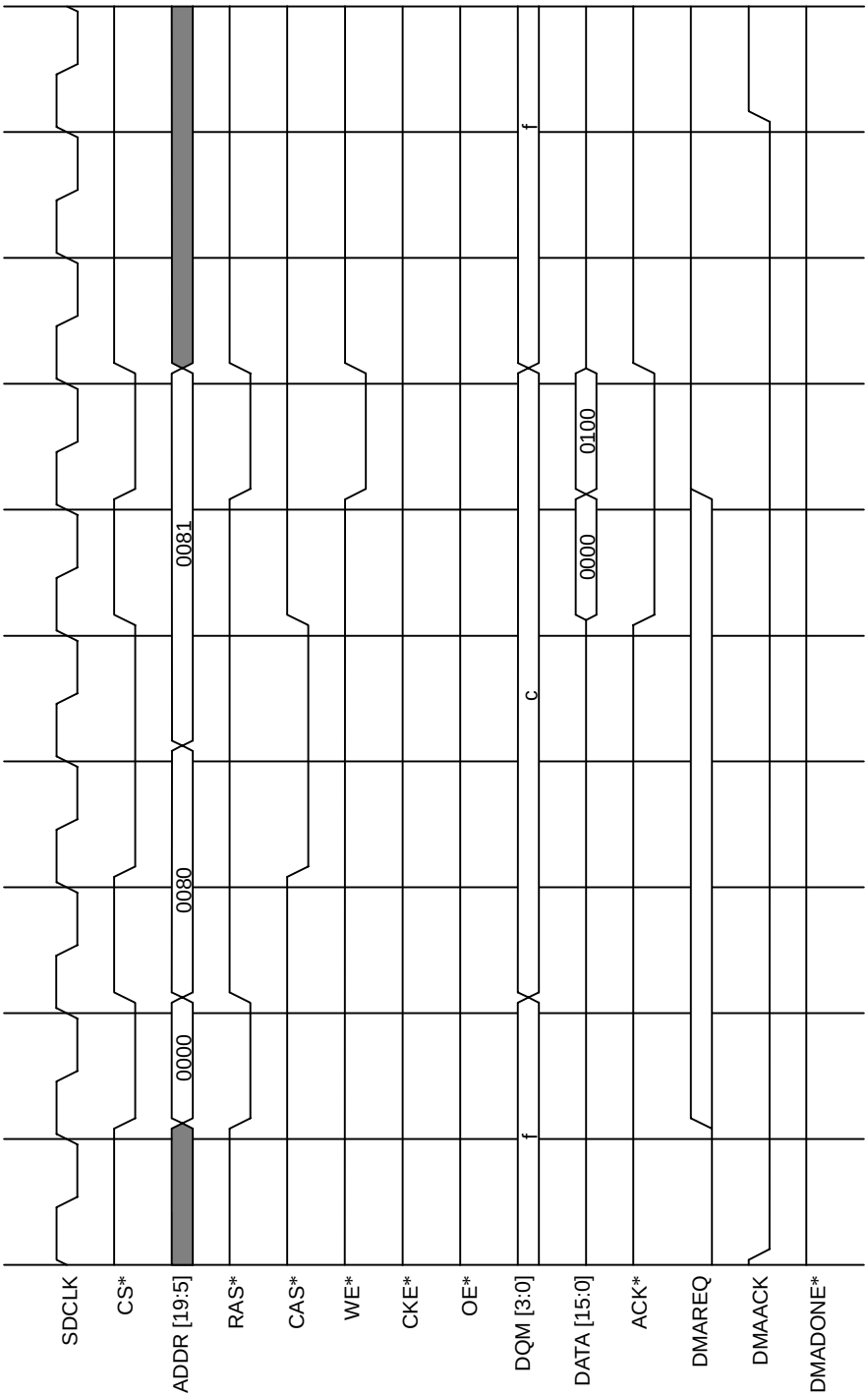


Figure 10.5.19 Single-Address Single-Read Timing (Reading 32-bit Data from a 16-bit SDRAM)

10.5.18 Single-Address Mode, 32-bit Write Operation (SDRAM)

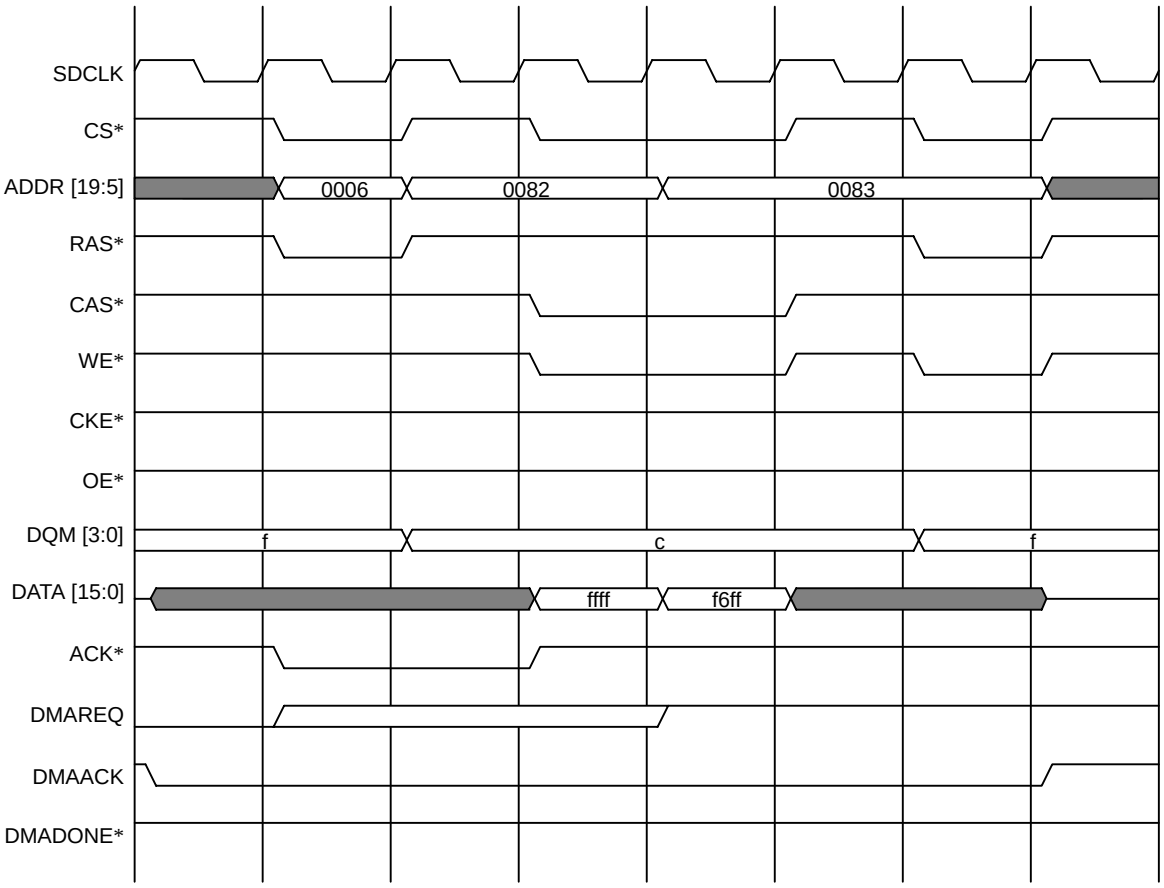


Figure 10.5.20 Single-Address Single-Write Timing (Writing 32-bit Data to a 16-bit SDRAM)

10.5.19 Single-Address Mode, 32-bit Burst-Write Operation (SDRAM)

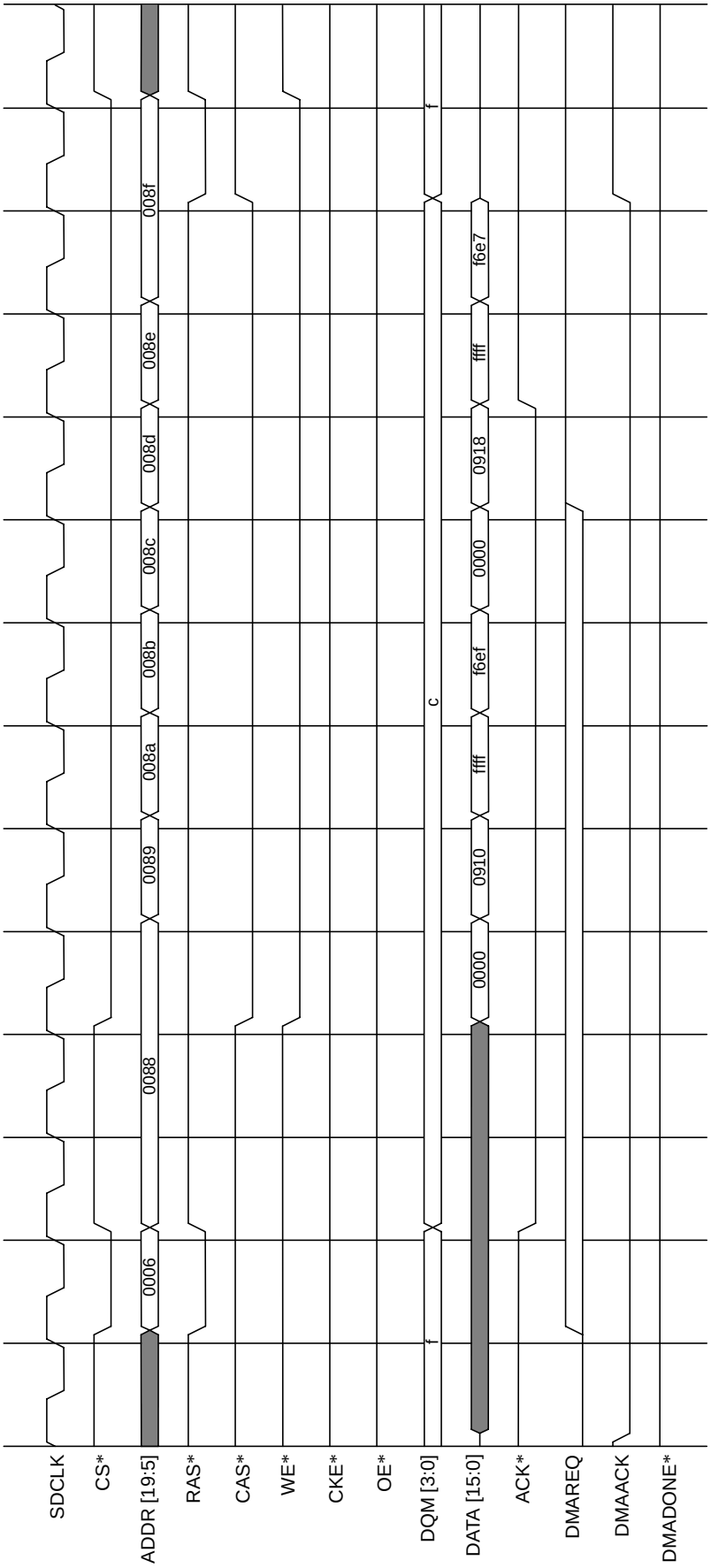


Figure 10.5.21 Single-Address Mode Burst-Write Timing (Burst-Write to a 16-bit SDRAM)

10.5.20 Dual-Address Mode Burst Operation (SRAM to SRAM)

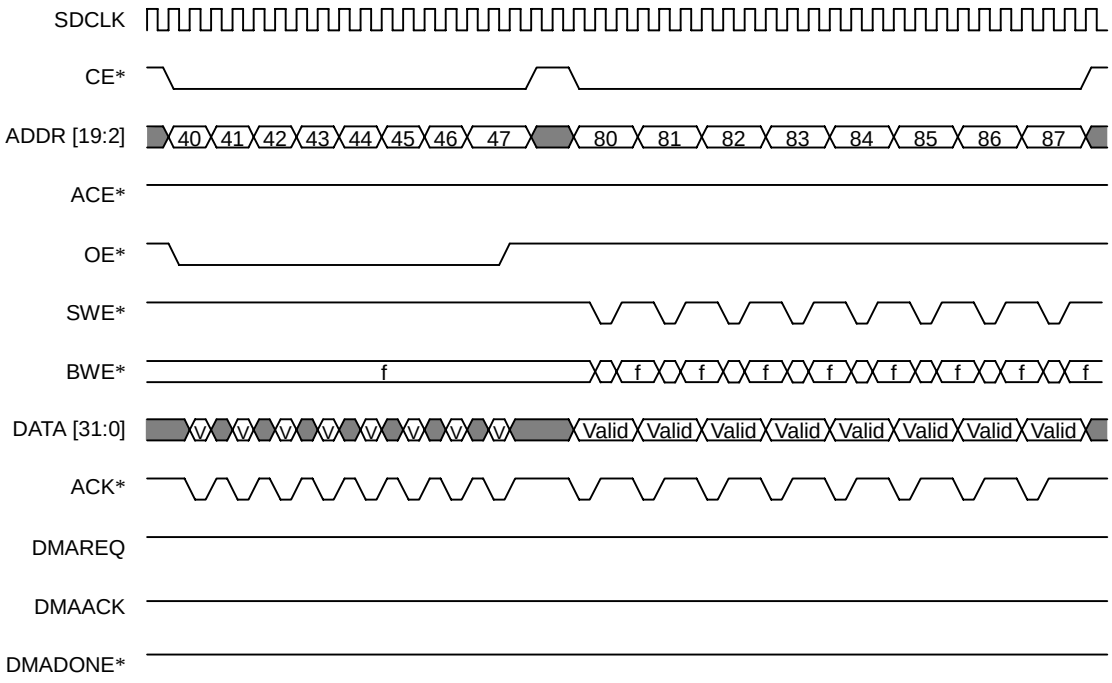


Figure 10.5.22 Dual-Address Mode Read/Write Timing (8-word Burst Transfer between 32-bit SRAMs)

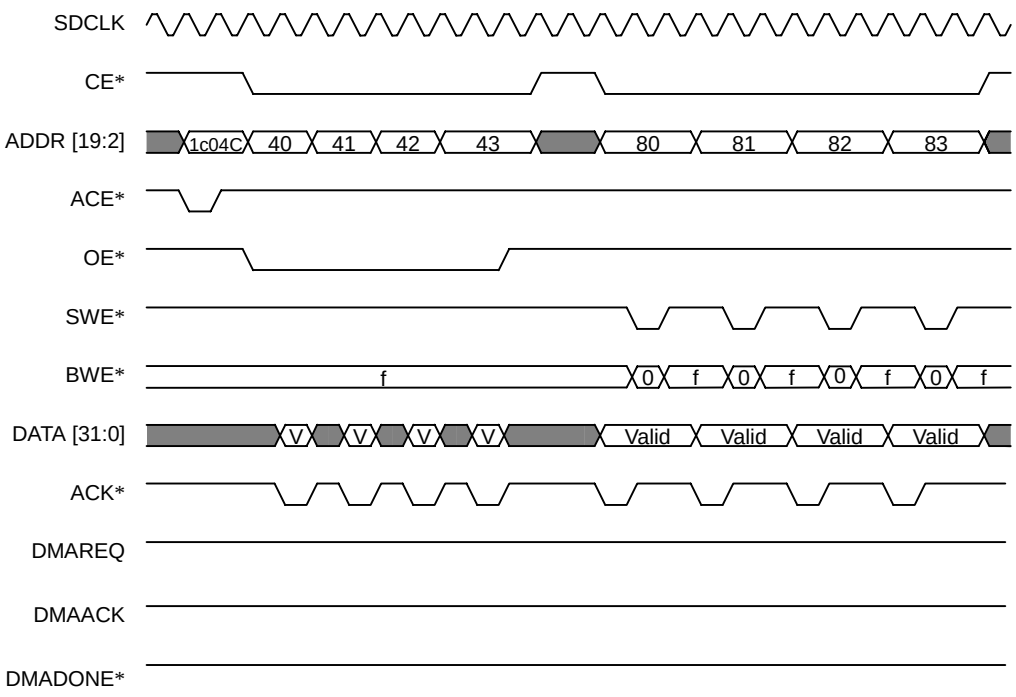


Figure 10.5.23 Dual-Address Mode Read/Write Timing (4-word Burst Transfer between 32-bit SRAMs)

10.5.21 Dual-Address Mode Burst Operation (SRAM to SDRAM)

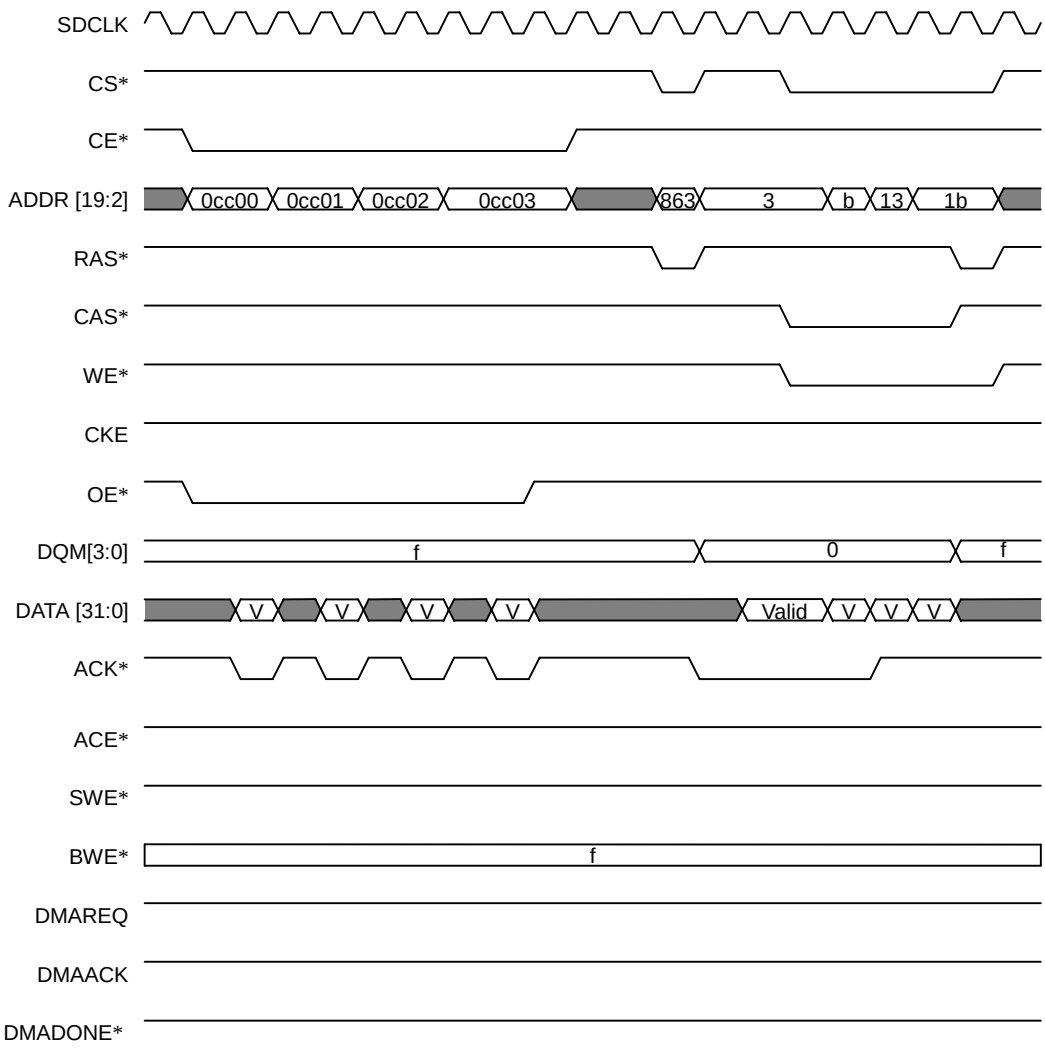


Figure 10.5.24 Dual-Address Mode Read/Write Timing
(4-word Burst Transfer from a 32-bit SRAM to a 32-bit SDRAM)

10.5.22 Dual-Address Mode Burst Operation (SDRAM to SRAM)

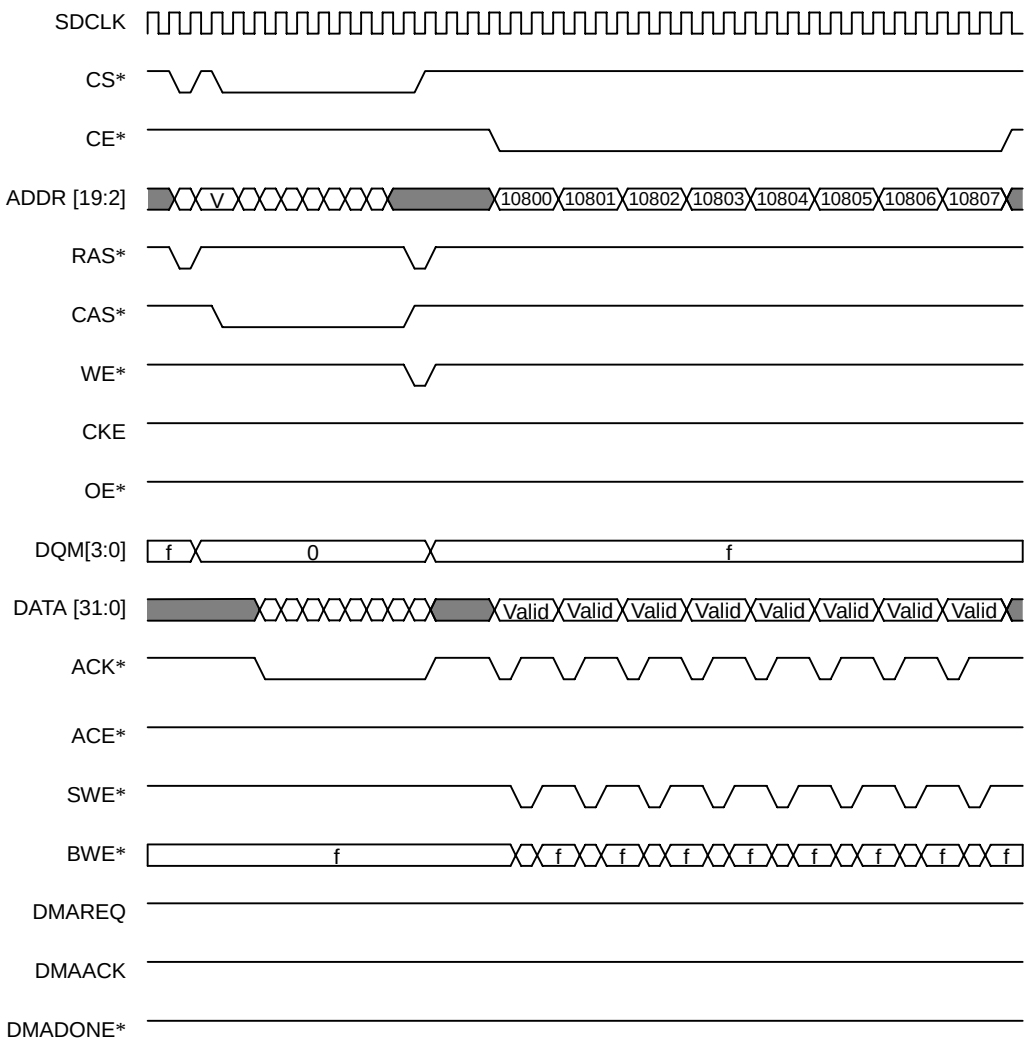


Figure 10.5.25 Dual-Address Mode Read/Write Timing
(8-word Burst Transfer from a 32-bit SDRAM to a 32-bit SRAM)

10.5.23 Dual-Address Mode Burst Operation (SDRAM to SDRAM)

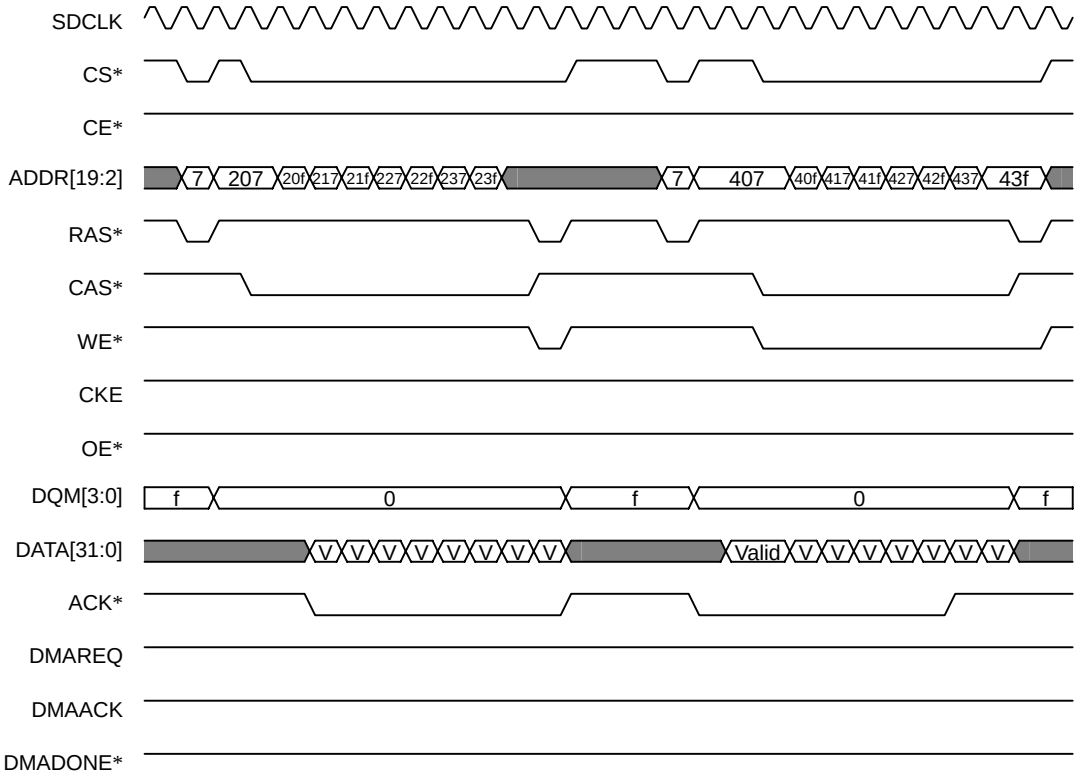


Figure 10.5.26 Dual-Address Mode Read/Write Timing (8-word Burst Transfer between 32-bit SDRAMs)

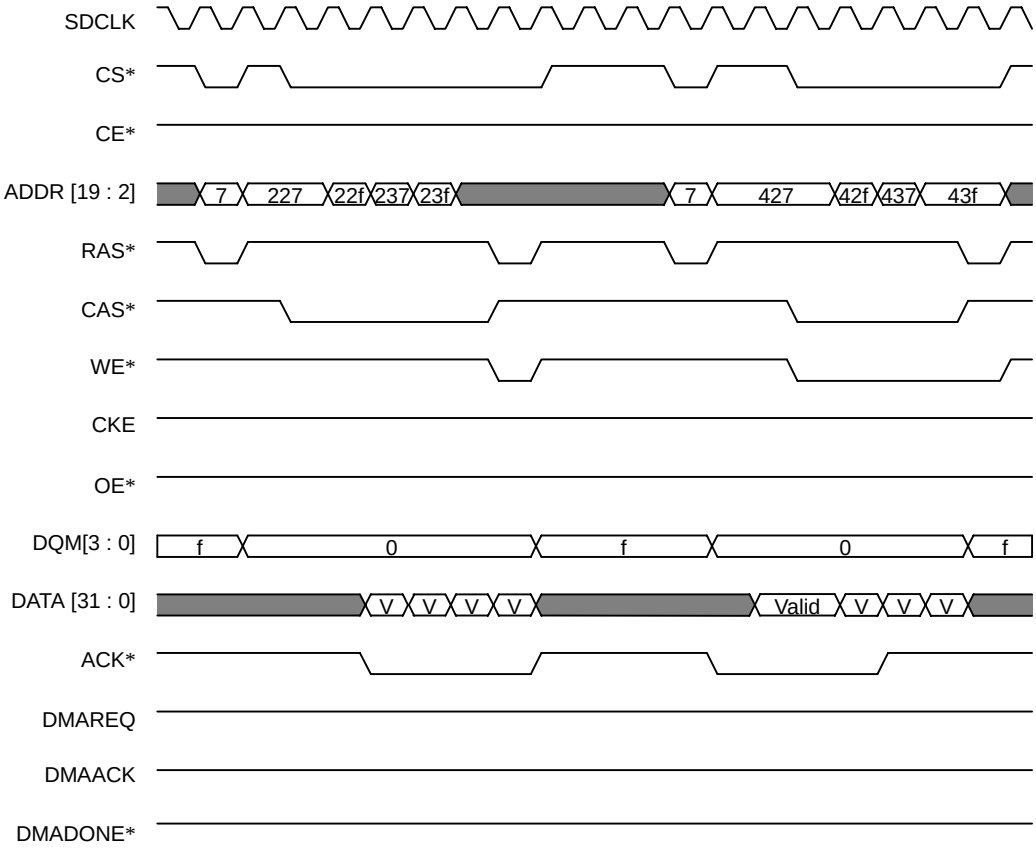


Figure 10.5.27 Dual-Address Mode Read/Write Timing (4-word Burst Transfer between 32-bit SDRAMs)

10.5.24 Dual-Address Mode Non-burst Operation (SDRAM to ROMC Device)

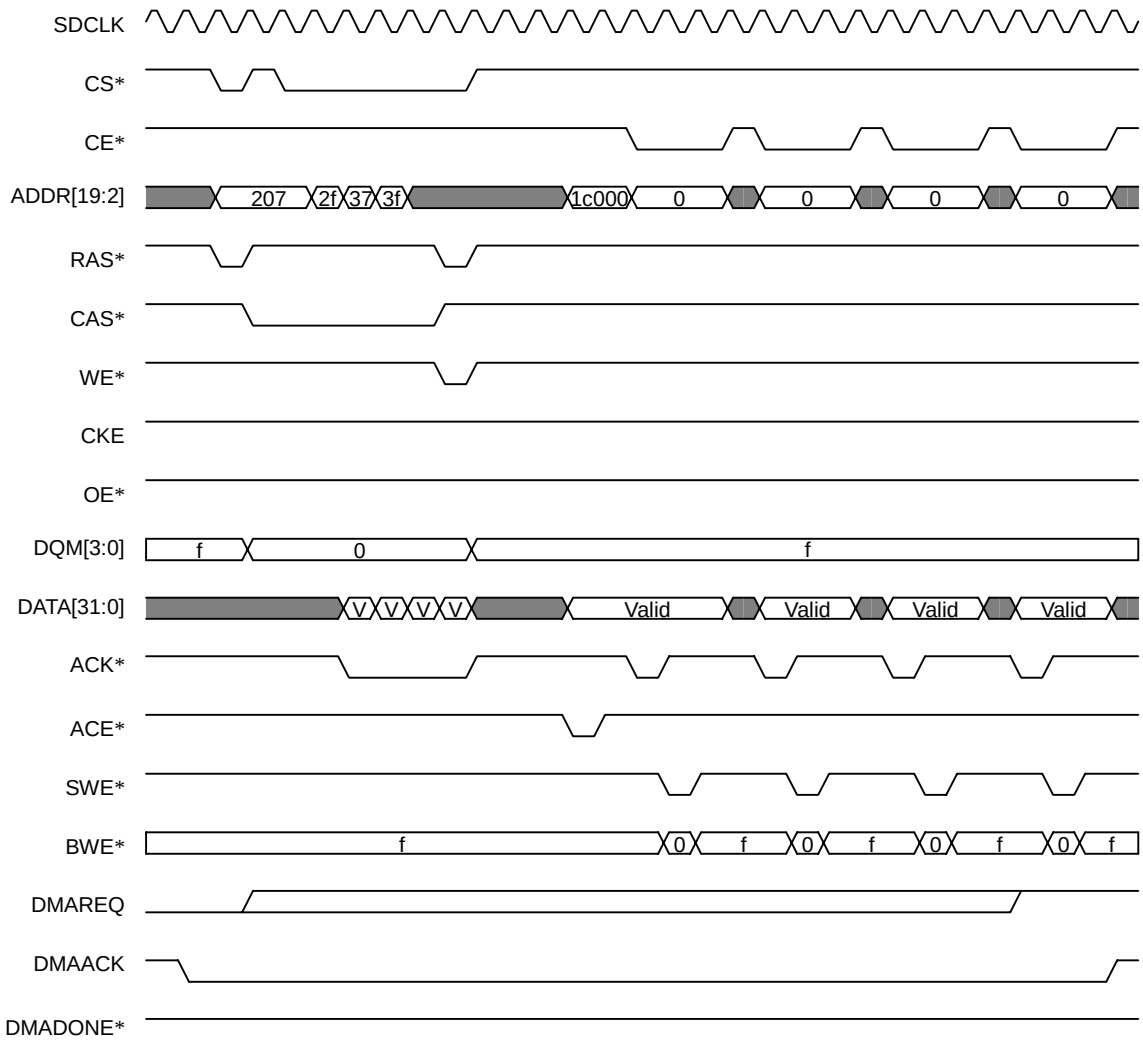


Figure 10.5.28 Dual-Address Mode Read/Write Timing
(4-word Transfer from a 32-bit Burst-Mode SDRAM to a 32-bit Non-burst ROMC Device)

10.5.25 Dual-Address Mode Non-burst Operation (ROMC Device to SDRAM)

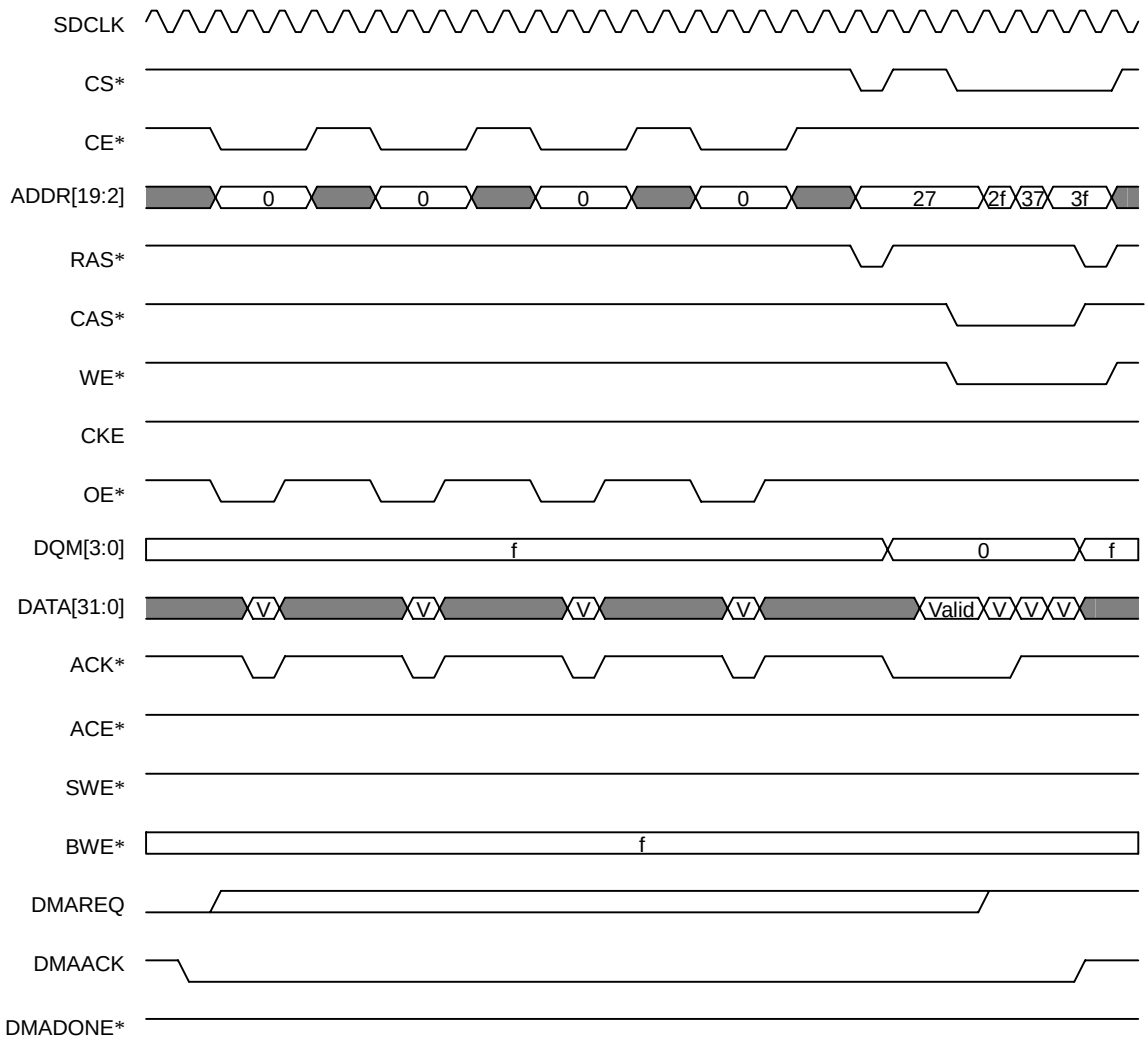


Figure 10.5.29 Dual-Address Read/Write Timing
(4-word Transfer from a 32-bit Non-burst ROMC Device to a 32-bit Burst-Mode SDRAM)

11. Interrupt Controller (IRC)

11.1 Features

The integrated Interrupt Controller (IRC) of the TX3927 coordinates all interrupt sources, both from on-chip peripherals and off-chip inputs, and provides interrupt requests to the TX39/H2 processor core with programmable priority levels.

The IRC has the following features:

- 8 internal interrupts and up to 6 external interrupts.
- Each interrupt source can be assigned one of eight priority levels (0-7).
- Each external interrupt pin can be individually programmed as either edge- or level-detected.

11.2 Block Diagram

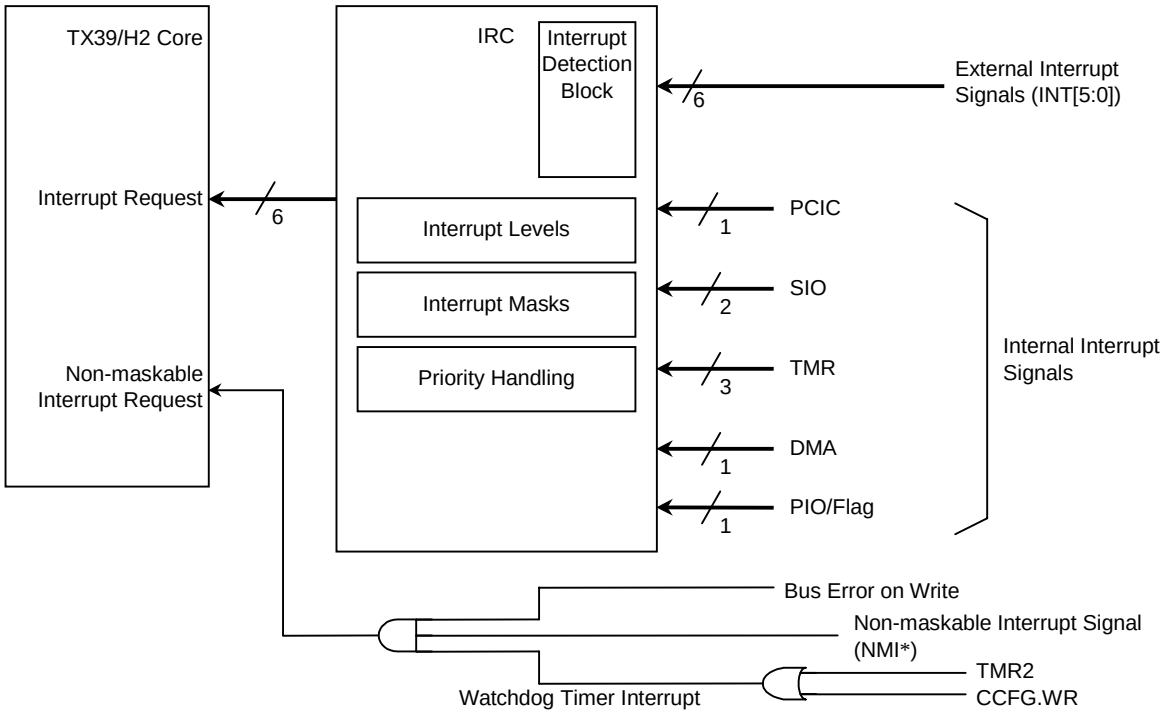


Figure 11.2.1 TX3927 IRC block diagram

The interrupt detection block monitors the states of the external interrupt request pins, INT[5:0]. Each interrupt pin can be individually programmed as either edge- or level-triggered and as either active-high or active-low. The Interrupt Control Mode Register 0 (IRCR0) specifies the trigger mode.

The IRC collects interrupt events from on- and off-chip peripherals and prioritizes them. After determining the highest-priority interrupt, the IRC drives the interrupt request lines to the TX39/H2 core to inform it of the interrupt request.

11.3 Registers

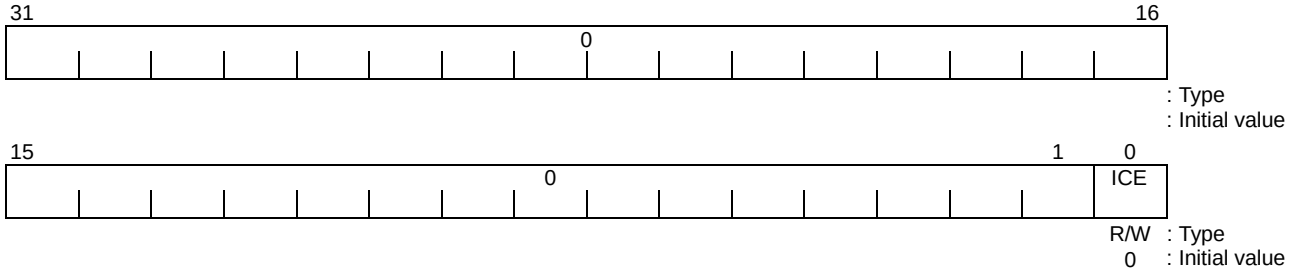
11.3.1 Register Map

The base address of the IRC registers is 0xFFFE_C000. All registers of the IRC can only be word-accessed. For the bits not defined in this section, the values shown in the figures must be written.

Table 11.3.1 IRC Registers

Address	Register Mnemonic	Register Name
0xFFFE_C0A0	IRCSR	Interrupt Current Status Register
0xFFFE_C080	IRSSR	Interrupt Source Status Register
0xFFFE_C060	IRSCR	Interrupt Status/Control Register
0xFFFE_C040	IRIMR	Interrupt Mask Register
0xFFFE_C02C	IRILR7	Interrupt Level Register 7
0xFFFE_C028	IRILR6	Interrupt Level Register 6
0xFFFE_C024	IRILR5	Interrupt Level Register 5
0xFFFE_C020	IRILR4	Interrupt Level Register 4
0xFFFE_C01C	IRILR3	Interrupt Level Register 3
0xFFFE_C018	IRILR2	Interrupt Level Register 2
0xFFFE_C014	IRILR1	Interrupt Level Register 1
0xFFFE_C010	IRILR0	Interrupt Level Register 0
0xFFFE_C008	IRCR1	Interrupt Control Mode Register 1
0xFFFE_C004	IRCR0	Interrupt Control Mode Register 0
0xFFFE_C000	IRCER	Interrupt Control Enable Register

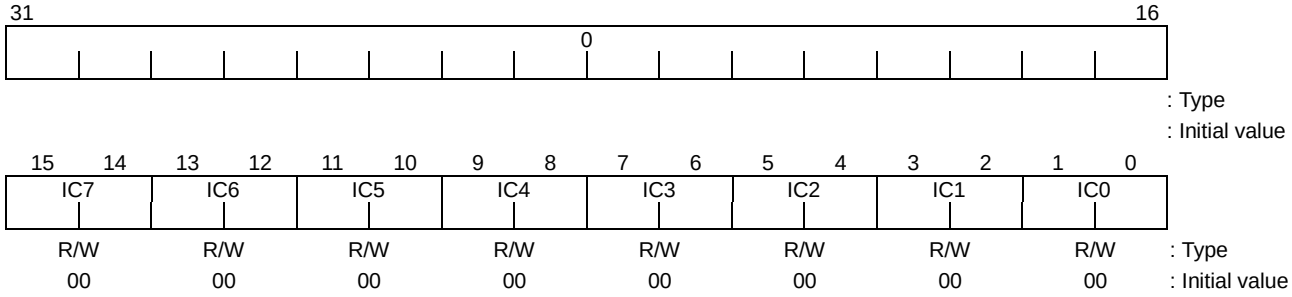
11.3.2 Interrupt Control Enable Register (IRCER) 0xFFFE_C000



Bits	Mnemonic	Field Name	Description
0	ICE	Interrupt Control Enable	Interrupt Control Enable (initial value: 0) Enables or disables interrupts detection. 0: Disabled 1: Enabled

Figure 11.3.1 Interrupt Control Enable Register

11.3.3 Interrupt Control Mode Register 0 (IRCR0) 0xFFFE_C004



Bits	Mnemonic	Field Name	Description
15:14	IC7	Interrupt Source Control 7	Interrupt Source Control 7 (initial value: 00) Specifies the polarity and trigger mode for SIO[1] interrupts. 00: Low level 01: Don't use. 10: Don't use. 11: Don't use.
13:12	IC6	Interrupt Source Control 6	Interrupt Source Control 6 (initial value: 00) Specifies the polarity and trigger mode for SIO[0] interrupts. 00: Low level 01: Don't use. 10: Don't use. 11: Don't use.
11:10	IC5	Interrupt Source Control 5	Interrupt Source Control 5 (initial value: 00) Specifies the polarity and trigger mode for INT[5] interrupts. 00: Low level 01: High level 10: Falling edge 11: Rising edge
9:8	IC4	Interrupt Source Control 4	Interrupt Source Control 4 (initial value: 00) Specifies the polarity and trigger mode for INT[4] interrupts. 00: Low level 01: High level 10: Falling edge 11: Rising edge
7:6	IC3	Interrupt Source Control 3	Interrupt Source Control 3 (initial value: 00) Specifies the polarity and trigger mode for INT[3] interrupts. 00: Low level 01: High level 10: Falling edge 11: Rising edge
5:4	IC2	Interrupt Source Control 2	Interrupt Source Control 2 (initial value: 00) Specifies the polarity and trigger mode for INT[2] interrupts. 00: Low level 01: High level 10: Falling edge 11: Rising edge
3:2	IC1	Interrupt Source Control 1	Interrupt Source Control 1 (initial value: 00) Specifies the polarity and trigger mode for INT[1] interrupts. 00: Low level 01: High level 10: Falling edge 11: Rising edge

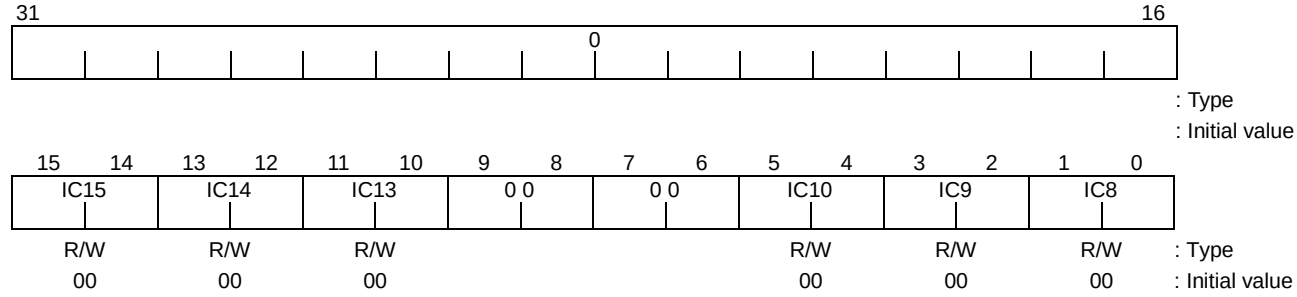
Note: IC7 and IC6 must be set to 00 for low-level detection.

Figure 11.3.2 Interrupt Control Mode Register 0 (1/2)

Bits	Mnemonic	Field Name	Description
1:0	IC0	Interrupt Source Control 0	Interrupt Source Control 0 (initial value: 00) Specifies the polarity and trigger mode for INT[0] interrupts. 00: Low level 01: High level 10: Falling edge 11: Rising edge

Figure 11.3.2 Interrupt Control Mode Register 0 (2/2)

11.3.4 Interrupt Control Mode Register 1 (IRCR1) 0xFFFE_C008

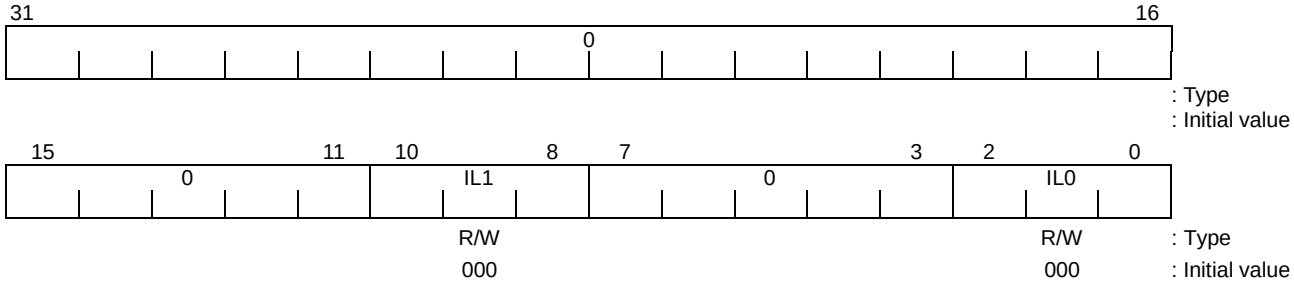


Bits	Mnemonic	Field Name	Description
15:14	IC15	Interrupt Source Control 15	Interrupt Source Control 15 (initial value: 00) Specifies the polarity and trigger mode for TMR[2] interrupts. 00: Low level 01: Don't use. 10: Don't use. 11: Don't use.
13:12	IC14	Interrupt Source Control 14	Interrupt Source Control 14 (initial value: 00) Specifies the polarity and trigger mode for TMR[1] interrupts. 00: Low level 01: Don't use. 10: Don't use. 11: Don't use.
11:10	IC13	Interrupt Source Control 13	Interrupt Source Control 13 (initial value: 00) Specifies the polarity and trigger mode for TMR[0] interrupts. 00: Low level 01: Don't use. 10: Don't use. 11: Don't use.
5:4	IC10	Interrupt Source Control 10	Interrupt Source Control 10 (initial value: 00) Specifies the polarity and trigger mode for PCI interrupts. 00: Low level 01: Don't use. 10: Don't use. 11: Don't use.
3:2	IC9	Interrupt Source Control 9	Interrupt Source Control 9 (initial value: 00) Specifies the polarity and trigger mode for PIO/Flag interrupts. 00: Low level 01: Don't use. 10: Don't use. 11: Don't use.
1:0	IC8	Interrupt Source Control 8	Interrupt Source Control 8 (initial value: 00) Specifies the polarity and trigger mode for DMA interrupts. 00: Low level 01: Don't use. 10: Don't use. 11: Don't use.

Note: IC15 to IC8 must be set to 00 for low-level detection.

Figure 11.3.3 Interrupt Control Mode Register 1

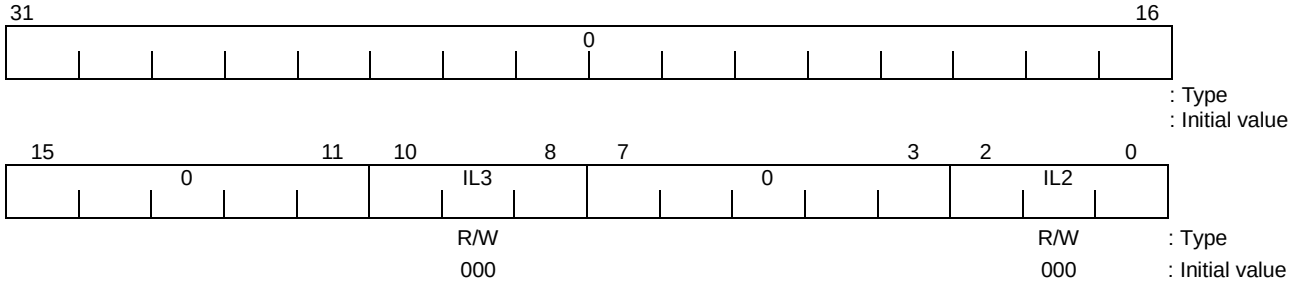
11.3.5 Interrupt Level Register 0 (IRILR0) 0xFFFE_C010



Bits	Mnemonic	Field Name	Description
10:8	IL1	Interrupt Level 1	Interrupt Level of INT[1] (initial value: 000) Specifies the interrupt priority level for the INT[1] interrupt. 000: Interrupt level 0 (interrupts disabled) 001: Interrupt level 1 010: Interrupt level 2 011: Interrupt level 3 100: Interrupt level 4 101: Interrupt level 5 110: Interrupt level 6 111: Interrupt level 7
2:0	IL0	Interrupt Level 0	Interrupt Level of INT[0] (initial value: 000) Specifies the interrupt priority level for the INT[0] interrupt. 000: Interrupt level 0 (interrupts disabled) 001: Interrupt level 1 010: Interrupt level 2 011: Interrupt level 3 100: Interrupt level 4 101: Interrupt level 5 110: Interrupt level 6 111: Interrupt level 7

Figure 11.3.4 Interrupt Level Register 0

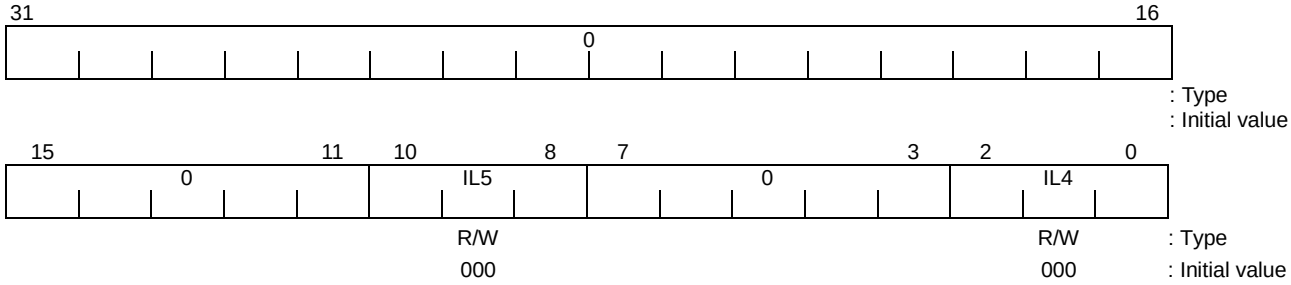
11.3.6 Interrupt Level Register 1 (IRILR1) 0xFFFE_C014



Bits	Mnemonic	Field Name	Description
10:8	IL3	Interrupt Level 3	Interrupt Level of INT[3] (initial value: 000) Specifies the interrupt priority level for the INT[1] interrupt. 000: Interrupt level 0 (interrupts disabled) 001: Interrupt level 1 010: Interrupt level 2 011: Interrupt level 3 100: Interrupt level 4 101: Interrupt level 5 110: Interrupt level 6 111: Interrupt level 7
2:0	IL2	Interrupt Level 2	Interrupt Level of INT[2] (initial value: 000) Specifies the interrupt priority level for the INT[1] interrupt. 000: Interrupt level 0 (interrupts disabled) 001: Interrupt level 1 010: Interrupt level 2 011: Interrupt level 3 100: Interrupt level 4 101: Interrupt level 5 110: Interrupt level 6 111: Interrupt level 7

Figure 11.3.5 Interrupt Level Register 1

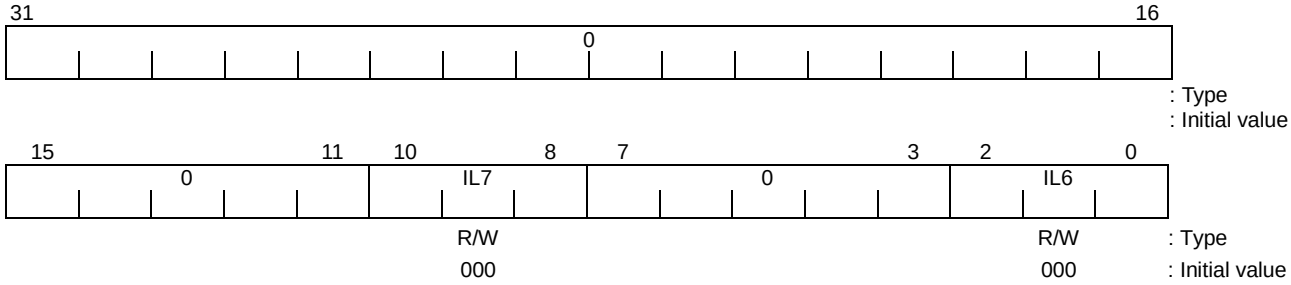
11.3.7 Interrupt Level Register 2 (IRILR2) 0xFFFE_C018



Bits	Mnemonic	Field Name	Description
10:8	IL5	Interrupt Level 5	Interrupt Level of INT[5] (initial value: 000) Specifies the interrupt priority level for the INT[5] interrupt. 000: Interrupt level 0 (interrupts disabled) 001: Interrupt level 1 010: Interrupt level 2 011: Interrupt level 3 100: Interrupt level 4 101: Interrupt level 5 110: Interrupt level 6 111: Interrupt level 7
2:0	IL4	Interrupt Level 4	Interrupt Level of INT[4] (initial value: 000) Specifies the interrupt priority level for the INT[4] interrupt. 000: Interrupt level 0 (interrupts disabled) 001: Interrupt level 1 010: Interrupt level 2 011: Interrupt level 3 100: Interrupt level 4 101: Interrupt level 5 110: Interrupt level 6 111: Interrupt level 7

Figure 11.3.6 Interrupt Level Register 2

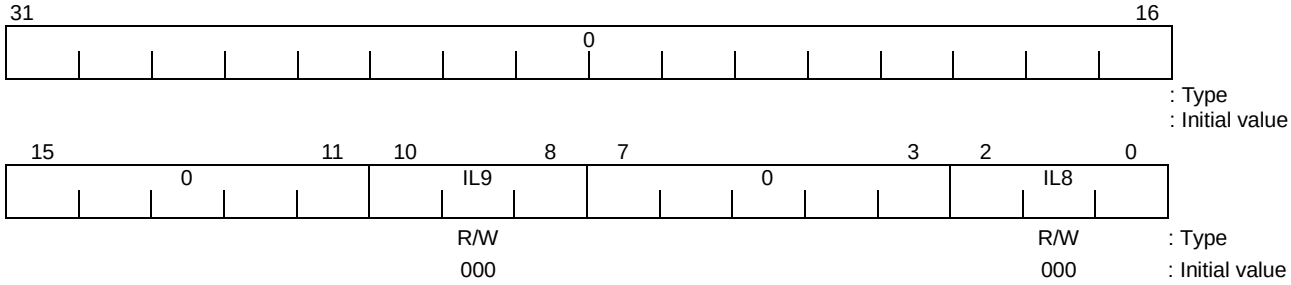
11.3.8 Interrupt Level Register 3 (IRILR3) 0xFFFE_C01C



Bits	Mnemonic	Field Name	Description
10:8	IL7	Interrupt Level 7	Interrupt Level of INT[7] (initial value: 000) Specifies the interrupt priority level for the SIO[1] interrupt. 000: Interrupt level 0 (interrupts disabled) 001: Interrupt level 1 010: Interrupt level 2 011: Interrupt level 3 100: Interrupt level 4 101: Interrupt level 5 110: Interrupt level 6 111: Interrupt level 7
2:0	IL6	Interrupt Level 6	Interrupt Level of INT[6] (initial value: 000) Specifies the interrupt priority level for the SIO[0] interrupt. 000: Interrupt level 0 (interrupts disabled) 001: Interrupt level 1 010: Interrupt level 2 011: Interrupt level 3 100: Interrupt level 4 101: Interrupt level 5 110: Interrupt level 6 111: Interrupt level 7

Figure 11.3.7 Interrupt Level Register 3

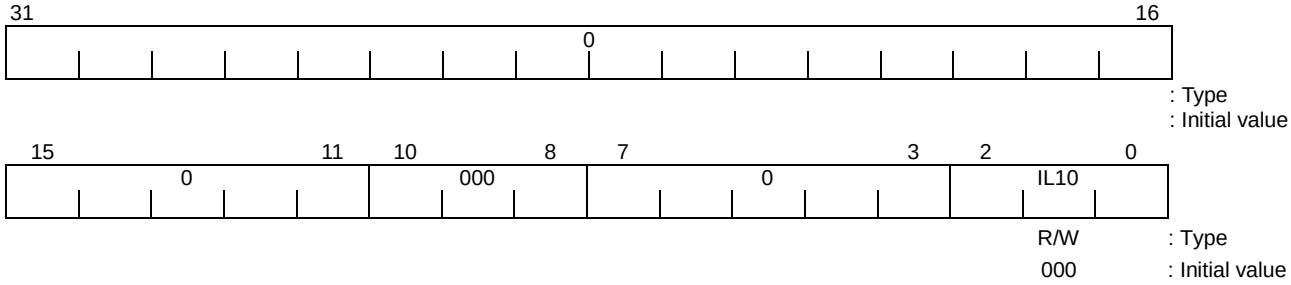
11.3.9 Interrupt Level Register 4 (IRILR4) 0xFFFE_C020



Bits	Mnemonic	Field Name	Description
10:8	IL9	Interrupt Level 9	Interrupt Level of INT[9] (initial value: 000) Specifies the interrupt priority level for the PIO/Flag interrupt. 000: Interrupt level 0 (interrupts disabled) 001: Interrupt level 1 010: Interrupt level 2 011: Interrupt level 3 100: Interrupt level 4 101: Interrupt level 5 110: Interrupt level 6 111: Interrupt level 7
2:0	IL8	Interrupt Level 8	Interrupt Level of INT[8] (initial value: 000) Specifies the interrupt priority level for the DMA interrupt. 000: Interrupt level 0 (interrupts disabled) 001: Interrupt level 1 010: Interrupt level 2 011: Interrupt level 3 100: Interrupt level 4 101: Interrupt level 5 110: Interrupt level 6 111: Interrupt level 7

Figure 11.3.8 Interrupt Level Register 4

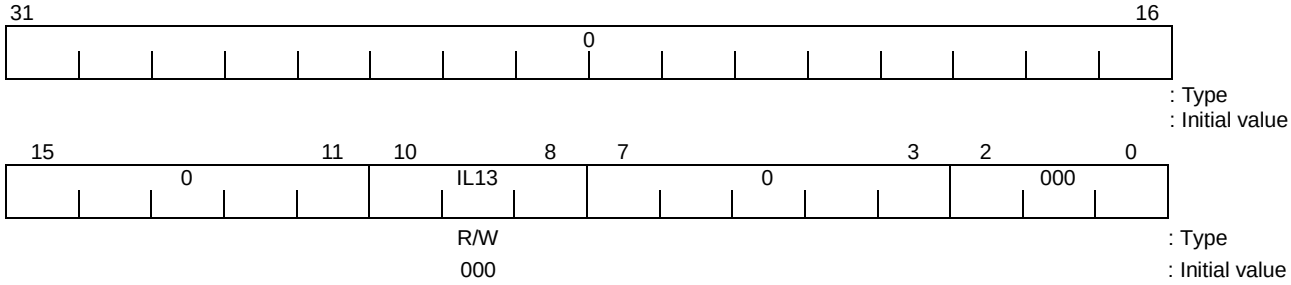
11.3.10 Interrupt Level Register 5 (IRILR5) 0xFFFE_C024



Bits	Mnemonic	Field Name	Description
2:0	IL10	Interrupt Level 10	Interrupt Level of INT[10] (initial value: 000) Specifies the interrupt priority level for the PCI interrupt. 000: Interrupt level 0 (interrupts disabled) 001: Interrupt level 1 010: Interrupt level 2 011: Interrupt level 3 100: Interrupt level 4 101: Interrupt level 5 110: Interrupt level 6 111: Interrupt level 7

Figure 11.3.9 Interrupt Level Register 5

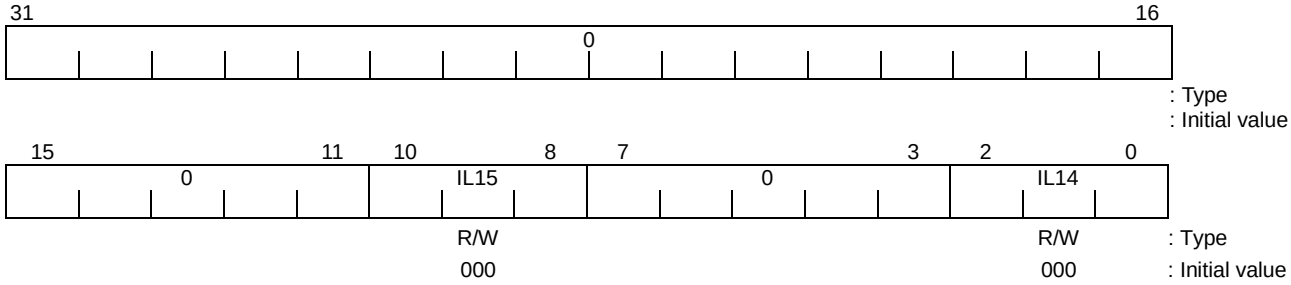
11.3.11 Interrupt Level Register 6 (IRILR6) 0xFFFE_C028



Bits	Mnemonic	Field Name	Description
10:8	IL13	Interrupt Level 13	Interrupt Level of INT[13] (initial value: 000) Specifies the interrupt priority level for the TMR[0] interrupt. 000: Interrupt level 0 (interrupts disabled) 001: Interrupt level 1 010: Interrupt level 2 011: Interrupt level 3 100: Interrupt level 4 101: Interrupt level 5 110: Interrupt level 6 111: Interrupt level 7

Figure 11.3.10 Interrupt Level Register 6

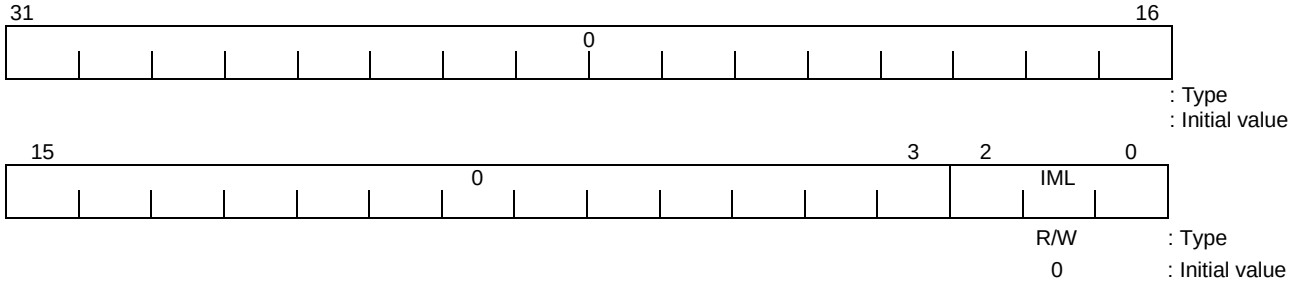
11.3.12 Interrupt Level Register 7 (IRILR7) 0xFFFE_C02C



Bits	Mnemonic	Field Name	Description
10:8	IL15	Interrupt Level 15	Interrupt Level of INT[15] (initial value: 000) Specifies the interrupt priority level for the TMR[2] interrupt. 000: Interrupt level 0 (interrupts disabled) 001: Interrupt level 1 010: Interrupt level 2 011: Interrupt level 3 100: Interrupt level 4 101: Interrupt level 5 110: Interrupt level 6 111: Interrupt level 7
2:0	IL14	Interrupt Level 14	Interrupt Level of INT[14] (initial value: 000) Specifies the interrupt priority level for the TMR[1] interrupt. 000: Interrupt level 0 (interrupts disabled) 001: Interrupt level 1 010: Interrupt level 2 011: Interrupt level 3 100: Interrupt level 4 101: Interrupt level 5 110: Interrupt level 6 111: Interrupt level 7

Figure 11.3.11 Interrupt Level Register 7

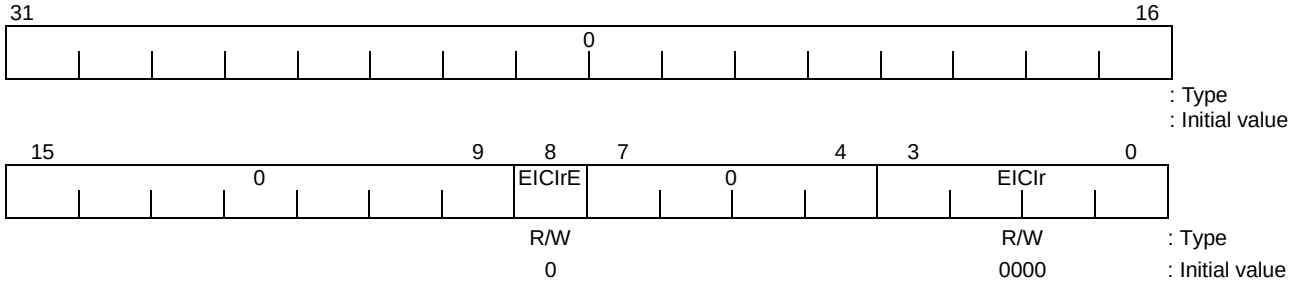
11.3.13 Interrupt Mask Register (IRIMR) 0xFFFE_C040



Bits	Mnemonic	Field Name	Description
2:0	IML	Interrupt Mask Level	Interrupt Mask Level (initial value: 000) Specifies an interrupt mask level. When an interrupt request has a priority lower than the value in the mask, the processor ignores that interrupt request. 000: Interrupt mask level 0 (all interrupts disabled) 001: Interrupt mask level 1 (levels 1-7 recognized) 010: Interrupt mask level 2 (levels 2-7 recognized) 011: Interrupt mask level 3 (levels 3-7 recognized) 100: Interrupt mask level 4 (levels 4-7 recognized) 101: Interrupt mask level 5 (levels 5-7 recognized) 110: Interrupt mask level 6 (levels 6-7 recognized) 111: Interrupt mask level 7 (only level 7 recognized)

Figure 11.3.12 Interrupt Mask Register

11.3.14 Interrupt Status/Control Register (IRSCR) 0xFFFE_C060

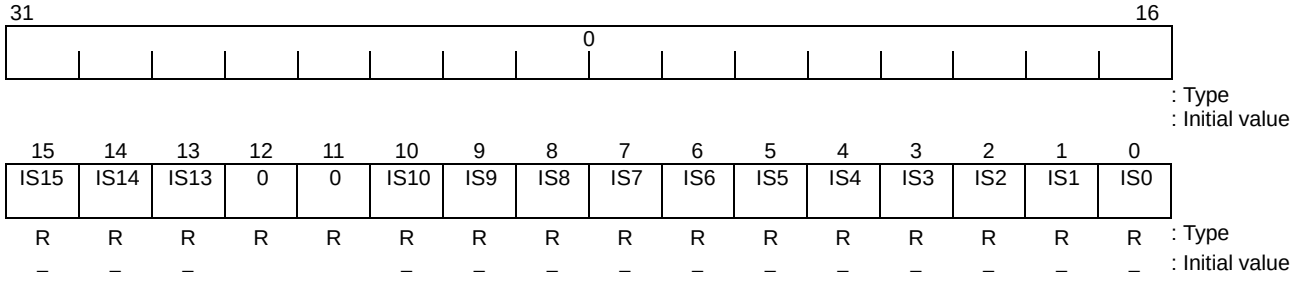


Bits	Mnemonic	Field Name	Description
8	EIClrE	Interrupt Clear Enable	Edge Interrupt Clear Enable for Sources (Initial value: 0) Clears the source of interrupt specified in the EIClr field. 0: The specified interrupt is unaffected. 1: The specified interrupt is cleared. This bit is immediately to cleared "0" after "1" is written. This bit returns a "0" on a read.
3:0	EIClr	Interrupt Clear for Sources	Edge Interrupt Clear for Sources (initial value: 0x0) Specifies the source of interrupt to be cleared. 1111: TMR[2] 1110: TMR[1] 1101: TMR[0] 1100: (Reserved) 1011: (Reserved) 1010: PCI 1001: PIO/Flag 1000: DMA 0111: SIO[1] 0110: SIO[0] 0101: INT[5] 0100: INT[4] 0011: INT[3] 0010: INT[2] 0001: INT[1] 0000: INT[0]

Figure 11.3.13 Interrupt Status/Control Register

11.3.15 Interrupt Source Status Register (IRSSR) 0xFFFE_C080

This register indicates which interrupts are pending, regardless of the settings the Interrupt Level (IRILR7 to IRILR0) and Interrupt Mask (IRIMR) registers.



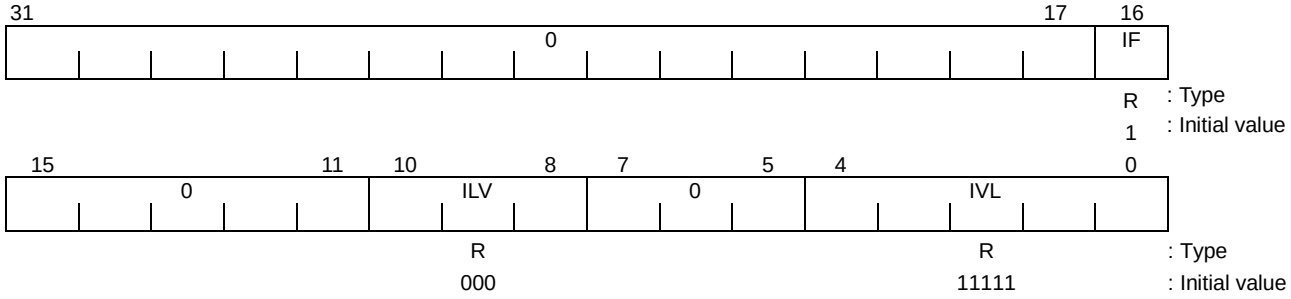
Bits	Mnemonic	Field Name	Description
15	IS15	Interrupt Status 15	IRINTREQ[15] Status Shows the status of the TMR[2] interrupt. 1: Interrupt pending 0: No interrupt pending
14	IS14	Interrupt Status 14	IRINTREQ[14] Status Shows the status of the TMR[1] interrupt. 1: Interrupt pending 0: No interrupt pending
13	IS13	Interrupt Status 13	IRINTREQ[13] Status Shows the status of the TMR[0] interrupt. 1: Interrupt pending 0: No interrupt pending
10	IS10	Interrupt Status 10	IRINTREQ[10] Status Shows the status of the PCI interrupt. 1: Interrupt pending 0: No interrupt pending
9	IS9	Interrupt Status 9	IRINTREQ[9] Status Shows the status of the PIO/Flag interrupt. 1: Interrupt pending 0: No interrupt pending
8	IS8	Interrupt Status 8	IRINTREQ[8] Status Shows the status of the DMA interrupt. 1: Interrupt pending 0: No interrupt pending
7	IS7	Interrupt Status 7	IRINTREQ[7] Status Shows the status of the SIO[1] interrupt. 1: Interrupt pending 0: No interrupt pending
6	IS6	Interrupt Status 6	IRINTREQ[6] Status Shows the status of the SIO[0] interrupt. 1: Interrupt pending 0: No interrupt pending
5	IS5	Interrupt Status 5	IRINTREQ[5] Status Shows the status of the INT[5] interrupt. 1: Interrupt pending 0: No interrupt pending
4	IS4	Interrupt Status 4	IRINTREQ[4] Status Shows the status of the INT[4] interrupt. 1: Interrupt pending 0: No interrupt pending

Figure 11.3.14 Interrupt Source Status Register (1/2)

Bits	Mnemonic	Field Name	Description
3	IS3	Interrupt Status 3	IRINTREQ[3] Status Shows the status of the INT[3] interrupt. 1: Interrupt pending 0: No interrupt pending
2	IS2	Interrupt Status 2	IRINTREQ[2] Status Shows the status of the INT[2] interrupt. 1: Interrupt pending 0: No interrupt pending
1	IS1	Interrupt Status 1	IRINTREQ[1] Status Shows the status of the INT[1] interrupt. 1: Interrupt pending 0: No interrupt pending
0	IS0	Interrupt Status 0	IRINTREQ[0] Status Shows the status of the INT[0] interrupt. 1: Interrupt pending 0: No interrupt pending

Figure 11.3.14 Interrupt Source Status Register (2/2)

11.3.16 Interrupt Current Status Register (IRCSR) 0xFFFE_C0A0



Bits	Mnemonic	Field Name	Description
16	IF	Interrupt Flag	Interrupt Flag (initial value: 1) Indicates whether there is any interrupt request or not. 0: An interrupt request detected 1: No interrupt request detected
10:8	ILV	Interrupt Level Vector	Interrupt Level Vector (initial value: 000) Indicates the interrupt priority level delivered to the TX39/H2 core. 000: Interrupt level 0 001: Interrupt level 1 : : 111: Interrupt level 7
4:0	IVL	Interrupt Vector	Interrupt Vector (initial value: 0x1F) Indicates the source of interrupt delivered to the TX39/H2 core. 00000: INT[0] 01001: PIO/Flag 00001: INT[1] 01010: PCI 00010: INT[2] 01011: (Reserved) 00011: INT[3] 01100: (Reserved) 00100: INT[4] 01101: TMR[0] 00101: INT[5] 01110: TMR[1] 00110: SIO[0] 01111: TMR[2] 00111: SIO[1] 10000-11111: (Reserved) 01000: DMA

Figure 11.3.15 Interrupt Current Status Register

11.4 Operation

11.4.1 Interrupt Sources

The TX3927 Interrupt Controller (IRC) recognizes eight internal interrupts and up to six external interrupts. Of the six external interrupts, INT[5:4] share device pins with the SIO. Refer to "3.3 Pin Multiplexing" for how the pin functions of these multiplexed pins are assigned.

Table 11.4.1 shows the interrupt sources the TX3927 supports. The Interrupt Controller arbitrates between interrupt requests from these sources and passes the highest-priority request to the TX39/H2 core. When an interrupt is taken, the TX39/H2 core's Cause register identifies the cause of the interrupt; while the IP[5] bit in the Cause register is set to 1 to indicate an occurrence of an interrupt, the IP[4:0] field captures the interrupt vector associated with its source, as shown below. When an interrupt occurs, processing branches to an appropriate interrupt exception vector address. The address depends on the value of the BEV bit in the TX39/H2 core's Status register. If BEV is 0, control is transferred to physical address 0x00000080. If BEV is 1, control is transferred to physical address 0x1FC00180.

Table 11.4.1

Priority*	Interrupt Vector	IP [5:0]	Interrupt Source
Highest	0	100000	INT[0]
	1	100001	INT[1]
	2	100010	INT[2]
	3	100011	INT[3]
	4	100100	INT[4]
	5	100101	INT[5]
	6	100110	SIO[0]
	7	100111	SIO[1]
	8	101000	DMA
	9	101001	PIO
	10	101010	PCI
	11	101011	(Reserved)
	12	101100	(Reserved)
	13	101101	TMR[0]
	14	101110	TMR[1]
Lowest	15	101111	TMR[2]

Note: An interrupt with a lower interrupt vector is given a higher priority if multiple interrupt requests having an equal priority level occur at the same time.

11.4.2 Interrupt Detection

To enable interrupts, the ICE bit of the Interrupt Control Enable Register (IRCER) must be set after initializing the other IRC registers. The IRCR0 and IRCR1 registers are used to select an interrupt detection mode from high and low level-sensitive, and rising and falling edge-triggered. The default is low level-sensitive. All the on-chip interrupt sources must always be programmed for to low-level detection.

When the IRC detects an interrupt request, it notifies the TX39/H2 core of the interrupt, which in turn records its interrupt vector in the IP field of its Cause register. If more than one interrupt request occurs simultaneously, the IRC delivers the highest-priority interrupt to the TX39/H2 core, based on its register settings. Software is responsible for clearing interrupt requests. If the interrupt source is programmed as high or low level-detected, the external circuitry asserting the interrupt request must be manipulated to deassert the request. If the interrupt source is programmed as rising or falling edge-detected, writing its interrupt vector to the EIClr field of the IRSCR register with EIClrE=1 causes the interrupt request to be

cleared.

Interrupt requests from on-chip peripheral modules must also be cleared through software.

11.4.3 Interrupt Priorities

- Eight levels of interrupt priorities are provided, numbered from 0 (000) to 7 (111). Priority levels can be programmed in the IRILR7 to IRILR0 registers, which contain a 3-bit priority-level field for each of the interrupt sources. Level 7 has the highest-priority, and level 1 the lowest. If any interrupt is programmed as 000, then that interrupt is effectively disabled. The response to simultaneous occurrence of equal priority interrupts is determined by a hardware priority-within-level resolver, according to interrupt vectors.

- The IRIMR register is available to mask interrupts. The 3-bit IML field of this register specifies the mask level. Interrupt requests having a priority level lower than the value in the IML field are masked. Those having the same priority level as the IML value are not masked. If the IML field is set to "000", all interrupts are globally disabled.

- If two or more interrupt requests occurs simultaneously, the IRC determines the highest-priority interrupt according to their programmed priority levels and the hardware-assigned interrupt vectors.

- A higher-priority interrupt always interrupts the servicing of a low-priority interrupt, causing the IRC to deliver a new interrupt vector to the TX39/H2 core. An interrupt is never interrupted by another interrupt of lower or equal priority, even by an interrupt with a smaller interrupt vector.

- The servicing of an interrupt may not be interrupted by another interrupt of equal or lower priority just by resetting the priority level of the current interrupt to 0 or a lower value in the Interrupt Level Register; i.e., resetting the priority level of the current interrupt to 0 does not affect the interrupt vector, if another interrupt of equal or lower priority is being requested at that time.

  If the user wants to interrupt the processing of the current interrupt by setting its priority level to 0, it is required to once change the interrupt mask level in the IRIMR register to a value that will mask the current interrupt and then change it back to the original value.

  For example:

  (1) IRIMR: Set the interrupt mask level to 7 (i.e., disable all interrupts), or any level that can mask the current interrupt.

  (2) IRILR: Set the interrupt level for the current interrupt to 0 (disable the current interrupt).

  (3) IRIMR: Restore the original interrupt mask level.

  The order of steps 1 and 2 is not significant.

Note: On the other hand, raising the interrupt mask level in the IML field of the IRIMR register will cause all lower-priority interrupts to be masked; consequently, interrupt vectors of any lower-priority interrupts will never be delivered.

12. PCI Controller (PCIC)

*** Notation used in this chapter ***

Note: In the TX39/H2 core documentation, the following terms are used to describe the size of data:

Byte = 8 bits
Halfword = 16 bits
Word = 32 bits

However, the PCI specification loosely uses these terms:

Byte = 8 bits
Word = 16 bits
Doubleword (D-word) = 32 bits

To be consistent with much of the PCI literature, this chapter refers to a 16-bit quantity as a word and a 32-bit quantity as a doubleword.

12.1 Features

The TX3927 PCI Controller (PCIC) is an embedded PCI core. The TX39/H2 processor core of the TX3927 uses the on-chip local bus (G-Bus) to access the PCIC.

The PCIC provides interface with a PCI bus and transfers data between the PCI bus and the local bus using the integrated memory controllers (SDRAMC and ROMC).

- Compliant with PCI Local Bus Specification, Revision 2.1 (32-bit bus/33 MHz)
- Support for initiator, target and bus arbiter functions
- Support for up to four external PCI masters
- Selectable PCI clock directions

Note: Because the TX39/H2 core can not retry a bus cycle, a PCI transaction request does not cause TX3927 internal bus operations to terminate. If a PCI target device has the same access characteristics, a deadlock situation might occur (with both sides repeatedly issuing retries). Deadlock avoidance is possible through the programming of transaction time-outs so as to generate a bus error.

12.1.1 PCI Interface Features

- 0- to 33-MHz PCI clock
- Independent initiator and target controllers
- Full PCI configuration registers with enhanced capability
- Programmable address mapping between the local bus (G-Bus) and the PCI bus
- Independent local bus (G-Bus) and PCI bus clocks

12.1.2 PCI Initiator Features

- Single-word transactions between the local bus (G-Bus) and the PCI bus
- Support for memory, I/O, configuration, special-cycle and interrupt-acknowledge transactions
- PCI write transactions from the local bus (G-Bus)
- Direct and Indirect data transfer modes

12.1.3 PCI Target Features

- Automatic data streaming between the PCI bus and the local bus (G-Bus)
  This allows the PCIC to transfer blocks of data larger than the FIFO depth, without requiring CPU or additional external logic
- Two 16 × 32-bit FIFOs
- Capable of zero-wait-state burst reads and writes
- Programmable burst lengths for the local bus (G-Bus) and the PCI bus, and memory prefetching of PCI bus reads
- Concurrent reads and writes by the PCI bus and the local bus (G-Bus)
- PCI write transactions from the PCI bus
- Support for fast back-to-back transactions
- Support for big-endian byte swapping

12.1.4 PCI Arbiter and Bus Parking Features

- Integrated PCI bus arbiter that supports up to four external PCI bus masters
- Programmable fairness algorithm: Two level, round-robin arbitration algorithm, with four PCI bus request/grant pairs
- Bus master parking using a Most Recently Used (MRU) algorithm
- Autonomous PCI bus arbiter/parking module
- Automatic disabling of an unused slot and broken masters after power-on reset
- Supports disabling the internal PCI bus arbiter to use an external arbiter
- PCI bus parking

12.2 Block Diagram

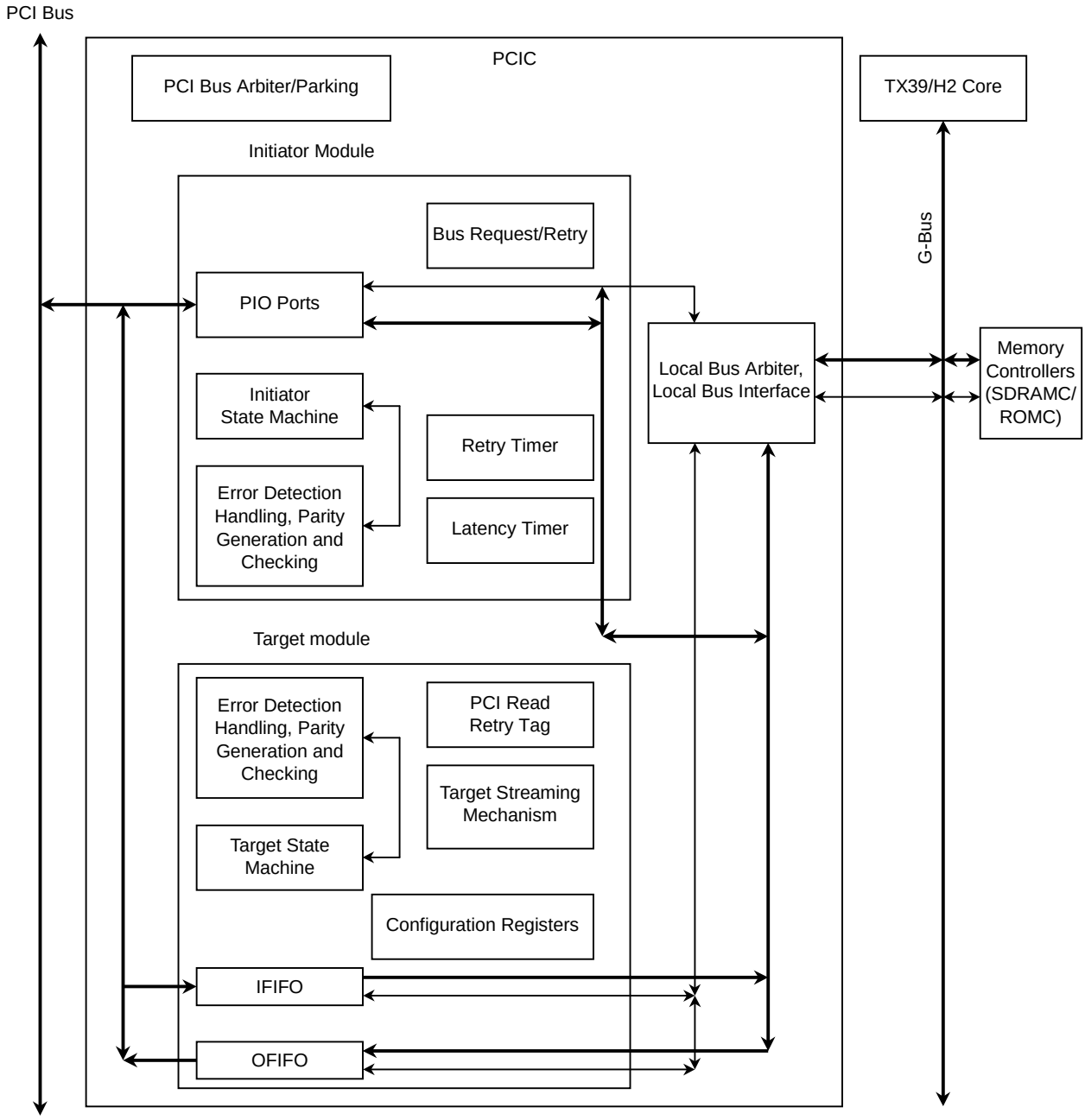


Figure 12.2.1 PCIC Block Diagram

12.3 Registers

The PCIC has two types of registers: PCI configuration space registers and local bus special registers:

1. PCI configuration space registers (0xFFFE_D0FF to 0xFFFE_D000)
   - PCI configuration header space registers
   - Initiator configuration space registers
   - Target configuration space registers

   All of the PCI configuration space registers can be accessed by both the TX39/H core and PCI bus masters. However, if the EPCAD bit of the Local Bus Control (LBC) register is set, PCI bus masters are denied access to these register.

2. Local bus special registers (0xFFFE_D1FF to 0xFFFE_D100)
   - PCI bus arbiter/parking registers
   - Local bus special registers

   These registers are not accessible from the PCI bus and must be programmed by the TX39/H2 core.

Reading reserved bits and registers may return undefined values. Software should not write any value to such bits or registers.

Note: All the internal registers of the PCIC are intrinsically little-endian. Non-32-bit registers are referenced with different addresses, depending on whether the CPU is in big-endian or little-endian mode. Such registers can be accessed as byte, word (16-bit), or even doubleword (32-bit) quantities, but care must be taken for endian differences between the CPU and the PCIC. In the PCIC register, all multi-byte numeric fields use little-endian ordering that assigns the lowest address to the lowest-order (rightmost) eight bits.

When reading from a PCIC register, software must mask reserved bits and should not rely on the value of any reserved bit remaining consistent. Also, writes must preserve the values of reserved bit positions by first reading the register, merging new values and writing the result back.

As a result of a hardware RESET*, the internal configuration registers of the PCIC are initialized to default states. The default states represent a minimum set of functions needed for proper system initialization, and do not necessarily provide an optimal system configuration. It is the responsibility of the system designer to program the PCIC registers with appropriate parameters and functions at system initialization time (typically using BIOS).

12.3.1 Register Map

In this section, the register addresses are shown for both big-endian and little-endian systems. In the following tables the first address is the address to use when the CPU is in big-endian mode. The address enclosed in parentheses is the offset from 0xFFFE_D000 when the CPU is in little-endian mode.

Example: The DID register is accessed with address 0xFFFE_D0000 in big-endian mode and with 0xFFFE_D002 in little-endian mode. This endian difference applies only to non-32-bit accesses.

Table 12.3.1 PCI Configuration Header Space Registers

Address	Mnemonic	Register Name
0xFFFE_D03F (+03c)	IL	PCI Interrupt Line Register
0xFFFE_D03E (+03d)	IP	PCI Interrupt Pin Register
0xFFFE_D03D (+03e)	MG	Minimum Grant Register
0xFFFE_D03C (+03f)	ML	Maximum Latency Register
0xFFFE_D037 (+034)	CAPPTR	Capabilities Pointer
0xFFFE_D030 (+030)	—	(Reserved)
0xFFFE_D02E (+02c)	SSVID	Subsystem Vendor ID Register
0xFFFE_D02C (+02e)	SVID	System Vendor ID Register
0xFFFE_D028 (+028)	—	(Reserved)
0xFFFE_D014 (+014)	MBA	Target Memory Base Address Register
0xFFFE_D010 (+010)	IOBA	Target I/O Base Address Register
0xFFFE_D00F (+00c)	—	(Reserved)
0xFFFE_D00E (+00d)	MLT	Master Latency Timer Register
0xFFFE_D00D (+00e)	HT	Header Type Register
0xFFFE_D00C (+00f)	—	(Reserved)
0xFFFE_D00B (+008)	RID	Revision ID Register
0xFFFE_D00A (+009)	RLPI	Register-Level Programming Interface Register
0xFFFE_D009 (+00a)	SCC	Subclass Code Register
0xFFFE_D008 (+00b)	CC	Class Code Register
0xFFFE_D006 (+004)	PCICMD	PCI Command Register
0xFFFE_D004 (+006)	PCISTAT	PCI Status Register
0xFFFE_D002 (+000)	VID	Vendor ID Register
0xFFFE_D000 (+002)	DID	Device ID Register

Table 12.3.2 Initiator Configuration Space Registers

Address	Mnemonic	Register Name
0xFFFE_D068 (+068)	ILBIOMAR	Initiator Local Bus I/O Mapping Address Register
0xFFFE_D064 (+064)	ILBMMAR	Initiator Local Bus Memory Mapping Address Register
0xFFFE_D060 (+060)	IPBIOMAR	Initiator PCI Bus I/O Mapping Address Register
0xFFFE_D05C (+05c)	IPBMMAR	Initiator PCI Bus Memory Mapping Address Register
0xFFFE_D04C (+04c)	RRT	Retry/Reconnect Timer Register
0xFFFE_D048 (+048)	IIM	Initiator Interrupt Mask Register
0xFFFE_D044 (+044)	ISTAT	Initiator Status Register
0xFFFE_D040 (+040)	—	(Reserved)

Table 12.3.3 Target Configuration Space Registers

Address	Mnemonic	Register Name
0xFFFE_D0E4 (+0e4)	PWMNGSR	Power Management Support Register
0xFFFE_D0E0 (+0e0)	PWMNGR	Power Management Register
0xFFFE_D0D0 (+0d0)	TBL	Target Burst Length Register
0xFFFE_D0CC (+0cc)	SC_BE	Special-Cycle Byte Enable Register
0xFFFE_D0C8 (+0c8)	SC_MSG	Special-Cycle Message Register
0xFFFE_D0C4 (+0c4)	TLBIOMAR	Target Local Bus I/O Mapping Address Register
0xFFFE_D0C0 (+0c0)	TLBMMAR	Target Local Bus Memory Mapping Address Register
0xFFFE_D0BC (+0bc)	TLBIAP	Target Local Bus IFIFO Address Pointer
0xFFFE_D0B8 (+0b8)	TLBOAP	Target Local Bus OFIFO Address Pointer
0xFFFE_D0A8 (+0a8)	PCIRRDT	PCI Read Retry Discard Timer Register
0xFFFE_D0A4 (+0a4)	PCIRRT_CMD	PCI Read Retry Timer Command Register
0xFFFE_D0A0 (+0a0)	PCIRRT	PCI Read Retry Tag Register
0xFFFE_D09C (+09c)	TCCMD	Target Current Command Register
0xFFFE_D098 (+098)	TIM	Target Interrupt Mask Register
0xFFFE_D094 (+094)	TSTAT	Target Status Register
0xFFFE_D090 (+090)	TC	Target Control Register

Table 12.3.4 PCI Bus Arbiter/Parked Master Registers

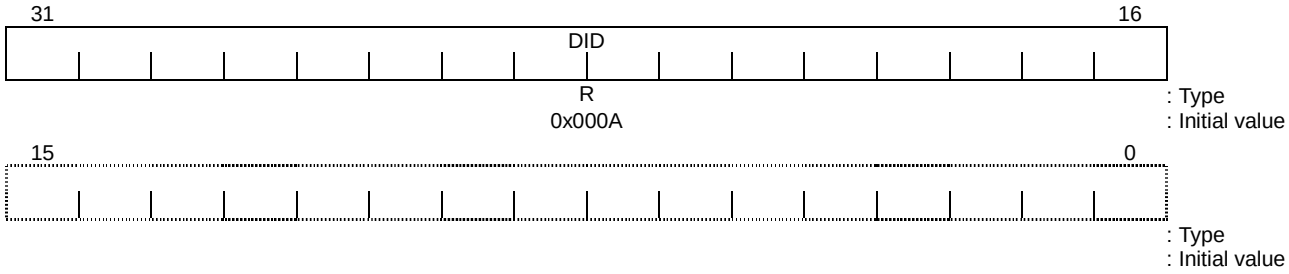
Address	Mnemonic	Register Name
0xFFFE_D11C (+11c)	PBACS	PCI Bus Arbiter Current State Register
0xFFFE_D118 (+118)	CPCIBGS	Current PCI Bus Grant Status Register
0xFFFE_D114 (+114)	CPCIBRS	Current PCI Bus Request Status Register
0xFFFE_D110 (+110)	BM	Broken Master Register
0xFFFE_D10C (+10c)	PBAPMIM	PCI Bus Arbiter/Parked Master Interrupt Mask Register
0xFFFE_D108 (+108)	PBAPMS	PCI Bus Arbiter/Parked Master Status Register
0xFFFE_D104 (+104)	PBAPMC	PCI Bus Arbiter/Parked Master Control Register
0xFFFE_D100 (+100)	REQ_TRACE	Request Trace Register

Table 12.3.5 Local Bus Special Registers

Address	Mnemonic	Register Name
0xFFFE_D158 (+158)	IPCICBE	Initiator Indirect Command/Byte Enable Register
0xFFFE_D154 (+154)	IPCIDATA	Initiator Indirect Data Register
0xFFFE_D150 (+150)	IPCIADDR	Initiator Indirect Address Register
0xFFFE_D14C (+14c)	IOMAS	Initiator I/O Mapping Address Size Register
0xFFFE_D148 (+148)	MMAS	Initiator Memory Mapping Address Size Register
0xFFFE_D144 (+144)	ISCDP	Initiator Special-Cycle Data Port Register
0xFFFE_D140 (+140)	IIADP	Initiator Interrupt-Acknowledge Data Port Register
0xFFFE_D13C (+13c)	ICDR	Initiator Configuration Data Register
0xFFFE_D138 (+138)	ICAR	Initiator Configuration Address Register
0xFFFE_D134 (+134)	PCISTATIM	PCI Status Interrupt Mask Register
0xFFFE_D130 (+130)	LBIM	Local Bus Interrupt Mask Register
0xFFFE_D12C (+12c)	LBSTAT	Local Bus Status Register
0xFFFE_D128 (+128)	LBC	Local Bus Control Register
0xFFFE_D124 (+124)	MBAS	Target Memory Base Address Size Register
0xFFFE_D120 (+120)	IOBAS	Target I/O Base Address Size Register

12.3.2 PCI Configuration Header Space Registers

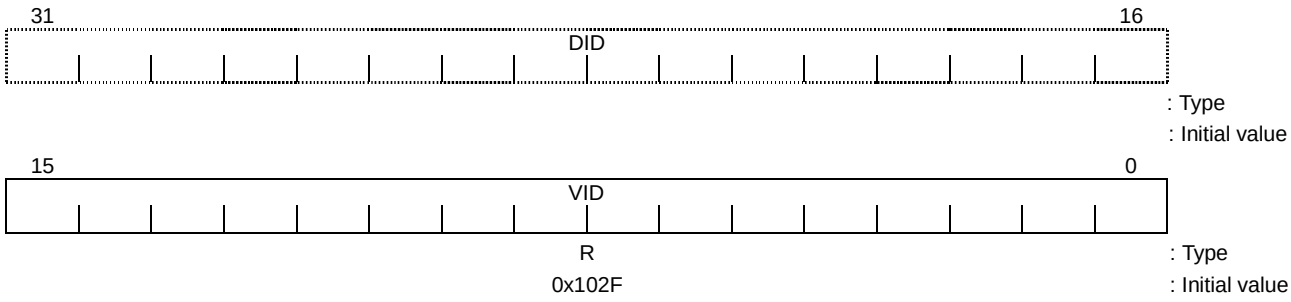
12.3.2.1 Device ID Register (DID) 0xFFFE_D000 (+002)



Bits	Mnemonic	Field Name	Description
31:16	DID	Device ID	Device ID This register contains the device identification number assigned to a particular device. TX3927=0x000A

Figure 12.3.1 Device ID Register

12.3.2.2 Vendor ID Register (VID) 0xFFFE_D002 (+000)

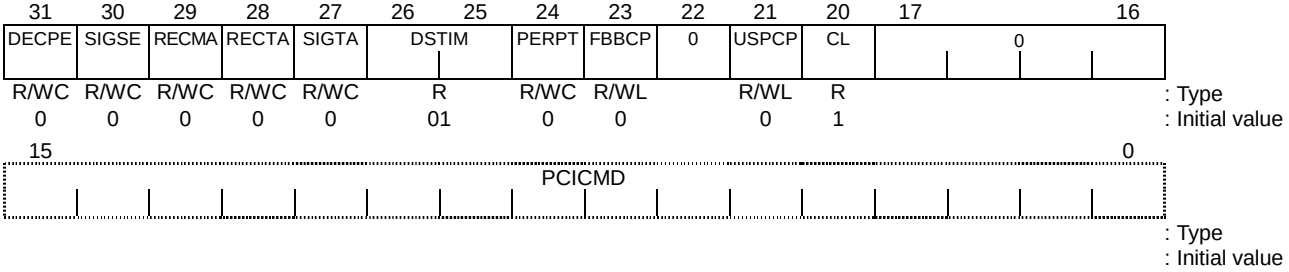


Bits	Mnemonic	Field Name	Description
15:0	VID	Vender ID	Vender ID This register contains the vendor identification number assigned by the PCI SIG. Toshiba=0x102F

Figure 12.3.2 Vendor ID Register

12.3.2.3 PCI Status Register (PCISTAT) 0xFFFE_D004 (+006)

The 16-bit PCI Status register records status information for PCI bus-related events such as PCI master-abort, PCI target-abort and data parity errors, and defines the DEVSEL* timing for the PCIC.



Bits	Mnemonic	Field Name	Description
31	DECPE	Detected Parity Error	Detected Parity Error (initial value: 0) Indicates that a parity error has been detected. This bit is set whenever a parity error is detected by the PCIC initiator or target module even if the Parity Error Response Enable bit of the PCI Command register (PCICMD.PEREN) is cleared. 1: Parity error detected. This bit is set in any of the following three cases: 1. The PCIC initiator module detects a data parity error during a PCI read. 2. The PCIC target module detects a data parity error during a PCI write. 3. The PCIC target module detects an address parity error. 0: Parity error not detected.
30	SIGSE	Signaled System Error	Signaled System Error (initial value: 0) Indicates the SERR* signal state. This bit is set to indicate that the PCIC target has asserted the SERR* pin to report an address parity error, or a data parity error on a special-cycle transaction. 1: SERR* asserted. 0: SERR* not asserted.
29	RECMA	Received Master-Abort	Received Master-Abort (initial value: 0) This bit is set by the PCIC initiator module when its host-to-PCI transaction is terminated with a master-abort (except for a special-cycle command).
28	RECTA	Received Target-Abort	Received Target-Abort (initial value: 0) This bit is set by the PCIC initiator module when its transaction is terminated with a target-abort. 1: Bus master transaction was terminated with a target-abort. 0: Bus master transaction was not terminated with a target-abort.
27	SIGTA	Signaled Target-Abort	Signaled Target-Abort (initial value: 0) This bit is set by the PCIC target module when it terminates a transaction with a target-abort. 1: Bus master transaction was terminated with a target-abort. 0: Bus master transaction was not terminated with a target-abort.
26:25	DSTIM	DEVSEL Timing	Device Select (DEVSEL) Timing (initial value: 01) The PCI 2.1 specification defines three different timings for the assertion of DEVSEL: 00b = fast, 01b = medium, 10b = slow, 11b = reserved. These read-only bits indicate the slowest DEVSEL timing for the PCIC target to claim an access except for the Configuration Read and Configuration Write commands. The TX3927 uses the medium DEVSEL timing (01).

Figure 12.3.3 PCI Status Register (1/2)

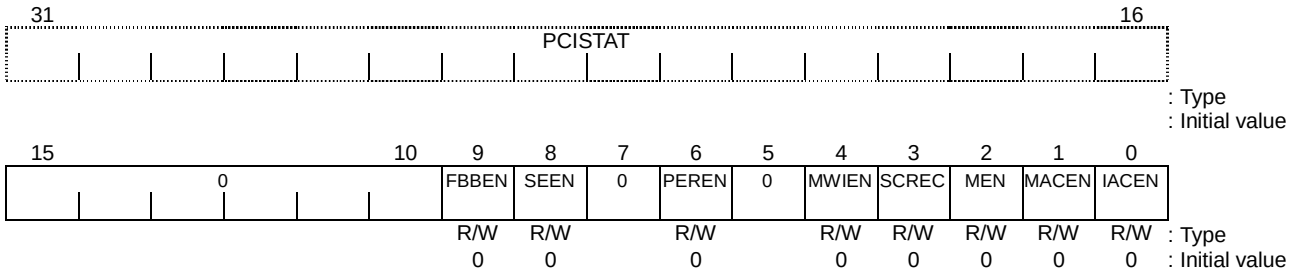
Bits	Mnemonic	Field Name	Description
24	PERPT	Parity Error Reported	Parity Error Reported (initial value: 0) This bit is used by the PCIC initiator to report a parity error to the system software. The target never sets this bit. It is set when all of the following conditions are met: • The PCIC initiator asserts PERR during a read transaction or samples PERR asserted by a target agent during a write transaction. • The PCIC initiator is the bus master for the PCI bus cycle in which the error occurred. • The PEREN bit in the PCICMD register is set.
23	FBBCP	Fast Back-to-Back Capable	Fast Back-to-Back Capable (initial value: 0) This bit indicates whether the PCIC is capable of fast back-to-back transactions. Applications can set this bit to 1 or 0. The default value is 0. 1: Capable of fast back-to-back transactions. 0: Not capable of fast back-to-back transactions.
21	USPCP	Up Speed Capable	Up Speed Capable (initial value: 0) This bit enables 66-MHz operation. However, the TX3927 is not 66-MHz capable.
20	CL	Capabilities List	Capabilities List (initial value: 1) This bit indicates whether the power management capability is implemented.

Note: Bits 31, 30, 29, 28, 27 and 24 are used to record events occurring on the PCI bus. These bits are set to 1 when the corresponding event occurs. These bits are cleared by writing a 1; writing a 0 has no effect. When these bits are currently cleared, writing a 1 has no effect.

Figure 12.3.3 PCI Status Register (2/2)

12.3.2.4 PCI Command Register (PCICMD) 0xFFFE_D006 (+004)

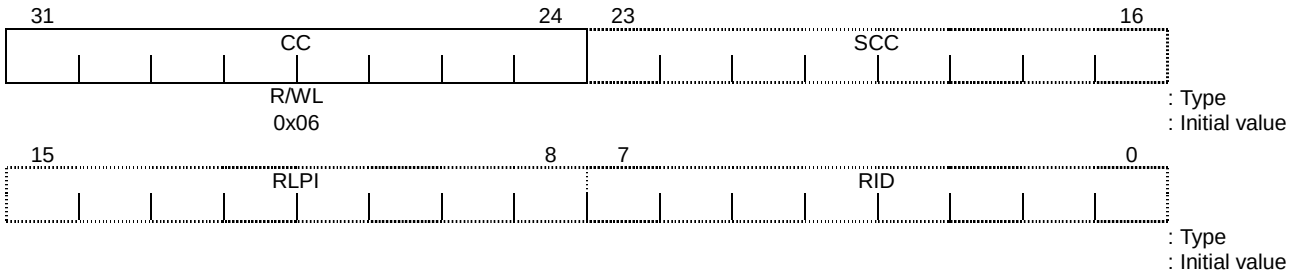
This 16-bit register provides basic control over the PCIC's ability to respond to PCI cycles.



Bits	Mnemonic	Field Name	Description
9	FBBEN	Fast Back-to-Back Enable	Fast Back-to-Back Enable (initial value: 0) The PCIC supports fast back-to-back transactions. During a fast back-to-back transaction, another transaction starts in the clock cycle immediately following the last data transfer for the last transaction. The FBBEN bit enables or disables fast back-to-back transactions. 1: Enabled 0: Disabled
8	SEEN	SERR* Enable	SERR* Enable (initial value: 0) Enables or disables the SERR* driver of the PCIC. When SEEN=1 and PEREN=1, SERR* is asserted for error signaling, including PCI bus address parity errors, and data parity errors on special-cycle transactions. When SEEN=0, SERR* is never asserted. 1: Enabled 0: Disabled
6	PEREN	PERR* Enable	Parity Error Response Enable (initial value: 0) Controls the PCIC's response to PCI address and data parity errors. The PEREN bit is used with the SEEN bit to enable the assertion of the SERR* pin for error signaling, including PCI address and data parity errors. If this bit is cleared, the PCIC ignores all parity errors and behaves as if a nothing is wrong. If this bit is set, the PCIC asserts PERR* when it detects a parity error. 1: PERR* is asserted upon detection of a parity error. If the SEEN bit is also set, SERR* is asserted when a PCI address or data parity error occurs. 0: Parity errors are ignored.
4	MWIEN	Memory Write and Invalidate Enable	Memory Write and Invalidate Enable (initial value: 0) Controls whether the master device generates the Memory-Write-and-Invalidate command. This bit has no effect on the TX3927.
3	SCREC	Special-Cycle Recognition	Special-Cycle Recognition (initial value: 0) 1: The PCIC target monitors all special-cycle operations. 0: The PCIC target ignores all special-cycles operations.
2	MEN	Master Enable	Master Enable (initial value: 0) 1: The PCIC initiator module can act as a bus master. 0: The PCIC initiator module can not act as a bus master.
1	MACEN	Memory Access Enable	Memory Access Enable (initial value: 0) 1: The PCIC target module responds to PCI memory space accesses. 0: The PCIC target module does not respond to PCI memory space accesses.
0	IACEN	I/O Access Enable	I/O Access Enable (initial value: 0) 1: The PCIC target module responds to PCI I/O space accesses. 0: The PCIC target module does not respond to PCI I/O space accesses.

Figure 12.3.4 PCI Command Register

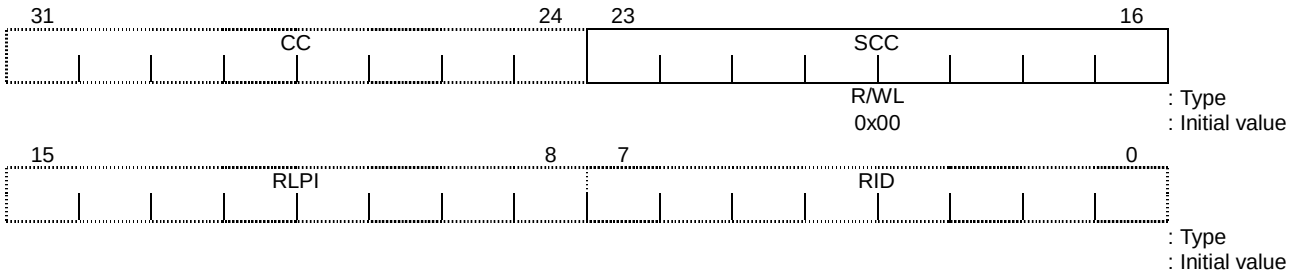
12.3.2.5 Class Code Register (CC) 0xFFFE_D008 (+00b)



Bits	Mnemonic	Field Name	Description
31:24	CC	Class Code	Class Code for PCIC (initial value: 0x6) Contains the base class code of the PCIC. Hardware reset to 06h (=bridge device) for the PCIC.

Figure 12.3.5 Class Code Register

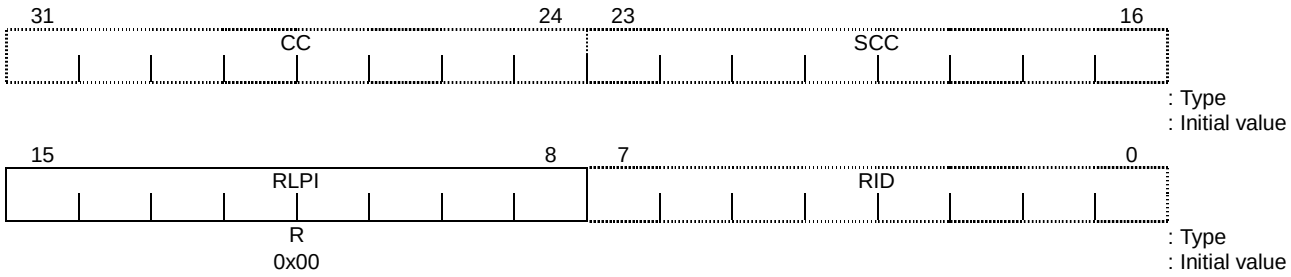
12.3.2.6 Subclass Code Register (SCC) 0xFFFE_D009 (+00a)



Bits	Mnemonic	Field Name	Description
23:16	SCC	Subclass Code	Subclass Code for PCIC (initial value: 0x00) Contains the subclass code of the PCIC. Hardware reset to 00h (=host bridge) for the PCIC.

Figure 12.3.6 Subclass Code Register

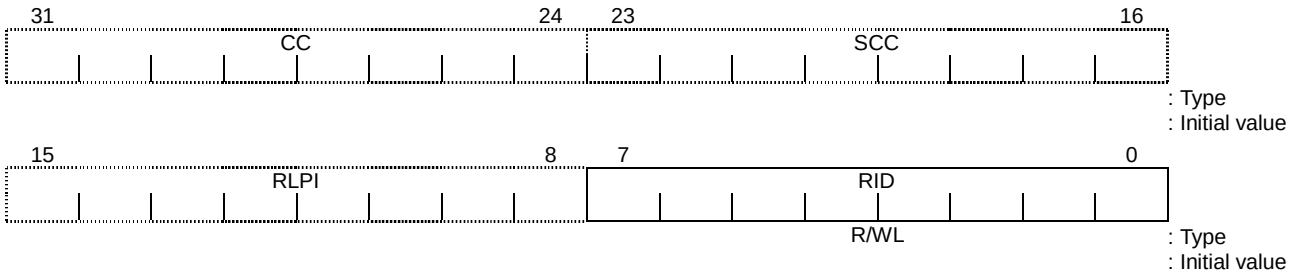
12.3.2.7 Register-Level Programming Interface Register (RLPI) 0xFFFE_D00A (+009)



Bits	Mnemonic	Field Name	Description
15:8	RLPI	Register-Level Programming Interface	Register-Level Programming Interface (initial value: 0x00) Where an industry-standard programming interface is available, this bit defines its classification. This register is not used in the TX3927. Reading this register return a 0.

Figure 12.3.7 Register-Level Programming Interface Register

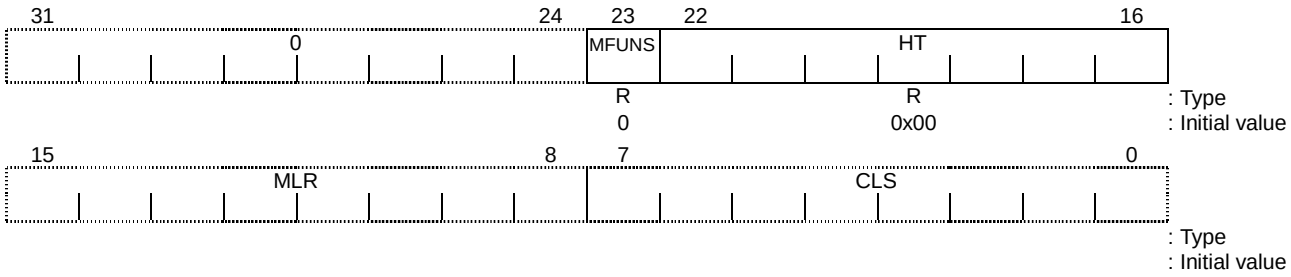
12.3.2.8 Revision ID Register (RID) 0xFFFE_D00B (+008)



Bits	Mnemonic	Field Name	Description
7:0	RID	Revision ID	Revision Identification Number for PCIC Contains the device revision number assigned by Toshiba. Contact the Toshiba technical staff.

Figure 12.3.8 Revision ID Register

12.3.2.9 Header Type Register (HT) 0xFFFE_D00D (+00e)



Bits	Mnemonic	Field Name	Description
23	MFUNS	Multi-Function	Multi-Function Status (initial value: 0) 0: Indicates a single-function device.
22:16	HT	Header Type	Header Type (initial value: 0x00) Defines the PCI configuration space header format. The TX3927 supports 0x00 only. 000000: Header type 0 (not PCI-to-PCI bridge).

Figure 12.3.9 Header Type Register

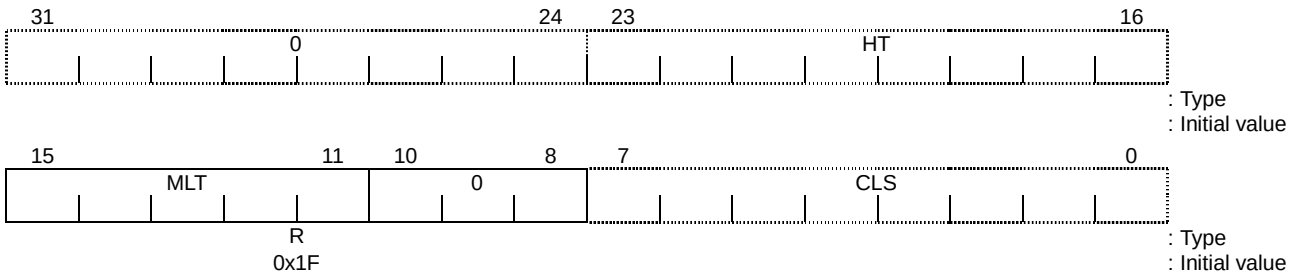
12.3.2.10 Master Latency Timer Register (MLT) 0xFFFE_D00E (+00d)

The 8-bit MLT register defines the maximum amount of time for which the PCIC, as a bus master, can hold the PCI bus to burst data. The timer is triggered automatically when the PCIC becomes the PCI bus master, and is cleared while the PCIC is not asserting FRAME*.

The MLT register begins decrementing when the PCIC asserts FRAME*. If the PCIC completes a transaction before the timer has expired, the MLT count value is ignored.

If the timer has expired before the PCIC concludes its intended transaction, the PCIC initiates a termination transaction as soon as its GNT* is negated. The number of PCI clocks programmed in the MLT register represents the guaranteed time slice allotted to the PCIC (or the total master latency).

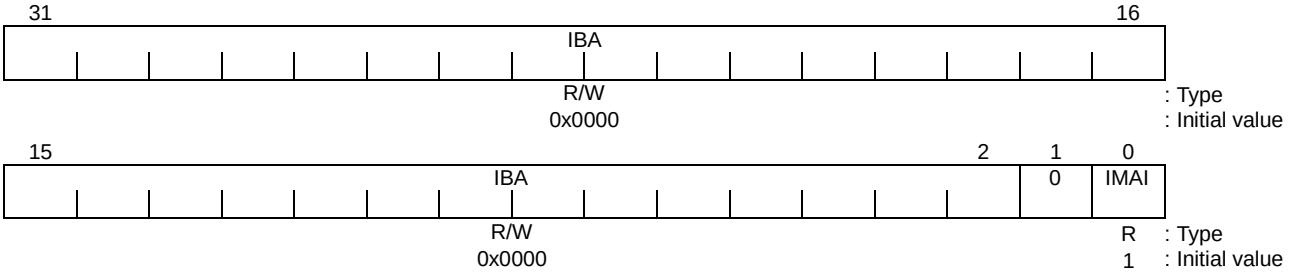
When GNT* is negated after the time slice has run out, the PCIC must immediately relinquish the bus. The total master latency is the value in bits 15-11 multiplied by 8 (because bits 10-8 are read-only as zeros).



Bits	Mnemonic	Field Name	Description
15:11	MLT	Master Latency Timer Count Value	Master Latency Timer Count Value (initial value: 0x1F) Specifies the total master latency. When GNT* is negated during a burst cycle, the PCIC must give up the bus within the programmed number of PCI clocks multiplied by 8.

Figure 12.3.10 Master Latency Timer Register

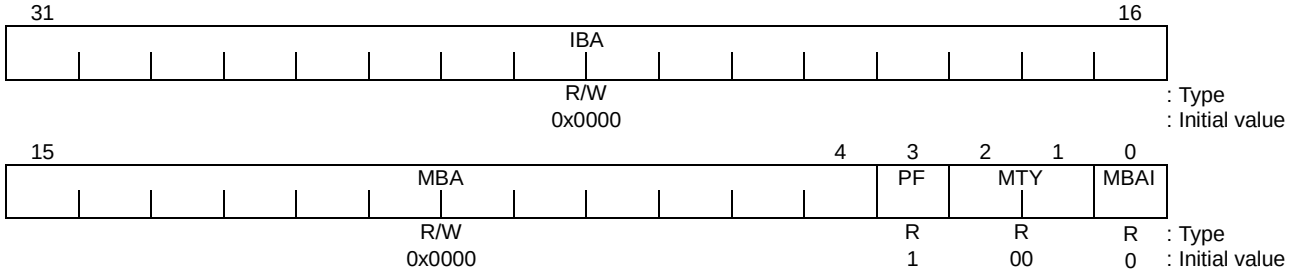
12.3.2.11 Target I/O Base Address Register (IOBA) 0xFFFE_D010 (+010)



Bits	Mnemonic	Field Name	Description
31:2	IBA	I/O Space Base Address	I/O Space Base Address for Target (initial value: 0x00000000) Provides the base address of the PCI I/O space assigned to the target. The Target I/O Base Address Size (IOBAS) register is used to define the size of I/O address space.
0	IMAI	I/O Base Address Indicator	I/O Space Base Address Indicator (fixed value: 1) 1: Indicates PCI I/O space.

Figure 12.3.11 Target I/O Base Address Register

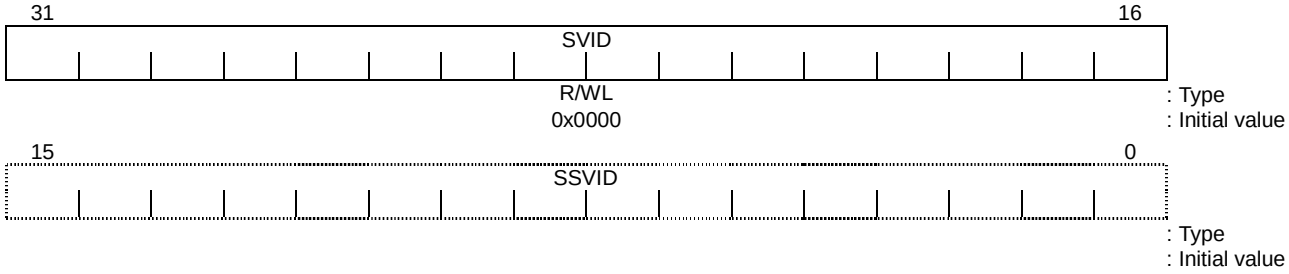
12.3.2.12 Target Memory Base Address Register (MBA) 0xFFFE_D014 (+014)



Bits	Mnemonic	Field Name	Description
31:4	MBA	Memory Base Address	Target Memory Base Address (initial value: 0x0000000) Provides the base address of the PCI memory space assigned to the target. The Target Memory Base Address Size (MBAS) register is used to define the memory space size.
3	PF	Prefetchable	Prefetchable (fixed value: 1) 1: Memory is prefetchable. A memory region is prefetchable when the following two conditions are met: 1. The device returns all bytes on reads, regardless of the state of the byte enables. 2. Reads from this memory space create no side effects. Linear frame buffers used in graphics devices are an example of prefetchable memory.
2:1	MTY	Memory Type	Memory Type (fixed value: 00) 00: The memory space is located anywhere in 32-bit address space.
0	MBAI	Memory Base Address Indicator	Address Space Indicator (fixed value: 0) 0: Indicates PCI memory sapce.

Figure 12.3.12 Target Memory Base Address Register

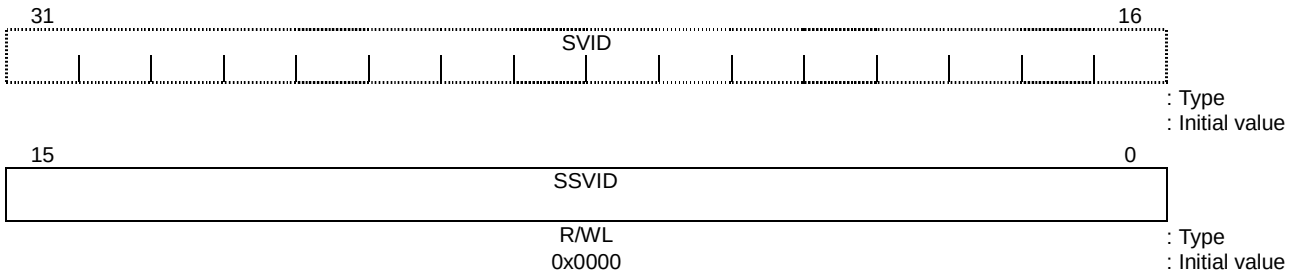
12.3.2.13 System Vendor ID Register (SVID) 0xFFFE_D02C (+02e)



Bits	Mnemonic	Field Name	Description
31:16	SVID	System Vender ID	System Vender ID Value (initial value: 0x0000) This register is used to identify the manufacturer of a subsystem or add-in board containing the PCI device. The system vendor is to obtain a unique identification number from the PCI SIG.

Figure 12.3.13 System Vendor ID Register

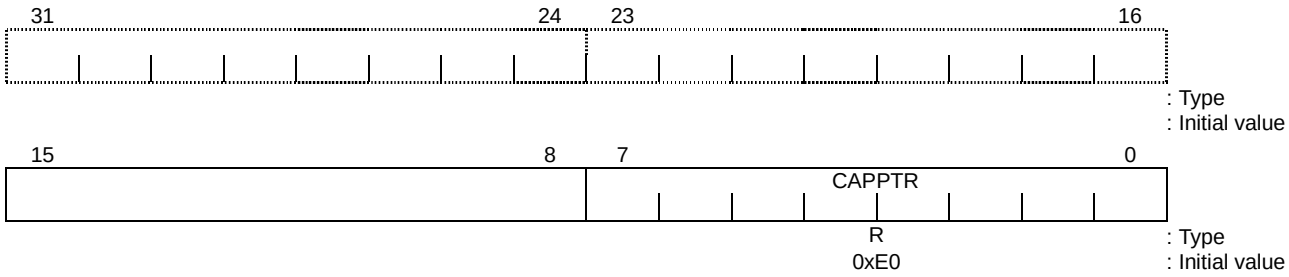
12.3.2.14 Subsystem Vendor ID Register (SSVID) 0xFFFE_D02E (+02c)



Bits	Mnemonic	Field Name	Description
15:0	SSVID	Subsystem Vender ID	Subsystem Vender ID Value (initial value: 0x0000) This register is used to identify the manufacturer of a subsystem or add-in board containing the PCI device. The system vendor is to obtain a unique identification number from the PCI SIG.

Figure 12.3.14 Subsystem Vendor ID Register

12.3.2.15 Capabilities Pointer (CAPPTR) 0xFFFE_D034 (+037)

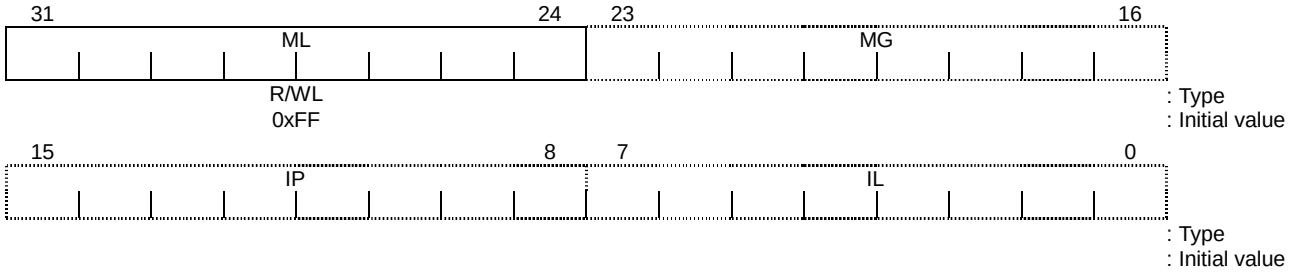


Bits	Mnemonic	Field Name	Description
7:0	CAPPTR	Capabilities Pointer	Capabilities Pointer (initial value: 0xE0) Provides an offset to the starting address of the register block used for the PCI power management function supported by the TX3927.

Figure 12.3.15 Capabilities Pointer

12.3.2.16 Maximum Latency Register (ML) 0xFFFE_D03C (+03f)

This register specifies how quickly the device needs to gain access to the PCI bus. The maximum latency value is specified in units of 250 ns (1/4 μs), assuming a 33-MHz PCICLK rate. This value is used by the configuration software to determine the priority level that the bus arbiter assigns to the PCIC initiator.

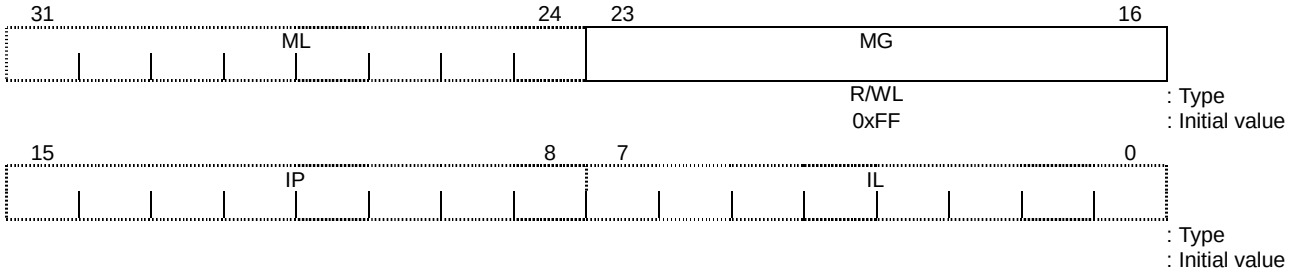


Bits	Mnemonic	Field Name	Description
31:24	ML	Maximum Latency	Maximum Latency Value (initial value: 0xFF) 00h: This register is not used in prioritizing PCI bus accesses. 01h-FFh: Defines how quickly the device requires access to the bus, assuming a PCICLK rate of 33 MHz. Value is a multiple of 250-ns increments.

Figure 12.3.16 Maximum Latency Register

12.3.2.17 Minimum Grant Register (MG) 0xFFFE_D03D (+03e)

This register specifies the length of the burst period required by the device. The minimum grant value is specified in units of 250 ns (1/4 μs), assuming a 33-MHz PCICLK rate. This value is used by applications to determine the latency timer value.

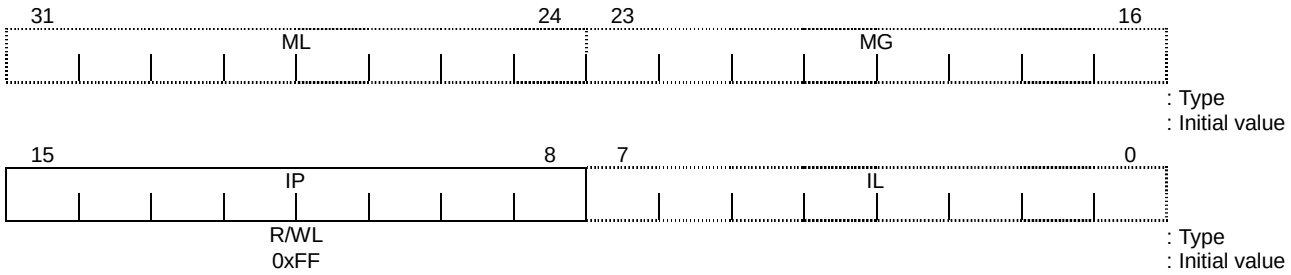


Bits	Mnemonic	Field Name	Description
23:16	MG	Minimum Grant	Minimum Grant Value (initial value: 0xFF) 00h: This register is not used in calculating latency timer values. 01h-FFh: Specifies how long a burst period the device needs, assuming a PCICLK rate of the 33 MHz. Value is a multiple of 250-ns increments.

Figure 12.3.17 Minimum Grant Register

12.3.2.18 Interrupt Pin Register (IP) 0xFFFE_D03E (+03d)

This register is used when the PCIC generates interrupt requests. This register indicates which PCI interrupt line (INTA* through INTD*) the device uses.



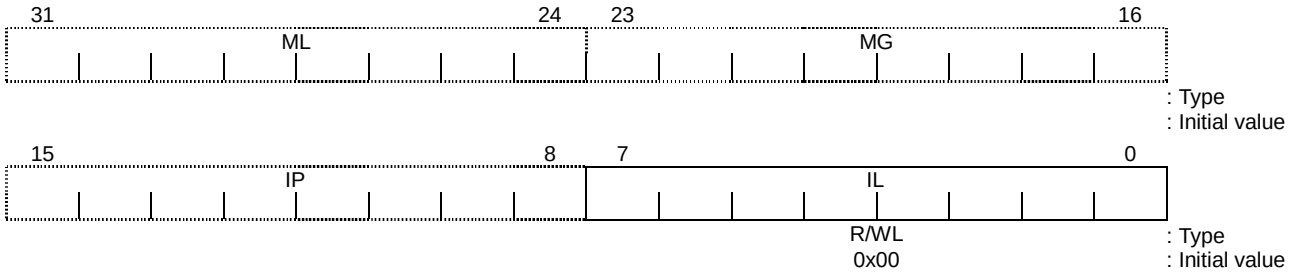
Bits	Mnemonic	Field Name	Description
15:8	IP	Interrupt Pin	Interrupt Pin Value (initial value: 0x01) Valid values are from 00h to 04h. 00h: No interrupt pin 01h: INTA* 02h: INTB* 03h: INTC* 04h: INTD* 05h-FFh: Reserved

Figure 12.3.18 Interrupt Pin Register

12.3.2.19 Interrupt Line Register (IL) 0xFFFE_D03F (+03c)

The Interrupt Line (IL) register indicates which of the system interrupt request lines on the Interrupt Controller the device's PCI interrupt request pin (specified in the Interrupt Pin register) is connected to.

The operating system or device driver can examine the Interrupt Line register to determine which of the system interrupt request line the device will use to issue interrupt service requests.

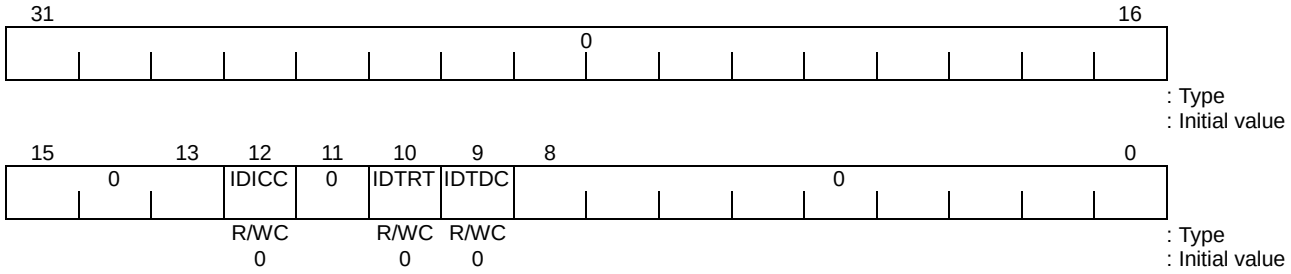


Bits	Mnemonic	Field Name	Description
7:0	IL	Interrupt Line	Interrupt Line Value (initial value: 0x00) Value in this field are system-architecture-specific. 00h-FEh: Interrupt line number that the device is connected to 00h ==> IRQ0 01h ==> IRQ1 0Eh ==> IRQ14 0Fh ==> IRQ15 FFh: No device interrupt line is connected to the system interrupt controller.

Figure 12.3.19 Interrupt Line Register

12.3.3 Initiator Configuration Space Registers

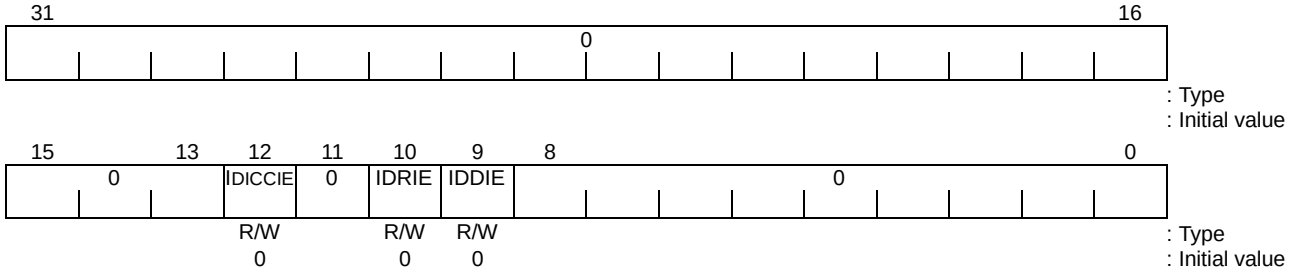
12.3.3.1 Initiator Status Register (ISTAT) 0xFFFE_D044 (+044)



Bits	Mnemonic	Field Name	Description
12	IDICC	Indirect Initiator Command Complete	Indirect Initiator Command Complete (initial value: 0) This bit is set when an indirect initiator command is complete. This bit is cleared by writing a 1.
10	IDTRT	Initiator Detected Target-Retry Cycle	Initiator Detected Target-Retry Cycle (initial value: 0) 1: A target-retry cycle has been detected. This bit is cleared by writing a 1.
9	IDTDC	Initiator Detected Target-Disconnect Cycle	Initiator Detected Target-Disconnect Cycle (initial value: 0) 1: A target-disconnect cycle has been detected. This bit is cleared by writing a 1.

Figure 12.3.20 Initiator Status Register

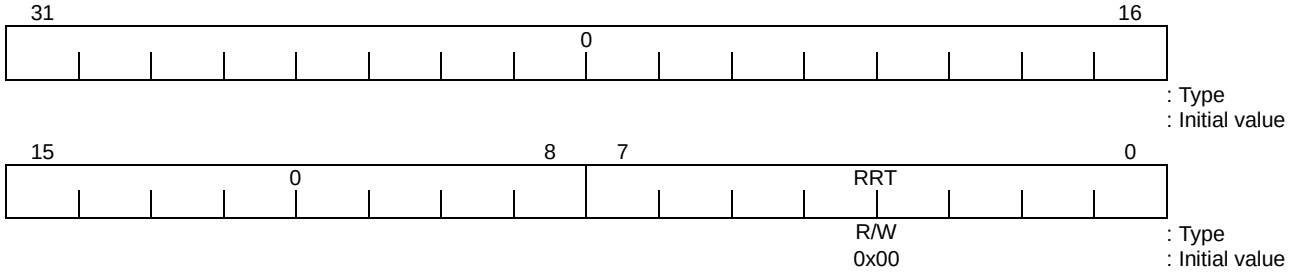
12.3.3.2 Initiator Interrupt Mask Register (IIM) 0xFFFE_D048 (+048)



Bits	Mnemonic	Field Name	Description
12	IDICCIE	Indirect Initiator Command Complete Interrupt Enable	Indirect Initiator Command Complete Interrupt Enable (initial value: 0) Provides a mask for the indirect-initiator command-complete interrupt. 1: Interrupts enabled 0: Interrupts disabled
10	IDRIE	Initiator Detected Target-Retry Cycle Interrupt Enable	Initiator Detected Target-Retry Cycle Interrupt Enable (initial value: 0) Provides a mask for the detected-target-retry-cycle interrupt. 1: Interrupts enabled 0: Interrupts disabled
9	IDDIE	Initiator Detected Target-Disconnect Cycle Interrupt Enable	Initiator Detected Target-Disconnect Cycle Interrupt Enable (initial value: 0) Provides a mask for the detected-target-disconnect-cycle interrupt. 1: Interrupts enabled 0: Interrupts disabled

Figure 12.3.21 Initiator Interrupt Mask Register

12.3.3.3 Retry/Reconnect Timer Register (RRT) 0xFFFE_D04C (+04c)

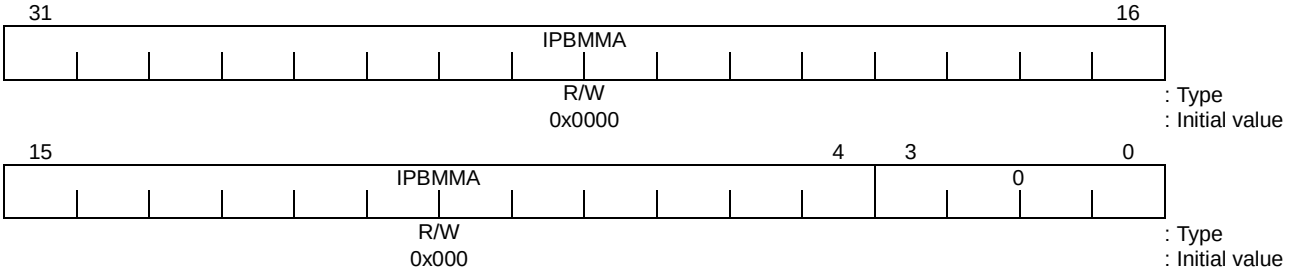


Bits	Mnemonic	Field Name	Description
7:0	RRT	Initiator Auto Retry/Reconnect Timer	Initiator Auto Retry/Reconnect Timer1 (initial value: 0x00) Specify the number of PCI bus clock (PCICLK) cycles before the initiator issues the same transaction request terminated by a target-retry or target-disconnect. This timer automatically begins decrementing when a transaction is target-terminated. The initiator waits for this timer to expire before repeating the same transaction. The TX39/H2 core can read the Initiator Status (ISTAT) register to find out whether the transaction was terminated by a target-retry or a target-disconnect.

Note: The PCI Local Bus Specification, Revision 2.1, suggests a value of 2 to 33 PCI clock cycles.

Figure 12.3.22 Retry/Reconnect Timer Register

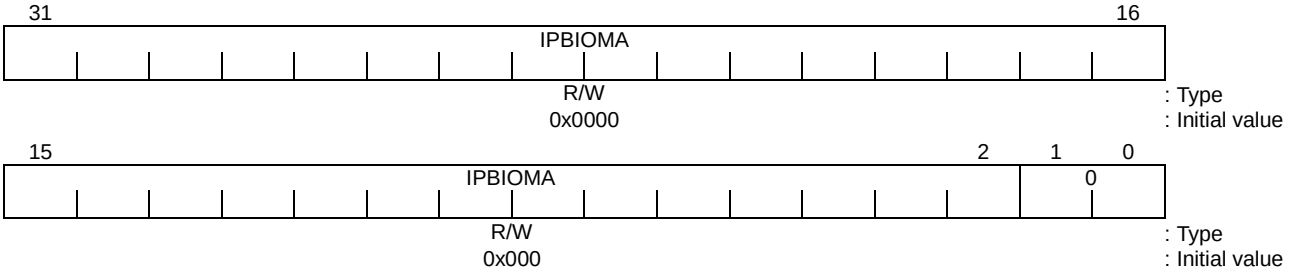
12.3.3.4 Initiator PCI Bus Memory Mapping Address Register (IPBMMAR) 0xFFFE_D05C (+05c)



Bits	Mnemonic	Field Name	Description
31:4	IPBMMA	Initiator PCI Bus Memory Mapping Address	Initiator PCI Bus Memory Mapping Address (initial value: 0x0000000) This register is used to decode a local-to-PCI memory access for a direct initiator memory cycle. It specifies the start address of a PCI memory address region. The range of the memory region is programmed by the Initiator Memory Mapping Address Size (MMAS) register. In local-to-PCI address translation, high-order bits of the local bus (G-Bus) address is replaced by the corresponding bits in this register and concatenated with the remaining low-order bits of the G-Bus address. This register is used for PCI memory read and PCI memory write commands. Because the IPBMMAR is internally masked by the MMAS, the MMAS must be programmed before programming the IPBMMAR.

Figure 12.3.23 Initiator PCI Bus Memory Mapping Address Register

12.3.3.5 Initiator PCI Bus I/O Mapping Address Register (IPBIOMAR) 0xFFFE_D060 (+060)

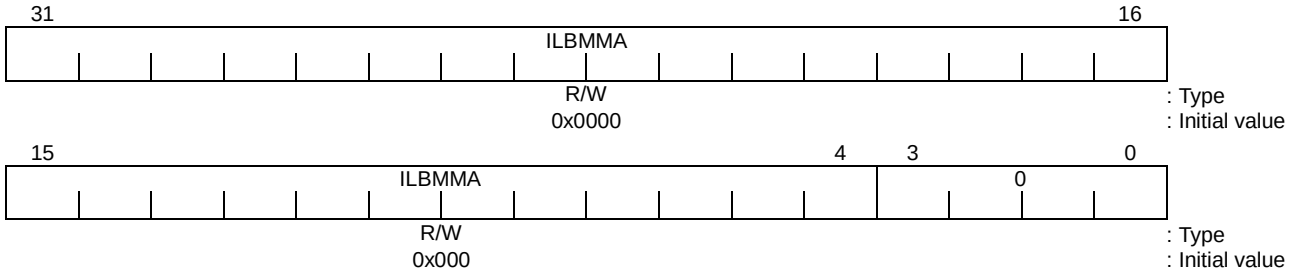


Bits	Mnemonic	Field Name	Description
31:2	IPBIOMA	Initiator PCI Bus I/O Mapping Address	Initiator PCI Bus I/O Mapping Address (initial value: 0x00000000) This register is used to decode a local-to-PCI I/O access for a direct initiator memory cycle. It specifies the start address of a PCI I/O address region. The range of the I/O space is programmed by the Initiator I/O Mapping Address Size (IOMAS) register. In local-to-PCI address translation, high-order bits of the local bus (G-Bus) address is replaced by the corresponding bits in this register and concatenated with the remaining low-order bits of the G-Bus address. This register is used for PCI I/O read and PCI I/O write commands. Because the IPBIOMAR is internally masked by the IOMAS, the IOMAS must be programmed before programming the IPBIOMAR.

Figure 12.3.24 Initiator PCI Bus I/O Mapping Address Register

12.3.3.6 Initiator Local Bus Memory Mapping Address Register (ILBMMAR)

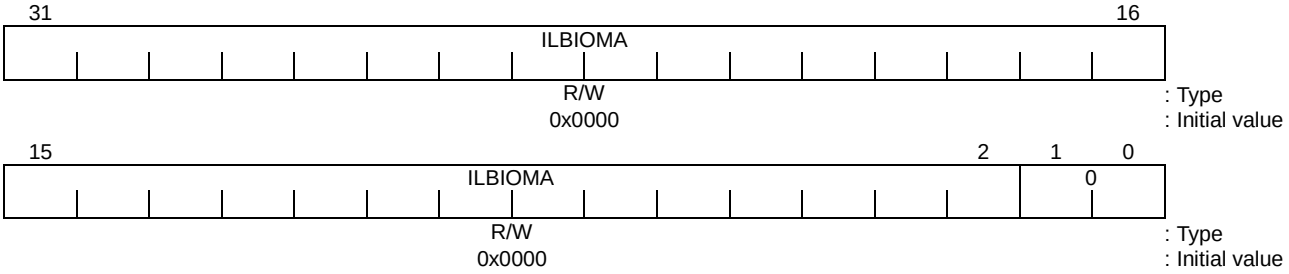
0xFFFE_D064 (+064)



Bits	Mnemonic	Field Name	Description
31:4	ILBMMA	Initiator Local Bus Memory Mapping Address	Initiator Local Bus Memory Mapping Address (initial value: 0x0000000) This register is used to decode a local-to-PCI memory access for a direct initiator memory cycle. High-order bits of an incoming local bus (G-Bus) address are compared with the value of this register, as programmed by the Initiator Memory Mapping Address Size (MMAS) register. A match occurs when G-Bus address falls in the address range specified by this register and MMAS; then the PCIC initiator generates a single PCI memory transaction. When a match occurs, a PCI address is generated by concatenating high-order bits of the IPBMMAR register with the remaining lower-order bits of the local bus (G-Bus) address. Because the ILBMMAR is internally masked by the MMAS, the MMAS must be programmed before programming the ILBMMAR.

Figure 12.3.25 Initiator Local Bus Memory Mapping Address Register

12.3.3.7 Initiator Local Bus I/O Mapping Address Register (ILBIOMAR) 0xFFFE_D068 (+068)

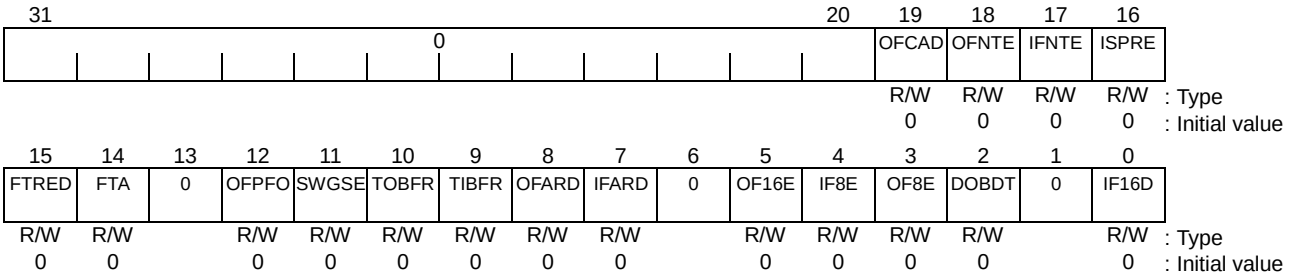


Bits	Mnemonic	Field Name	Description
31:2	ILBIOMA	Initiator Local Bus I/O Mapping Address	Initiator Local Bus I/O Mapping Address (initial value: 0x00000000) This register is used to decode a local-to-PCI I/O access for a direct initiator I/O cycle. High-order bits of an incoming local bus (G-Bus) address are compared with the value of this register, as programmed by the Initiator I/O Mapping Address Size (IOMAS) register. A match occurs when the G-Bus address falls in the address range specified by this register and IOMAS; then the PCIC initiator generates a single PCI I/O transaction. When a match occurs, a PCI address is generated by concatenating high-order bits of the IPBIOMAR register with the remaining low-order bits of the local bus (G-Bus) address. Because the ILBIOMAR is internally masked by the IOAMS, the IOMAS must be programmed before programming the ILBIOMAR.

Figure 12.3.26 Initiator Local Bus I/O Mapping Address Register

12.3.4 Target Configuration Space registers

12.3.4.1 Target Control Register (TC) 0xFFFE_D090 (+090)



Bits	Mnemonic	Field Name	Description
19	OFCAD	OFIFO Caching Enable	OFIFO Caching Capability Disable (initial value: 0) Controls the OFIFO data cache. 1: Unused OFIFO data is discarded after a PCI transaction is completed. 0: Unused OFIFO data is retained. If the next PCI read requires data already in the OFIFO, then the data is returned immediately. Otherwise, the OFIFO is reset (i.e., the OFIFO is flushed) and the new data is fetched via the local bus. If a PCI write address to the IFIFO falls within the range of addresses of data currently in the OFIFO, then the OFIFO is reset (in order to prevent the OFIFO from returning stale data that has been changed after being read).
18	OFNTE	OFIFO 16/8 Clocks Never Time-Out Enable	OFIFO 16/8 Clocks Never Time-Out Enable (initial value: 0) Specifies whether to observe the PCI 8- and 16-clock rules. 1: When the OFIFO 16-clock rule is enabled (OF16E=1), the OFIFO will not time out according to the PCI 16-clock rule for the first data phase. When the OFIFO 8-clock rule is enabled (OF8E=1), the OFIFO will not time out according to the PCI 8-clock rule for a subsequent data phase. In either case, the target does not issue a retry or disconnect to the PCI initiator. 0: If the OFIFO 16/8-clock rule is enabled (OF16E=1 / OF8E=1), the OFIFO will observe the 16/8-clock rule and will issue a retry or disconnect to the PCI initiator when a time-out occurs.
17	IFNTE	IFIFO 8 Clocks Never Time-Out Enable	IFIFO 8 Clocks Never Time-Out Enable (initial value: 0) 1: When the IFIFO 8-clock rule is enabled (IF8E=1), the IFIFO will ignore the PCI 8-clock rule and will not issue a disconnect to the PCI initiator even when a time-out occurs. 0: When the IFIFO 8-clock rule is enabled, the IFIFO will observe the PCI 8-clock rule and will issue a disconnect to the PCI initiator when a time-out occurs.
16	ISPRE	I/O Space Prefetch Enable	I/O Space Prefetch Enable (initial value: 0) 1: I/O space is prefetchable. The I/O read and I/O write commands are treated the same as the memory read and memory write commands. 0: I/O space is not prefetchable.
15	FTRED	Force Target Retry/Disconnect	Force Target Retry/Disconnect (initial value: 0) This bit allows software to terminate the current transaction. It is useful when the host system is ready to shut down and wants to force the PCI target to terminate the current transaction. 1: Causes the PCIC target module to generate a retry termination (if no data has been transferred) or a disconnect termination (if some data has been transferred). This bit is automatically cleared on transaction termination. 0: Writing a 0 to this bit has no effect.
14	FTA	Force Target-Abort	Force Target-Abort (initial value: 0) This bit allows software to abort the transaction. It is useful when the host system wants to force the PCI target to terminate the current PCI transaction. 1: Causes the PCIC target module to generate a target-abort termination. This bit is automatically cleared at the end of the operation. 0: Writing a 0 to this bit has no effect.

Figure 12.3.27 Target Control Register (1/3)

Bits	Mnemonic	Field Name	Description
12	OFPFO	Single Burst Enable	Single Burst Enable (initial value: 0) 1: The OFIFO performs only one read transaction on the local bus. The burst-read size is determined by the Target Burst Length (TBL) register. 0: Allows data streaming from the local bus. The OFIFO accepts data from the local bus as long as it is not full, or until the entire PCI transaction completes. While the PCI transaction is in progress or a subsequent PCI transaction is in progress (provided OFIFO caching is enabled by OFCAE=0), the OFIFO sustains the streaming from the local bus as long as there is room for the next burst data in the OFIFO (the size of which is programmed in the Target Burst Length register).
11	SWGSE	Software-Generated System Error Enable	Software-Generated System Error (initial value: 0) 1: If the SERR Enable (SEEN) bit of the PCI Command register is set, writing a 1 to this bit cause the SERR* signal to be asserted. 0: Writing a 0 to this bit has no effect.
10	TOBFR	Target Outbound FIFO Reset	Target Outbound FIFO Reset (initial value: 0) Once this bit is set, it remains set until a 0 is written. 1: Resets the target OFIFO. 0: Activates the target OFIFO.
9	TIBFR	Target Inbound FIFO Reset	Target Inbound FIFO Reset Once this bit is set, it remains set until a 0 is written. 1: Resets the target IFIFO. 0: Activates the target IFIFO.
8	OFARD	OFIFO Address Range Check Disable	OFIFO Address Range Check Disable (initial value: 0) The target memory base address (MBA) and target I/O base address (IOBA) specify the location of the PCIC address space. The last four D-words of this space are reserved for the PCIC. If this bit is cleared, the PCIC will issue a target-abort if an attempt is made by an external PCI bus master to access addresses outside the defined PCI address space or to access the reserved area. 1: Disables OFIFO address range checking. 0: Enables OFIFO address range checking.
7	IFARD	IFIFO Address Range Check Disable	IFIFO Address Range Check Disable (initial value: 0) The target memory base address (MBA) and target I/O base address (IOBA) specify the location of the PCIC address space. The last four D-words of this space are reserved for the PCIC. If this bit is cleared, the PCIC will issue a target-abort if an attempt is made by an external PCI bus master to access addresses outside the reserved area. 1: Disables IFIFO address range checking. 0: Enables IFIFO address range checking.
5	OF16E	OFIFO 16-Clocks Rule Enable	OFIFO (PCI Read) 16-Clock Rule Enable (initial value: 0) Controls the PCIC operation regarding the OFIFO 16-clock rule. 1: If the OFIFO is not ready to send out the first data of a burst in response to a master read request within 16 PCI clock cycles, the PCIC target state machine disconnects the PCI bus master. (This mode is for fast memory devices capable of delivering the first read data to the PCI bus within 16 PCI clock cycles.) 0: If the OFIFO is not ready to send out the first data of a burst in response to a master read request within 16 PCI clock cycles, the PCIC target module issues a retry to the PCI bus master. (This mode is for slow memory devices that are not able to deliver the first read data to the PCI bus within 16 PCI clock cycles.) In either case, the PCI master is required to retry the transaction before the OFIFO discard timer expires or the transaction is discarded.
4	IF8E	IFIFO 8-Clock Rule Enable	IFIFO (PCI Write) 8-Clock Rule Enable (initial value: 0) 1: Enables target IFIFO 8-clock rule checking. 0: Disables target IFIFO 8-clock rule checking.
3	OF8E	OFIFO 8-Clock Rule Enable	OFIFO (PCI Read) 8-Clock Rule Enable (initial value: 0) 1: Enables target OFIFO 8-clock rule checking. 0: Disables target OFIFO 8-clock rule checking.

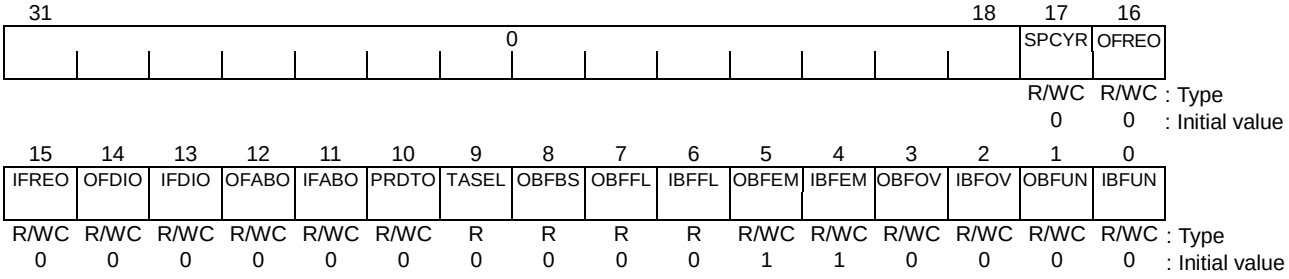
Figure 12.3.27 Target Control Register (2/3)

Bits	Mnemonic	Field Name	Description
2	DOBDT	Disable Outbound Discard Timer	Disable Outbound (Read) Discard Timer (initial value: 0) 1: Disables the read discard timer. 0: Enables the read discard timer.
0	IF16D	IFIFO 16-Clock Rule Disable	IFIFO (Write) 16-Clock Rule Disable (initial value: 0) 1: The target immediately requests the PCI bus master to retry the transaction if the IFIFO is not ready. 0: The target waits for 16 PCI clock cycles before requesting the PCI initiator to retry the transaction when the IFIFO is not ready (i.e., the previous transaction has not been completed on the local bus).

Figure 12.3.27 Target Control Register (3/3)

12.3.4.2 Target Status Register (TSTAT) 0xFFFE_D094 (+094)

Each bit in this register corresponds to an enable bit in the Target Interrupt Mask (TIM) register. Setting a bit in this register triggers an interrupt if the corresponding TIM register bit is set.



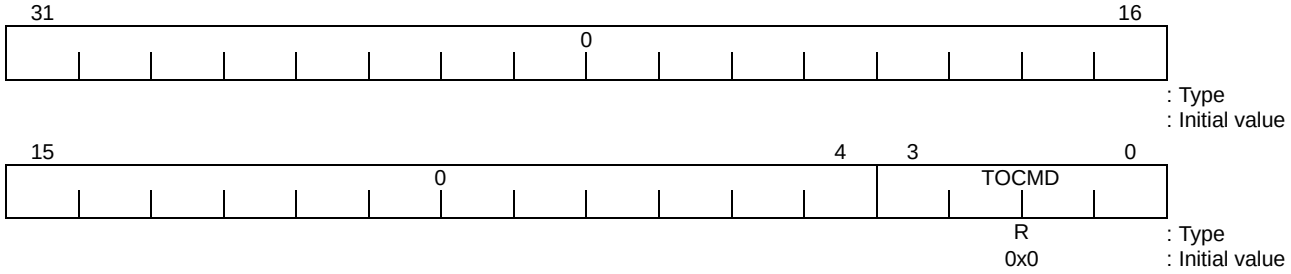
Bits	Mnemonic	Field Name	Description
17	SPCYR	Special-Cycle Received	Special-Cycle Received (initial value: 0) This bit is set when special-cycle data is received. 1: Special-cycle data has been received. 0: Special-cycle data has not been received.
16	OFREO	OFIFO Retry Occurred	OFIFO Retry Occurred (initial value: 0) This bit is set when the PCIC target state machine issues target-retry during a PCI read transaction. 1: A retry has been issued. 0: A retry has not been issued.
15	IFREO	IFIFO Retry Occurred	IFIFO Retry Occurred (initial value: 0) This bit is set when the PCIC target state machine issues target-retry during a PCI write transaction. 1: A retry has been issued. 0: A retry has not been issued.
14	OFDIO	OFIFO Disconnect Occurred	OFIFO Disconnect Occurred (initial value: 0) This bit is set when the PCIC target state machine issues a target-disconnect during a PCI read transaction. 1: A disconnect has been issued. 0: A disconnect has not been issued.
13	IFDIO	IFIFO Disconnect Occurred	IFIFO Disconnect Occurred (initial value: 0) This bit is set when the PCIC target state machine issues a target-disconnect during a PCI write transaction. 1: A disconnect has been issued. 0: A disconnect has not been issued.
12	OFABO	OFIFO Abort Occurred	OFIFO Abort Occurred (initial value: 0) This bit is set when the PCIC target detects a master attempting a PCI read to an address outside the defined memory space.
11	IFABO	IFIFO Abort Occurred	IFIFO Abort Occurred (initial value: 0) This bit is set when the target detects the master attempting a PCI write to an address outside the defined memory space.
10	PRDTO	PCI Read Discard Timer Time-Out	PCI Read Discard Timer Time-out (initial value: 0) This bit is set when the read discard timer expires. The bit is valid when the Disable Outbound Discard Timer (DOBDT) bit is cleared in the Target Control (TC) register. 1: Discard time-out has occurred. 0: Discard time-out has not occurred.

Figure 12.3.28 Target Status Register (1/2)

Bits	Mnemonic	Field Name	Description
9	TASEL	Target Selected	Target Selected (initial value: 0) This bit indicates the state of the PCIC DEVSEL* signal. The bit is set when PCIC is selected as a target. The bit is automatically cleared when the transaction terminates (i.e., FRAME* is deasserted). The local host can read the Target Current Command (TCCMD) register to find which PCI command is being executed.
8	OBFBS	Outbound FIFO Busy	Outbound (PCI Read) FIFO Busy (initial value: 0) This bit is set when the target is selected with a memory or I/O read command and remains set until the master terminates the transaction (i.e., until completion of the current PCI command). When the OBFBS bit is set, subsequent outbound memory or I/O commands (C_BE0*=0) are not latched into the PCI Read Retry Tag (PCIRRT) register.
7	OBFFL	Outbound FIFO Full	Outbound FIFO Full Flag (initial value: 0) 1: The OFIFO is full. 0: The OFIFO is not full.
6	IBFFL	Inbound FIFO Full	Inbound FIFO Full Flag (initial value: 0) 1: The IFIFO is full. 0: The IFIFO is not full.
5	OBFEM	Outbound FIFO Empty	Outbound FIFO Empty Flag (initial value: 0) 1: The OFIFO is empty. 0: The OFIFO is not empty.
4	IBFEM	Inbound FIFO Empty	Inbound FIFO Empty Flag (initial value: 0) 1: The IFIFO is empty. 0: The IFIFO is not empty.
3	OBFOV	Outbound FIFO Overrun	Outbound FIFO Overrun Flag (initial value: 0) This bit is set if an attempt is made to write to the OFIFO when the Outbound FIFO Full (OBFFL) bit is set. The data is corrupted; the TX39/H2 core should send either the SERR* signaling or software-triggered the PCI bus master target-abort. An interrupt is generated if the corresponding Target Interrupt Enable bit is set.
2	IBFOV	Inbound FIFO Overrun	Inbound FIFO Overrun Flag (initial value: 0) This bit is set if an attempt is made to write to the IFIFO when the Outbound FIFO Full (IBFFL) bit is set. The data is corrupted; the TX39/H2 core should send either the SERR* signaling or software-triggered target-abort to the PCI bus master. An interrupt is generated if the corresponding target interrupt enable bit is set.
1	OBFUN	Outbound FIFO Underrun	Outbound FIFO Underrun Flag (initial value: 0) This bit is set if an attempt is made to read from the OFIFO when the Outbound FIFO Empty (OBFEM) bit is set. The data is corrupted; the TX39/H2 core should send either the SERR* signaling or software-triggered target-abort to the PCI bus master. An interrupt is generated if the corresponding target interrupt enable bit is set.
0	IBFUN	Inbound FIFO Underrun	Inbound FIFO Underrun Flag (initial value: 0) This bit is set by an attempt is made to read from the IFIFO when the Inbound FIFO Empty (IBFEM) bit is set. The data is corrupted; the TX39/H2 core should send either the SERR* signaling or software-triggered target-abort to the PCI bus master. An interrupt is generated if the corresponding target interrupt enable bit is set.

Figure 12.3.28 Target Status Register (2/2)

12.3.4.3 Target Current Command Register (TCCMD) 0xFFFE_D09C (+09c)

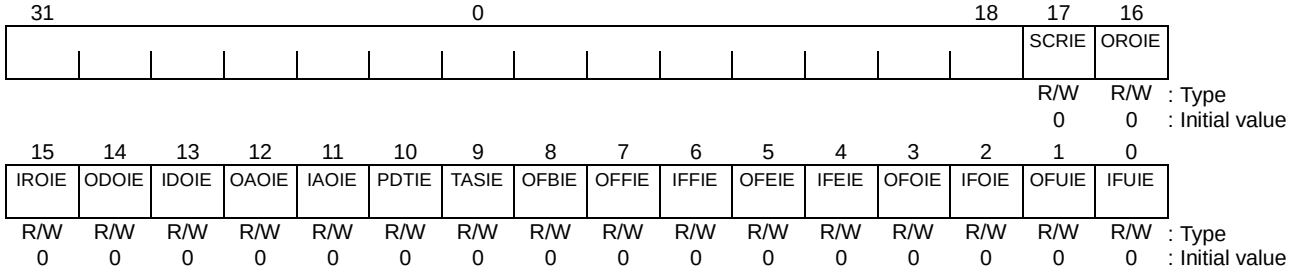


Bits	Mnemonic	Field Name	Description
3:0	TCCMD	Target Current Command	Target Current Command (initial value: 0x0) Indicates which PCI command is currently being executed during a target PCIC access.

Figure 12.3.29 Target Current Command Register

12.3.4.4 Target Interrupt Mask Register (TIM) 0xFFFE_D098 (+098)

Each bit in this register has a corresponding bit in the Target Status (TSTAT) register. Setting a bit in the TSTAT register triggers an interrupt if the corresponding enable bit in this register is set.



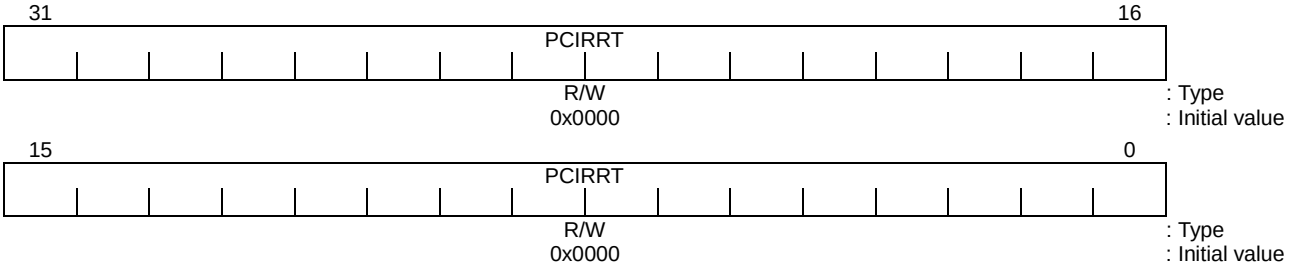
Bits	Mnemonic	Field Name	Description
17	SCRIE	Special-Cycle Received Interrupt Enable	Special-Cycle Received Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.SPCYR bit is set. 0: An interrupt request is not generated even if the TSTAT.SPCYR bit is set.
16	OROIE	OFIFO Retry Occurred Interrupt Enable	OFIFO Retry Occurred Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.OFREO bit is set. 0: An interrupt request is not generated even if the TSTAT.OFREO bit is set.
15	IROIE	IFIFO Retry Occurred Interrupt Enable	IFIFO Retry Occurred Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.IFREO bit is set. 0: An interrupt request is not generated even if the TSTAT.IFREO bit is set.
14	ODOIE	OFIFO Disconnect Occurred Interrupt Enable	OFIFO Disconnect Occurred Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.OFDIO bit is set. 0: An interrupt request is not generated even if the TSTAT.OFDIO bit is set.
13	IDOIE	IFIFO Disconnect Occurred Interrupt Enable	IFIFO Disconnect Occurred Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.IFDIO bit is set. 0: An interrupt request is not generated even if the TSTAT.IFDIO bit is set.
12	OAOIE	OFIFO Abort Occurred Interrupt Enable	OFIFO Abort Occurred Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.OFABO bit is set. 0: An interrupt request is not generated even if the TSTAT.OFABO bit is set.
11	IAOIE	IFIFO Abort Occurred Interrupt Enable	IFIFO Abort Occurred Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.IFABO bit is set. 0: An interrupt request is not generated even if the TSTAT.IFABO bit is set.
10	PDTIE	PCI Read Discard Timer Time-Out Interrupt Enable	PCI Read Discard Timer Time-Out Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.PRDTO bit is set. 0: An interrupt request is not generated even if the TSTAT.PRDTO bit is set.
9	TASIE	Target Selected Interrupt Enable	Target Selected Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.TASEL bit is set. 0: An interrupt request is not generated even if the TSTAT.TASEL bit is set.
8	OFBIE	Outbound (Read) FIFO Busy Interrupt Enable	Outbound (Read) FIFO Busy Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.OBFBS bit is set. 0: An interrupt request is not generated even if the TSTAT.OBFBS bit is set.
7	OFFIE	Outbound FIFO Full	Outbound FIFO Full (initial value: 0) 1: An interrupt request is generated if the TSTAT.OBFFL bit is set. 0: An interrupt request is not generated even if the TSTAT.OBFFL bit is set.
6	IFFIE	Inbound FIFO Full Interrupt Enable	Inbound FIFO Full Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.IBFFL bit is set. 0: An interrupt request is not generated even if the TSTAT.IBFFL bit is set.

Figure 12.3.30 Target Interrupt Mask Register (1/2)

Bits	Mnemonic	Field Name	Description
5	OFEIE	Outbound Empty Interrupt Enable	Outbound Empty Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.OBFEM bit is set. 0: An interrupt request is not generated even if the TSTAT.OBFEM bit is set.
4	IFEIE	Inbound FIFO Empty Interrupt Enable	Inbound FIFO Empty Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.IBFEM bit is set. 0: An interrupt request is not generated even if the TSTAT.IBFEM bit is set.
3	OFOIE	Outbound FIFO Overrun (overflow) Interrupt Enable	Outbound FIFO Overrun (overflow) Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.OBFOV bit is set. 0: An interrupt request is not generated even if the TSTAT.OBFOV bit is set.
2	IFOIE	Inbound FIFO Overrun (overflow) Interrupt Enable	Inbound FIFO Overrun (overflow) Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.IBFOV bit is set. 0: An interrupt request is not generated even if the TSTAT.IBFOV bit is set.
1	OFUIE	Outbound FIFO Underrun (underflow) Interrupt Enable	Outbound FIFO Underrun (underflow) Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.OBFUN bit is set. 0: An interrupt request is not generated even if the TSTAT.OBFUN bit is set.
0	IFUIE	Inbound FIFO Underrun (underflow) Interrupt Enable	Inbound FIFO Underrun (underflow) Interrupt Enable (initial value: 0) 1: An interrupt request is generated if the TSTAT.IBFUN bit is set. 0: An interrupt request is not generated even if the TSTAT.IBFUN bit is set.

Figure 12.3.30 Target Interrupt Mask Register (2/2)

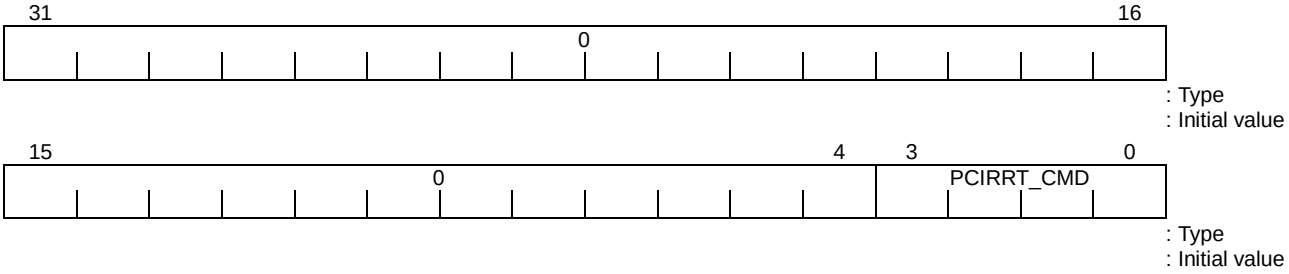
12.3.4.5 PCI Read Retry Tag Register (PCIRRT) 0xFFFE_D0A0 (+0a0)



Bits	Mnemonic	Field Name	Description
31:0	PCIRRT	PCI Read Retry Tag Address	PCI Read Retry Tag Address (initial value: 0x00000000) Contains the tag address of the command that has caused a read retry.

Figure 12.3.31 PCI Read Retry Tag Register

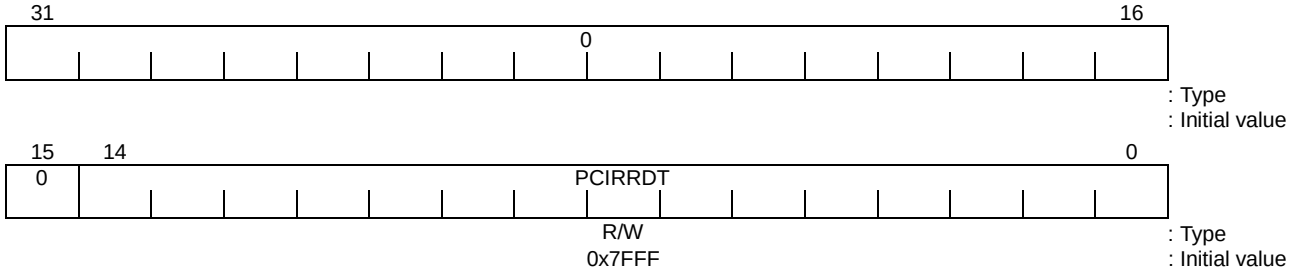
12.3.4.6 PCI Read Retry Timer Command Register (PCIRRT_CMD) 0xFFFE_D0A4 (+0a4)



Bits	Mnemonic	Field Name	Description
3:0	PCIRRT_CMD	PCI Read Retry Timer Command	PCI Read Retry Timer Command (initial value: 0x0) Contains the PCI command that has caused a read retry.

Figure 12.3.32 PCI Read Retry Timer Command Register

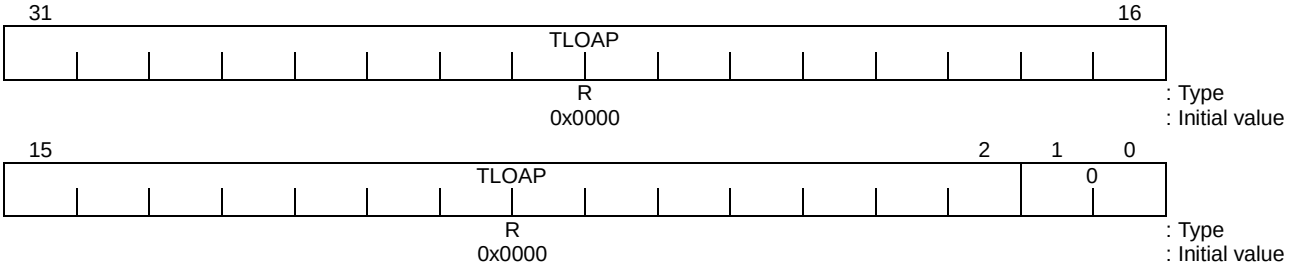
12.3.4.7 PCI Read Retry Discard Timer Register (PCIRRDT) 0xFFFE_D0A8 (+0a8)



Bits	Mnemonic	Field Name	Description
14:0	PCIRRDT	PCI Read Retry Discard Timer	PCI Read Retry Discard Timer (initial value: 0x7FFF) Defines the period of time (as the number of PCICLK cycles) in which the bus master can retry a read transaction. At the beginning of an outbound delayed transaction, the discard timer is loaded with this value and starts counting down. The bus master is required to retry a delayed transaction before the timer expires. Otherwise, the PCI target module will discard the data in the FIFO. When the discard timer expires, the PCI Read Discard Timer Time-Out (PRDTO) bit in the TSTAT register is set.

Figure 12.3.33 PCI Read Retry Discard Timer Register

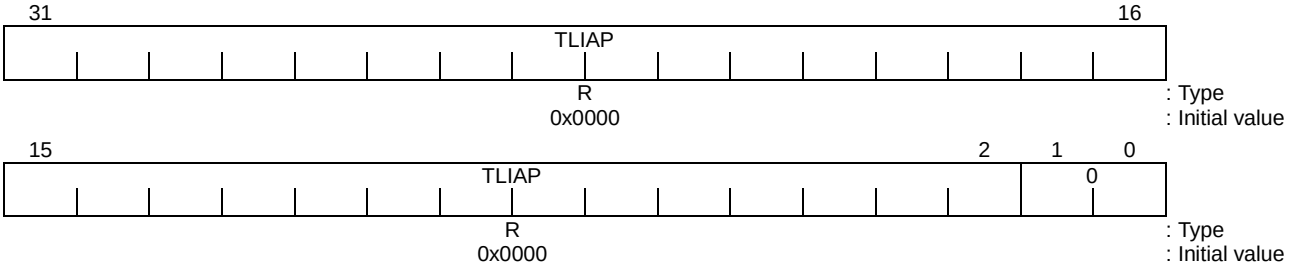
12.3.4.8 Target Local Bus OFIFO Address Pointer (TLBOAP) 0xFFFE_D0B8 (+0b8)



Bits	Mnemonic	Field Name	Description
31:2	TLOAP	Target Local Bus Outbound OFIFO Address Pointer	Target Local Bus Outbound FIFO Address Pointer (initial value: 0x00000000) Contains an OFIFO address pointer for local bus (G-Bus) addresses. High-order bits of an address is supplied by either the Target Local Bus Memory Mapping Address register (TLBMMAR) or the Target Local Bus I/O Mapping Address register (TLBIOMAR), depending on the PCI command that the target has received. The remaining low-order bits of the address come from the external PCI bus master.

Figure 12.3.34 Target Local Bus OFIFO Address Pointer

12.3.4.9 Target Local Bus IFIFO Address Pointer (TLBIAP) 0xFFFE_D0BC (+0bc)

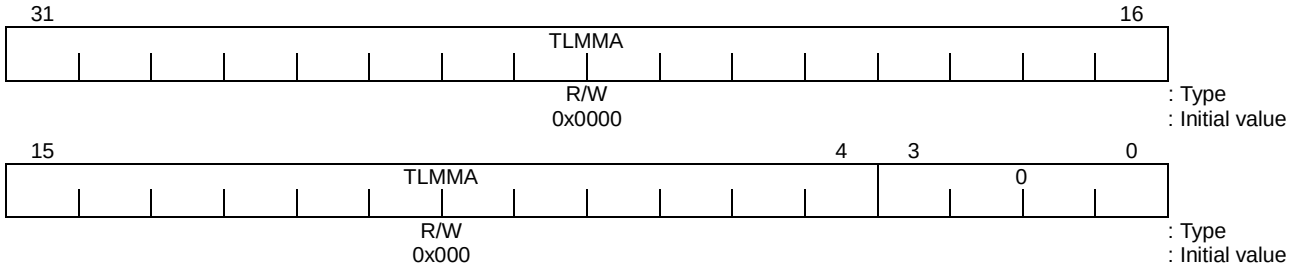


Bits	Mnemonic	Field Name	Description
31:2	TLIAP	Target Local Bus Inbound FIFO Address Pointer	Target Local Bus Inbound FIFO Address Pointer (initial value: 0x00000000) Contains an IFIFO address pointer for local bus (G-Bus) addresses. The high-order bits of an address is supplied by either the Target Local Bus Memory Mapping address register (TLBMMAR) or the Target Local Bus I/O Mapping Address register (TLBIOMAR), depending on the PCI command that the target has received. The remaining low-order bits of the address come from the external PCI bus master.

Figure 12.3.35 Target Local Bus IFIFO Address Pointer

12.3.4.10 Target Local Bus Memory Mapping Address Register (TLBMMAR)

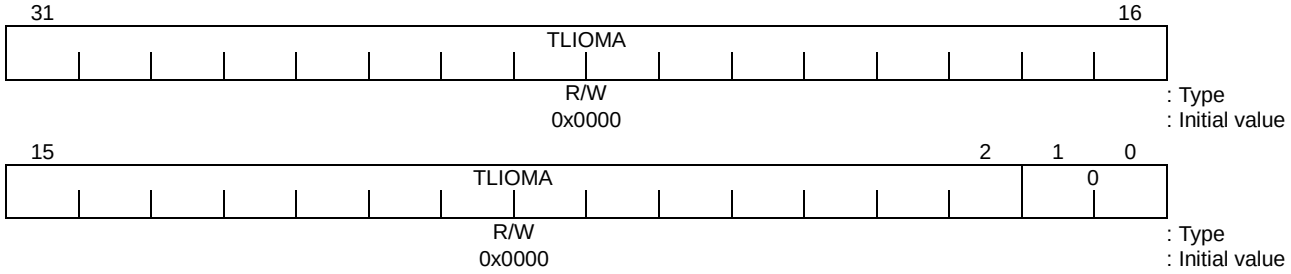
0xFFFE_D0C0 (+0c0)



Bits	Mnemonic	Field Name	Description
31:4	TLMMA	Target Local Bus Memory Address Mapping	Target Local Bus Memory Address Mapping (initial value: 0x0000000) This register is used to decode a PCI-to-local memory access for a target memory cycle. It specifies the start address of a local bus (G-Bus) memory region. The range of the memory region is programmed by the Target Memory Base Address Size (MBAS) register. In PCI-to-local address translation, high-order bits of the PCI bus address is replaced by the corresponding bits in this register and concatenated with the remaining low-order bits of the PCI bus address. This register is used by the PCI memory read and PCI memory write commands. Because the TLBMMAR is internally masked by the MBAS, the MBAS must be programmed before programming the TLBMMAR.

Figure 12.3.36 Target Local Bus Memory Mapping Address Register

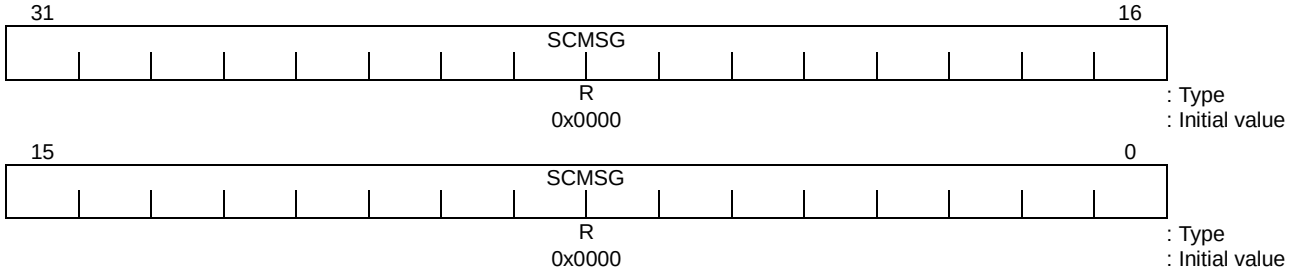
12.3.4.11 Target Local Bus I/O Mapping Address Register (TLBIOMAR) 0xFFFE_D0C4 (+0c4)



Bits	Mnemonic	Field Name	Description
31:2	TLIOMA	Target Local Bus IO Address Mapping Address	Target Local Bus IO Address Mapping Address (initial value: 0x00000000) This register is used to decode a local-to-PCI I/O access for a target memory cycle. It specifies the start address of a local bus (G-Bus) I/O address region. The range of the I/O space is programmed by the Target I/O Base Address Size (IOBAS) register. In local-to-PCI address translation, high-order bits of the PCI bus address is replaced by the corresponding bits in this register and concatenated with the remaining low-order bits of the PCI bus address. This register is used by the PCI I/O read and PCI I/O write commands. Because the TLBIOMAR is internally masked by the IOBAS, the IOBAS must be programmed before programming the TLBIOMAR.

Figure 12.3.37 Target Local Bus I/O Mapping Address Register

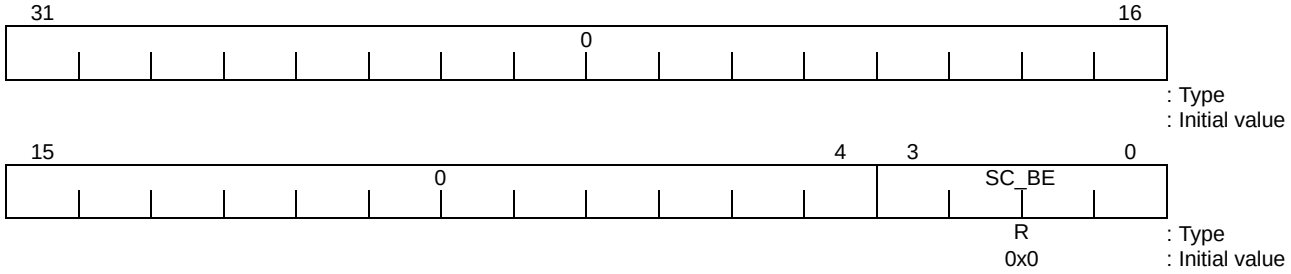
12.3.4.12 Special-Cycle Message Register (SC_MSG) 0xFFFE_D0C8 (+0c8)



Bits	Mnemonic	Field Name	Description
31:0	SCMSG	Special-Cycle Message	Special-Cycle Message (initial value: 0x00000000) Captures a special-cycle message on detection of a special-cycle command. To enable the capturing of a special-cycle message, the Special Cycle Recognition (SCREC) bit in the PCI Command (PCICMD) register must be set.

Figure 12.3.38 Special-Cycle Message Register

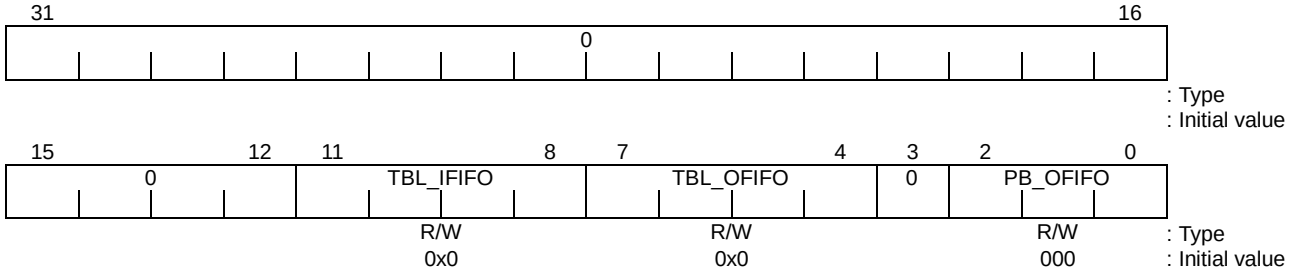
12.3.4.13 Special-Cycle Byte Enable Register (SC_BE) 0xFFFE_D0CC (+0cc)



Bits	Mnemonic	Field Name	Description
3:0	SC_BE	Special-Cycle Byte Enable	Special-Cycle Byte Enable (initial value: 0x0) Captures the byte enable signals for a special-cycle message on detection of a special-cycle command. To enable the capturing of the byte enables, the Special-Cycle Recognition (SCREC) bit in the PCI Command (PCICMD) register must be set.

Figure 12.3.39 Special-Cycle Byte Enable Register

12.3.4.14 Target Burst Length Register (TBL) 0xFFFE_D0D0 (+0d0)



Bits	Mnemonic	Field Name	Description
11:8	TBL_IFIFO	IFIFO Local Bus Burst Length	IFIFO Local Bus Burst Length (initial value: 0x0) Defines the number of D-words required in the IFIFO before data is written to the local bus. Bit 11 is used as a burst transfer enable bit. 0000: Single D-word. When one or more D-words are available in the IFIFO, one D-word is written to the local bus. The following settings enable repeated fast single accesses to write multiple words. 0001: Two D-words. When two or more D-words are available in the IFIFO, two D-words are written to the local bus. 0010: Four D-words. When four or more D-words are available in the IFIFO, four D-words are written to the local bus. 0011: Eight D-words. When eight or more D-words are available in the IFIFO, eight D-words are written to the local bus. 01XX: Sixteen D-words. When sixteen or more D-words are available in the IFIFO, sixteen D-words are written to the local bus. The following settings enable burst writes. 1X00: Four D-words. When four or more D-words are available in the IFIFO, four D-words are written to the local bus. 1X01: Eight D-words. When eight or more D-words are available in the IFIFO, eight D-words are written to the local bus. 1X1X: Sixteen D-words. When sixteen or more D-words are available in the IFIFO, sixteen D-words are written to the local bus.

Note 1: The IFIFO accepts data from the PCI master as long as there is room for it in the IFIFO. Once the IFIFO becomes full, the PCIC issues a target-disconnect to the PCI master. Thereafter, the PCIC continues to issue targe-retries until there is room in the IFIFO.

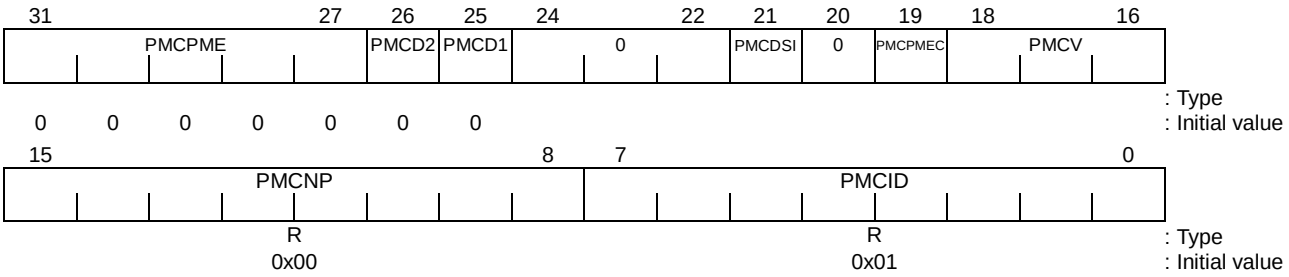
Note 2: A PCI write cycle may terminate with data short of the programmed burst size remaining in the IFIFO. In that case, the PCIC writes all the data in the IFIFO to local memory as 32-bit single write transactions after the PCI write cycle is complete. This prevents data from being left in the IFIFO.

Figure 12.3.40 Target Burst Length Register (1/2)

Bits	Mnemonic	Field Name	Description
7:4	TBL_OFIFO	OFIFO Local Bus Burst Length	OFIFO Local Bus Burst Length (initial value: 0x0) Defines the number of empty D-word locations required in the OFIFO before D-word data is read from the local bus. For non-prefetchable PCI read commands (non-prefetchable memory read and I/O read commands), a value of 000b is assumed automatically. Bit 7 is used as the burst transfer enable bit. The following settings enable repeated fast single accesses to read multiple words. 000X: Two D-words. When two or more empty D-word locations are available in the OFIFO, two D-words are read from the local bus. 0010: Four D-words. When four or more empty D-word locations are available in the OFIFO, four D-words are read from the local bus. 0011: Eight D-words. When eight or more empty D-word locations are available in the OFIFO, eight D-words are read from the local bus. 01XX: Sixteen D-words. When sixteen or more empty D-word locations are available in the OFIFO, sixteen D-words are read from the local bus. The following settings enable burst reads. 1X00: Four D-words. When four or more empty D-word locations are available in the OFIFO, four D-words are read from the local bus. 1X01: Eight D-words. When eight or more empty D-word locations are available in the OFIFO, eight D-words are read from the local bus. 1X1X: Sixteen D-words. When sixteen or more empty D-word locations are available in the OFIFO, sixteen D-words are read from the local bus.
2:0	PB_OFIFO	OFIFO PCI Bus Burst Length	OFIFO PCI Bus Burst Length (initial value: 0x0) Defines the number of D-words required in the OFIFO before data is sent to the PCI bus. 000: Single D-word. When one or more D-words are available in the OFIFO, one D-word is sent to the PCI bus. The following settings enable repeated fast single accesses to write multiple words. 001: Two D-words. When two or more D-words are available in the OFIFO, two D-words are sent to the PCI bus. 100: Four D-words. When four or more D-words are available in the OFIFO, four D-words are sent to the PCI bus. 101: Eight D-words. When eight or more D-words are available in the OFIFO, eight D-words are sent to the PCI bus. 11X: Sixteen D-words. When sixteen or more D-words are available in the OFIFO, sixteen D-words are sent to the PCI bus.

Figure 12.3.40 Target Burst Length Register (2/2)

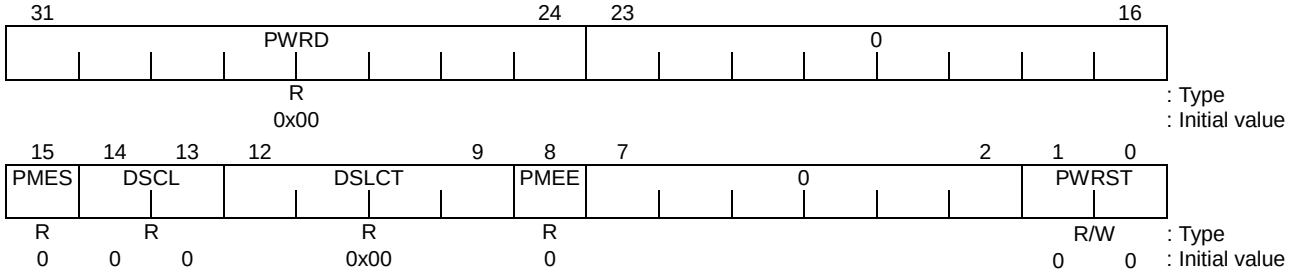
12.3.4.15 Power Management Register (PWMNGR) 0xFFFE_D0E0 (+0e0)



Bits	Mnemonic	Field Name	Description
31:27	PMCPME	Power Management Event	Power Management Event (initial value: 00000) Not supported by the TX3927.
26	PMCD2	D2 State	D2 State (initial value: 0) Not supported by the TX3927.
25	PMCD1	D1 State	D1 State (initial value: 0) Not supported by the TX3927.
21	PMCDSI	Device-Specific Initialization	Device-Specific Initialization (initial value: 1) Indicates whether it is necessary to initialize the PCIC. Initializing the TX3927 PCIC places it into the D0 uninitialized state; a device-specific initialization sequence is required following transition to the D0 uninitialized state.
19	PMCPMEC	Power Management Event Clock	Power Management Event Clock (initial value: 0) Indicates that no PCI clock is required to generate the power management event signal (PME#).
18:16	PMCV	Version	Version (initial value: 001) Indicates that the PCIC power management function complies with the PCI Bus Power Management Interface Specification, Version 1.1.
15:8	PMCNP	Next Pointer	Next Pointer (initial value: 0x00) Points to the location of the next item in the capabilities list. In the TX3927, this field contains 0x00 because there are no more items in the capabilities list.
7:0	PMCID	ID	ID (initial value: 0x01) In the TX3927, this field identifies the linked list item as being the PCI power management register.

Figure 12.3.41 Power Management Register

12.3.4.16 Power Management Support Register (PWMNGSR) 0xFFFE_D0E4 (+0e4)



Bits	Mnemonic	Field Name	Description
31:24	PWRD	Power Data	Power Data (initial value: 0x00) This field reflects the values of the DSCL and DSLCT fields in this register. Not supported by the TX3927.
15	PMES	PME Status	PME Status (initial value: 0) The TX3927 does not support PME generation from D3cold.
14:13	DSCL	Data Scale	Data Scale (initial value: 00) This field contains the scaling factor to be indicated in the PWRD field. Not supported by the TX3927.
12:9	DSLCT	Data Select	Data Select (initial value: 0x0) This field contains the data to be indicated in the PWRD field. Not supported by the TX3927.
8	PMEE	PME Enable	PME Enable (initial value: 0) The TX3927 does not support PME generation from D3cold.
1:0	PWRST	Power State	Power State (initial value: 00) This field is used to determine the current power state and to set the function into a new power state. 00: D0 (no change) 01: D1 (Don't use.) 10: D2 (Don't use.) 11: D3hot

Figure 12.3.42 Power Management Support Register

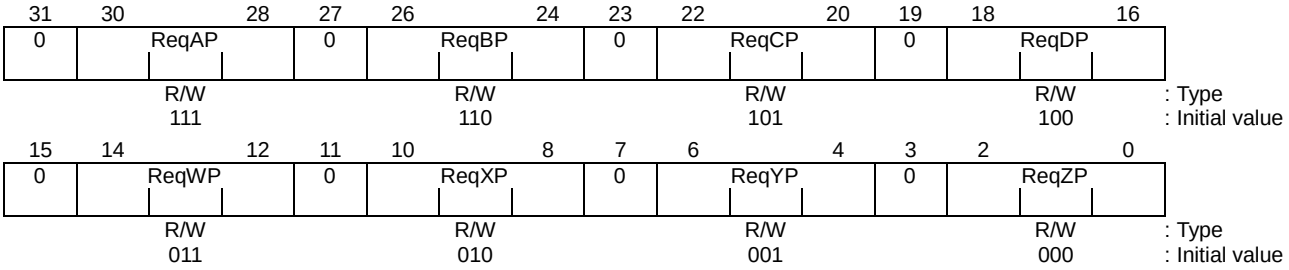
12.3.5 PCI Bus Arbiter/Parked Master Registers

Note: In external arbiter mode (boot configuration signal PCIXARB=0), these registers are not used.

12.3.5.1 Request Trace Register (REQ_TRACE) 0xFFFE_D100 (+100)

The PCIC bus arbiter provides unique request inputs for it and four other PCI bus masters.

The Request Trace register defines the assignment of the five request input ports (PCIC and REQ[3:0]) to the arbiter's internal request ports (masters A to D and W to Z). This assigns priorities to the request input ports. The corresponding grant trace values are set automatically. Each time this register is reprogrammed, the Broken Master (BM) register must be cleared.



Bits	Mnemonic	Field Name	Description
30:28	ReqAP	Request A Port	Request A Port (initial value: 111) Defines which PCI bus master is connected to the internal PCI bus arbiter request A port (master A). 111: Master A = PCIC 110: Don't use. 101: Don't use. 100: Don't use. 011: Master A = REQ*[3] 010: Master A = REQ*[2] 001: Master A = REQ*[1] 000: Master A = REQ*[0]
26:24	ReqBP	Request B Port	Request B Port (initial value: 110) Defines which PCI bus master is connected to the internal PCI bus arbiter request B port (master B). 111: Master B = PCIC 110: Don't use. 101: Don't use. 100: Don't use. 011: Master B = REQ*[3] 010: Master B = REQ*[2] 001: Master B = REQ*[1] 000: Master B = REQ*[0]

Figure 12.3.43 Request Trace Register (1/3)

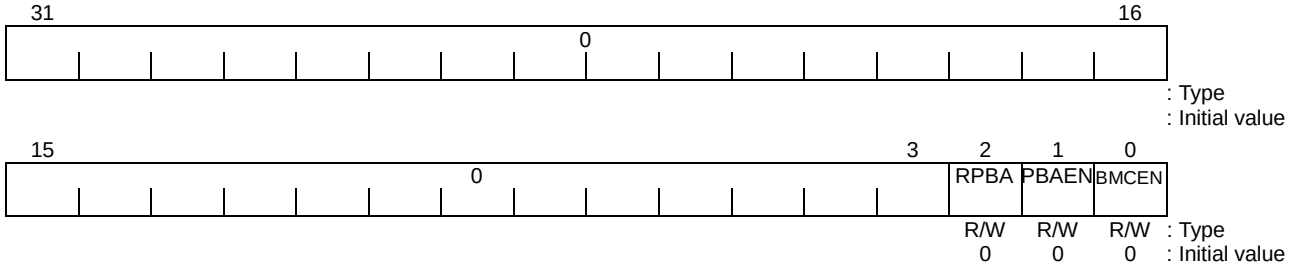
Bits	Mnemonic	Field Name	Description
22:20	ReqCP	Request C Port	Request C Port (initial value: 101) Defines which PCI bus master is connected to the internal PCI bus arbiter request C port (master C). 111: Master C = PCIC 110: Don't use. 101: Don't use. 100: Don't use. 011: Master C = REQ*[3] 010: Master C = REQ*[2] 001: Master C = REQ*[1] 000: Master C = REQ*[0]
18:16	ReqDP	Request D Port	Request D Port (initial value: 100) Defines which PCI bus master is connected to the internal PCI bus arbiter request D port (master D). 111: Master D = PCIC 110: Don't use. 101: Don't use. 100: Don't use. 011: Master D = REQ*[3] 010: Master D = REQ*[2] 001: Master D = REQ*[1] 000: Master D = REQ*[0]
14:12	ReqWP	Request W Port	Request W Port (initial value: 011) Defines which PCI bus master is connected to the internal PCI bus arbiter request W port (master W). 111: Master W = PCIC 110: Don't use. 101: Don't use. 100: v 011: Master W = REQ*[3] 010: Master W = REQ*[2] 001: Master W = REQ*[1] 000: Master W = REQ*[0]
10:8	ReqXP	Request X Port	Request X Port (initial value: 010) Defines which PCI bus master is connected to the internal PCI bus arbiter request X port (master X). 111: Master X = PCIC 110: Don't use. 101: Don't use. 100: Don't use. 011: Master X = REQ*[3] 010: Master X = REQ*[2] 001: Master X = REQ*[1] 000: Master X = REQ*[0]
6:4	ReqYP	Request Y Port	Request Y Port (initial value: 001) Defines which PCI bus master is connected to the internal PCI bus arbiter request Y port (master Y). 111: Master Y = PCIC 110: Don't use. 101: Don't use. 100: Don't use. 011: Master Y = REQ*[3] 010: Master Y = REQ*[2] 001: Master Y = REQ*[1] 000: Master Y = REQ*[0]

Figure 12.3.43 Request Trace Register (2/3)

Bits	Mnemonic	Field Name	Description
2:0	ReqZP	Request Z Port	Request Z Port (initial value: 000) Defines which PCI bus master is connected to the internal PCI bus arbiter request Z port (master Z). 111: Master Z = PCIC 110: Don't use. 101: Don't use. 100: Don't use. 011: Master Z = REQ*[3] 010: Master Z = REQ*[2] 001: Master Z = REQ*[1] 000: Master Z = REQ*[0]

Figure 12.3.43 Request Trace Register (3/3)

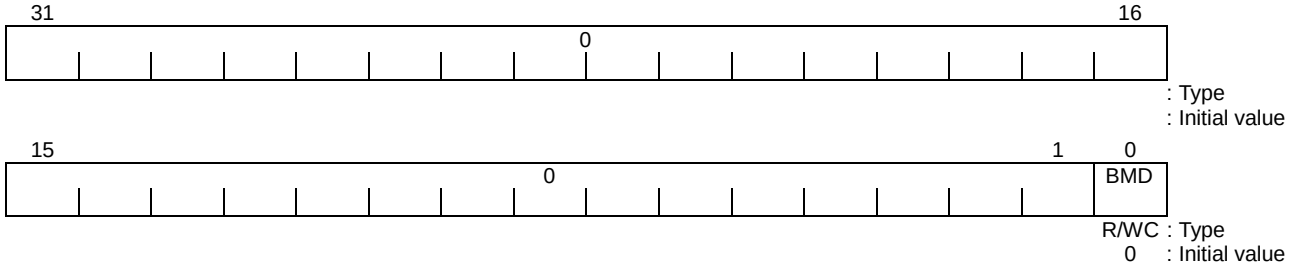
12.3.5.2 PCI Bus Arbiter/Parked Master Control Register (PBAPMC) 0xFFFE_D104 (+104)



Bits	Mnemonic	Field Name	Description
2	RPBA	Reset PCI Bus Arbiter	Reset PCI Bus Arbiter (initial value: 0) Resets the PCI bus arbiter. 1: The round-robin priority of the PCI bus arbiter is being reset. 0: The round-robin priority of the PCI bus arbiter is not being reset.
1	PBAEN	PCI Bus Arbiter Enable	PCI Bus Arbiter Enable (initial value: 0) After reset, external PCI bus requests to the PCI arbiter are blocked until this bit is set. The PCI bus is parked on the PCIC by default. 1: Enables the PCI bus arbiter. 0: Disables the PCI bus arbiter.
0	BMCEN	Broken Master Check Enable	Broken Master Check Enable (initial value: 0) This bit controls whether the PCI arbiter negates the bus grant to a requesting master that does not assert FRAME* within 16 PCI clock cycles from the time the bus is idle. This master is treated as a broken master. If this bit is set, identification of the broken master is recorded in the Broken Master (BM) register. 1: Enables checking for broken masters. 0: Disables checking for broken masters.

Figure 12.3.44 PCI Bus Arbiter/Parked Master Control Register

12.3.5.3 PCI Bus Arbiter/Parked Master Status Register (PBAPMS) 0xFFFE_D108 (+108)



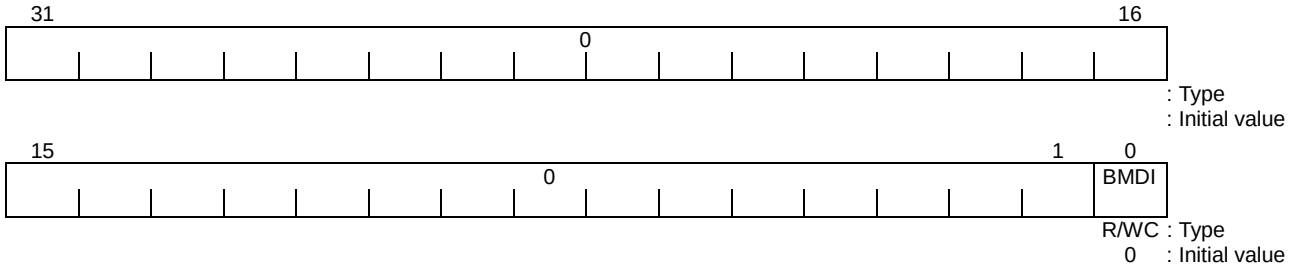
Bits	Mnemonic	Field Name	Description
0	BMD	Broken Master Detected	Broken Master Detected (initial value: 0) Indicates that a broken master has been detected. This bit is set if at least one bit in the Broken Master (BM) register is set. 1: At least one bit in BMR.BM is set. 0: None of the bits in BMR.BM are set.

Figure 12.3.45 PCI Bus Arbiter/Parked Master Status Register

12.3.5.4 PCI Bus Arbiter/Parked Master Interrupt Mask Register (PBAPMIM)

0xFFFE_D10C (+10c)

Setting the BMDI bit enables interrupts generated by the Broken Master Detected (BMD) bit in the PCI Bus Arbiter/Parked Master Status (PBAPMS) register.

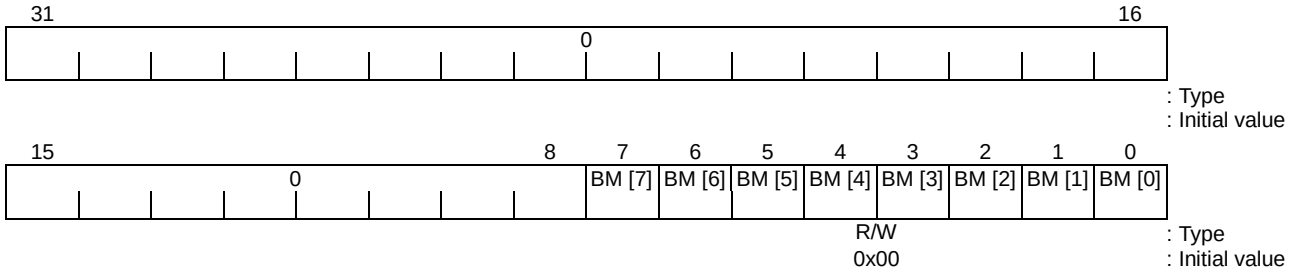


Bits	Mnemonic	Field Name	Description
0	BMDI	Broken Master Detected Interrupt Enable	Broken Master Detected Interrupt Enable (initial value: 0)

Figure 12.3.46 PCI Bus Arbiter/Parked Master Interrupt Mask Register

12.3.5.5 Broken Master Register (BM) 0xFFFE_D110 (+110)

This register shows the current broken master status. This register identifies any broken masters if the Broken Master Check Enable (BMCEN) bit in the PCI Bus Arbiter/Parked Master Control (PBAPMC) register is set. Each bit in this register represents a PCI master. The broken master feature allows the PCIC to lock out any masters that are broken or ill-behaved. Also, any masters can be locked out by programming the BM register, regardless of the setting of the BMCEN bit of the PBAPMC register. This register must be cleared whenever the contents of the REQ_TRACE register is changed.

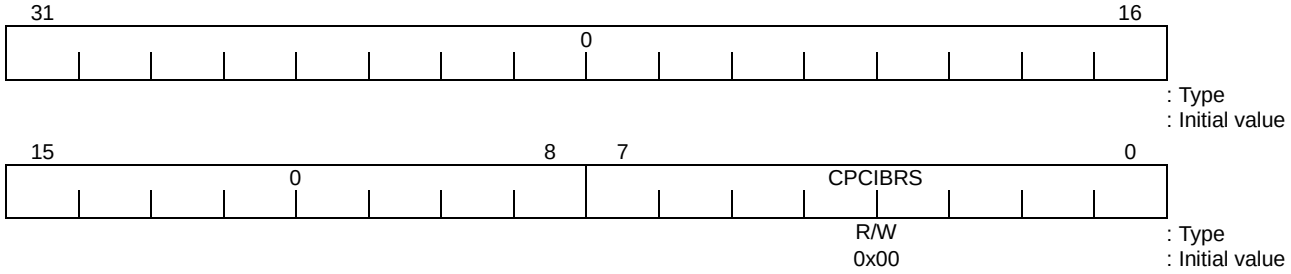


Bits	Mnemonic	Field Name	Description
7:0	BM [7:0]	Broken Master	Broken Master (initial value: 0x00) Indicates whether or not each PCI bus master is broken. BM[7:0] correspond to bus masters A, B, C, D, W, X, Y and Z in this order. 1: Assumed to be a broken master. 0: Not assumed to be a broken master.

Note: Writing a value of 0FFh to this register causes the PCI bus arbiter to be frozen.

Figure 12.3.47 Broken Master Register

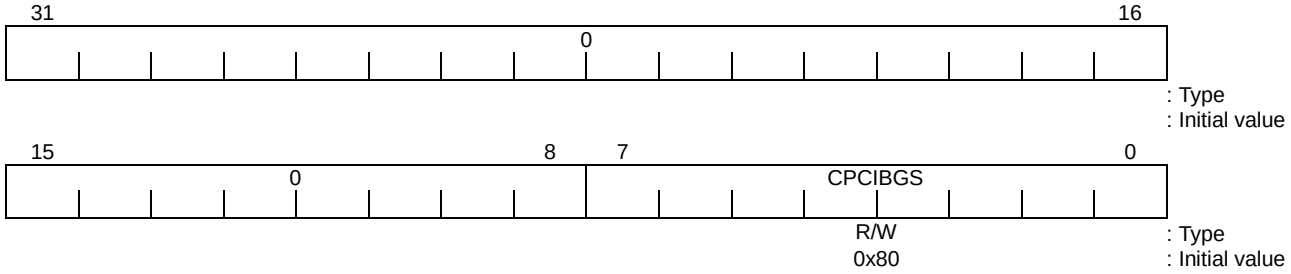
12.3.5.6 Current PCI Bus Request Status Register (CPCIBRS) 0xFFFE_D114 (+114)



Bits	Mnemonic	Field Name	Description
7:0	CPCIBRS	Current PCI Bus Request Status	Current PCI Bus Request Status (initial value: 0x00) Indicates the current state of the PCI bus request input signals (PCIC and REQ*[3:0]). CPCIBRS[7] corresponds to PCIC, and CPCIBRS[3:0] correspond to REQ*[3:0].

Figure 12.3.48 Current PCI Bus Request Status Register

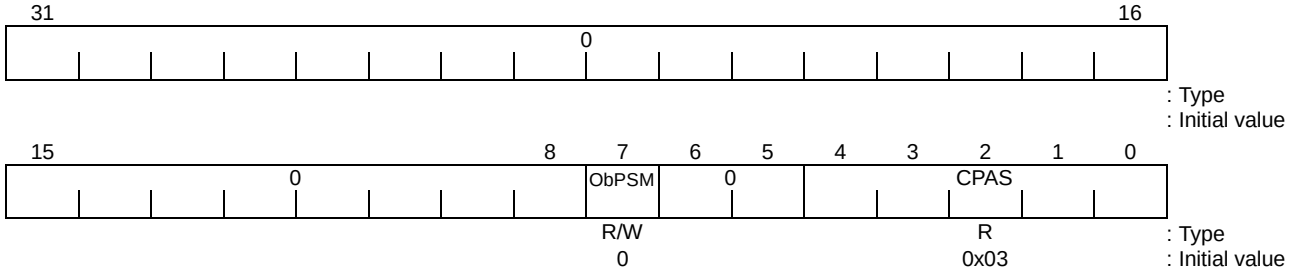
12.3.5.7 Current PCI Bus Grant Status Register (CPCIBGS) 0xFFFE_D118 (+118)



Bits	Mnemonic	Field Name	Description
7:0	CPCIBGS	Current PCI Bus Grant Status	Current PCI Bus Grant Status (initial value: 0x80) Indicates the current state of the PCI bus grant sent signals (PCIC and GNT*[3:0]). CPCIBGS[7] corresponds to PCIC, and CPCIBGS[3:0] correspond to GNT*[3:0].

Figure 12.3.49 Current PCI Bus Grant Status Register

12.3.5.8 PCI Bus Arbiter Current State Register (PBACS) 0xFFFE_D11C (+11c)



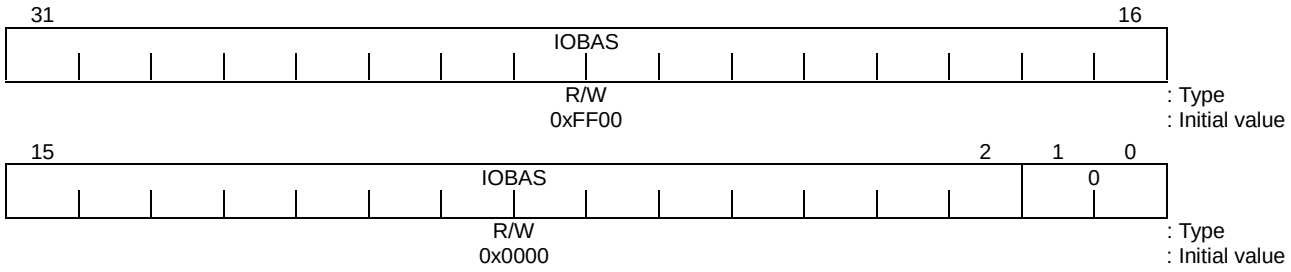
Bits	Mnemonic	Field Name	Description
7	ObPSM	Observe PCI State Machine	Observe PCI State Machine (initial value: 0) Specifies which state machine to observe. 1: Observes the Level-1 state machine. 0: Observes the Level-2 state machine.
4:0	CPAS	Current PCI Bus Arbiter State	Current PCI Bus Arbiter State (initial value: 0x03) Indicates the current state of the state machine selected by the ObPSM bit. See Figures 12.3.43 and 12.4.3 for agents/grants A to W and Level-2. The following describes the states when the ObPSM bit is 1. For cases where the ObPSM bit is 0, read A as W, B as X, C as Y and D as Z, and ignore references to Level-2. When ObPSM = 1: 0x00: The PCI bus arbiter prepares to grant bus mastership to PCI agent A. 0x01: Grant A is being awarded to PCI agent A when another agent holds the PCI bus. 0x02: Grant A is being awarded to PCI agent A when no agent holds the PCI bus. 0x03: Grant A has been awarded to agent A. If another agent is requiring bus mastership, the PCI bus arbiter prepares to grant it bus mastership. 0x04: The PCI bus arbiter prepares to grant bus mastership to PCI agent B. 0x05: Grant B is being awarded to PCI agent B when another agent holds the PCI bus. 0x06: Grant B is being awarded to PCI agent B when no agent holds the PCI bus. 0x07: Grant B has been awarded to agent B. If another agent is requiring bus mastership, the PCI bus arbiter prepares to grant it bus mastership. 0x08: The PCI bus arbiter prepares to grant bus mastership to PCI agent C. 0x09: Grant C is being awarded to PCI agent C when another agent holds the PCI bus. 0x0A: Grant C is being awarded to PCI agent C when no agent holds the PCI bus. 0x0B: Grant C has been awarded to agent C. If another agent is requiring bus mastership, the PCI bus arbiter prepares to grant it bus mastership. 0x0C: The PCI bus arbiter prepares to grant bus mastership to PCI agent D. 0x0D: Grant D is being awarded to PCI agent D when another agent holds the PCI bus. 0x0E: Grant D is being awarded to PCI agent D when no agent holds the PCI bus. 0x0F: Grant D has been awarded to agent D. If another agent is requiring bus mastership, the PCI bus arbiter prepares to grant it bus mastership.
4:0	CPAS	Current PCI Bus Arbiter State	0x10: The PCI bus arbiter prepares to grant bus mastership to a Level-2 PCI agent. 0x11: The Level-2 grant is being awarded to a Level-2 PCI agent when another agent holds the PCI bus. 0x12: The Level-2 grant is being awarded to a Level-2 PCI agent when no agent holds the PCI bus. 0x13: The Level-2 has been awarded to a Level-2 PCI agent. If another agent is requiring bus mastership, the PCI bus arbiter prepares to grant it bus mastership.

Figure 12.3.50 PCI Bus Arbiter Current State Register

12.3.6 Local Bus Special Registers

The local bus special registers do not reside in the PCI configuration space because they are assigned special functions.

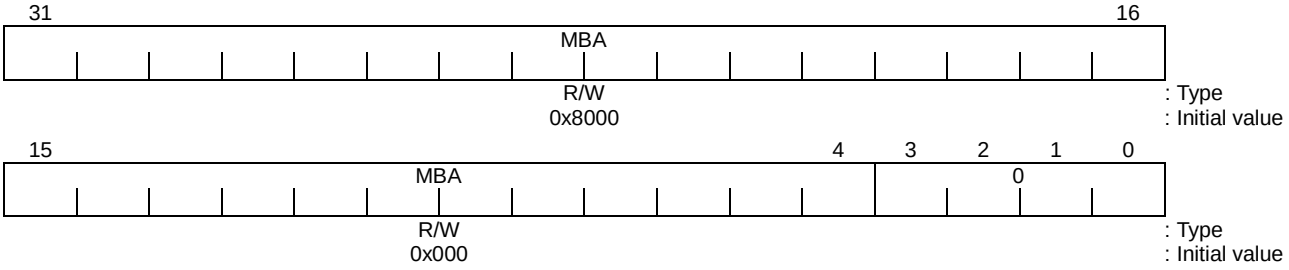
12.3.6.1 Target I/O Base Address Size Register (IOBAS) 0xFFFE_D120 (+120)



Bits	Mnemonic	Field Name	Description
31:2	IOBAS	Target IO Base Address Size	Target IO Base Address Size (initial value: 0xFF000000) This register is used together with the Target I/O Base Address (IOBAS) and Target Local Bus I/O Mapping Address (TLBIOMAR) registers to define the size of the I/O address region in PCI-to-local address translation. An all-zero value disables the target I/O address region. The default value after reset is 16 Mbytes (FF000000h). Legal values are equal to those that, after writing 1s to all locations in the IOBA space, an external PCI bus master receives when reading it. Because the IOBAS register internally masks the TLBMMAR and MBA registers, the IOBAS register must be programmed before programming them. The size must be a power of 2, or the Illegal Address Size (IAS) status bit in the Local Bus Status (LBSTAT) register will be set and an interrupt will be generated if the corresponding interrupt mask bit (LBIM.IASIE) is set. 00000000h: IOBA is disabled. FFFFFFFCh: 4 bytes FFFFFFF8h: 8 bytes FFFFFFF0h: 16 bytes FFFFFFE0h: 32 bytes FFFFFFC0h: 64 bytes FFFFFF80h: 128 bytes FFFFFF00h: 256 bytes : FE000000h: 32 Mbytes FC000000h: 64 Mbytes F8000000h: 128 Mbytes F0000000h: 256 Mbytes E0000000h: 512 Mbytes C0000000h: 1 Gbytes 80000000h: 2 Gbytes

Figure 12.3.51 Target I/O Base Address Size Register

12.3.6.2 Target Memory Base Address Size Register (MBAS) 0xFFFE_D124 (+124)

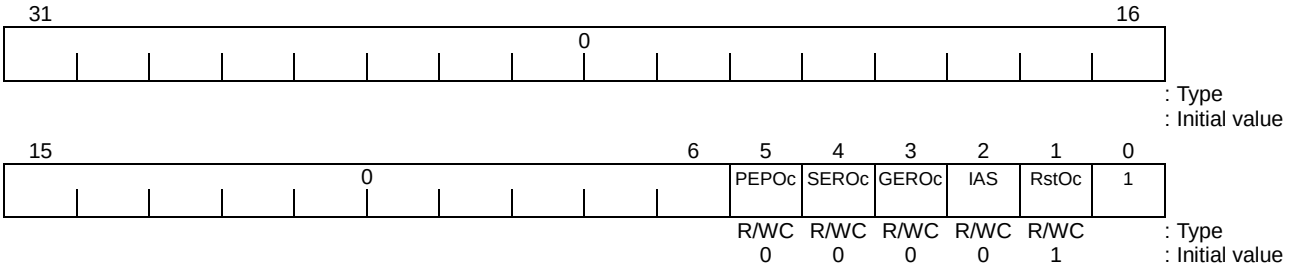


Bits	Mnemonic	Field Name	Description
31:4	MBA	Target Memory Base Address Size	Target Memory Base Address Size (initial value: 0x80000000) This register is used together with the Target Memory Base Address (MBA) and Target Local Bus Memory Mapping Address (TLBMMAR) registers to define the size of the memory address region in PCI-to-local address translation. An all-zero value disables the target memory address region. The default value after reset is 2 Gbytes (80000000h). Legal values are equal to those that, after writing 1s to all locations in the MBA space, an external PCI bus master receives when reading it. Because the MBAS register internally masks the TLBMMAR and MBA registers, the MBAS register must be programmed before programming them. The size must be a power of 2, or the Illegal Address Size (IAS) status bit in the Local Bus Status (LBSTAT) register will be set and an interrupt will be generated if the corresponding interrupt mask bit (LBIM.IASIE) is set. 00000000h: MBA is disabled. FFFFFFF0h: 16 bytes FFFFFFE0h: 32 bytes FFFFFFC0h: 64 bytes FFFFFF80h: 128 bytes FFFFFF00h: 256 bytes : FE000000h: 32 Mbytes FC000000h: 64 Mbytes F8000000h: 128 Mbytes F0000000h: 256 Mbytes E0000000h: 512 Mbytes C0000000h: 1 Gbytes 80000000h: 2 Gbytes

Figure 12.3.52 Target Memory Base Address Size Register

12.3.6.3 Local Bus Status Register (LBSTAT) 0xFFFE_D12C (+12c)

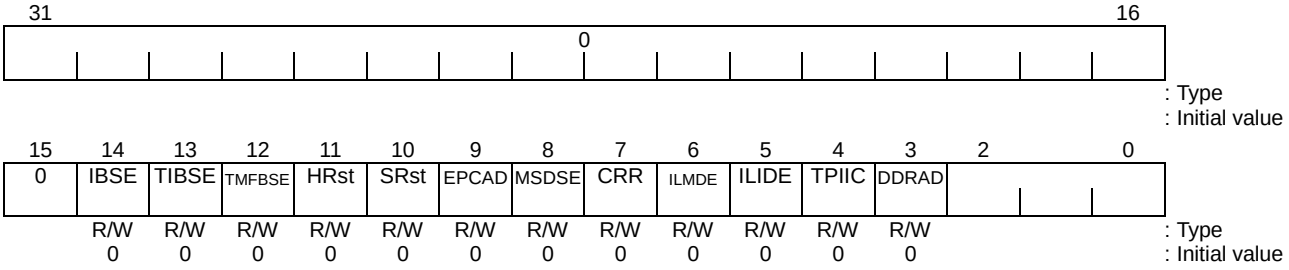
Each bit in this register has a corresponding bit in the Local Bus Interrupt Mask (LBIM) register. Setting a bit in this register triggers an interrupt to the TX3927 CPU if the corresponding LBIM register bit is set.



Bits	Mnemonic	Field Name	Description
5	PEROc	Parity Error Occurred	PERR* occurred. (initial value: 0) This bit is set when the PCIC or an external agent asserts PERR*. 1: PERR* asserted. 0: PERR* not asserted.
4	SEROc	System Error Occurred	SERR* occurred. (initial value: 0) This bit is set when the PCIC or an external agent asserts SERR*. 1: SERR* asserted. 0: SERR* not asserted.
3	GEROc	G-Bus Error Occurred	G-Bus Error Occurred. (initial value: 0) Indicates that a bus error has occurred on the G-Bus while the PCIC owns the G-Bus.
2	IAS	Illegal Address Size	Illegal Address Size (initial value: 0) This bit is set when the IOBAS, MBAS, IOMAS or MMAS size is illegal.
1	RstOc	RESET occurred	RESET occurred (initial value: 1) Indicates that the PCIC has been initialized.

Figure 12.3.53 Local Bus Status Register

12.3.6.4 Local Bus Control Register (LBC) 0xFFFE_D128 (+128)



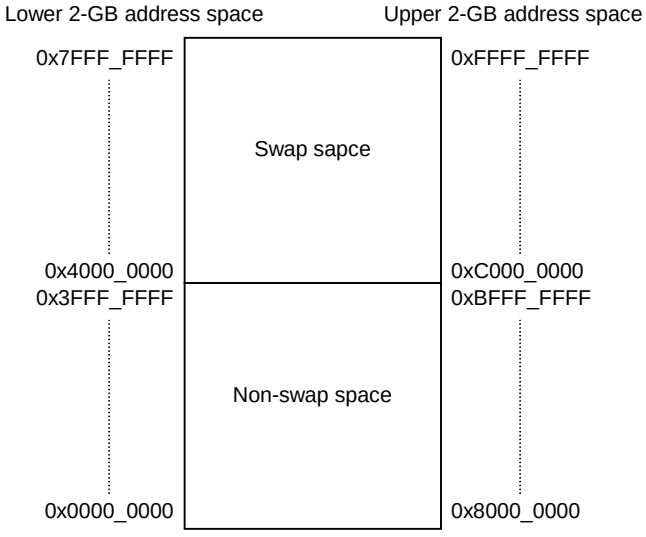
Bits	Mnemonic	Field Name	Description
14	IBSE	Initiator Byte Swapping Enable	Initiator Byte Swapping Enable (initial value: 0) Enables or disables byte swapping during accesses to the PCI bus as an initiator. Byte swapping occurs during all transactions: memory cycles, I/O cycles, configuration cycles, interrupt-acknowledge cycles and special-cycles. 0: Disable 1: Enable
13	TIBSE	Target I/O Byte Swapping Enable	Target I/O Byte Swapping Enable (initial value: 0) Enables or disables byte swapping during I/O-cycle accesses to the PCI bus as a target. 0: Disable 1: Enable
12	TMFBSE	Target Memory Full Area Byte Swapping Enable	Target Memory Full area Byte Swapping Enable (initial value: 0) Setting the MSDSE bit enables byte swapping during memory-cycle accesses to the PCI bus as a target. The TMFBSE bit selects the byte swapping region. 0: The upper half of the programmed target memory space is configured to perform byte swapping. Example: Assume that the Target Memory Base Address Size (MBAS) register is programmed as 0x80000000 (2 Gbytes) and that the MSDSE bit is 1. When AD[30]=1, byte swapping occurs because the target address falls within the upper 1 Gbytes. When AD[30]=0, byte swapping does not occur.  1: The entire target memory space is configured to perform byte swapping.
11	HRst	Hard Reset	Hard Reset (initial value: 0) Setting this bit is equivalent to a hard reset and an ordinary reset. Setting this bit causes the TX3927 to hold the reset signal active for 16 PCI clock cycles. Thereafter, this bit is cleared automatically. This bit should not be polled to determine if the reset is complete. An attempt to read this register during a reset causes a local bus error.

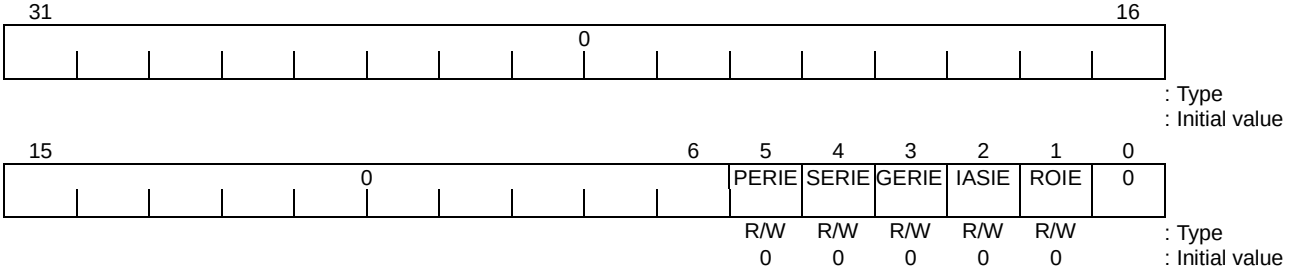
Figure 12.3.54 Local Bus Control Register (1/2)

Bits	Mnemonic	Field Name	Description
10	SRst	Soft Reset	Soft Reset (initial value: 0) Setting this bit initializes the PCIC. The soft reset does not initialize the PCI configuration header space registers or any status registers with R/WC access attributes.
9	EPCAD	External PCI Configuration Access Disable	External PCI Configuration Access Disable (initial value: 0) 1: Disables external PCI configuration accesses. This is mainly used by host-PCI bridge applications. 0: Enables external PCI bus masters to access the configuration registers.
8	MSDSE	Memory Space Dynamic Swapping Enable	Memory Space Dynamic Swapping Enable (initial value: 0) 1: Enables dynamic word swapping for PCI memory read/write commands. (The swapping region is selected by the TMFBSE bit.) The following illustrates the byte swapping mechanism: Little-endian word <--> Big-endian word Byte 0 Bits 7:0 J <--> n Bits 31:24 Byte 0 Byte 1 Bits 15:8 o <--> h Bits 23:16 Byte 1 Byte 2 Bits 23:16 h <--> o Bits 15:8 Byte 2 Byte 3 Bits 31:24 n <--> J Bits 7:0 Byte 3 If the input word is "John," the output word will be "nhoJ." 0: Disables dynamic word swapping. (No swapping occurs.)
7	CRR	Configuration Registers Ready	Configuration Registers Ready for Access (initial value: 0) 1: The configuration registers are ready for access. An external PCI bus master is allowed to access the target configuration registers. 0: The configuration registers are not ready for access. An attempt to access any target configuration register causes a target-retry termination. Software must set this bit to permit accesses to the configuration registers.
6	ILMDE	Initiator Local Bus Memory Address Space Decoder Enable	Initiator Local Bus Memory Address Space Decoder Enable (initial value: 0) Controls whether to compare the address on the G-Bus to the value programmed in the IPBMMAR register. This bit is used in conjunction with the Initiator Memory Mapping Address Size (MMAS) register. If the upper bits of the address match the corresponding bits of the MMAS register, the initiator generates a PCI memory read or write transaction. 1: Enables the PCIC initiator local bus memory address decoder. 0: Disables the PCIC initiator local bus memory address decoder. (No PCI transaction is generated.)
5	ILIDE	Initiator Local Bus I/O Address Space Decoder Enable	Initiator Local Bus I/O Address Space Decoder Enable (initial value: 0) Controls whether to compare the address on the G-Bus to the value programmed in the IPBIOMAR register. This bit is used in conjunction with the Initiator I/O Mapping Address Size (IOMAS) register. If the upper bits of the address match the corresponding bits of the IOMAS register, the initiator generates a PCI I/O read or write transaction. 1: Enables the PCI initiator local bus I/O address decoder. 0: Disables the PCI initiator local bus I/O address decoder. (No PCI transaction is generated.)
4	TPIIC	Test	Test (initial value: 0) This bit is reserved for test purposes; it should be fixed at 0.
3	DDRAD	Device-Dependent Region Access Disable	Device-Dependent Region Access Disable (initial value: 0) Controls whether to allow an external PCI bus master to access the registers for local bus settings (40h to dfh and e8h to ffh). 0: Allowed 1: Not allowed Reading any register at the above addresses will result in the value of the Device ID register being read. Writes to those registers will be ignored.

Figure 12.3.54 Local Bus Control Register (2/2)

12.3.6.5 Local Bus Interrupt Mask Register (LBIM) 0xFFFE_D130 (+130)

Each bit in this register has a corresponding bit in the Local Bus Status (LBSTAT) register. Setting a bit in the LBSTAT register triggers an interrupt if the corresponding enable bit in this register is set.

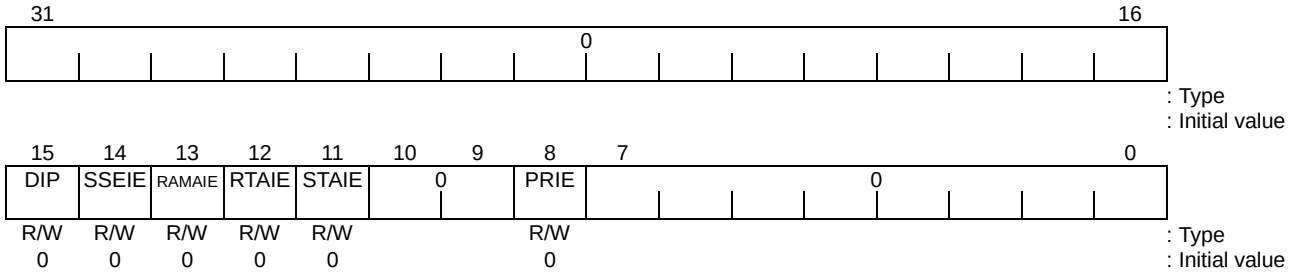


Bits	Mnemonic	Field Name	Description
5	PERIE	Parity Error occurred Interrupt Enable	PERR* occurred Interrupt Enable (initial value: 0) 1: An interrupt request is generated if LBSTAT.PEROc is set. 0: An interrupt request is not generated even if LBSTAT.PEROc is set.
4	SERIE	System Error occurred Interrupt Enable	SERR* occurred Interrupt Enable (initial value: 0) 1: An interrupt request is generated if LBSTAT.SEROc is set. 0: An interrupt request is not generated even if LBSTAT.SEROc is set.
3	GERIE	G-Bus Error occurred Interrupt Enable	G-Bus Error occurred Interrupt Enable (initial value: 0) 1: An interrupt request is generated if LBSTAT.GEROc is set. 0: An interrupt request is not generated even if LBSTAT.GEROc is set.
2	IASIE	Illegal Address Size Interrupt Enable	Illegal BARS Size Interrupt Enable (initial value: 0) 1: An interrupt request is generated if LBSTAT.ISA is set. 0: An interrupt request is not generated even if LBSTAT.ISA is set.
1	ROIE	RESET occurred Interrupt Enable	RESET occurred Interrupt Enable (initial value: 0) 1: An interrupt request is generated if LBSTAT.RstOc is set. 0: An interrupt request is not generated even if LBSTAT.RstOc is set.

Figure 12.3.55 Local Bus Interrupt Mask Register

12.3.6.6 PCI Status Interrupt Mask Register (PCISTATIM) 0xFFFE_D134 (+134)

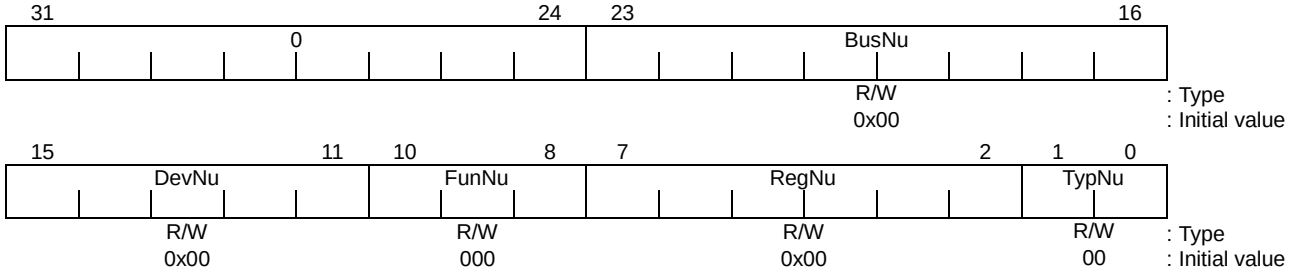
Each bit in this register has a corresponding bit in the PCI Status (PCISTAT) register. Setting a bit in the PCISTAT register triggers an interrupt if the corresponding enable bit in this register is set.



Bits	Mnemonic	Field Name	Description
15	DIP	Detected Parity Error Interrupt Enable	Detected Parity Error Interrupt Enable (initial value: 0) 1: An interrupt request is generated if PCISTAT.DECPE is set. 0: An interrupt request is not generated even if PCISTAT.DECPE is set.
14	SSEIE	Signaled System Error Interrupt Enable	Signaled System Error Interrupt Enable (initial value: 0) 1: An interrupt request is generated if PCISTAT.SIGSE is set. 0: An interrupt request is not generated even if PCISTAT.SIGSE is set.
13	RAMAIE	Received Master-Abort Interrupt Enable	Received Master-Abort Interrupt Enable (initial value: 0) 1: An interrupt request is generated if PCISTAT.RECMA is set 0: An interrupt request is not generated even if PCISTAT.RECMA is set.
12	RTAIE	Received Target-Abort Interrupt Enable	Received Target-Abort Interrupt Enable (initial value: 0) 1: An interrupt request is generated if PCISTAT.RECTA is set. 0: An interrupt request is not generated even if PCISTAT.RECTA is set.
11	STAIE	Signaled Target-Abort Interrupt Enable	Signaled Target-Abort Interrupt Enable (initial value: 0) 1: An interrupt request is generated if PCISTAT.SIGTA is set. 0: An interrupt request is not generated even if PCISTAT.SIGTA is set.
8	PRIE	Parity Error Reported Interrupt Enable	Parity Error Reported Interrupt Enable (initial value: 0) 1: An interrupt request is generated if PCISTAT.PERPT is set. 0: An interrupt request is not generated even if PCISTAT.PERPT is set.

Figure 12.3.56 PCI Status Interrupt Mask Register

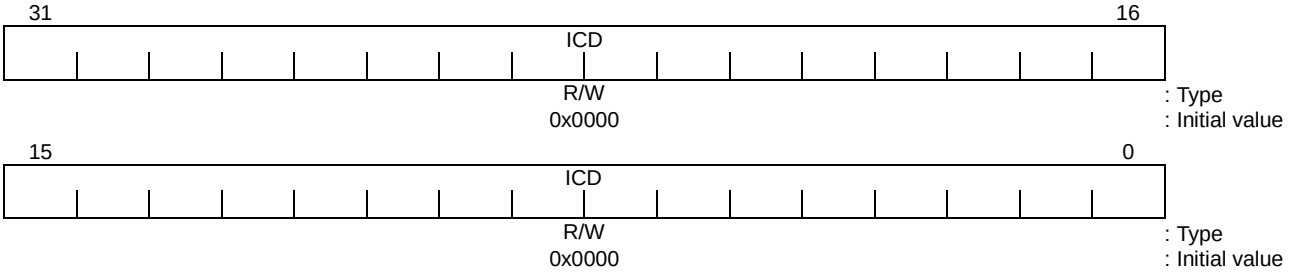
12.3.6.7 Initiator Configuration Address Register (ICA) 0xFFFE_D138 (+138)



Bits	Mnemonic	Field Name	Description
23:16	BusNu	Bus Number	Bus Number (initial value: 0x00) Selects one of 256 possible target PCI buses in a system.
15:11	DevNu	Device Number	Device Number (initial value: 0x00) Selects the target physical device to address on the selected PCI bus (the PCIC uses 21 devices out of 32 possible devices). The high-order 21 bits of the address lines, AD[31:11], are not used during the address phase of type 0 configuration cycles. The PCIC derives one of the following signals as an IDSEL signal to a physical PCI device. 0x00: AD[11] is driven asserted as IDSEL. 0x01: AD[12] is driven asserted as IDSEL. 0x02: AD[13] is driven asserted as IDSEL. 0x03: AD[14] is driven asserted as IDSEL. 0x04: AD[15] is driven asserted as IDSEL. 0x05: AD[16] is driven asserted as IDSEL. 0x06: AD[17] is driven asserted as IDSEL. 0x07: AD[18] is driven asserted as IDSEL. 0x08: AD[19] is driven asserted as IDSEL. 0x09: AD[20] is driven asserted as IDSEL. 0x0A: AD[21] is driven asserted as IDSEL. 0x0B: AD[22] is driven asserted as IDSEL. 0x0C: AD[23] is driven asserted as IDSEL. 0x0D: AD[24] is driven asserted as IDSEL. 0x0E: AD[25] is driven asserted as IDSEL. 0x0F: AD[26] is driven asserted as IDSEL. 0x10: AD[27] is driven asserted as IDSEL. 0x11: AD[28] is driven asserted as IDSEL. 0x12: AD[29] is driven asserted as IDSEL. 0x13: AD[30] is driven asserted as IDSEL. 0x14: AD[31] is driven asserted as IDSEL. 0x15-0x1f: Not used
10:8	FunNu	Function Number	Function Number (initial value: 000) Selects one of eight possible logical functions within the device.
7:2	RegNu	Register Number	Register Number (initial value: 0x00) Indexes one of 64 possible D-words in the configuration space of the intended target.
1:0	TypNu	Type Number	Type Number (initial value: 00) Specifies the address type in the address phase of the target configuration cycle.

Figure 12.3.57 Initiator Configuration Address Register

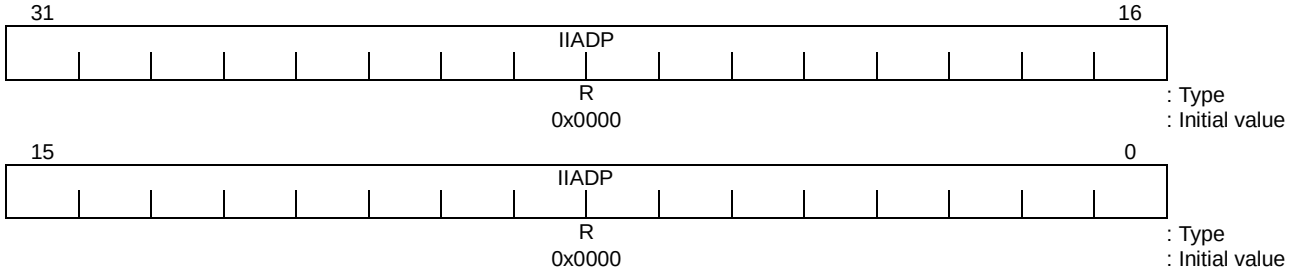
12.3.6.8 Initiator Configuration Data Register (ICDR) 0xFFFE_D13C (+13c)



Bits	Mnemonic	Field Name	Description
31:0	ICD	Initiator Configuration Data Register	Initiator Configuration Data Register (initial value: 0x00000000) This register is used as a data port for the configuration access cycle. Software reads or writes this register to initiate a PCI bus configuration transaction after setting up the Initiator Configuration Address (ICA) register.

Figure 12.3.58 Initiator Configuration Data Register

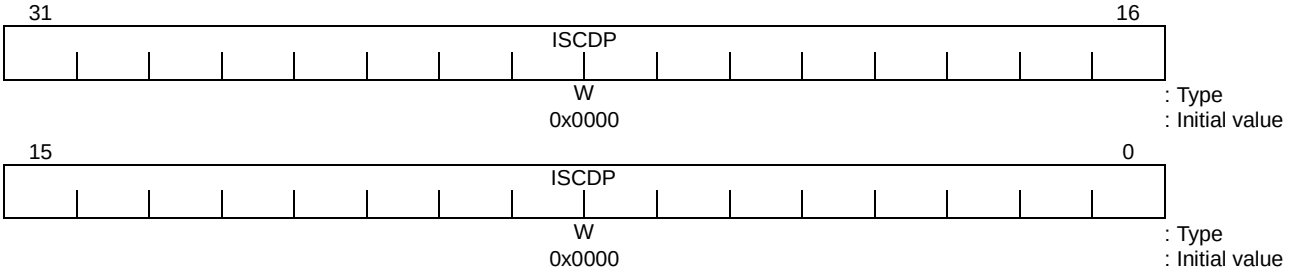
12.3.6.9 Initiator Interrupt-Acknowledge Data Port Register (IIADP) 0xFFFE_D140 (+140)



Bits	Mnemonic	Field Name	Description
31:0	IIADP	Initiator Interrupt Acknowledge Address Port	Initiator Interrupt-Acknowledge Address Port (initial value: 0x0000) A read of this register generates an interrupt-acknowledge cycle on the PCI bus. The read data is driven onto AD[31:0] during the data phase of the interrupt-acknowledge cycle.

Figure 12.3.59 Initiator Interrupt-Acknowledge Data Port Register

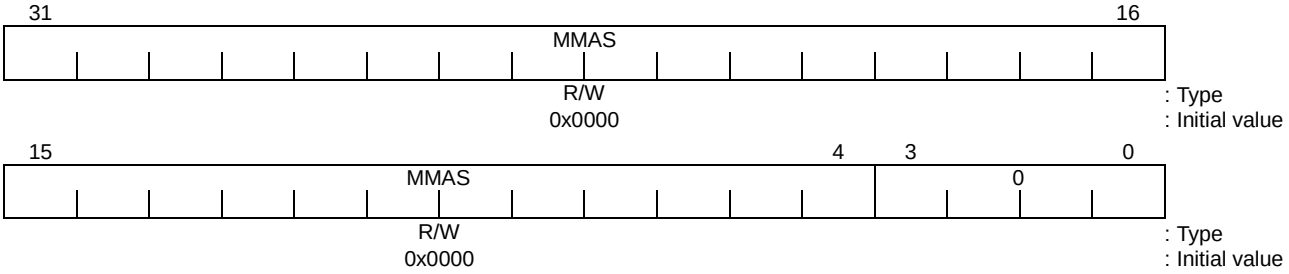
12.3.6.10 Initiator Special-Cycle Data Port Register (ISCDP) 0xFFFE_D144 (+144)



Bits	Mnemonic	Field Name	Description
31:0	ISCDP	Initiator Special-Cycle Data Port	Initiator Special-Cycle Data Port (initial value: 0x00000000) A write to this register generates a special-cycle transaction on the PCI bus with the written message.

Figure 12.3.60 Initiator Special-Cycle Data Port Register (ISCDP)

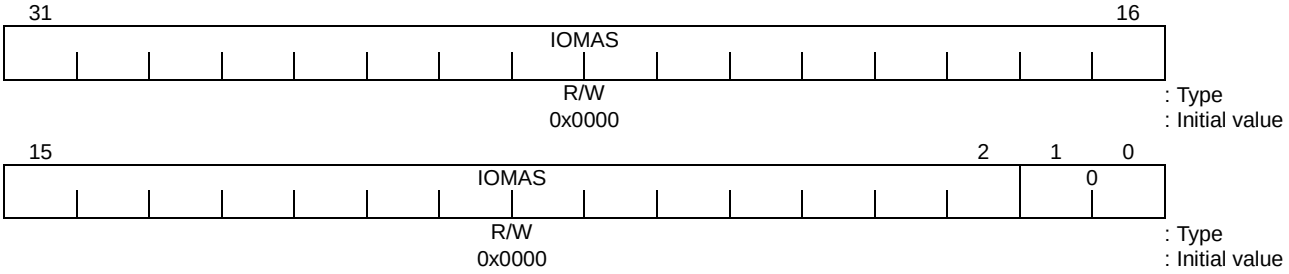
12.3.6.11 Initiator Memory Mapping Address Size Register (MMAS) 0xFFFE_D148 (+148)



Bits	Mnemonic	Field Name	Description
31:4	MMAS	Initiator Memory Mapping Address Size	Initiator Memory Mapping Address Size (initial value: 0x0000000) This register is used together with the Initiator PCI Bus Memory Mapping Address register (IPBMMAR) and the Initiator Local Bus Memory Mapping Address register (ILBMMAR) to define the size of the memory address region in local-to-PCI address translation. An all-zero value disables the PCI memory address region. Because the MMAS register internally masks the ILBMMAR and IPBMMAR registers, the MMAS register must be programmed before programming them. The size must be a power of 2, or the Illegal Address Size (IAS) status bit in the Local Bus Status (LBSTAT) register will be set and an interrupt will be generated if the corresponding interrupt mask bit (LBIM.IASIE) is set. 00000000h: The ILBMMAR address decoder is disabled (default POR value). FFFFFFF0h: 16 bytes FFFFFFE0h: 32 bytes FFFFFFC0h: 64 bytes FFFFFF80h: 128 bytes FFFFFF00h: 256 bytes : : FF000000h: 16 Mbytes FE000000h: 32 Mbytes FC000000h: 64 Mbytes F8000000h: 128 Mbytes F0000000h: 256 Mbytes E0000000h: 512 Mbytes C0000000h: 1 Gbytes 80000000h: 2 Gbytes

Figure 12.3.61 Initiator Memory Mapping Address Size Register

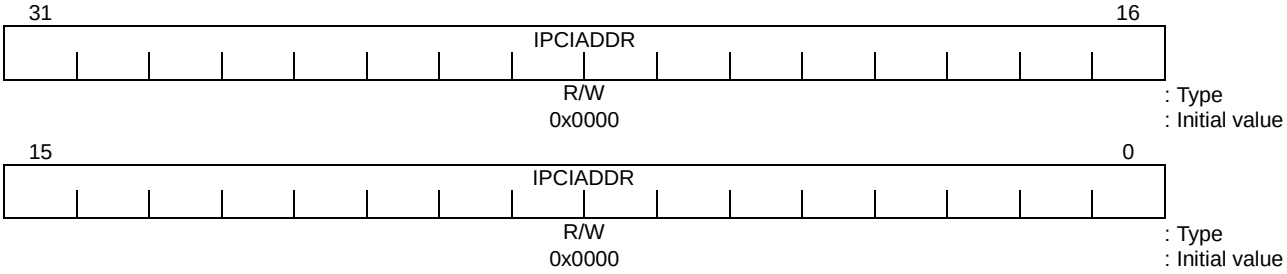
12.3.6.12 Initiator I/O Mapping Address Size Register (IOMAS) 0xFFFE_D14C (+14c)



Bits	Mnemonic	Field Name	Description
31:2	IOMAS	Initiator IO Mapping Address Size	Initiator IO Mapping Address Size (initial value: 0x0000000) This register is used together with the Initiator PCI Bus I/O Mapping Address register (IPBIOMAR) and the Initiator Local Bus I/O Mapping Address register (ILBIOMAR) to define the size of the I/O address region in local-to-PCI a address translation. An all-zero value disables the PCI I/O address region. Because the IOMAS register internally masks the ILBIOMAR and IPBIOMAR registers, the IOMAS register must be programmed before programming them. The size must be a power of 2, or the Illegal Address Size (IAS) status bit in the Local Bus Status (LBSTAT) register will be set and an interrupt will be generated if the corresponding interrupt mask bit (LBIM.IASIE) is set. 00000000h: The ILBIOMAR address decoder is disabled (default POR value). FFFFFFF0h: 16 bytes FFFFFFE0h: 32 bytes FFFFFFC0h: 64 bytes FFFFFF80h: 128 bytes FFFFFF00h: 256 bytes : : FF000000h: 16 Mbytes FE000000h: 32 Mbytes FC000000h: 64 Mbytes F8000000h: 128 Mbytes F0000000h: 256 Mbytes E0000000h: 512 Mbytes C0000000h: 1 Gbytes 80000000h: 2 Gbytes

Figure 12.3.62 Initiator I/O Mapping Address Size Register

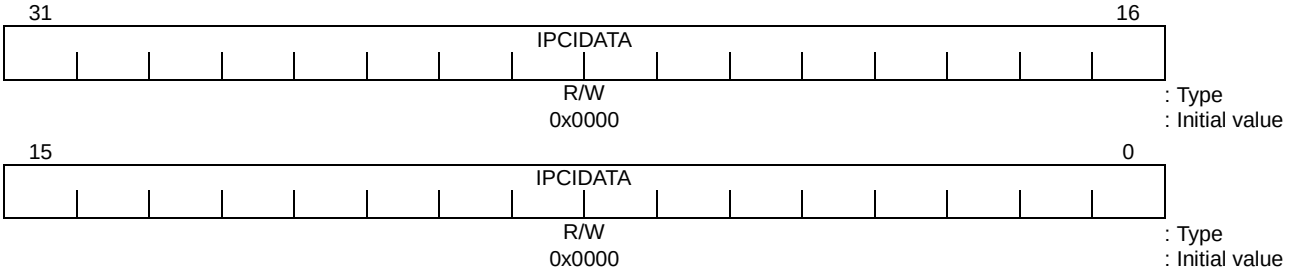
12.3.6.13 Initiator Indirect Address Register (IPCIADDR) 0xFFFE_D150 (+150)



Bits	Mnemonic	Field Name	Description
31:0	IPCIADDR	Initiator Indirect Address	Initiator Indirect Address (initial value: 0x00000000) Specifies the PCI address to be accessed for an indirect initiator cycle.

Figure 12.3.63 Initiator Indirect Address Register

12.3.6.14 Initiator Indirect Data Register (IPCIDATA) 0xFFFE_D154

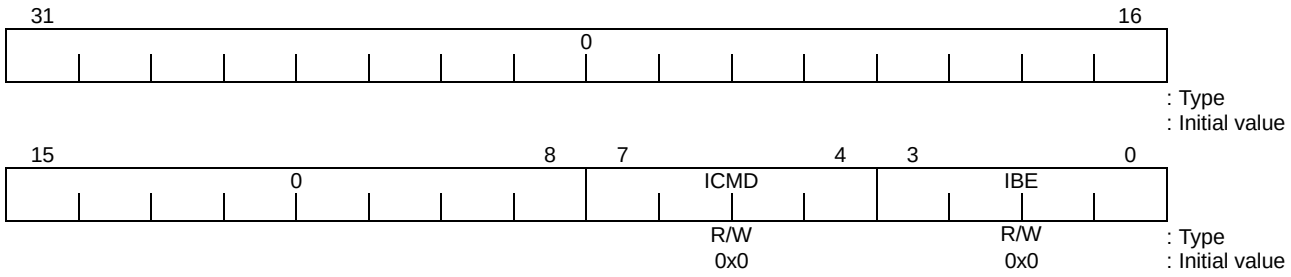


Bits	Mnemonic	Field Name	Description
31:0	IPCIDATA	Initiator Indirect Data	Initiator Indirect Data (initial value: 0x00000000) Contains the data for an indirect initiator PCI cycle. For a read cycle, the data read during the data phase is captured in this register. For a write cycle, the data in this register is driven onto the PCI AD bus during the data phase.

Figure 12.3.64 Initiator Indirect Data Register

12.3.6.15 Initiator Indirect Command/Byte Enable Register (IPCICBE) 0xFFFE_D158

A write to this register generates an indirect initiator cycle. The IDICC bit of the ISTAT register will be set upon completion of the cycle.



Bits	Mnemonic	Field Name	Description
7:4	ICMD	Initiator Indirect Command	Initiator Indirect Command (initial value: 0x0) Provides a PCI command for an indirect initiator access. The value in this field is directly driven onto C_BE[3:0].
3:0	IBE	Initiator Indirect Byte Enable	Initiator Indirect Byte Enable (initial value: 0x0) Provides byte enables for an indirect initiator access. The value in this field is directly driven onto C_BE[3:0].

Figure 12.3.65 Initiator Indirect Command/Byte Enable Register

12.4 Operation

12.4.1 Transfer Modes

As an initiator, the PCI Controller (PCIC) supports PIO interface; as a target, the PCIC supports data streaming.

- Initiator PIO mode

  The PCIC initiator module incorporates a general write buffer for PCI memory and I/O transactions. When the PCIC is acting as an initiator, data is transferred to and from the PCI bus via the PIO port shown in Figure 12.2.1. In this mode, the PCIC functions as a G-Bus slave. The PCIC initiator module supports direct and indirect transfer modes for PCI cycles.

  In PIO mode, the RF field of the Config register within the TX39/H2 CPU core must be 00 (x1 processor clock frequency). Other settings could yield improper operation.

  Direct mode: In this mode, part of the local bus (G-Bus) address space is directly mapped to the PCI bus. A read of this space generates a read cycle on the PCI bus. The local bus (G-Bus) cycle does not finish until the PCI bus cycle finishes.
  Writes uses a 1-doubleword buffer; thus, the PCI write cycle is run asynchronously from the local bus. Another write to this local bus (G-Bus) region will not finish until the previous PCI write has finished and the write buffer becomes available.

  Indirect mode: In this mode, initiator PCI cycles run asynchronously to the local bus (G-Bus). For reads, write the appropriate PCI address and C_BE values into the IPCIADDR and IPCICBE registers and then poll a status bit (or interrupt). When the PCIC performs a PCI cycle and puts the data into the IPCIDATA register, the status bit is set. This signals to the CPU that the data is ready to be read.
  Writes operate similarly, except that the data is written to the IPCIDATA register before the IPCIADDR and IPCICBE registers are programmed. In indirect transfer mode, PCI-to-local address translation does not occur.

  Note: When the PCIC is acting as an initiator, direct accesses can create a deadlock situation. A deadlock occurs when the TX3927 is performing a direct bus master access to the same PCI target that is trying to access the TX3927. If that happens, the TX3927 can not respond until it completes its direct master access and frees up the local bus (G-Bus). Consequently, both the TX3927 and the PCI device result in a deadlock, retrying PCI requests. For deadlock avoidance, indirect accesses can be used.

- Target streaming mode

  As a target, the PCIC can provide continuous data to PCI master through FIFO buffers. In this mode, the PCIC is the bus master within the TX3927, and data can be transferred from memory to OFIFO to PCI bus, or from PCI bus to IFIFO to memory (see Figure 12.2.1), without any intervention from the TX3927 DMA Controller. When the PCIC target module receives a PCI command, the PCIC performs address translation as described in Section 12.4.3, "Address Translation," and transfers the data.

12.4.2 Configuration Cycles

To perform a PCI configuration transaction as an initiator in PIO mode, the Initiator Configuration Address register (ICAR) must programmed. Then, a read or write to the Initiator Configuration Data register (ICDR) causes the PCIC to translate the access into a PCI configuration cycle. The TX3927 performs type 0 configuration accesses.

12.4.3 Address Translation

The PCIC allows remapping of PCI transactions to the G-Bus space and G-Bus transactions to the PCI space.

- When the PCIC is the initiator running in direct mode (G-Bus to PCI bus address translation)

  When the TX39/H2 core accesses memory on the PCI bus, the PCIC compares the G-Bus address from the TX39/H2 core with the contents of the ILBMMAR register. The MMAS register holds a comparison mask that sets the variable memory region size. If there is an address match, the PCI bus address is generated by replacing the high-order bits of the G-Bus address with the corresponding bits of the IPBMMAR register. The remaining low-order bits of the G-Bus address are passed unchanged. When the target is an I/O device, the registers shown in parentheses in Figure 12.4.1 are used in address translation.

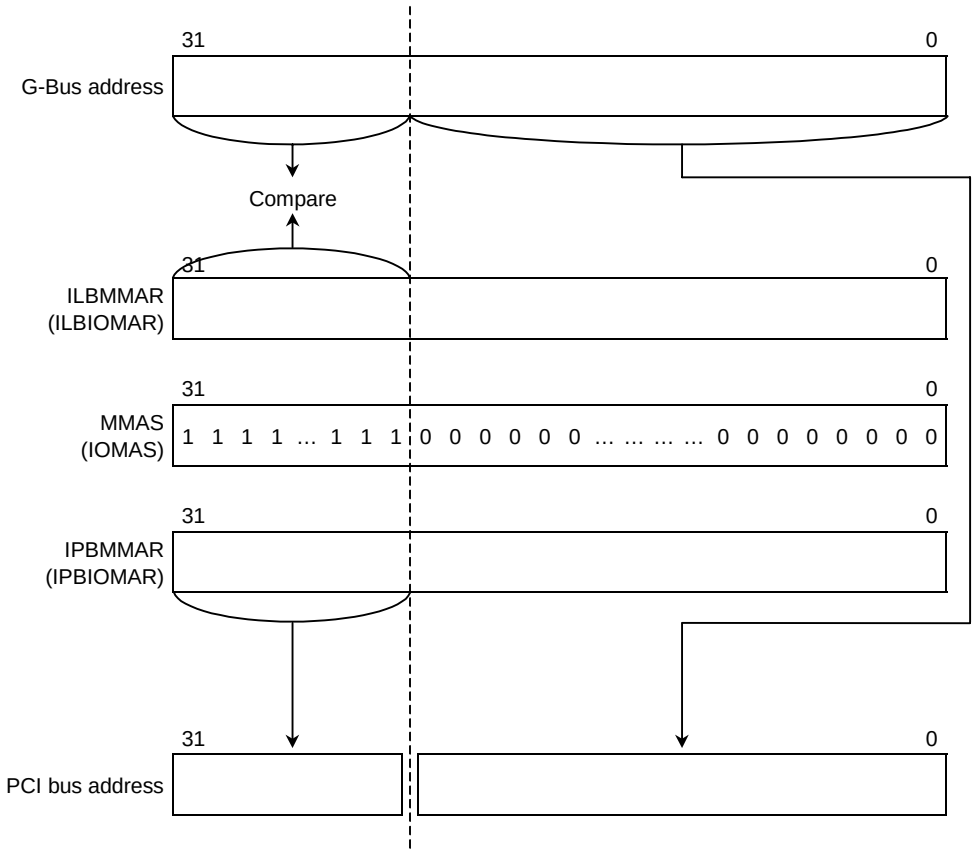


Figure 12.4.1 Address Translation in Direct Initiator Mode (G-Bus to PCI bus address translation)

- When the PCIC is the target (PCI bus to G-Bus address translation)

  When a bus master on the PCI bus accesses G-Bus memory mapped by a memory controller, the PCIC compares the PCI address with the contents of the MBA field in the Target Memory Base Address (MBA) register. The MBAS register holds a comparison mask that sets the variable memory region size. If there is an address match, the G-Bus address is generated by replacing the high-order bits of the PCI bus address with the corresponding bits of the TLBMMAR register. The remaining low-order bits of the PCI bus address are passed unchanged. When the target is an I/O device, the registers shown in parentheses in Figure 12.4.2 are used in address translation.

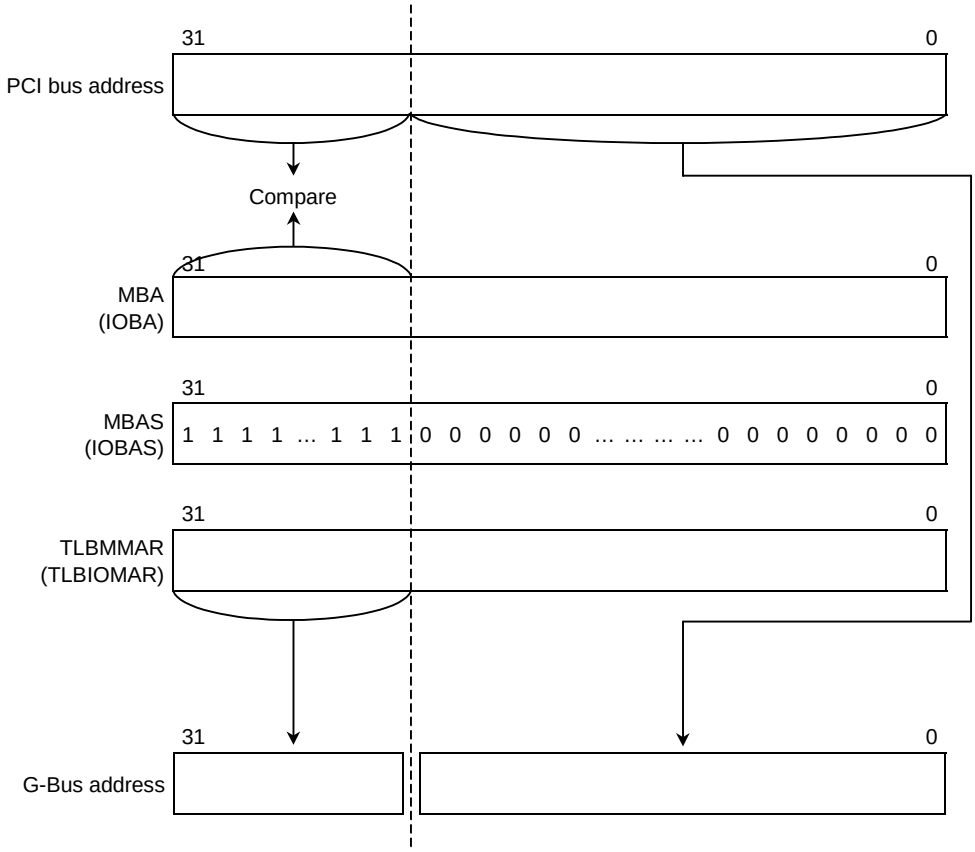


Figure 12.4.2 PCI Bus to G-Bus Address Translation

12.4.4 PCIC Clock

The external PCI clock signal, PCICLK, can be configured as an input or output. The state of the PCICLKEN boot configuration signal (ADDR[18]) determines the direction of the PCI clock.

When PCICLKEN=1, PCICLK[0:3] are set to output mode and drive a clock that is used as the PCIC internal reference clock (CLKPCI). The PCIC internal reference clock is generated by dividing down the G-Bus clock (GBUSCLK) by 2 or 3. The PCI clock continues to run even while the CPU is halted. Either the PCI3 bit of the CCFG register or the state of the PCI3* boot configuration signal (ADDR[17]) determines the clock divide ratio.

When PCICLKEN=0, PCICLK is set to input mode and the PCIC operates with the clock supplied from PCICLK[0]. In this mode, PCICLK[1:3] cannot be used.

12.4.5 PCI Bus Arbitration

The internal state machine that arbitrates between PCI bus accesses provides arbitration for up to eight PCI bus masters. However, in the TX3927, three of them are not used; The PCI bus arbiter controls the arbitration for the PCIC itself and four external bus masters connected to the REQ[3:0]* and GNT[3:0]* pins.

The PCIXARB boot configuration signal (ADDR[11]) sampled at the rising edge of the RESET* signal determines if the on-chip PCI arbiter is enabled (high) or disabled (low). If the on-chip PCI bus arbiter is disabled, REQ[0] is configured as an output, and GNT[0] as an input. Also, REQ[1] is configured as an interrupt output whose state is controlled by PIO module flags.

As shown in Figure 12.4.3, the on-chip PCI bus arbiter uses a two-level, round-robin arbitration algorithm. The low-priority group (level 2) consists of masters W to Z, while the high priority group (level 1) consists of masters A to D. The low-priority group collectively has one bus transaction slot in the high-priority group. The provision for eight bus masters is to allow for future expansion; the TX3927 PCIC uses five of the eight masters. The Request Trace (REQ_TRACE) register is used to program the assignment of A to D and W to Z to the PCIC and four external bus masters.

The order of progression of priorities between these accesses is shown in Figure 12.4.3.

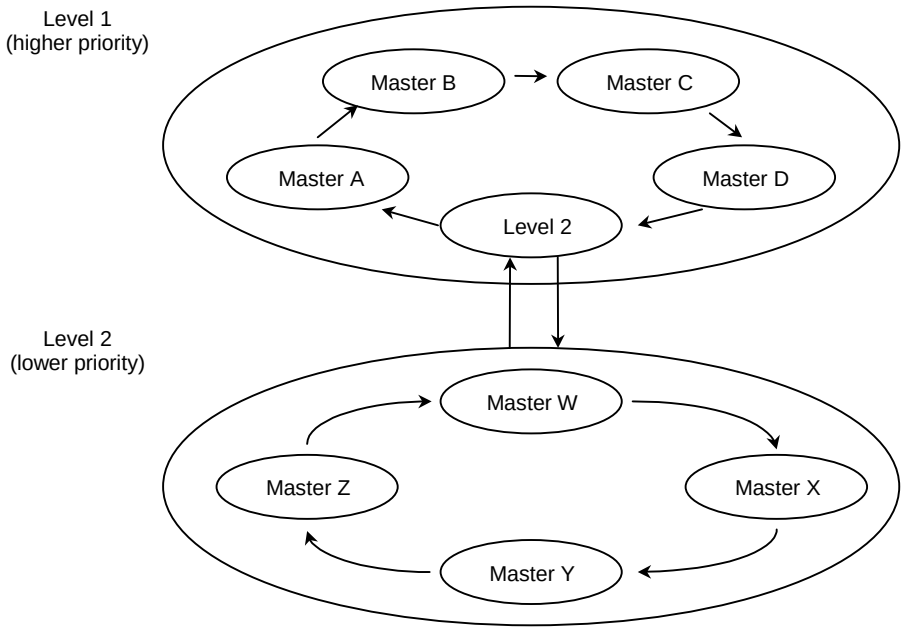


Figure 12.4.3 PCI Bus Arbitration/Parked Master Priority

All Level-1 agents have the same priority and are granted the bus equally (rotating in round-robin sequence within Level 1). If there are any agents at Level 2, one of them is guaranteed to get at least one of five bus transaction. All Level-2 agents have the same priority level and receive an equal number of bus grants in round-robin fashion (within Level 2).

Not all of the eight transaction slots can be used with the TX3927. However, assuming there are eight bus masters requesting the bus at the same time, the grant sequence is A → B → C → D → W → A → B → C → D → X → A → B → C → D → Y → A → B → C → D → Z → and repeating.

The round-robin sequence moves only in the direction of the arrows. Suppose that there are two high-priority devices assigned to A and B, and one low-priority device assigned to W. If A and W request the bus simultaneously when B is conducting a transaction, the grant sequence will be B →W → A.

The round-robin sequences can be initialized by setting the RPBA bit in the PCI Bus Arbiter/Parked Master Control (PBAPMC) register.

12.4.6 FIFO Depth

To support data streaming to and from local memory, the PCIC target module incorporates two 16-deep FIFO buffers: one for inbound memory transactions and one for outbound memory transactions. The IFIFO is used for the streaming of data from the PCI bus to memory. The OFIFO is used for the streaming of data from memory to the PCI bus.

12.4.7 Accessing the PCIC Target Module

The PCIC target module can be accessed from the PCI bus in one of the following two ways:

1. FIFO access

   A FIFO access is performed using an I/O or memory cycle. This type of access allows data to be transferred from memory to the PCI bus and from the PCI bus to memory.

2. Register access

   A register access is performed using a configuration cycle or a special cycle.

12.4.8 PCIC Register Access

The PCI configuration space registers are comprised of the following groups of registers:

- PCI configuration header space registers
- Initiator configuration space registers
- Target configuration space registers

PCI bus masters can access these registers. They are located in the 256-byte PCI configuration space (0xFFFE_D0FF to 0xFFFE_D000). All the other registers are located at addresses 0xFFFE_D1FF to 0xFFFE_D100 and can be accessed by the TX39/H2 core but not by PCI bus masters.

12.4.9 Address Mapping Between the Local Bus and PCI Bus

The PCIC allows address translation between the PCI and local bus address space through use of the base address registers for both of the PCI and local buses. Linear address mapping is used. The size of the mapped address space must be a power of 2. The TX39/H2 core must keep track of which section of the memory space belongs to memory or I/O space.

12.4.10 ACPI Power Management

The TX3927 provides partial support for the ACPI power management specifications. This section explains the support of the ACPI specifications.

12.4.10.1 Power Management Interface

This section describes the format of the PCI configuration space registers used by power management operations.

The CL bit in the PCI Status (PCISTAT) register indicates the presence or absence of the capabilities list. The TX3927 has a capabilities list compliant with the ACPI power management specifications; so this bit is 1.

The Capabilities Pointer (CAPPTR) register gives the location of the PCIC registers containing the first item in the list. The 8-bit CAPPTR field in this register provides an offset into the PCI power management register block, relative to the PCIC base address 0xFFFE_D000. In the TX3927, this offset value is 0xE0.

The Power Management (PWMNGR) register has the 8-bit PMCID field which identifies the type of the data structure currently being pointed to. In the TX3927, this field is read as "01," indicating ACPI power management registers.

The TX3927 has two registers used for ACPI power management: the Power Management register (PWMNGR) and the Power Management Support register (PWMNGSR).

12.4.10.2 PCI Function Power States

The PCI Bus Power Management Interface Specification defines four power management states for PCI functions, D0 to D3. D0 is the most power-consuming state, D3 is the least power-consuming state, and D1 and D2 represent intermediate power management states.

The TX3927 does not support D1 and D2.

The PCIC must initially be put into the D0 state before being used. Upon entering the D0 state as a result of a power-on reset or transition from D3hot, the PCIC will be in an uninitialized state. Once the PCIC is initialized, it will be in the D0 active state. A reset will force the PCIC to the uninitialized D0 state.

Setting the Power State (PWRST) field in the Power Management Support register (PWMNGSR) to "11" causes the PCIC to enter the D3hot state. In the D3hot state, the PCIC initiator and either the target memory or I/O decoder are disabled.

The D3 state has two variants, D3hot and D3cold, where "hot" and "cold" refer to the presence and absence of power, respectively. The PCIC in the D3hot state can be transitioned to the uninitialized D0 state by writing "00" to the Power State (PWRST) field in the Power Management Support register (PWMNGSR) or by having the RESET* signal asserted. The RESET* signal resets the entire TX3927.

The PCIC in the D3cold state can only be transitioned to the uninitialized D0 state by asserting the RESET* signal.

Figure 12.4.4 shows the PCI function power management state transitions.

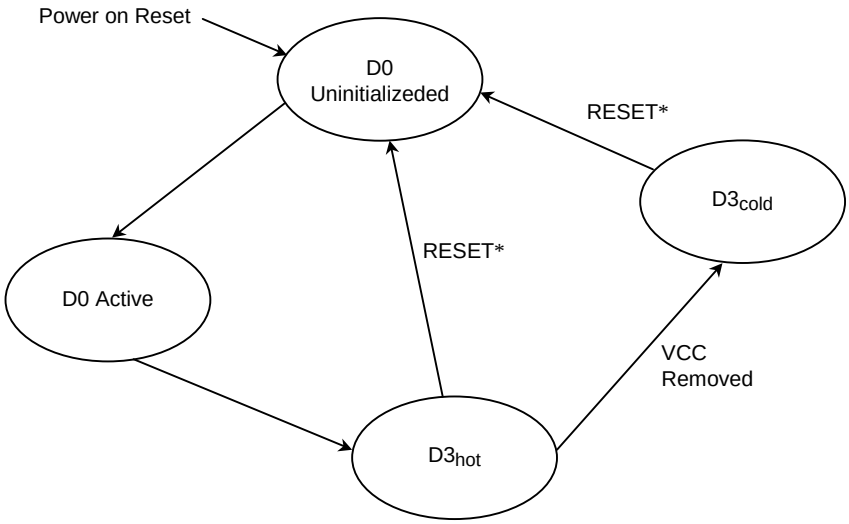


Figure 12.4.4 Power Management State Transitions

12.4.11 Byte Swapping

The TX3927 has the ability to perform byte swapping. The byte swapping can be enabled or disabled using the Local Bus Control (LBC) register.

Bit 14 (IBSE) in the LBC register specifies whether to enable byte swapping in initiator mode. The setting of this bit applies to all transactions: initiator memory cycles, initiator I/O cycles, initiator configuration cycles, initiator interrupt-acknowledge cycles and initiator special-cycles.

Bit 13 (TIBSE) in the LBC register specifies whether to enable byte swapping in target I/O cycles.

In target memory cycles, bit 8 (MSDSE) in the LBC register specifies whether to enable byte swapping. Bit 12 (TMFBSE) in the LBC register specifies the byte swapping area. When TMFBSE is 0, only the upper half of the programmed target memory space is configured to perform byte swapping. When TMBSE is 1, the entire target memory space is configured to perform byte swapping.

To recap, the TX3927 supports byte swapping for all the supported PCI commands, except for target configuration cycles.

The following table summarizes the byte swapping capability.

C/BE Value	PCI Command	TX3927 Supports as an Initiator	TX3927 Supports as a target
0000	Interrupt-acknowledge	Yes	—
0001	Special-cycle	Yes	—
0010	I/O read	Yes	Yes
0011	I/O write	Yes	Yes
0100	(Reserved)	—	—
0101	(Reserved)	—	—
0110	Memory read	Yes	Yes
0111	Memory write	Yes	Yes
1000	(Reserved)	—	—
1001	(Reserved)	—	—
1010	Configuration read	Yes	No
1011	Configuration write	Yes	No
1100	Memory read multiple	—	Yes
1101	Dual address cycle	—	—
1110	Memory read line	—	Yes
1111	Memory write and invalidate	—	Yes

Yes: Byte swap selectable

No: No byte swap

—: Commed not supported by the TX3927

Note: When the TX3927 CPU core accesses the PCIC registers, the data is not byte-swapped.

The following shows examples of byte swapping operations for different values of the data, addresses and byte enable signals.

The assumption is that the TX3927 is operating in big-endian mode. The tables show the data, addresses and byte enable signals when byte swapping is enabled and those when byte swapping is disabled.

Although the TX3927 address bus does not have the two least-significant bits of the address (A[1:0]), addresses are shown as if those bits were present in order to clarify byte offsets.

The data bus is shown as 32 bits; the valid byte lanes are underscored.

(1) I/O access when byte swapping is disabled

Big-endian TX3927 data bus			Byte swap disabled PCI bus		
A[1:0]	BE	Data	A[1:0]	BE	Data
0	7	01234567	3	7	01234567
1	B	01234567	2	B	01234567
2	D	01234567	1	D	01234567
3	E	01234567	0	E	01234567
0	3	01234567	2	3	01234567
2	C	01234567	0	C	01234567
0	0	01234567	0	0	01234567

(2) I/O accesses when byte swapping is enabled

Big-endian TX3927 data bus			Byte swap enabled PCI bus		
A[1:0]	BE	Data	A[1:0]	BE	Data
0	7	01234567	0	E	67452301
1	B	01234567	1	D	67452301
2	D	01234567	2	B	67452301
3	E	01234567	3	7	67452301
0	3	01234567	0	C	67452301
2	C	01234567	2	3	67452301
0	0	01234567	0	0	67452301

(3) Memory accesses when byte swapping is disabled

Big-endian TX3927 data bus			Byte swap disabled PCI bus		
A[1:0]	BE	Data	A[1:0]	BE	Data
0	7	01234567	0	7	01234567
1	B	01234567	0	B	01234567
2	D	01234567	0	D	01234567
3	E	01234567	0	E	01234567
0	3	01234567	0	3	01234567
2	C	01234567	0	C	01234567
0	0	01234567	0	0	01234567

(4) Memory access when byte swapping is enabled

Big-endian TX3927 data bus			Byte swap enabled PCI bus		
A[1:0]	BE	Data	A[1:0]	BE	Data
0	7	01234567	0	E	67452301
1	B	01234567	0	D	67452301
2	D	01234567	0	B	67452301
3	E	01234567	0	7	67452301
0	3	01234567	0	C	67452301
2	C	01234567	0	3	67452301
0	0	01234567	0	0	67452301

12.4.12 Disabling Access to a Part of the PCI Configuration Space

The TMPR3927AF has an additional bit in the PCIC that is not present in the TMPR3927F. This bit is used to prevent external PCI masters from accessing the PCI configuration space other than the predefined 64-byte PCI configuration header, and ACPI power management registers.

Bit 3 in the PCIC Local Bus Control (LBC) register serves that purpose. When this bit is cleared, external PCI masters are allowed to access the entire PCI configuration space (00h to ffh), in both the TMPR3927F and TMPR3927AF. When this bit is set, external PCI masters can only access the PCI configuration header space (00h to 3fh) and the ACPI registers (e0h to e7h). An attempt to read any other register in the PCI configuration space (registers used for TX3927 local settings) will result in 00h being read. Writes to those registers will be ignored.

This bit is initialized to 0 on reset. Therefore, the TMPR3927AF operates the same way as the TMPR3927F unless the bit setting is changed.

Table 12.4.1 summarizes the access permission for the PCI configuration space, depending on the settings of the LBC register.

Table 12.4.1 Access Permission for the PCI Configuration Area

	LBC Register		PCI Configuration Space			
	LBC[9]	LBC[3]	0-3f	40-df	e0-e7	e8-ff
TMPR3927F	0	—	Yes	Yes	Yes	Yes
	1	—	No	No	No	No
TMPR3927AF	0	0	Yes	Yes	Yes	Yes
	0	1	Yes	No	Yes	No
	1	0/1	No	No	No	No

Yes: An external PCI master can access the registers.

No: 00h is returned on a read; writes are ignored.

12.5 Timing Diagrams

Sections 12.5.1 to 12.5.6 show PCIC operations as an initiator. Sections 12.5.7 to 12.5.9 show PCIC operations as a target. It is assumed that the on-chip PCI bus arbiter is used.

12.5.1 Initiator Configuration Read

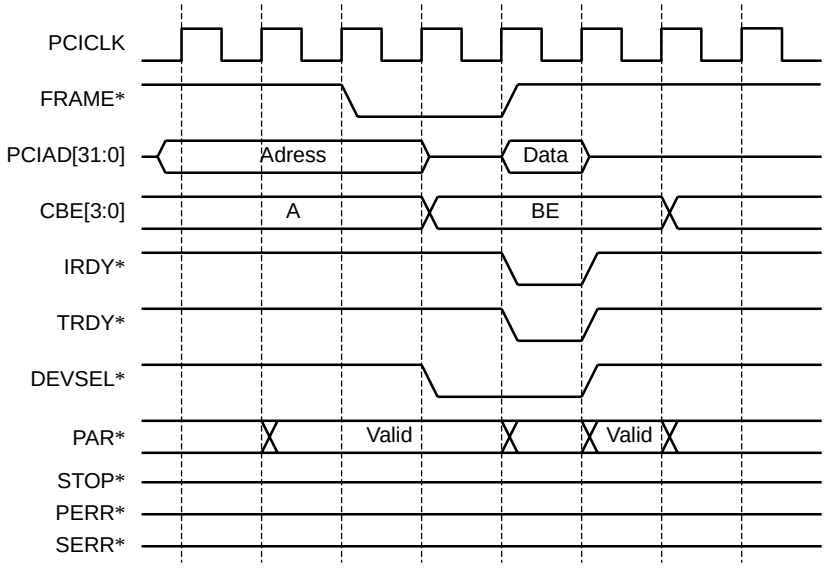


Figure 12.5.1 Initiator Configuration Read

12.5.2 Initiator Memory Read

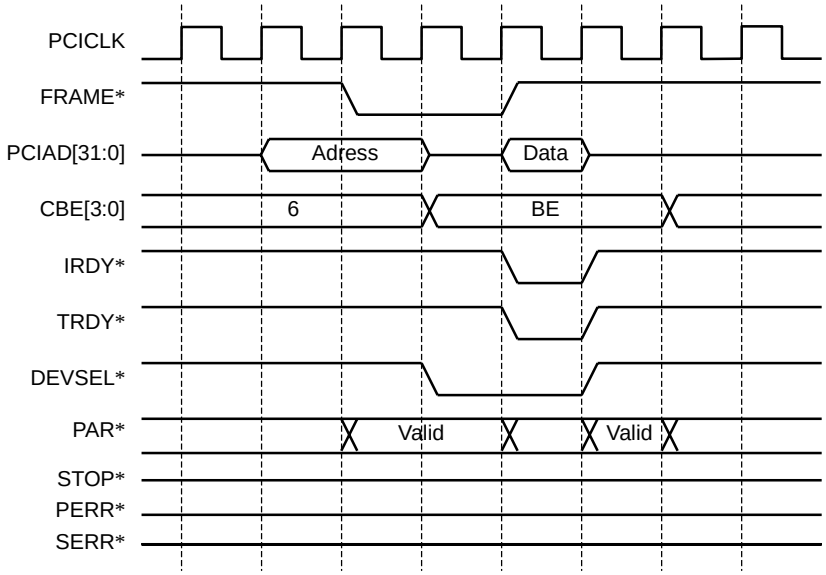


Figure 12.5.2 Initiator Memory Read

12.5.3 Initiator Memory Write

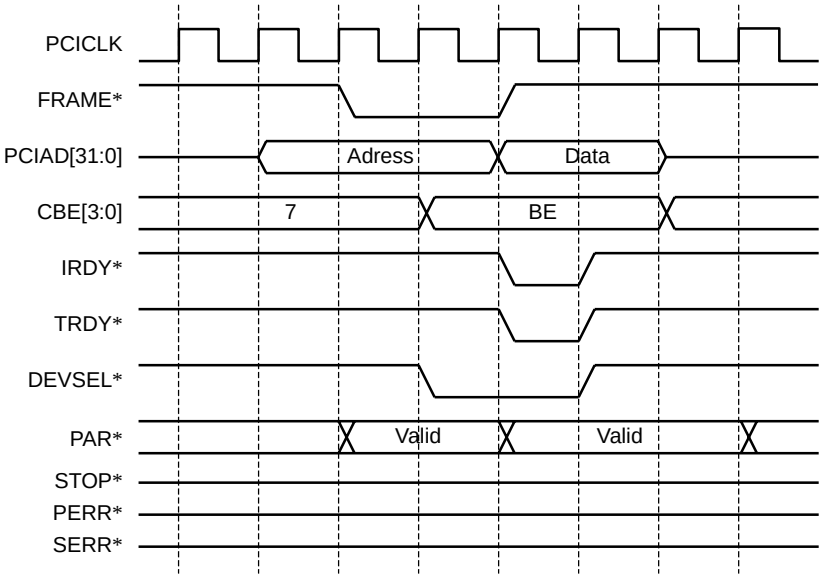


Figure 12.5.3 Initiator Memory Write

12.5.4 Initiator I/O Read

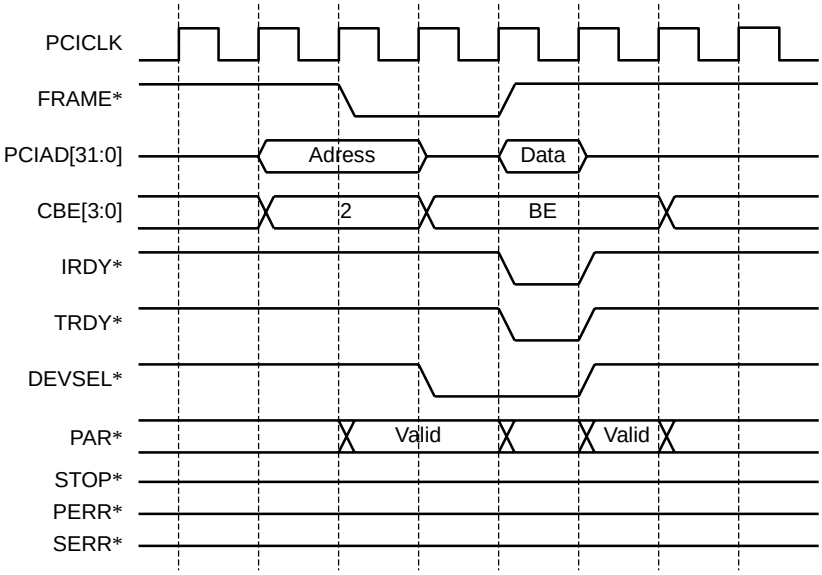


Figure 12.5.4 I/O Read

12.5.5 Initiator I/O Write

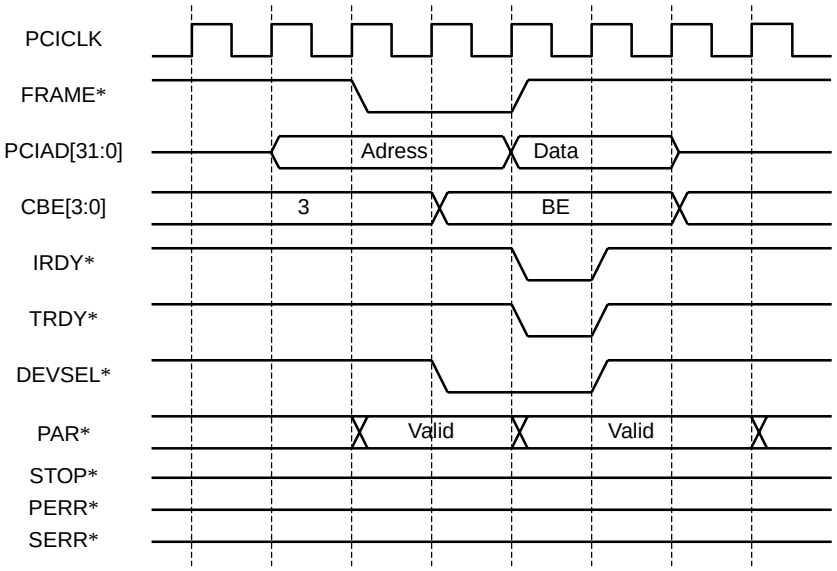


Figure 12.5.5 I/O Write

12.5.6 Special Cycle

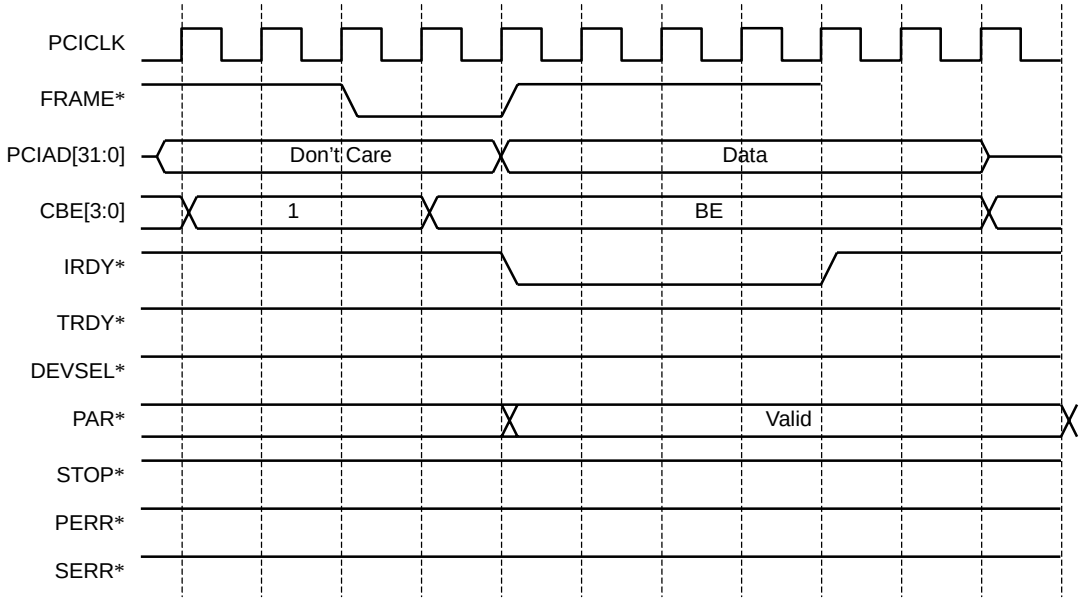


Figure 12.5.6 Special Cycle

12.5.7 Target Configuration Read

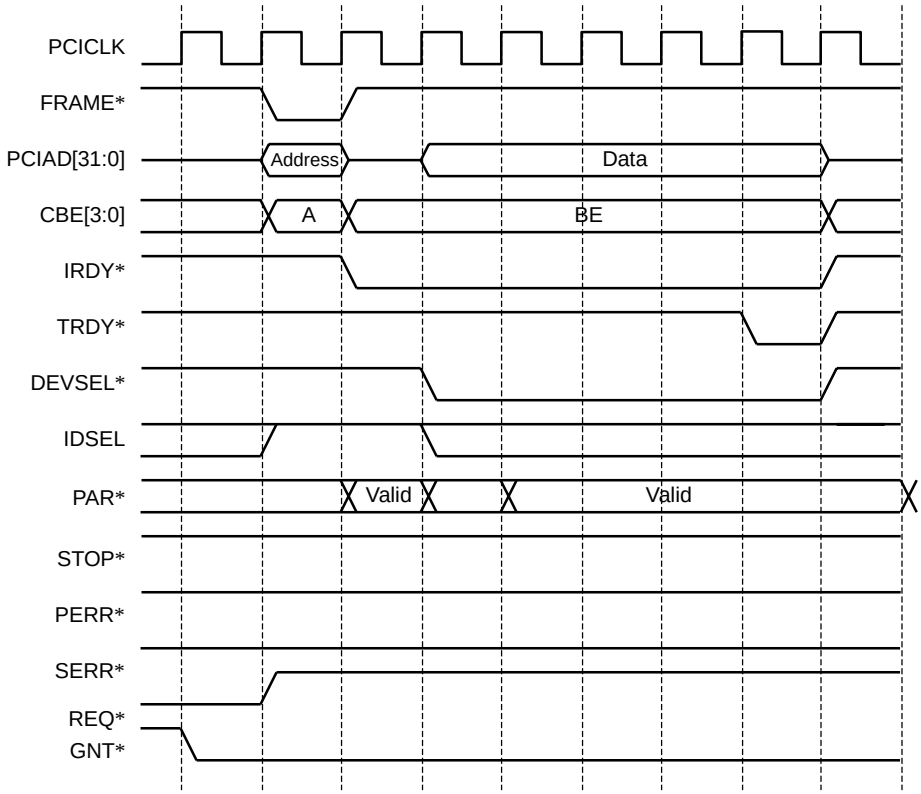
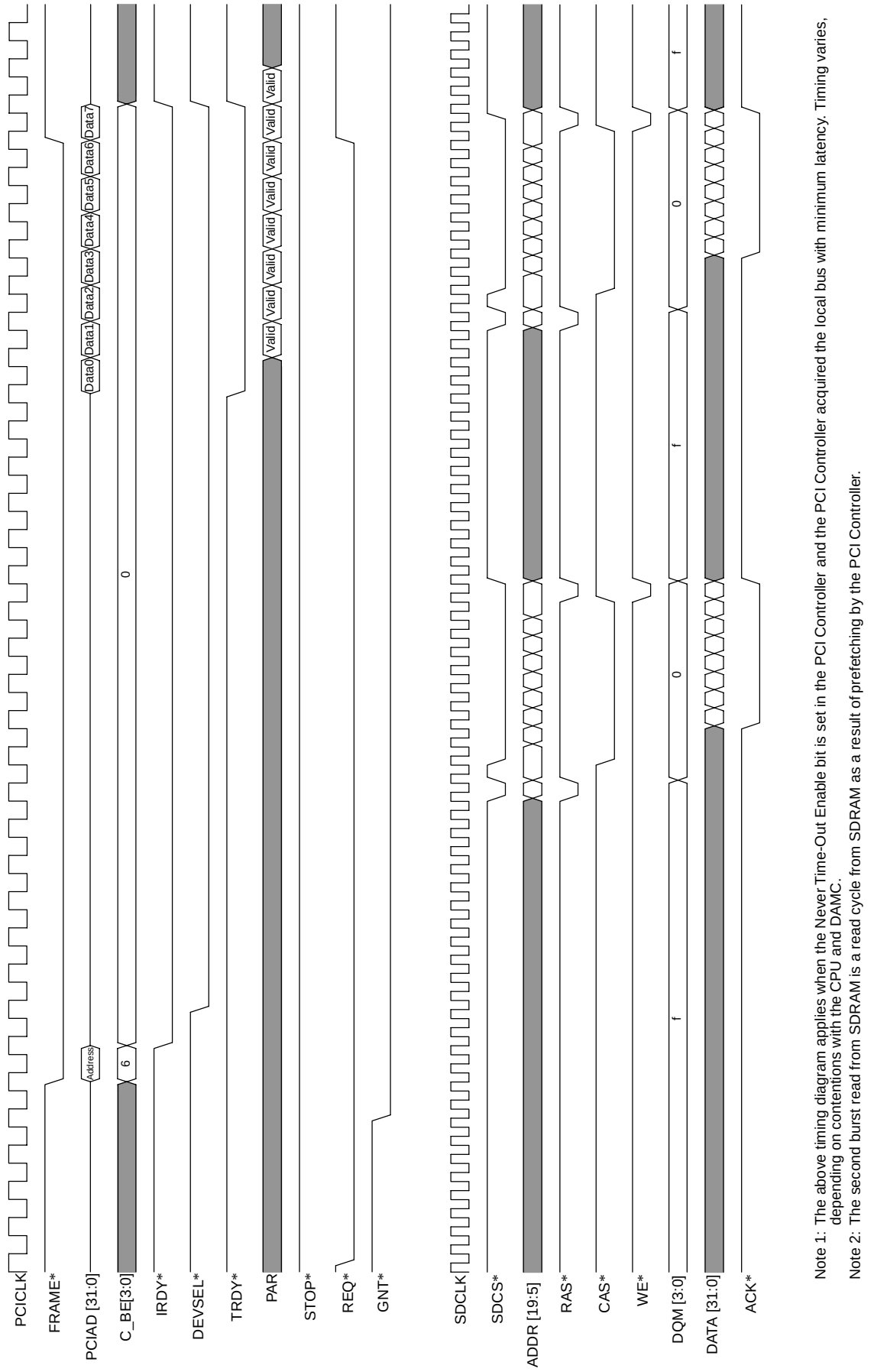


Figure 12.5.7 Target Configuration Read

12.5.8 8-word Burst Transfer from SDRAM to PCI



Note 1: The above timing diagram applies when the Never Time-Out Enable bit is set in the PCI Controller and the PCI Controller acquired the local bus with minimum latency. Timing varies, depending on contentions with the CPU and DAMC.

Note 2: The second burst read from SDRAM is a read cycle from SDRAM as a result of prefetching by the PCI Controller.

Figure 12.5.8 8-word Burst Transfer from SDRAM to PCI (t_{RCD}=2, t_{CASL}=2)

12.5.9 8-word Burst Transfer from PCI to SDRAM

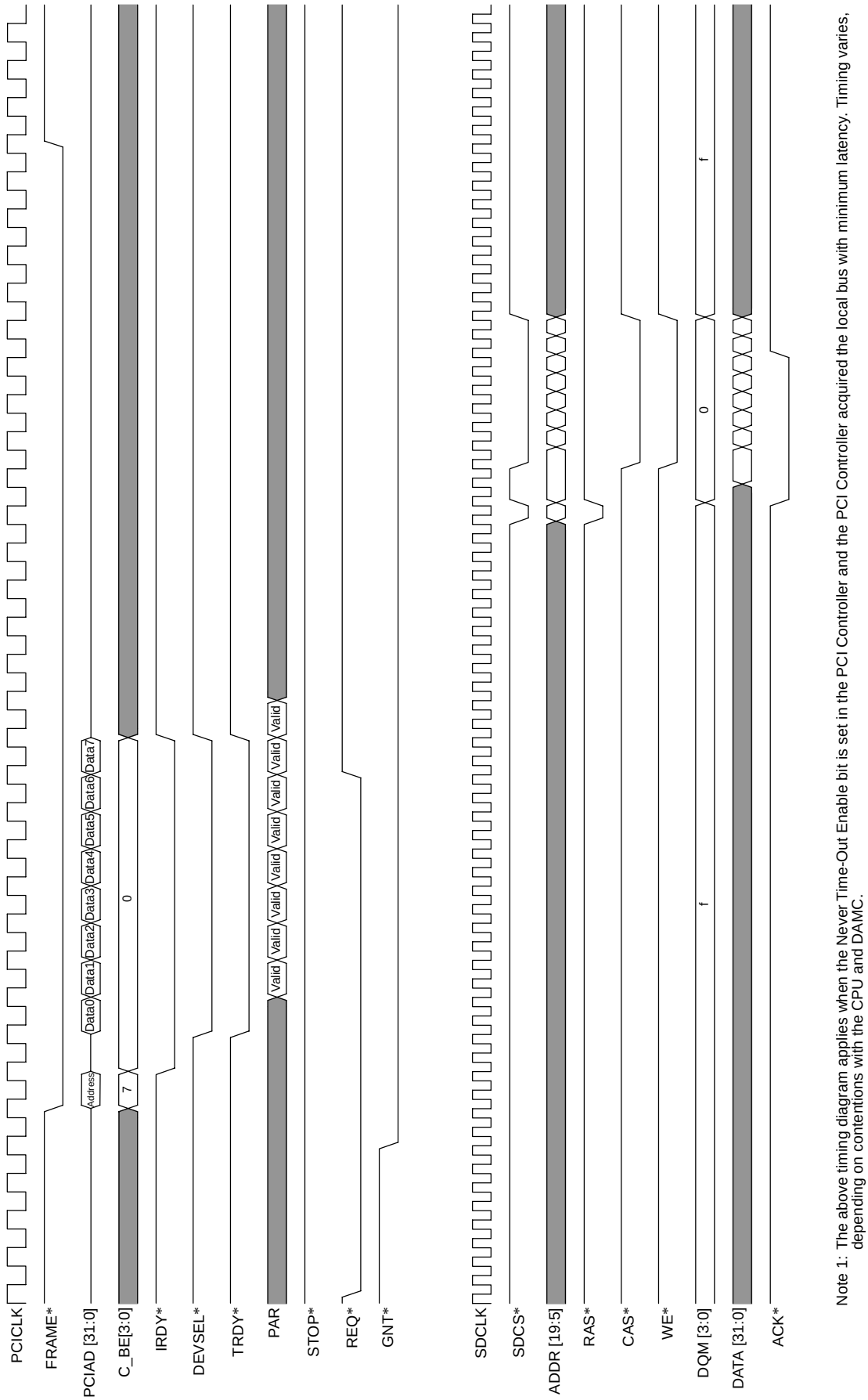


Figure 12.5.9 8-word Burst Transfer from PCI to SDRAM (t_{RCD}=2, t_{CASL}=2)

13. Serial I/O Ports (SIO)

13.1 Features

The TX3927 asynchronous serial interface has two full-duplex UART channels: SIO0 and SIO1.

The UART channels has the following features:

(1) Full-duplex (data can be sent in both directions)
(2) Baud rate generator
(3) Modem flow control (CTS*/RTS*)
(4) FIFOs
- Transmit FIFO: 8 × 8 bits
- Receive FIFO: 16 × 13 bits (8 bits for data and 5 bits for status)

(5) Multidrop operation
- Master and slave modes

13.2 Block Diagram

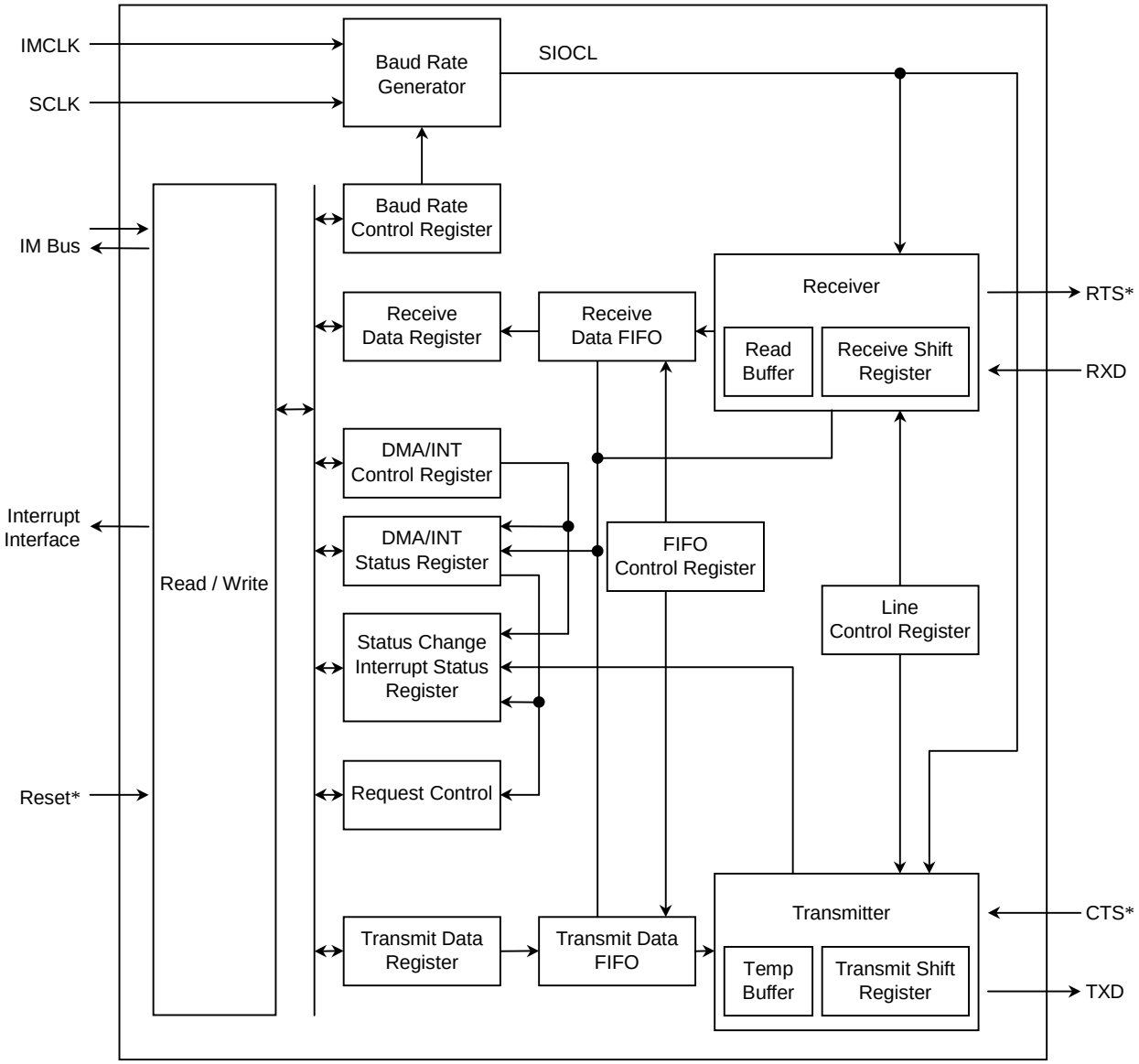


Figure 13.2.1 SIO Block Diagram

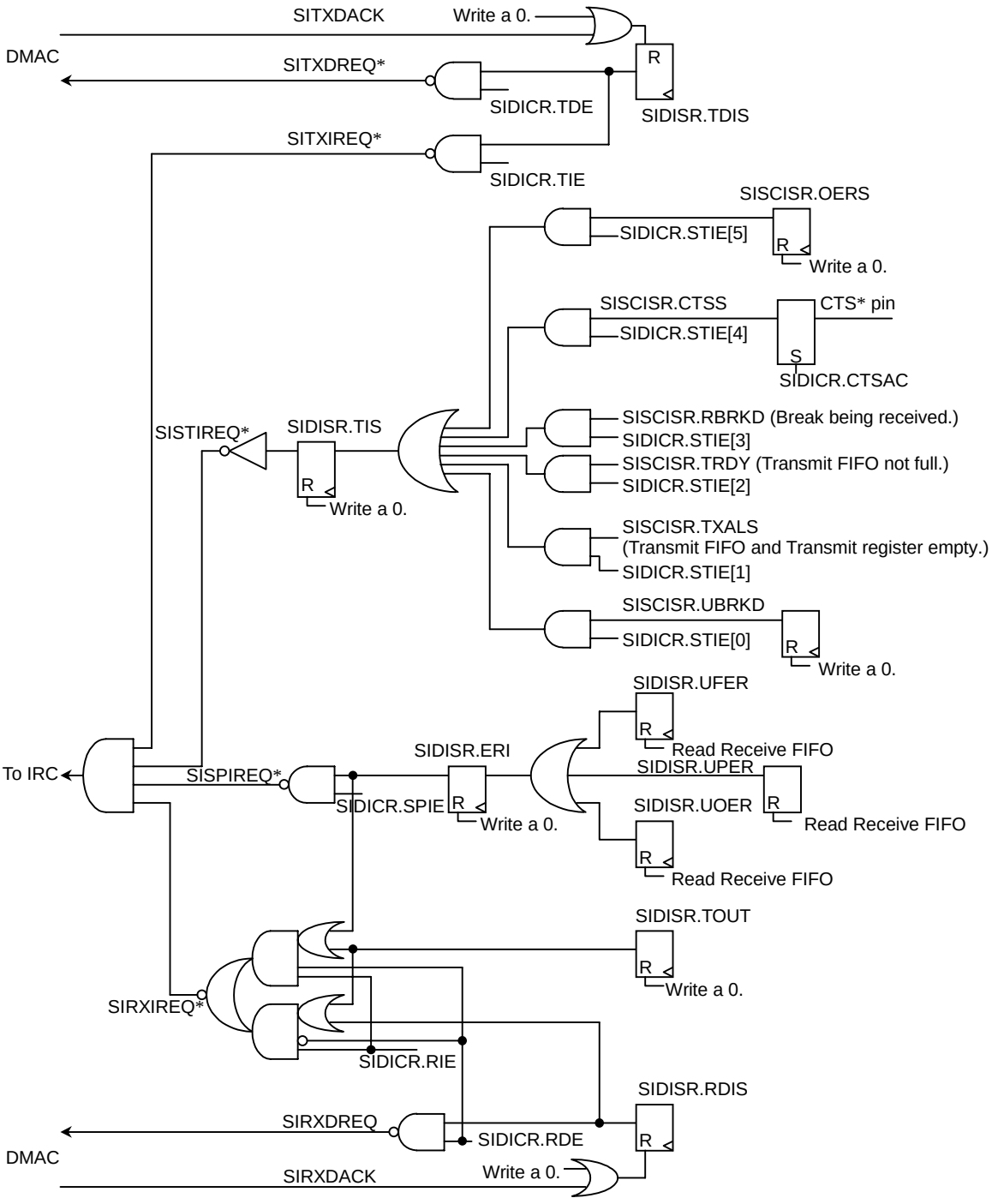


Figure 13.2.2 SIO Status Bits and an Interrupt Request Signal

13.3 Registers

13.3.1 Register Map

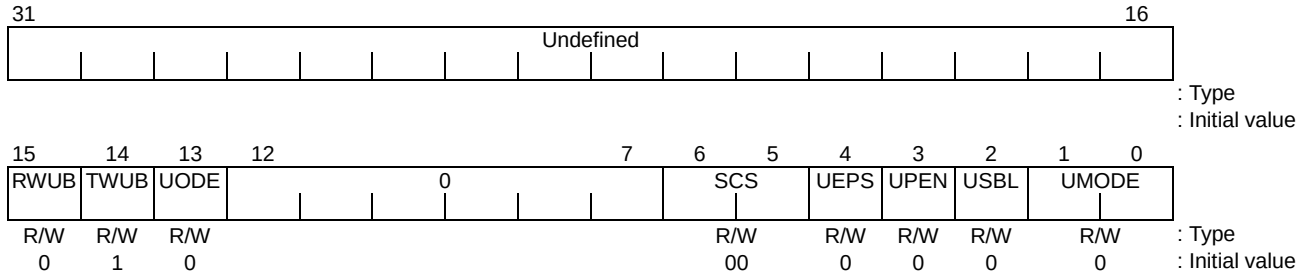
All the registers in the SIO should be accessed as a word quantity. For the bits other than those defined in this section, the values shown in the figures must be written.

Table 13.3.1 SIO Registers

Address	Register Mnemonic	Register Name
SIO1 (channel 1)		
0xFFFE_F420	SIRFIFO1	Receive FIFO Register 1
0xFFFE_F41C	SITFIFO1	Transmit FIFO Register 1
0xFFFE_F418	SIBGR1	Baud Rate Control Register 1
0xFFFE_F414	SIFLCR1	Flow Control Register 1
0xFFFE_F410	SIFCR1	FIFO Control Register 1
0xFFFE_F40C	SISCISR1	Status Change Interrupt Status Register 1
0xFFFE_F408	SIDISR1	DMA/Interrupt Status Register 1
0xFFFE_F404	SIDICR1	DMA/Interrupt Control Register 1
0xFFFE_F400	SILCR1	Line Control Register 1
SIO0 (channel 0)		
0xFFFE_F320	SIRFIFO0	Receive FIFO Register 0
0xFFFE_F31C	SITFIFO0	Transmit FIFO Register 0
0xFFFE_F318	SIBGR0	Baud Rate Control Register 0
0xFFFE_F314	SIFLCR0	Flow Control Register 0
0xFFFE_F310	SIFCR0	FIFO Control Register 0
0xFFFE_F30C	SISCISR0	Status Change Interrupt Status Register 0
0xFFFE_F308	SIDISR0	DMA/Interrupt Status Register 0
0xFFFE_F304	SIDICR0	DMA/Interrupt Control Register 0
0xFFFE_F300	SILCR0	Line Control Register 0

13.3.2 Line Control Registers (SILCRn) 0xFFFE_F300 (Ch. 0)
0xFFFE_F400 (Ch. 1)

The Line Control registers are used to configure the format of the asynchronous serial frame for data transmission and reception.



Bits	Mnemonic	Field Name	Description
15	RWUB	Receive Wake Up Bit	Receive Wake Up Bit (initial value: 0) 0: In multidrop operation, software must clear this bit when the TX3927 used in a slave station has recognized that it was addressed. The slave controller receives data from the master controller while this bit is cleared. 1: In multidrop operation, software must set this bit while the TX3927 used in a slave station is monitoring for an address (ID) frame from the master controller. The frame with the WUB bit set to 1 is interpreted as an address character, causing an interrupt request to be sent to the host upon reception. The frame with the WUB bit set to 0 is interpreted as a data character and discarded.
14	TWUB	Transmit Wake Up Bit	Transmit Wake Up Bit (initial value: 1) Selects the polarity of the wakeup bit (WUB) when the TX3927 is a master controller in a multidrop system. 0: Data frame 1: Address (ID) frame (default)
13	UODE	TXD Open-Drain Enable	TXD Open-Drain Enable (initial value: 0) In a multidrop system, slave controllers must have the TXD pin configured for open-drain operation. 0: Configured as a normal CMOS output. 1: Configured as an open-drain output.
6:5	SCS	SIO Clock Select	SIO Clock Select (initial value: 00) Selects the serial clock. The serial clock is always 16 times the baud rate. 00: Internal system clock (IMCLK, which is 1/4 of the 133-MHz CPU clock) 01: Baud rate generator (input clock: IMCLK) 10: External clock (SCLK) 11: Baud rate generator (input clock: SCLK)
4	UEPS	UART Even Parity Select	UART Even Parity Select (initial value: 0) Selects even or odd parity. 0: Odd parity 1: Even parity UPEN UEPS Description 1 0 Odd parity 1 1 Even parity 0 * Parity disabled
3	UPEN	UART Parity Check Enable	UART Parity Enable (initial value: 0) 0: Parity disabled. 1: Parity enabled. This bit must be cleared in a multidrop system (UMODE = 10 or 11).

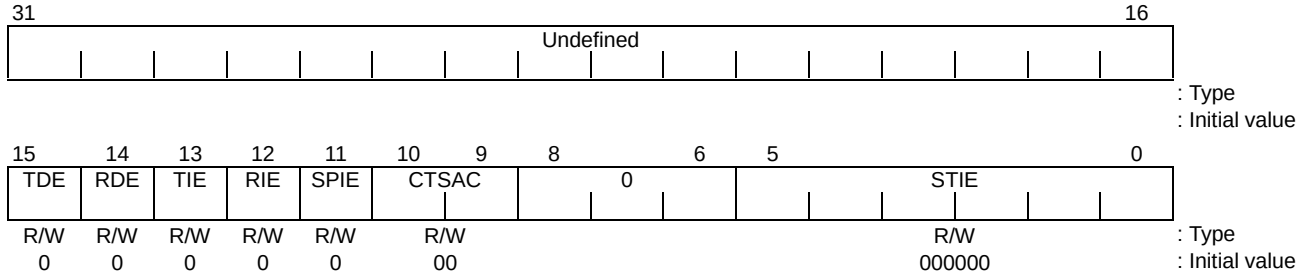
Figure 13.3.1 Line Control Register (1/2)

Bits	Mnemonic	Field Name	Description
2	USBL	UART Stop Bit Length	UART Stop Bit Length (initial value: 0) Specifies the number of stop bits. 1: 1 stop bit 0: 2 stop bits
1:0	UMODE	UART Mode	UART Mode (initial value: 00) Specifies the SIO data word length. 00: 8 data bits 01: 7 data bits 10: 8 data bits in multidrop mode 11: 7 data bits in multidrop mode

Figure 13.3.1 Line Control Register (2/2)

13.3.3 DMA/Interrupt Control Registers (SIDICRn) 0xFFFE_F304 (Ch. 0)
0xFFFE_F404 (Ch. 1)

These registers are used to control host interfacing using DMA or interrupts.



Bits	Mnemonic	Field Name	Description
15	TDE	Transmit DMA Enable (SITXDREQ*)	Transmit DMA Enable (initial value: 0) Controls whether to assert the internal SITXDREQ* signal when the SIDISR.TDIS bit is set. 0: Not asserted 1: Asserted
14	RDE	Receive DMA Enable (SIRXDREQ)	Receive DMA Enable (initial value: 0) Controls whether to assert the internal SIRXDREQ signal when the SIDISR.RDIS bit is set. However, SIRXDREQ is not asserted if the SIDISR.ERI bit is set. 0: Not asserted 1: Asserted
13	TIE	Transmit Data Interrupt Enable (SITXIREQ*)	Transmit Interrupt Enable (initial value: 0) Controls whether to assert the internal SITXIREQ* signal when the SIDISR.TDIS bit is set. 0: Not asserted 1: Asserted
12	RIE	Receive Data Interrupt Enable (SIRXIREQ*)	Receive Interrupt Enable (initial value: 0) In DMA receive mode (RDE = 1) Controls whether to asserts the internal SIRXIREQ* signal when either the SIDISR.ERI or SIDISR.TOUT bit is set. 0: Asserted 1: Not asserted (Don't use.) In any other mode (RDE = 0) Controls whether to asserts the internal SIRXIREQ* signal when either the SIDISR.TOUT or SIDISR RDIS bit is set. 0: Not asserted 1: Asserted
11	SPIE	Special Receive Interrupt Enable (SISPIREQ*)	Special Receive Interrupt Enable (initial value: 0) Controls whether to asserts the internal SISPIREQ* signal when the SIDISR.ERI bit is set. 0: Not asserted 1: Asserted
10:9	CTSAC	CTSS Active Condition	CTSS Active Condition (initial value: 00) Specifies the change on the CTS* pin to be treated as a modem status change regarding the interrupt enable bit in the STIE field. 00: Disable 01: Rising edge on the CTS* pin 10: Falling edge on the CTS* pin 11: Both rising and falling edges on the CTS* pin

Note: Refer to Table 13.4.3 for the possible combinations of bit settings.

Figure 13.3.2 DMA/Interrupt Control Registers (1/2)

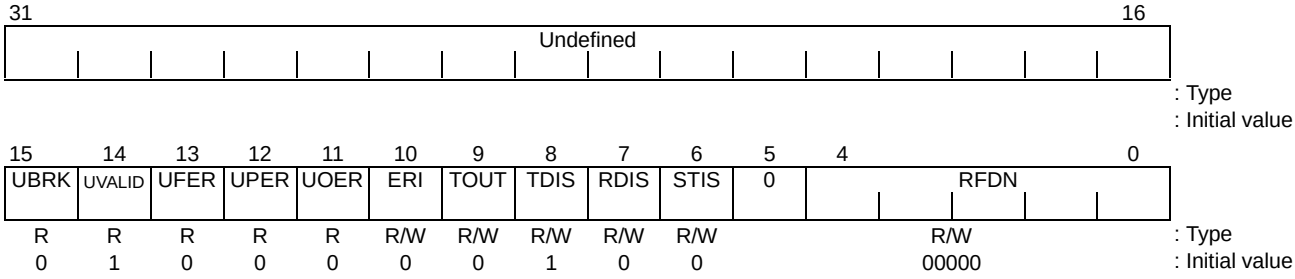
Bits	Mnemonic	Field Name	Description
5:0	STIE	Status Change Interrupt Request (SISTIREQ*)	Status Change Interrupt Enable Channel 1 (initial value: 0x00) Specifies when to set the SIDISR.STIS bit. The bits in this field correspond to the status conditions available in the Status Change Interrupt Status register (SISCISR). Multiple bits can be 1. The internal SISTIREQ* signal is asserted when the SIDISR.STIS bit is set. 000000: Disable 1*****: Sets STIS to 1 when OERS is set. *1****: Sets STIS to 1 when the change specified by CTSAC occurs in CTSS. **1***: Sets STIS to 1 when RBRKD is set. ***1**: Sets STIS to 1 when TRDY is set. ****1*: Sets STIS to 1 when TXALS is set. *****1: Sets STIS to 1 when UBRKD is set.

Figure 13.3.2 DMA/Interrupt Control Registers (2/2)

13.3.4 DMA/Interrupt Status Registers (SIDISRn)

0xFFFE_F308 (Ch. 0)
0xFFFE_F408 (Ch. 1)

These registers provide DMA and interrupt status information.



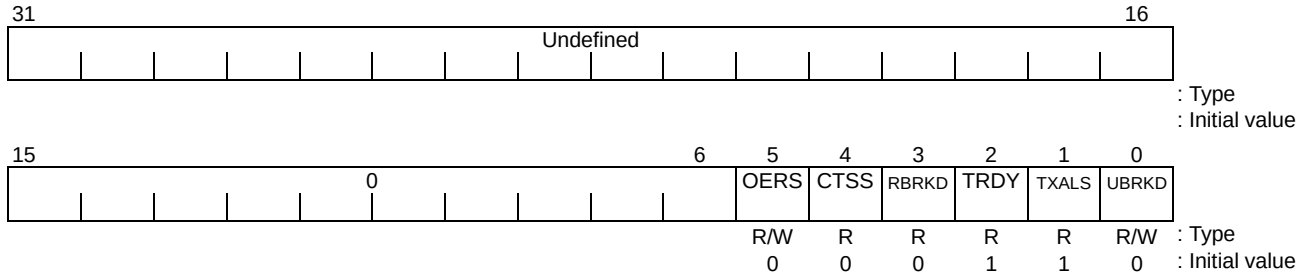
Bits	Mnemonic	Field Name	Description
15	UBRK	UART Break Reception	UART Break (initial value: 0) This bit is set when an entire frame that was received into the FIFO with break indication is now at the top of the FIFO. This status bit is automatically updated by a read of the Receive FIFO register (SIRFIFO). 0: Not break 1: Break detected
14	UVALID	UART Receiver FIFO Available Status	UART Available Data (initial value: 1) This bit is set when there is a byte in the Receive FIFO register (SIRFIFO). 0: Data is present in the Receive FIFO. 1: No received data is ready to read.
13	UFER	UART Frame Error	UART Frame Error (initial value: 0) This bit is set when the frame that was received into the FIFO with a framing error status is now at the top of the FIFO. This status bit is automatically updated by a read of the Receive FIFO register (SIRFIFO). 0: Frame error not detected 1: Frame error detected
12	UPER	UART Parity Error	UART Parity Error (initial value: 0) This bit is set when the frame that was received into the FIFO with a parity error status is now at the top of the FIFO. This status bit is automatically updated by a read of the Receive FIFO register (SIRFIFO). 0: Parity error not detected 1: Parity error detected
11	UOER	UART Overrun Error	UART Overrun Error (initial value: 0) This bit is set when the frame that was received into the FIFO with an overrun error status is now at the top of the FIFO. This status bit is automatically updated by a read of the Receive FIFO register (SIRFIFO). 0: No overrun error 1: Overrun error detected
10	ERI	Error Interrupt	Error Interrupt (initial value: 0) This bit is immediately set on detection of a receive error (framing error, parity error or overrun error). This bit is cleared by writing a 0. Writing a 1 has no effect.
9	TOUT	Receive Time-Out	Time Out (initial value: 0) This bit is set when a receive time-out occurs. This bit is cleared by writing a 0. Writing a 1 has no effect.

Figure 13.3.3 DMA/Interrupt Status Registers (1/2)

Bits	Mnemonic	Field Name	Description
8	TDIS	Transmit Data Empty	Transmit DMA/Interrupt Status (initial value: 1) This bit is set when the Transmit FIFO now has empty locations specified by SIFCR.TDIL. - Interrupt mode (SIDICR.TIE = 1) The internal SITXIREQ* signal is asserted when this bit is set. Writing a 0 to this bit clears it and deasserts the SITXIREQ* signal. - DMA transfer mode (SIDICR.TDE = 1) The internal SITXDREQ* signal is asserted when this bit is set. The SITXDACK signal from the DMA Controller clears this bit and deasserts the SITXDREQ* signal (see Figure 13.5.2).
7	RDIS	Receive Data Full	Receive DMA/Interrupt Status (initial value: 0) This bit is set when the Receive FIFO now has valid data above the threshold specified by SIFCR.RDIL. - Interrupt mode (SIDICR.RIE = 1) The internal SIRXIREQ* signal is asserted when this bit is set. Writing a 0 to this bit clears it and deasserts the SIRXIREQ* signal. - DMA transfer mode (SIDICR.RDE = 1) The internal SIRXDREQ signal is asserted when this bit is set. The SIRXDACK signal from the DMA Controller clears this bit and deasserts the SIRXDREQ signal (see Figure 13.5.2).
6	STIS	Status Change Interrupt Status	Status Change Interrupt Status (initial value: 0) This bit is set when at least one status bit selected by SIDICR.STIE is set. The internal SISTREQ signal is asserted when this bit is set. Writing a 0 to this bit clears it and deasserts the SISTREQ signal.
4:0	RFDN	Receive Data Stage Status	Receive FIFO Data Number (initial value: 00000) Indicates the number of valid characters present in the Receive FIFO (0 to 16).

Figure 13.3.3 DMA/Interrupt Status Registers (2/2)

13.3.5 Status Change Interrupt Status Registers (SISCISRn) 0xFFFE_F30C (Ch. 0) 0xFFFE_F40C (Ch. 1)

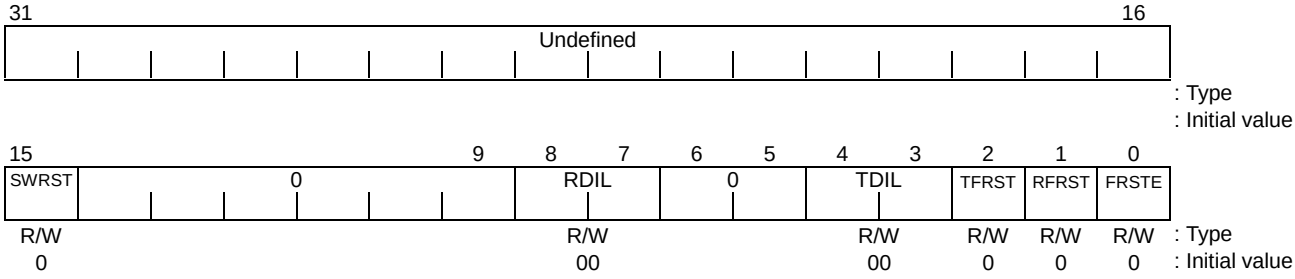


Bits	Mnemonic	Field Name	Description
5	OERS	Overrun Error Status	Overrun Error Status (initial value: 0) This bit is set when an overrun error is detected. This bit is cleared by writing a 0.
4	CTSS	CTS* Terminal Status	CTS* Terminal Status (initial value: 0) Provides the current status of the CTS* signal. 1: The CTS* signal is high. 0: The CTS* signal is low.
3	RBRKD	Receive Break	Receive Break (initial value: 0) This bit is set on detection of a break. It is automatically cleared on reception of a non-break frame. 1: There is a break indication associated with the character being received. 0: There is no break indication.
2	TRDY	Transmit Data Empty	Transmit Ready (initial value: 1) This bit is set when there is at least one empty location in the Transmit FIFO.
1	TXALS	Transmission Completed	Transmit All Sent (initial value: 1) This bit is set when the Transmit FIFO is empty and the transmit shift register has no valid data.
0	UBRKD	UART Break Detect	UART Break Detect (initial value: 0) This bit is set on detection of a break. Once set, this bit remains set until it is cleared by writing a 0.

Figure 13.3.4 Status Change Interrupt Status Registers

13.3.6 FIFO Control Registers (SIFCRn) 0xFFFE_F310 (Ch. 0)
0xFFFE_F410 (Ch. 1)

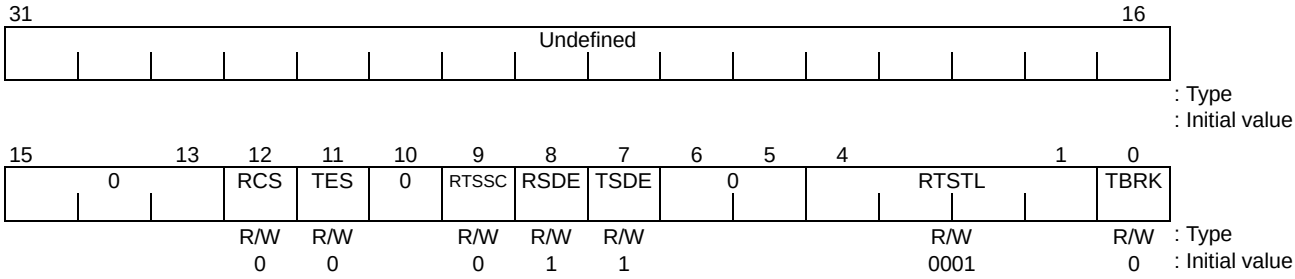
These registers are used to control the Receive and Transmit FIFO buffers.



Bits	Mnemonic	Field Name	Description
15	SWRST	Software Reset	Software Reset (initial value: 0) Writing a 1 to this bit performs a software reset of the SIO. This bit clears itself when the SIO reset. 0: Normal operation 1: SIO software reset
8:7	RDIL	Receive FIFO Request Trigger Level	Receive FIFO DMA/Interrupt Trigger Level (initial value: 00) Selects the number of characters required in the Received FIFO before the RDIS bit in the SIDISR register is set to report that received data is available. 00: 1 byte 01: 4 bytes 10: 8 bytes 11: 12 bytes
4:3	TDIL	Transmit FIFO Request Trigger Level	Transmit FIFO DMA/Interrupt Trigger Level (initial value: 00) Selects the number of unfilled locations required in the Transmit FIFO before the TDIS bit in the SIDISR register is set to report that it can accept next data. 00: 1 byte 01: 4 bytes 10: 8 bytes 11: Don't use.
2	TFRST	Transmit FIFO Reset	Transmit FIFO Reset (initial value: 0) Writing a 1 to this bit resets the Transmit FIFO. It is valid when the FRSTE bit is 1. 1: Reset the Transmit FIFO.
1	RFRST	Receive FIFO Reset	Receive FIFO Reset (initial value: 0) Writing a 1 to this bit resets the Receive FIFO. It is valid when the FRSTE bit is 1. 1: Reset the Receive FIFO.
0	FRSTE	FIFO Reset Enable	FIFO Reset Enable (initial value: 0) Enables the resetting the Receive and Transmit FIFOs. When this bit is 1, writing a 1 to the TFRST or RFRST bit resets the corresponding FIFO. 1: Reset enable.

Figure 13.3.5 FIFO Control Registers

13.3.7 Flow Control Registers (SIFLCRn) 0xFFFE_F314 (Ch. 0)
0xFFFE_F414 (Ch. 1)

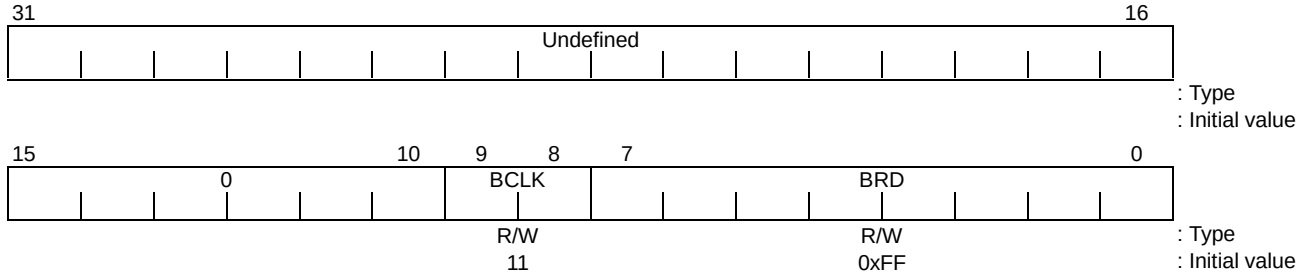


Bits	Mnemonic	Field Name	Description
12	RCS	RTS Control Select	RTS Control Select (initial value: 0) Specifies how the RTS* signal is controlled. 0: The RTS* signal is software controllable (RTSSC bit). 1: The RTS* signal is both software (RTSSC bit) and hardware (RTSTL bit) controllable.
11	TES	Transmit Enable Select	Transmit Enable Select (initial value: 0) Selects the type of a transmission request. 0: Software command (TSDE bit) 1: Software command and CTS* hardware signal
9	RTSSC	RTS Software Control	RTS Software Control (initial value: 0) This bit controls the RTS* signal. 0: The RTS* signal is forced low. 1: The RTS* signal is forced high.
8	RSDE	Receive Serial Data Enable	Receive Serial Data Enable (initial value: 1) This bit is used to request the reception of serial data under software control. If this bit is set, the received data is discarded. 0: Reception enabled 1: Reception disabled
7	TSDE	Transmit Serial Data Enable	Transmit Serial Data Enable (initial value: 1) This bit is used to request the transmission of serial data under software control. If this bit is set, the SIO halts transmission after completing any current character. 0: Transmission enabled 1: Transmission disabled
4:1	RTSTL	RTS Active Trigger Level	RTS Trigger Level (initial value: 0001) Specifies the number of characters required in the Receive FIFO before the RTS* hardware signal is asserted. 0000: Don't use. 0001: 1 data byte : 1111: 15 data bytes
0	TBRK	Break Transmit	Break Transmit (initial value: 0) When set, the transmitter sends a break. 0: Disabled 1: Enabled

Figure 13.3.6 Flow Control Registers

13.3.8 Baud Rate Control Registers (SIBGRn) 0xFFFE_F318 (Ch. 0)
0xFFFE_F418 (Ch. 1)

These registers specify the clock source and divisor for the baud rate generator.

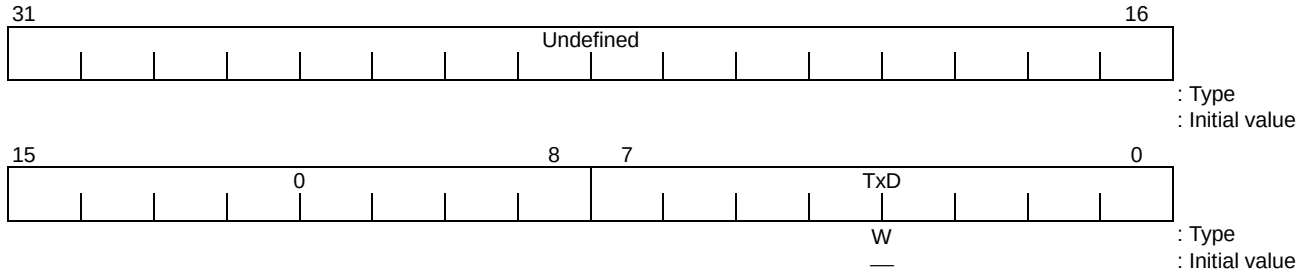


Bits	Mnemonic	Field Name	Description
9:8	BCLK	Baud Rate Generator Clock	Baud Rate Generator Clock (initial value: 11) Selects the clock source for the baud rate generator. 00: T0 prescaler output (IMCLK/2) 01: T2 prescaler output (IMCLK/8) 10: T4 prescaler output (IMCLK/32) 11: T6 prescaler output (IMCLK/128)
7:0	BRD	Baud Rate Divide Value	Baud Rate Divide Value (initial value: 0xFF) Baud rate is selected by writing a binary division ratio into this field.

Figure 13.3.7 Baud Rate Control Registers

13.3.9 Transmit FIFO Registers (SITFIFOn)

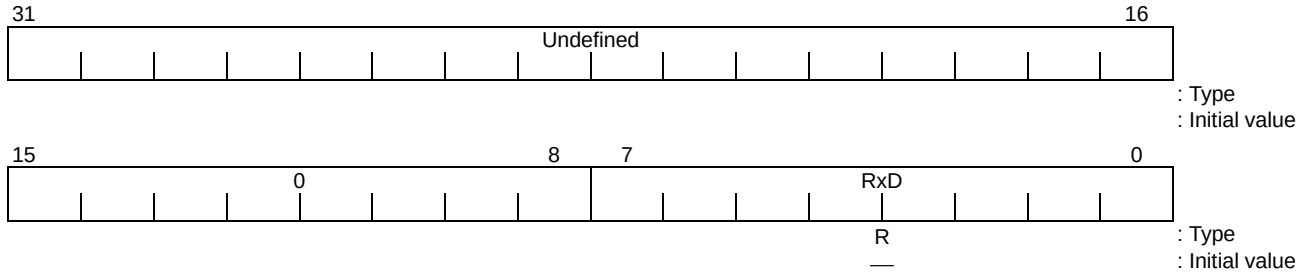
0xFFFE_F31C (Ch. 0)
0xFFFE_F41C (Ch. 1)



Bits	Mnemonic	Field Name	Description
7:0	TxD	Transmit Data	Transmit Data This register is used to write the transmit data to the Transmit FIFO.

Figure 13.3.8 Transmit FIFO Registers

13.3.10 Receive FIFO Registers (SIRFIFOn) 0xFFFE_F320 (Ch. 0)
0xFFFE_F420 (Ch. 1)



Bits	Mnemonic	Field Name	Description
7:0	RxD	Receive Data	Receive Data This register is used to read the received data from the Receive FIFO. Reading this register advances the FIFO read pointer.

Figure 13.3.9 Receive FIFO Registers

13.4 Operation

13.4.1 Overview

The TX3927 SIO converts serial data received on the external RXD pin into parallel data using the internal receive shift register. Once an entire UART frame has been received, the character is transferred from the internal receive shift register into the Receive FIFO buffer. The received data can then be picked up by the CPU or DMA Controller.

The TX3927 SIO also converts parallel data into serial data for transmission off the chip on the TXD pin. For transmission, data is written to the Transmit FIFO buffer by the CPU or DMA Controller. The first FIFO entry is then transferred into the internal transmit shift register. Once data has been latched into the transmit shift register, it is directly shifted out of the TDX pin.

Both the receiver and transmitter use a clock 16× faster than the baud rate. To generate the baud rate of the transfer, a specified clock is divided by a divisor value chosen by the programmer. The baud rate generator automatically calculates the baud rate from the divisor value programmed into the Baud Rate Control Register (SIBGRn).

13.4.2 Data Format

The TX3927 SIO's programmable serial interface includes:

Data length: 9, 8, or 7 bits (9-bit data supports multidrop operation.)
Stop bits: 1 or 2 bits
Parity bit: Optional
Parity type: Even or odd

The start bit is fixed at 1 bit.

Figure 13.4.1 shows the data frame configuration.

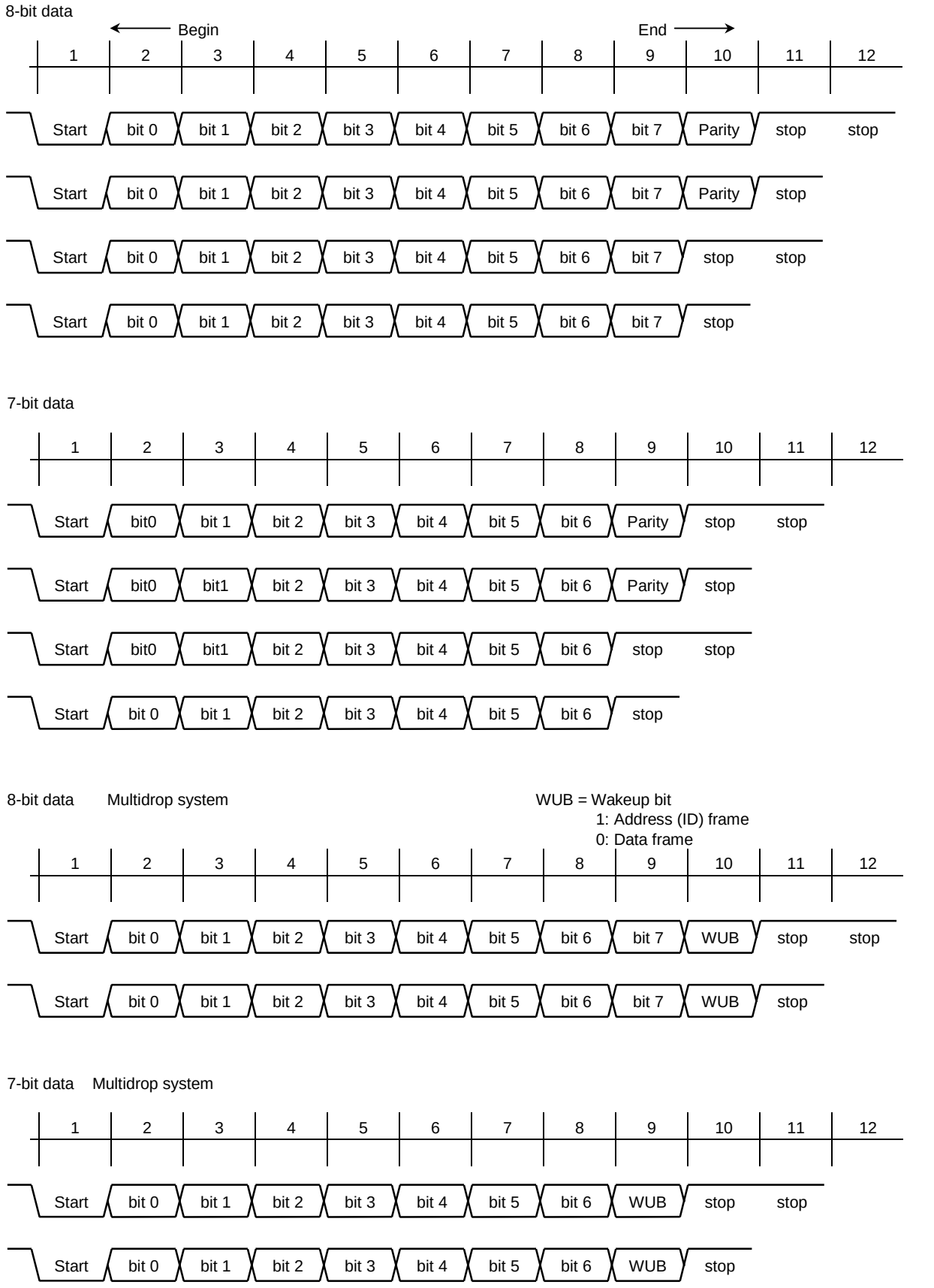


Figure 13.4.1 Data Frame Configuration

13.4.3 Serial Clock Generator

SIOCLK, the transmit/receive clock which determines the serial transfer rate, can be selected from the output from the baud rate generator, the internal system clock (IMCLK), or the external serial clock input (SCLK). IMCLK is one-fourth the normal CPU clock frequency. (If the CPU frequency is 133 MHz, then IMCLK is 33.25 MHz.)

The SCLK frequency must be less than half the IMCLK frequency; refer to Chapter 17, "Electrical Characteristics."

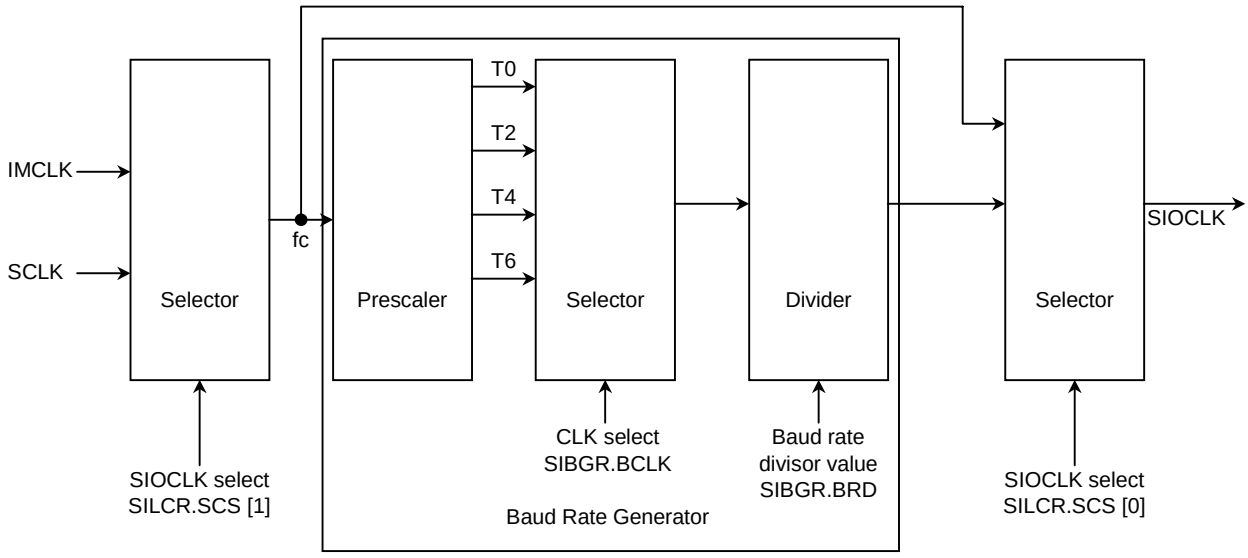


Figure 13.4.2 Baud Rate Generator and SIOCLK Generator

13.4.4 Baud Rate Generator

The frequency used to transmit and receive data through the SIO is derived from the baud rate generator. The baud rate is determined by the following equation:

$$\text{Baud rate} = \frac{\text{Baud rate generator clock input}}{\text{Baud rate generator divisor value}} \div 16$$

Where the baud rate generator clock input is selectable from prescaler outputs T0, T2, T4 and T6. The prescaler clock input (fc) can come from either IMCLK or an external clock (SCLK) input.

The baud rate generator divides the rate of the selected baud rate generator clock input by divisors of 1 to 255 programmed in the Baud Rate Control Register.

The baud rate generator produces a clock with a frequency 16 times that of the desired baud rate. Table 13.4.1 shows the decimal values of the divisors required to program into the latches of the SIO to obtain the required baud rate for various IMCLK frequencies. Table 13.4.2 lists divisor values to use to achieve common baud rates.

Table 13.4.1 Divisors and Baud Rates (in kbps)

IMCLK [MHz]	BRD divisor	Baud Rate Generator Output Clock			
		T0	T2	T4	T6
		fc/2	fc/8	fc/32	fc/128
33.25	18	57.73	14.43	3.61	0.90
33.25	28	37.11	9.28	2.32	0.58
33.25	54	19.24	4.81	1.20	0.30
33.25	72	14.43	3.61	0.90	0.23
33.25	108	9.62	2.41	0.60	0.15
33.25	216	4.81	1.20	0.30	0.08
32.00	26	38.46	9.62	2.40	0.60
32.00	52	19.23	4.81	1.20	0.30
32.00	104	9.62	2.40	0.60	0.15
32.00	208	4.81	1.20	0.30	0.08
24.57	10	76.78	19.20	4.80	1.20
24.57	20	38.39	9.60	2.40	0.60
24.57	40	19.20	4.80	1.20	0.30
24.57	80	9.60	2.40	0.60	0.15
24.57	160	4.80	1.20	0.30	0.07
16.00	26	19.23	4.81	1.20	0.30
16.00	52	9.62	2.40	0.60	0.15

Table 13.4.2 Baud Rates and Divisors for 33.25-MHz Clock

IMCLK [MHz]	kbps	fc/2	fc/8	fc/32	fc/128
33.25	0.11				148
33.25	0.15				108
33.25	0.3			216	54
33.25	0.6			108	28
33.25	1.2		216	54	14
33.25	2.4		108	28	6
33.25	4.8	216	54	14	4
33.25	9.6	108	27	7	2
33.25	14.4	72	18	5	1
33.25	19.2	54	14	3	
33.25	28.8	36	9	2	
33.25	38.4	27	7	2	
33.25	57.6	18	5	1	
33.25	76.8	14	3		

13.4.5 Receive Controller

The receive controller enables the receiver for data reception when the SIFLCR.RSDE bit is set. Reception of a frame is initiated when a start bit is received on the RXD pin.

The receiver controller looks for the high-to-low transition of a start bit on the RXD pin. Therefore, if RXD is low when the SIFLCR.RSDE bit is set, the start bit is invalid. When a valid start bit has been detected, the receive controller begins sampling data received on the RXD pin.

Data reception is based on 2-of-3 majority vote. The receiver uses SIOCLK 16× faster than the baud rate, and oversamples the start bit and each bit of the incoming data three times around their center (with 7th to 9th clocks). The value of the bit is determined by the majority of those samples.

13.4.6 Receive Shift Register

The receive shift register is eight bits in length. Received data is serially shifted into the receive shift register, least-significant bit (bit 0) first.

13.4.7 Receive Read Buffer

The receiver's read buffer is placed between the receive shift register and the Receive FIFO. Once an entire data frame has been received, it is transferred to the read buffer and a parity check is performed.

13.4.8 Transmit Controller

Data is transferred from the Transmit FIFO buffer to the transmit shift register when the shift register has completed transmission of the previous character. One bit is sent out every 16 SIOCLK cycles.

13.4.9 Transmit Shift Register

The transmit shift register is eight bits in length. Transmit data is serially shifted out, least-significant bit (bit 0) first.

13.4.10 Host Interface

The SIO asserts the internal interrupt or DMA request signal to indicate that it is ready to accept data in the Transmit FIFO. The Transmit FIFO has a trigger level programmable via the SIFCR.TDIL bit field at 1, 4 or 8 bytes present.

The SIO also asserts the internal interrupt or DMA request signal to indicate that it has data in the Receive FIFO to be picked up. The Receive FIFO has a trigger level programmable via the SIFCR.RDIL bit field at 1, 4, 8 or 12 bytes present.

Table 13.4.3 Register Bit Settings and Transmit/Receive Operations

TDE	RDE	TIE	RIE	Transmit	Receive
0	0	0	0	TDIS bit polled	RDIS bit polled
0	0	0	1	TDIS bit polled	Interrupt-driven
0	0	1	0	Interrupt-driven	RDIS bit polled
0	0	1	1	Interrupt-driven	Interrupt-driven
0	1	0	0	TDIS bit polled	DMA
0	1	1	0	Interrupt-driven	DMA
1	0	0	0	DMA	RDIS bit polled
1	0	0	1	DMA	Interrupt-driven
1	1	0	0	DMA	DMA

Note: Any other combinations are prohibited.

13.4.11 Flow Controller

Transmission of serial data can be solely software-initiated (SIFLCR.TSDE) or both software-initiated and hardware-triggered (CTS* signal). Selection of which is programmed in the SIFLCR.TES bit.

When the SIFLCR.TSDE bit is set, the SIO halts transmission after completing any current character. Transmission remains disabled until the SIFLCR.TSDE bit is cleared again.

The state of the CTS* pin can be monitored so that an interrupt is generated upon CTS* change.

The SIFLCR.RSDE and SIFLCR.RTSTL bits allow for the initiation of data reception. The SIFLCR.RSDE bit, when cleared, forces the RTS* pin to its active state. The SIFLCR.RTSTL bit specifies the Receive FIFO trigger level at which RTS* pin is asserted. The SIFLCR.RSDE bit can be used alone, or both the SIFLCR. RSDE and SIFLCR.RTSTL bits can be used in combination.

The RTS* pin is driven high when the Receive FIFO has reached the trigger level programmed in the RTSTL field of the Flow Control register (SIFLCR).

During data reception, a high on the RTS* pin indicates a request to temporarily stop transmission to the transmitter. Once the receiver is ready again, asserting the RTS* pin low informs the transmitter that it can restart transmission.

The handshaking can be set up, by programming the transmitter for hardware control (SIFLCR.TES = 1) and the receiver to 1-byte trigger level (SIFLCR.RTSTL = 0001).

13.4.12 Parity Controller

During transmission, the parity controller automatically generates parity for the data in the transmit shift register. The parity bit is stored in bit 7 (MSB) of the transmit shift register when the data length is 7 bits and in the TWUB bit of the Line Control register (SILCRn) when the data length is 8 bits.

During reception, a parity check is performed when data has been transferred from the receive shift register to the read buffer. The parity bit for the character is compared to bit 7 (MSB) of the read buffer when the data length is 7 bits and with the RWUB bit of the Line Control register (SILCRn) when the data length is 8 bits. If they do not match, a parity error is reported.

13.4.13 Error Flags

- Overrun error

  An overrun error is reported if a new character is received into the read buffer when the 16-byte Receive FIFO is already 100% full. The overrun status bit of the 16th byte in the Receive FIFO is set.

- Parity error

  A parity error is reported when received data’s parity does not match the parity bit.

- Framing error

  A framing error is reported when a 0 is detected where a stop bit was expected. (The middle 3 of the 16 samples taken on the 7th to 9th SIOCLK cycles are used to determine the bit value.)

  (The same sampling timing is used, regardless of whether one stop bit or two stop bits are used.)

13.4.14 Break Indication

A break is reported when a framing error occurs on the received data, with all bits in the frame being 0s. At this time, the SISCISR.RBRKD and SISCISR.UBRKD bits are set. The SISCISR.UBRKD bit remains set until it is cleared by software, whereas the SISCISR.RBRKD bit is automatically cleared when a non-break frame is received.

Two characters loaded into the Receive FIFO on a break indication are always 0x00.

13.4.15 Receive Timeout

A receive timeout is reported when there is at least one byte in the Receive FIFO and the receive shift register has not been accessed within 2 character times of the last byte. The SIDISR.TOUT bit is set to indicate that a receive timeout has occurred.

The timer for the receive timeout is reset when a new character is received and when all characters in the Receive FIFO have been read; the timer does not restart until the next character is received.

13.4.16 Receive Data Transfer and the Handling of Receive FIFO Status Bits

The Receive FIFO stores the following status bits, along with the received data:

- Break detection (UBRK)
- Receive FIFO data available status (UVALID)
- Framing error (UFER)
- Parity error (UPER)
- Overrun error (UOER)

The SIDISR register contains a copy of these status bits. The contents of this register is updated each time a character is read from the Receive FIFO (SIRFIFO).

The interrupt handler for the receive data interrupt (SIRXIREQ*) can examine the receive status before reading the receive data in order to obtain a one-to-one correspondence between receive errors and received characters. This enables the interrupt handler to process those characters in the exact order in which they are received. If such ordering is not necessary, special receive interrupt (SISPIREQ*) and status-change interrupt (SISTIREQ*) requests can be used, which occur on detection of a receive error before the character present in the SIRFIFO is read.

In DMA transfer mode, the Receive FIFO transfers only error-free characters. If an error (framing error, parity error or overrun error) or the receive timeout (TOUT) occurs, the Receive Data Transfer Request (SIRXIREQ*) signal is asserted to report a receive error.

If a receive error occurs in DMA transfer mode, the Receive FIFO must be cleared.

Note: The UVALID flag is always set for the 16th character in the Receive FIFO even though when it has been picked up by the CPU or DMA Controller, no more character may be present in the Receive FIFO. Examine the RFDN field of the SIDISR register to check the number of characters remaining in the Receive FIFO.

13.4.17 Multidrop System

Multidrop mode is selected when the SILCR.UMODE field is set to 10 or 11. In this mode of operation, the master station's controller transmits an address (ID) character followed by a block of data characters targeted for one or more of the slave stations. When a slave station's address matches the received address, it enables the receiver if it wants to receive the subsequent data from the master station. The UART frame is extended one bit to distinguish an address character from standard data character. The character is interpreted as an address character if the WUB bit is set to 1, or interpreted as a data character if it is set to 0. Software must be used to perform the address comparison.

13.4.17.1 Protocol

(1) Both the master and slave controllers are configured to operate in multidrop mode by setting the UMODE field of the Line Control register (SILCR.) to 10 or 11.

(2) If the RWUB bit in the SILCR register is set, slave controllers continuously monitor the data stream from the master controller for the address character.

(3) If the Transmit Wakeup (TWUB) bit of the SILCR register is set, the master controller transmits a slave's station address (8-bit or 7-bit), with the WUB bit set to 1.

(4) The slave controller generates an interrupt to the host when the RWUB bit of the SILCR register is set and the WUB bit of the received character is 1 (indicating an address frame). The host compares the received address to its own address; and if they match, the host clears the RWUB bit to 0.

(5) The master controller transmits a block of characters to the designated slave controller. This time, the TWUB bit of the SILCR register is set so that the master controller transmits data frames with the WUB bit cleared.

(6) The slave controller with the RWUB bit cleared receives the data. Other slaves whose RWUB bit remains set do not generate an interrupt because the WUB bit of the received data frame is 0. Therefore, the slave stations that are not addressed ignore the received data.

(7) Slave controllers can transmit data only to the master controller.

Figure 13.4.3 shows an example configuration of a multidrop system.

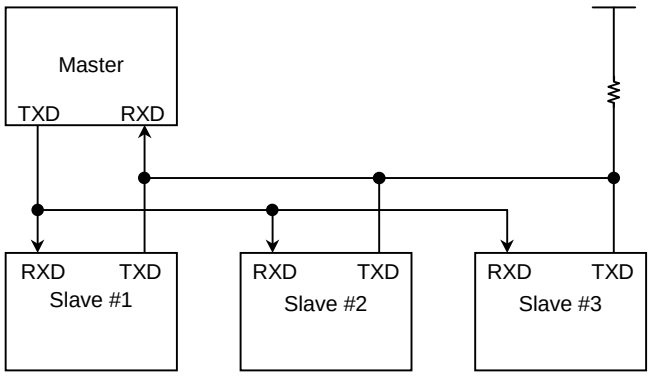


Figure 13.4.3 Example Configuration of a Multidrop System

The TXD outputs of the slave controllers must be open-drain. When the UODE bit of the Line Control register (SILCR.) is set, the TXD output is configured as open-drain. When the UODE bit is cleared, the TXD output is configured as a normal CMOS output.

13.4.18 Software Reset

Writing a 1 to the SWRST bit (bit 15) of the FIFO Control register (SIFCR) performs a software reset of the SIO. This bit clears itself when the SIO resets. Software reset is required in the following cases because even if the FIFO is cleared, data remains in the temporary buffer:

1. When transmission is discontinued halfway after having placed transmit data in the FIFO and started transmission.

2. When an overrun error occurs during reception

13.4.19 DMA Transfer Mode

The SIO Transmit and Receive FIFOs support DMA. The TX3927's DMA interface provides up to four DMA channels to support the two SIO channels, as listed below. For each of the DMA channels, bits 7 to 4 (INTDMA[3:0]) of the Pin Configuration (PCFG) register determine whether the DMA request source is an SIO channel or an external DMA request signal (DMAREQ[3:0]).

SIO channel 0 reception	-	DMA channel 0
SIO channel 1 reception	-	DMA channel 1
SIO channel 0 transmission	-	DMA channel 2
SIO channel 1 transmission	-	DMA channel 3

Both the DMA request output and the DMA acknowledge input of the SIO are active-low. Set the DMA Controller (DMAC) request and acknowledge polarities to active-low (CCRn.ACKPOL=0b, CCRn.REQPL = 0b).

The SIO FIFO Control register (SIFCR) determines how many bytes are transferred with a single DMA transfer request. Set the DMAC to one-byte transfer size (CCRn.XFZ = 000b) and dual-address transfer mode (CCRn.ONEAD = 0b). Therefore, the DMAC Source or Destination Address register must specify the byte address of the SIO register.

In DMA transfer mode, the SIO receive timeout error must be not used to determine whether a DMA transaction has completed (on the assumption that when no more data has arrived for a given time, the DMA transaction has completed). Regardless of whether DMA transfer mode is used or not, a receive timeout occurs when there is at least one byte in the Receive FIFO and the receive shift register has not been accessed within 2 character time of the last byte. Therefore, receive timeout error is not reported when the DMAC has read all the received characters from the Receive FIFO.

13.5 Timing Diagrams

13.5.1 Receiver Operation (7- and 8-bit Data Lengths)

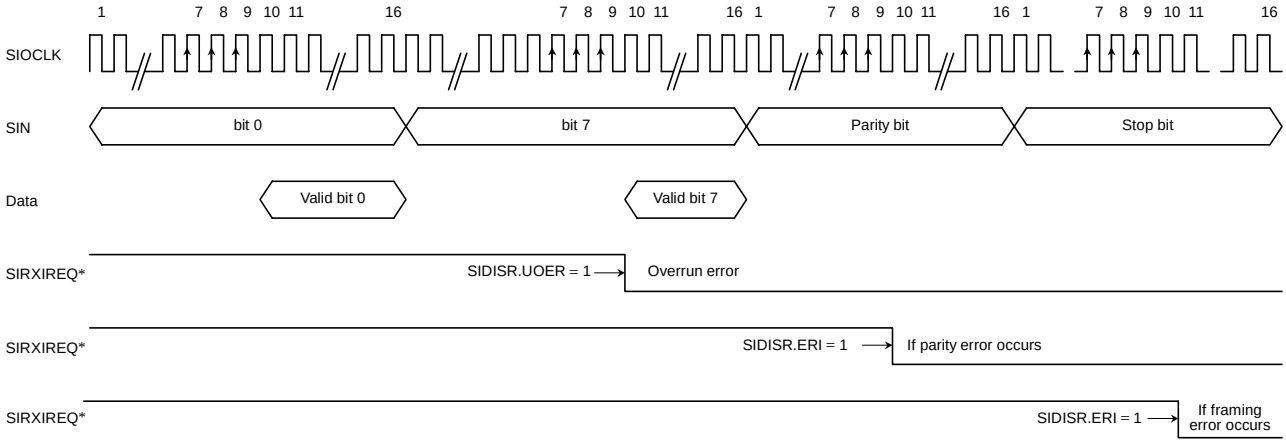
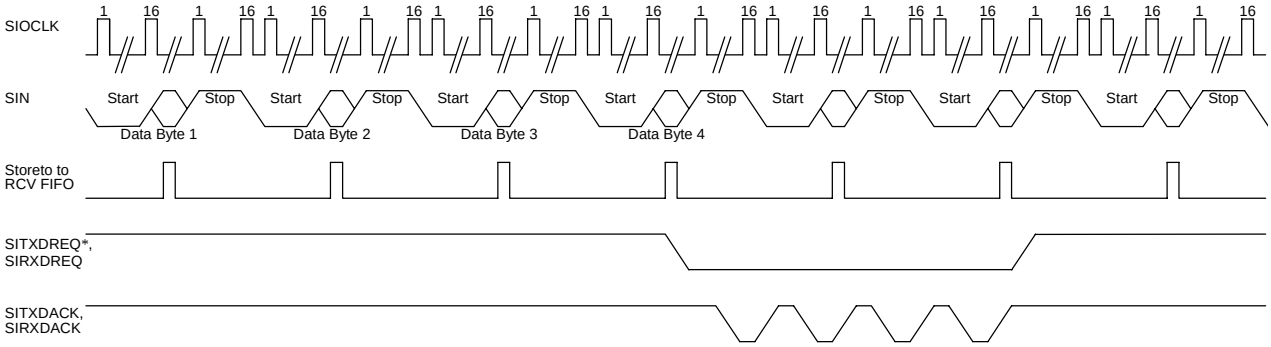


Figure 13.5.1 Receiver Timing

13.5.2 SITXDREQ*/SITXDACK and SIRXDREQ/SIRXDACK Timing for DMA Interface (DMA Level 4)



SDMAREQ is deasserted after recognizing the third assertion of DMAACK.

SDMAACK is sampled at the IMCLK frequency.

Figure 13.5.2 DMA Request and Acknowledge Signals (DMA Level 4)

13.5.3 SITXDREQ*/SITXDACK and SIRXDREQ/SIRXDACK Timing for DMA Interface (DMA Level 8)

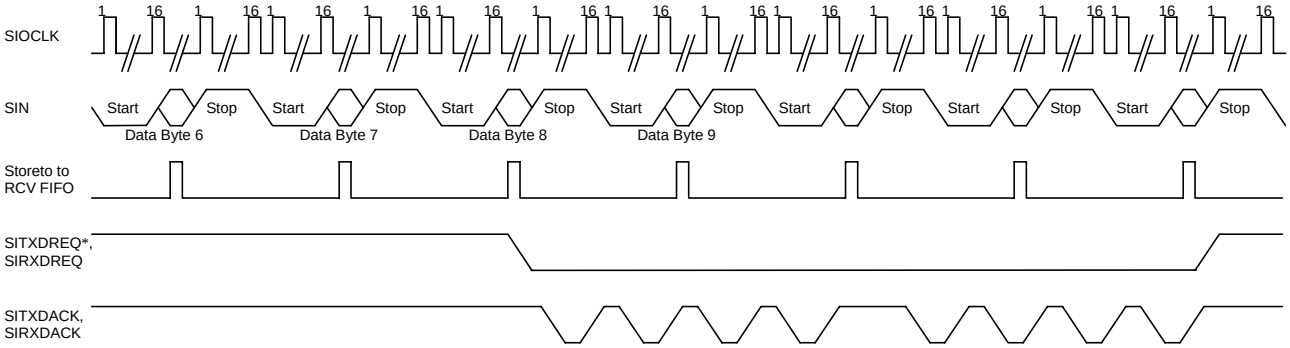


Figure 13.5.3 DMA Request and Acknowledge Signals (DMA Level 8)

13.5.4 Receiver Operation (7- and 8-bit Lengths in Multidrop System Mode, RWUB = 1, Waiting for an ID Frame)

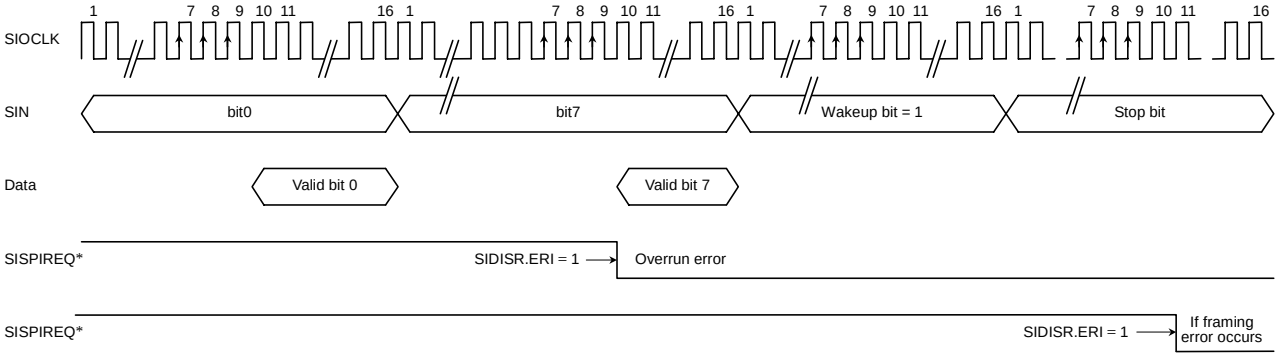


Figure 13.5.4 Receiver Timing

13.5.5 Receiver Operation (7- and 8-bit Lengths in Multidrop System Mode, RWUB = 0, Waiting for a Data Frame)

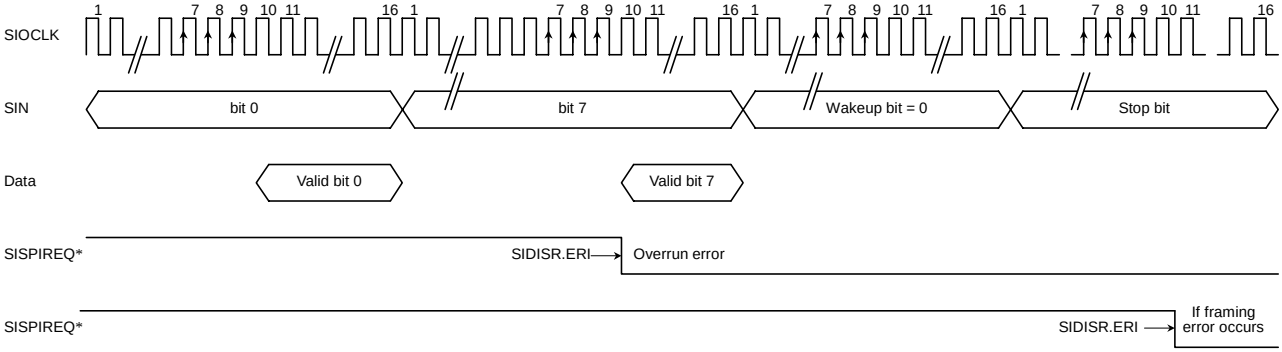


Figure 13.5.5 Receiver Timing

13.5.6 Receiver Operation (7- and 8-bit Lengths in Multidrop System Mode, RWEB = 1, Skipping Data Read)

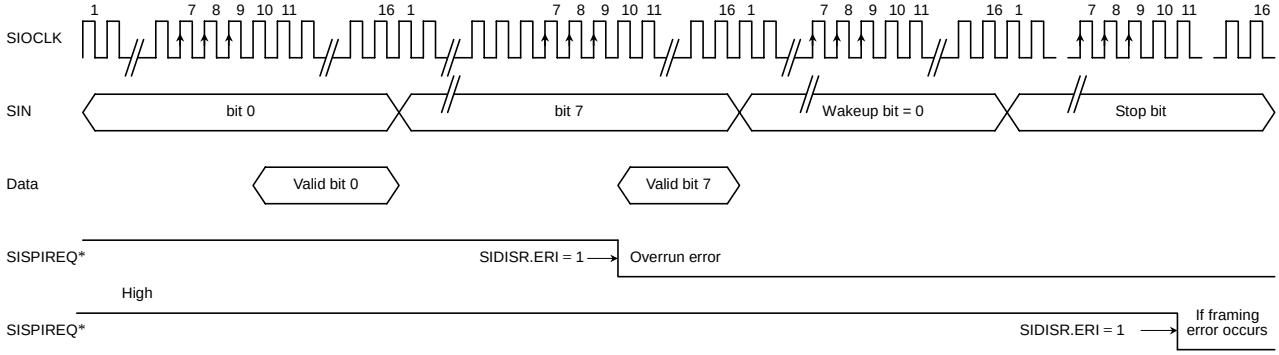


Figure 13.5.6 Receiver Timing

13.5.7 Transmitter Operation

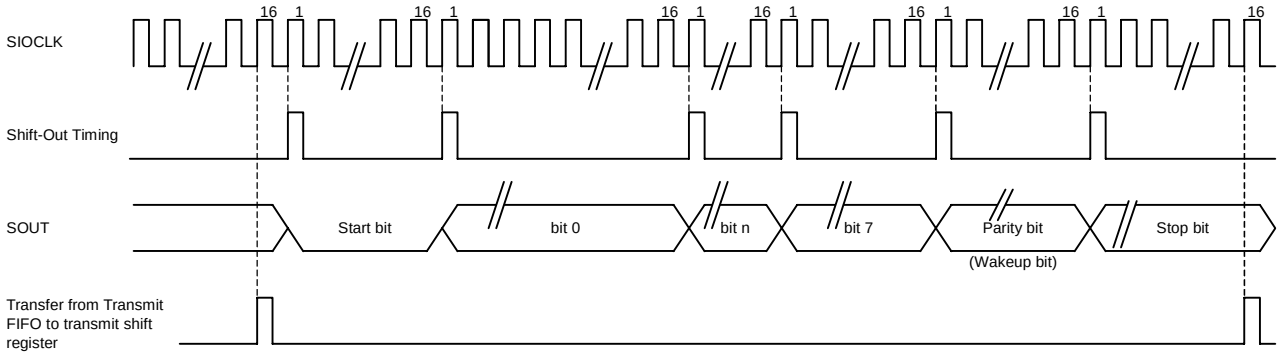


Figure 13.5.7 Transmitter Timing

13.5.8 Timing for Stopping Transmission by CTS*

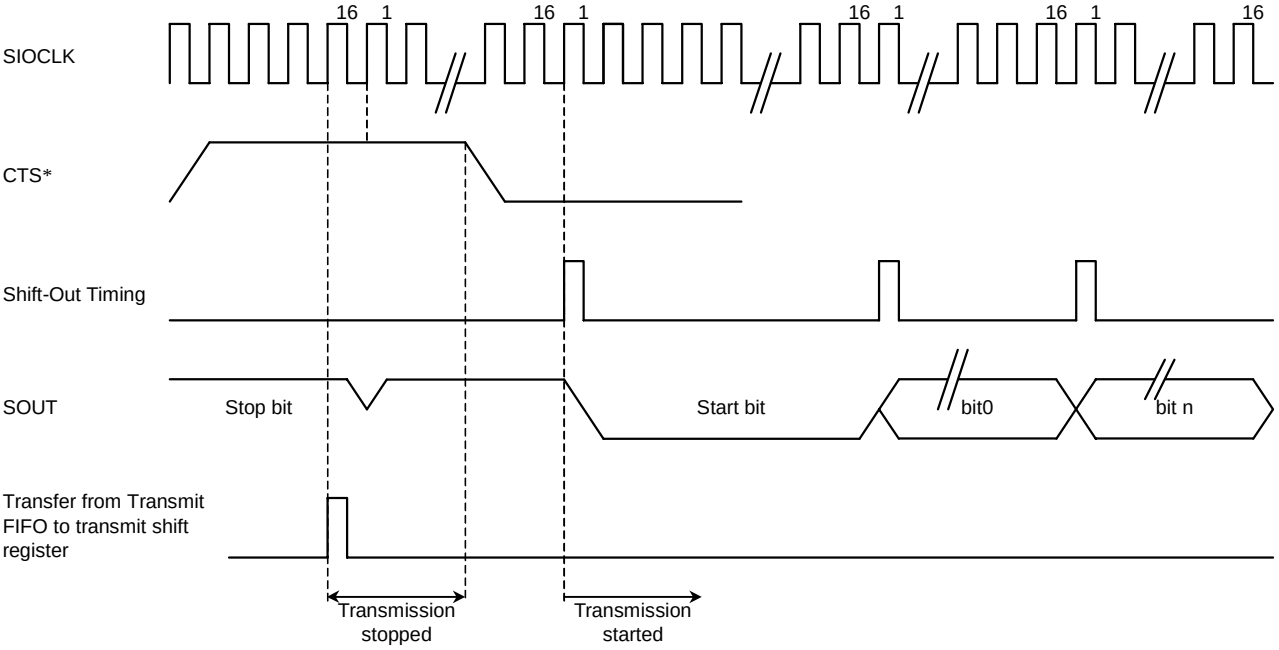


Figure 13.5.8 CTS* Timing

When CTS* goes high during transmission, transmission is stopped after completing the transfer of the current character. However, the next character has been transferred from the FIFO to the transfer shift register.

When CTS* goes low, transmission is resumed at the first shift-out start timing.

14. Timer/Counter

14.1 Features

The TX3927 includes three 24-bit timer/counter channels.

All three channels (Timer 0-2) can be used as an interval timer, operating as a 24-bit up counter. Timers 0 and 1 can also be used as pulse generators, and Timer 2 also functions as a watchdog timer.

(1) Interval timer mode
- Can generate interrupt requests at a regular interval time.

(2) Pulse generator mode
- Outputs the state of the timer flip-flop onto the timer output pin.

(3) Watchdog timer mode
- Protects against system failures.

The counter clock sources can be an external clock pin (TCLK) or the internal clock divider output. The clock divider is programmed to divide the internal clock (IMCLK) by 2, 4, 8, 16, 32, 64, 128 or 256. The IMCLK frequency is half of the G-Bus clock speed. For details, refer to Chapter 6, "Clocks."

When selected, TCLK is used by all three timers for internal timing reference.

TCLK is multiplexed with the DMAREQ[2] and PIO[13] signals on pin 127. When this pin is configured for DMAREQ[2] or PIO[13], an external clock cannot be used as the timer’s clock source.

When TCLK is used, all three timers can be used as 24-bit event counters. The program determines which edge polarity (rising or falling) is detected. The TCLK frequency must not exceed IMCLK/2.

14.2 Block Diagram

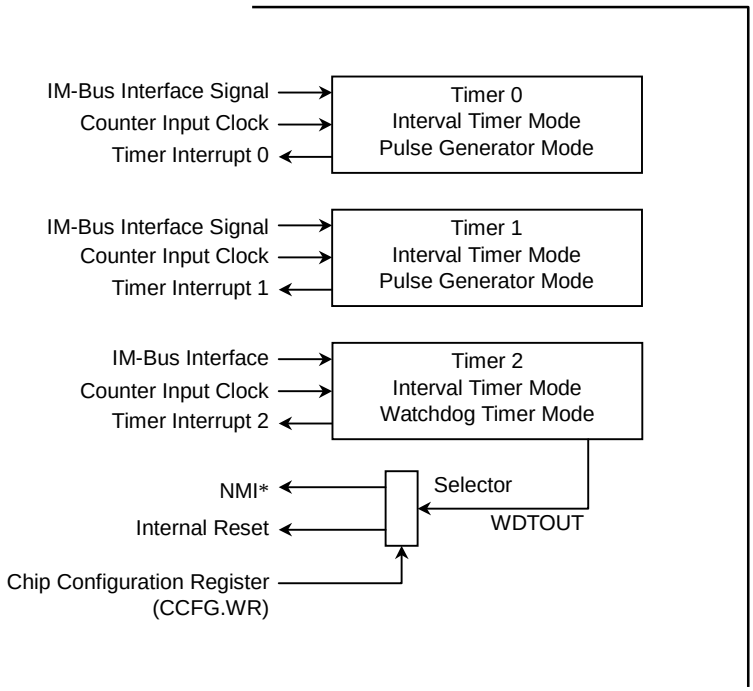


Figure 14.2.1 Timer Module Interface within the TX3927

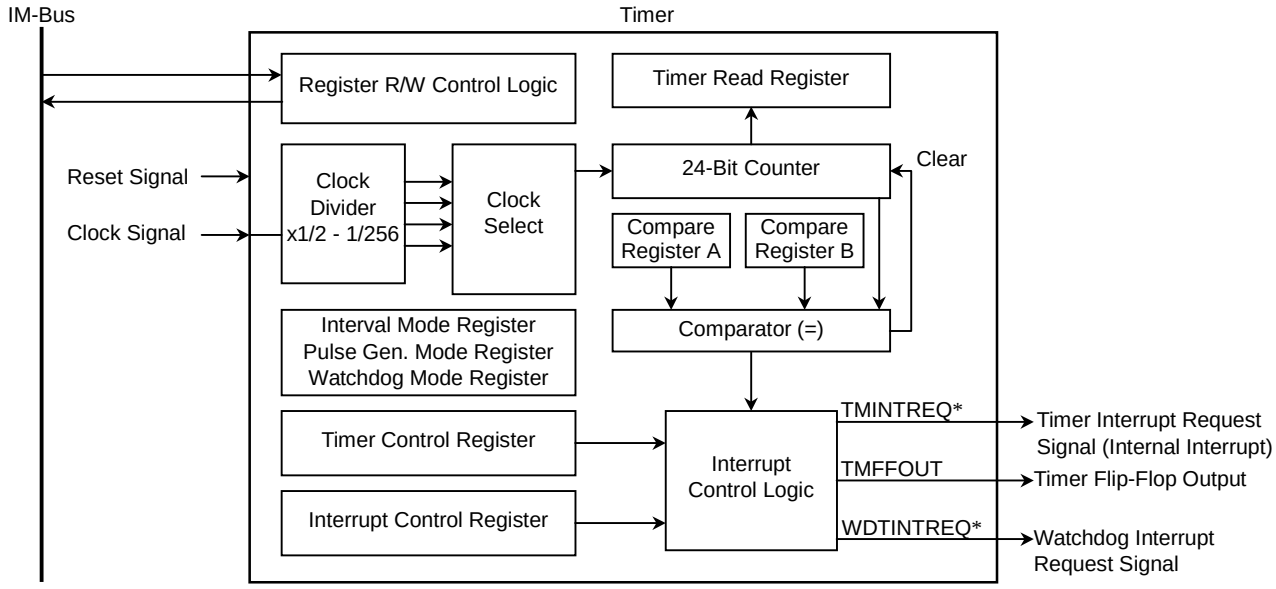


Figure 14.2.2 Internal Block Diagram of a Timer

14.3 Registers

14.3.1 Register Map

All the registers in the Timer/Counter module should be accessed as a word quantity. For the bits other than those defined in this section, the values shown in the figures must be written.

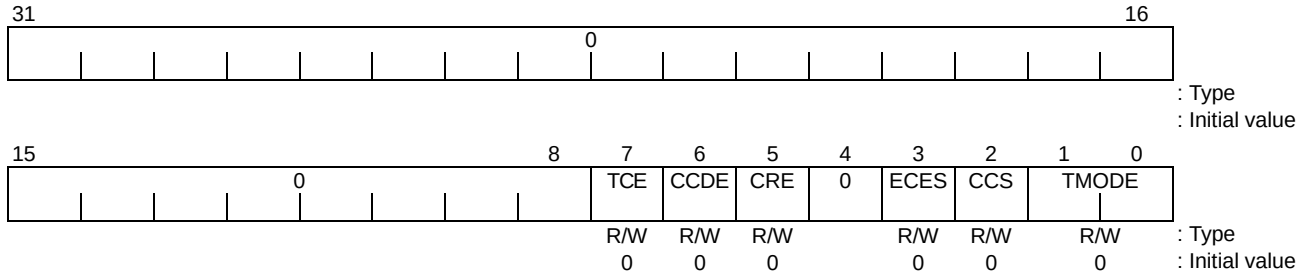
Table 14.3.1 Timer/Counter Registers

Address	Register Mnemonic	Register Name
Timer 2 (TMR2)		
0xFFFE_F2F0	TMTRR2	Timer Read Register 2
0xFFFE_F240	TMWTMR2	Watchdog Timer Mode Register 2
0xFFFE_F230	TMPGMR2	(Reserved)
0xFFFE_F220	TMCCDR2	Clock Divider Register 2
0xFFFE_F210	TMITMR2	Interval Timer Mode Register 2
0xFFFE_F20C	TMCPRB2	(Reserved)
0xFFFE_F208	TMCPRA2	Compare Register A 2
0xFFFE_F204	TMTISR2	Timer Interrupt Status Register 2
0xFFFE_F200	TMTCR2	Timer Control Register 2
Timer 1 (TMR1)		
0xFFFE_F1F0	TMTRR1	Timer Read Register 1
0xFFFE_F140	TMWTMR1	(Reserved)
0xFFFE_F130	TMPGMR1	Pulse Generator Mode Register 1
0xFFFE_F120	TMCCDR1	Clock Divider Register 1
0xFFFE_F110	TMITMR1	Interval Timer Mode Register 1
0xFFFE_F10C	TMCPRB1	Compare Register B 1
0xFFFE_F108	TMCPRA1	Compare Register A 1
0xFFFE_F104	TMTISR1	Timer Interrupt Status Register 1
0xFFFE_F100	TMTCR1	Timer Control Register 1
Timer0 (TMR0)		
0xFFFE_F0F0	TMTRR0	Timer Read Register 0
0xFFFE_F040	TMWTMR0	(Reserved)
0xFFFE_F030	TMPGMR0	Pulse Generator Mode Register 0
0xFFFE_F020	TMCCDR0	Clock Divider Register 0
0xFFFE_F010	TMITMR0	Interval Timer Mode Register 0
0xFFFE_F00C	TMCPRB0	Compare Register B 0
0xFFFE_F008	TMCPRA0	Compare Register A 0
0xFFFE_F004	TMTISR0	Timer Interrupt Status Register 0
0xFFFE_F000	TMTCR0	Timer Control Register 0

Note: All registers can be accessed as words.

14.3.2 Timer Control Registers (TMTCRn)

0xFFFE_F000 (Ch. 0)
0xFFFE_F100 (Ch. 1)
0xFFFE_F200 (Ch. 2)

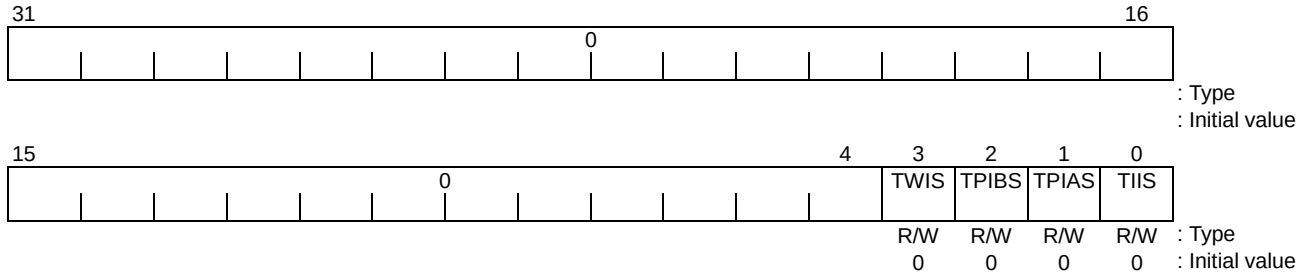


Bits	Mnemonic	Field Name	Description
7	TCE	Timer Count Enable	Timer Count Enable (initial value: 0) Enables or disables counting. The CRE bit determines the action taken when the counter stops. 0: Disables counting (and resets the counter if CRE = 1). 1: Enables counting.
6	CCDE	Counter Clock Divide Enable	Counter Clock Divide Enable (initial value: 0) Enables or disables the clock divider for the internal clock (IMCLK). Clearing this bit disables the clock divider. 0: Disables the clock divider. 1: Enables the clock divider.
5	CRE	Counter Reset Enable	Counter Reset Enable (initial value: 0) Determines the action taken when the counter is stopped by clearing the TCE bit. If the CRE bit is set, the counter is inhibited from counting and is reset. If the CRE bit is cleared, the counter is inhibited from counting, but is not reset.
3	ECES	External Clock Edge Select	External Clock Edge Select (initial value: 0) Selects the active edge of the clock when the external clock is used. 0: Falling edge 1: Rising edge
2	CCS	Counter Clock Select	Counter Clock Select (initial value: 0) Selects the timer's clock source. 0: Internal system clock (IMCLK) 1: External input clock (TCLK)
1:0	TMODE	Timer Mode	Timer Mode (initial value: 00) Selects one of the three modes of operation for the timer. 11: Don't use. 10: Watchdog timer mode (only for channel 2) 01: Pulse generator mode (only for channels 0 and 1) 00: Interval timer mode

Figure 14.3.1 Timer Control Registers

14.3.3 Timer Interrupt Status Registers (TMTISRn)

0xFFFE_F004 (Ch. 0)
0xFFFE_F104 (Ch. 1)
0xFFFE_F204 (Ch. 2)



Bits	Mnemonic	Field Name	Description
3	TWIS	Timer Watchdog Interrupt Status	Timer Watchdog Interrupt Status (initial value: 0) This bit is set when the channel 2 counter value matches the value programmed in Compare Register A (TMCPRA). At this time, if the TWIE bit of the Watchdog Timer Mode register is set, the TMWDTREQ* signal is asserted. Clearing this bit negates the TMWDTREQ* signal. Writing a 1 to this bit has no effect. On reads: 0: No interrupt condition has occurred. 1: An interrupt condition has occurred; i.e., the counter value has reached the value in Compare Register A. On writes: 0: Negates the interrupt request signal. 1: Ignored
2	TPIBS	Timer Pulse Generator Interrupt by TMCPRB Status	Timer Pulse Generator Interrupt by TMCPRB Status (initial value: 0) This bit is set when the counter value matches the value programmed in Compare Register B (TMCPRB). At this time, if the TPIBE bit of the Pulse Generator Mode register is set, the TMINTREQ* signal is asserted. Clearing this bit negates the TMINTREQ* signal. Writing a 1 to this bit has no effect. On reads: 0: No interrupt condition has occurred. 1: An interrupt condition has occurred; i.e., the counter value has reached the value in Compare Register B. On writes: 0: Negates the interrupt request signal. 1: Ignored
1	TPIAS	Timer Pulse Generator Interrupt by TMCPRA Status	Timer Pulse Generator Interrupt by TMCPRA Status (initial value: 0) This bit is set when the counter value matches the value programmed in Compare Register A (TMCPRA). At this time, if the TPIAE bit of the Pulse Generator Mode register is set, the TMINTREQ* signal is asserted. Clearing this bit negates the TMINTREQ* signal. Writing a 1 to this bit has no effect. On reads: 0: No interrupt condition has occurred. 1: An interrupt condition has occurred; i.e., the counter value has reached the value in Compare Register A. On writes: 0: Negates the interrupt request signal. 1: Ignored

Figure 14.3.2 Timer Interrupt Status Registers (1/2)

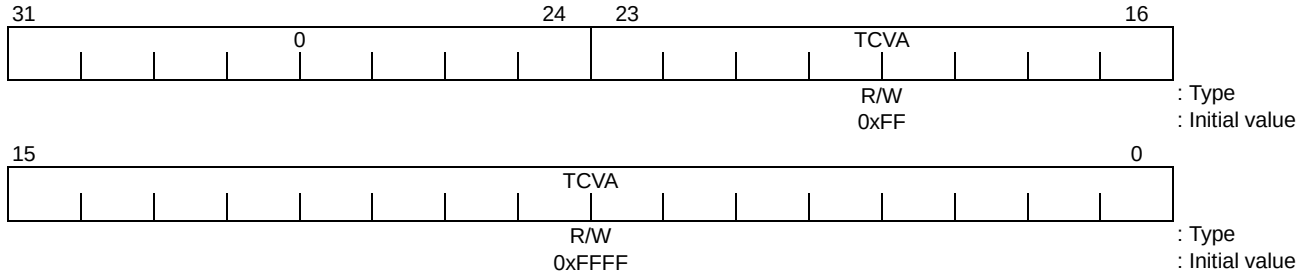
Bits	Mnemonic	Field Name	Description
0	TIIS	Timer Interval Interrupt Status	Timer Interval Interrupt Status (initial value: 0) This bit is set when the counter value matches the value programmed in Compare Register A (TMCPRA). At this time, if the TIIE bit of the Interval Timer Mode register is set, the TMINTREQ* signal is asserted. Clearing this bit negates the TMINTREQ* signal. Writing a 1 has no effect. On reads: 0: No interrupt condition has occurred. 1: An interrupt condition has occurred; i.e., the counter value has reached the value in Compare Register A. On writes: 0: Negate the interrupt. 1: Ignored

Note: Bit 3 is valid only for channel 2. Bits 2 and 1 are valid only for channels 0 and 1.

14.3.3 Timer Interrupt Status Registers (2/2)

14.3.4 Compare Registers A (TMCPRAn)

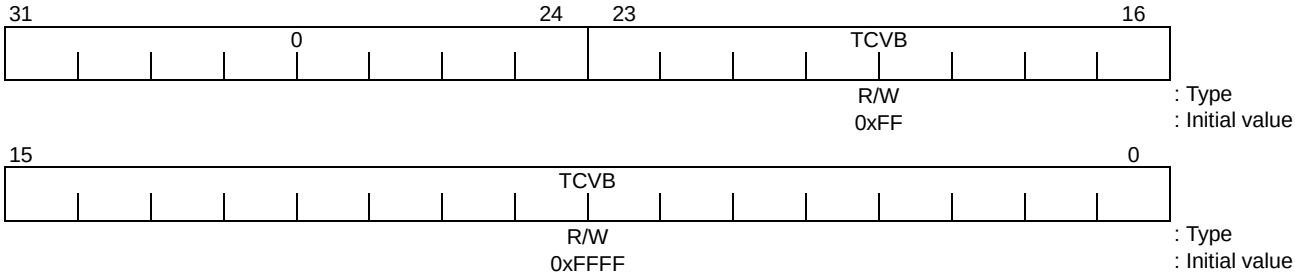
0xFFFE_F008 (Ch. 0)
0xFFFE_F108 (Ch. 1)
0xFFFE_F208 (Ch. 2)



Bits	Mnemonic	Field Name	Description
23:0	TCVA	Timer Compare Value A	Timer Compare Value A (initial value: 0xffffff) Specifies the compare value (24-bit binary value) for the timer. This register can be used in any mode.

Figure 14.3.3 Compare Registers A

14.3.5 Compare Registers B (TMCPRBn) 0xFFFE_F00C (Ch. 0)
0xFFFE_F10C (Ch. 1)

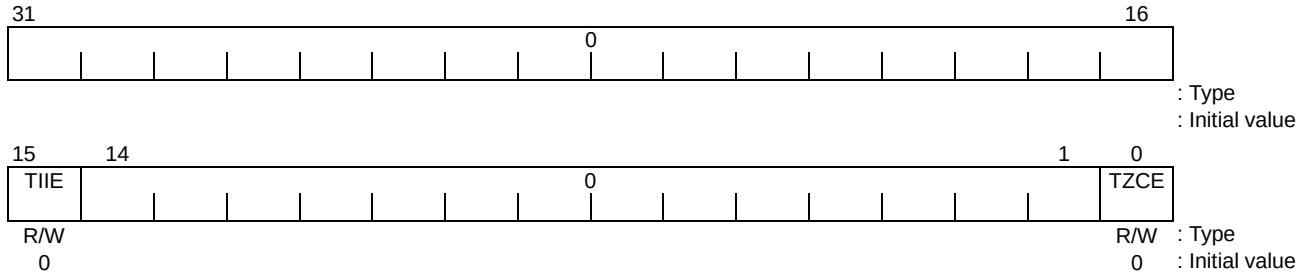


Bits	Mnemonic	Field Name	Description
23:0	TCVB	Timer Compare Value B	Timer Compare Value B (initial value: 0xffffff) Compare Register B is used in pulse generator mode. Specify a 24-bit binary value greater than the value in Compare Register A.

Figure 14.3.4 Compare Registers B

14.3.6 Interval Timer Mode Registers (TMITMRn)

0xFFFE_F010 (Ch. 0)
0xFFFE_F110 (Ch. 1)
0xFFFE_F210 (Ch. 2)

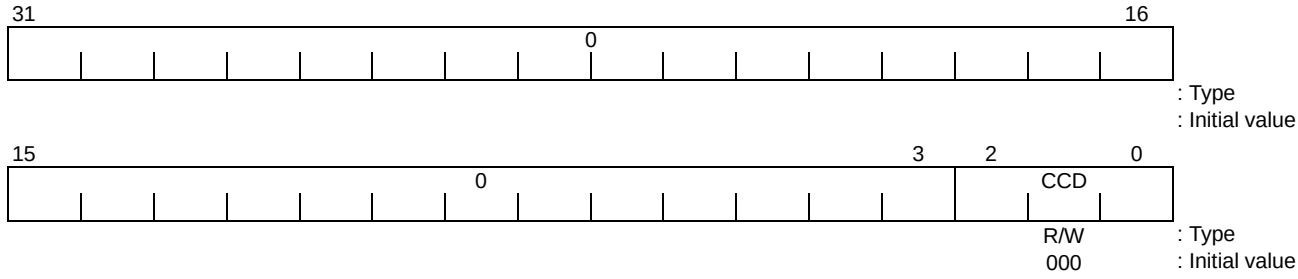


Bits	Mnemonic	Field Name	Description
15	TIIE	Timer Interval Interrupt Enable	Timer Interval Interrupt Enable (initial value: 0) Enables or disables interval timer mode interrupts. 0: Disables interrupts. (mask) 1: Enables interrupts.
0	TZCE	Interval Timer Zero Clear Enable	Interval Timer Zero Clear Enable (initial value: 0) Controls whether to reset the counter to 0 whenever the count value reaches the value in Compare Register A. If this bit is cleared, the counter halts when the count reaches the compare value. Even if the counter is in halt state at the compare value (with this bit cleared), another interrupt will not occur on return from the interrupt. 0: Do not clear (mask) 1: Clear

Figure 14.3.5 Interval Timer Mode Registers

14.3.7 Clock Divider Registers (TMCCDRn)

0xFFFE_F020 (Ch. 0)
0xFFFE_F120 (Ch. 1)
0xFFFE_F220 (Ch. 2)

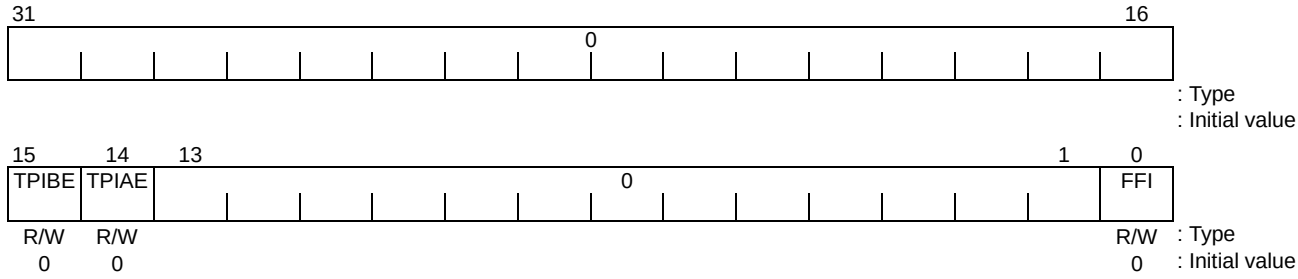


Bits	Mnemonic	Field Name	Description
2:0	CCD	Counter Clock Divide	Counter Clock Divide (initial value: 000) These bits determine clock divisor when the internal clock (IMCLK) is used as the counter clock source. The binary value n divides the clock by 2^{n+1}. 000: Divide by 2^1 001: Divide by 2^2 010: Divide by 2^3 011: Divide by 2^4 100: Divide by 2^5 101: Divide by 2^6 110: Divide by 2^7 111: Divide by 2^8

Figure 14.3.6 Clock Divider Registers

14.3.8 Pulse Generator Mode Registers (TMPGMRn)

0xFFFE_F030 (Ch. 0)
0xFFFE_F130 (Ch. 1)

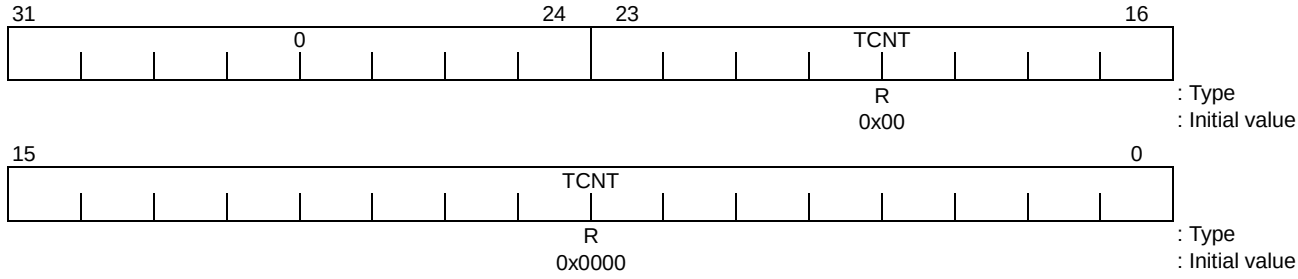


Bits	Mnemonic	Field Name	Description
15	TPIBE	TMCPRB Interrupt Enable	Timer Pulse Generator Interrupt by TMCPRB Enable (initial value: 0) Enables or disables interrupts generated when the counter value matches the value programmed in Compare Register B in pulse generator mode. 0: Disables interrupts. 1: Enables interrupts.
14	TPIAE	TMCPRA Interrupt Enable	Timer Pulse Generator Interrupt by TMCPRA Enable (initial value: 0) Enables or disables interrupts generated when the counter value matches the value programmed in Compare Register A in pulse generator mode. 0: Disables interrupts. 1: Enables interrupts.
0	FFI	Timer Flip-Flop Initial Value	Timer Flip-Flop Initial (initial value: 0) Specifies the value to which the timer flip-flop is initially set. 0: Low 1: High

Figure 14.3.7 Pulse Generator Mode Registers

14.3.9 Timer Read Registers (TMTRRn)

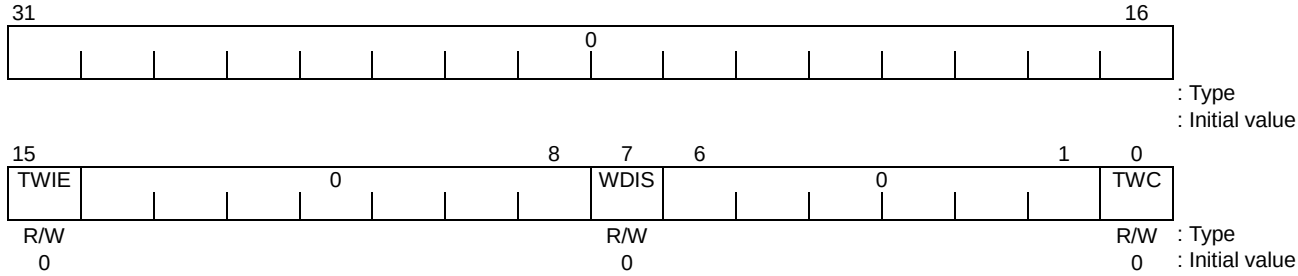
0xFFFE_F0F0 (Ch. 0)
0xFFFE_F1F0 (Ch. 1)
0xFFFE_F2F0 (Ch. 2)



Bits	Mnemonic	Field Name	Description
23:0	TCNT	Timer Count	Timer Count (initial value: 0x000000) This register follows the contents of the 24-bit counter. A read cycle to this register causes the current value of the counter to be read. This register is read-only. Don't write to this register; any write to this register will result in improper operation.

Figure 14.3.8 Timer Read Registers

14.3.10 Watchdog Timer Mode Register (TMWTMR2) 0xFFFE_F240 (Ch. 2)



Bits	Mnemonic	Field Name	Description
15	TWIE	Timer Watchdog Interrupt Enable	Timer Watchdog Interrupt Enable (initial value: 0) Enables or disables watchdog timer interrupts. 0: Disables interrupts. (mask) 1: Enables interrupts.
7	WDIS	Watchdog Timer Disable	Watchdog Timer Disable (initial value: 0) Watchdog timer mode can be disabled by setting this bit and clearing the TCE bit of the Timer Control register. This bit is self-clearing. Writing a 0 to this bit has no effect.
0	TWC	Timer Watchdog Clear	Timer Watchdog Clear (initial value: 0) 1: Setting this bit resets the counter. This bit is self-clearing. Writing a 0 to this bit has no effect.

Figure 14.3.9 Watchdog Timer Mode Register

14.4 Operation

All timers (2, 1, and 0) are identical in operation, so only single timer (Timer 0) will be described. Remember that only Timer 2 supports watchdog timer mode.

Any differences among the three timer channels will be described.

14.4.1 Interval Timer Mode

The timer is configured for interval timer mode by setting the TMODE field of the Timer Control register (TMTCR) to 00.

The CCS bit of the TMTCR selects either the internal clock (IMCLK) or an external clock input.

When the internal clock (IMCLK) is selected, the clock divider divides IMCLK by the value programmed into the CCD field of the Clock Divider register (TMCCDR). The output of the clock divider is used as input to the 24-bit counter. Clock frequency division is enabled by setting the CCDE bit of the TMTCR. The division factor can be from 2^1 to 2^8.

When the external pin is selected as the clock source, the ECES bit of the TMTCR determines the active clock edge for the counter.

Setting the TCE bit of the TMTCR enables counting.

When the count value reaches the Compare Register A (TMCPRA) value, the TIIS flag in the Timer Interrupt Status register (TMTISR) is set. An interrupt request is generated if the TIIE bit of the Interval Timer Mode register (TMITMR) is set. The timer interrupt request generation can be disabled by clearing the TIIE bit. If the internal timer interrupt request signal has been asserted, it is negated when the TIIS bit of the TMTISR is cleared. A write of 1 has no effect on this bit.

Regardless of the setting of the TIIE bit, the TIIS bit is set when the count value matches the Compare Register A value. The TIIS bit, when set, indicates a pending interrupt; thus a timer interrupt request is issued if the TIIE bit is then set. To prevent this, the TIIS bit must be cleared before setting the TIIE bit.

If the TZCE bit of the TMITMR is set, the timer count is reset to 0 whenever it reaches the compare value programmed in the TMCPRA, and the timer immediately begins counting again. Consequently, the interrupt request is generated continuously at a regular interval time. If the TZCE bit is cleared, the timer halts when the count value has reached the TMCPRA value. Table 14.4.1 summarizes interrupt request generation.

Table 14.4.1 Interrupt Request Generation, Depending on the TIIE and TZCE Settings

TIIE	TZCE	Interrupt operation when the counter reaches the specified value
0	*	No interrupt occurs.
1	0	An interrupt request is issued. No further interrupt will occur if the TZCE bit is 0 upon return from the interrupt (i.e., when the TIIS bit is cleared). Otherwise, another interrupt is requested, as is the case with TIIE = 1 and TZCE = 1.
1	1	An interrupt request is issued.

The Timer Read register (TMTRR) follows the contents of the 24-bit counter. A read of this register causes the current value of the counter to be read.

Figure 14.4.1 depicts interval timer mode operation and interrupt request generation.

Figure 14.4.2 depicts interval timer mode operation using the external clock.

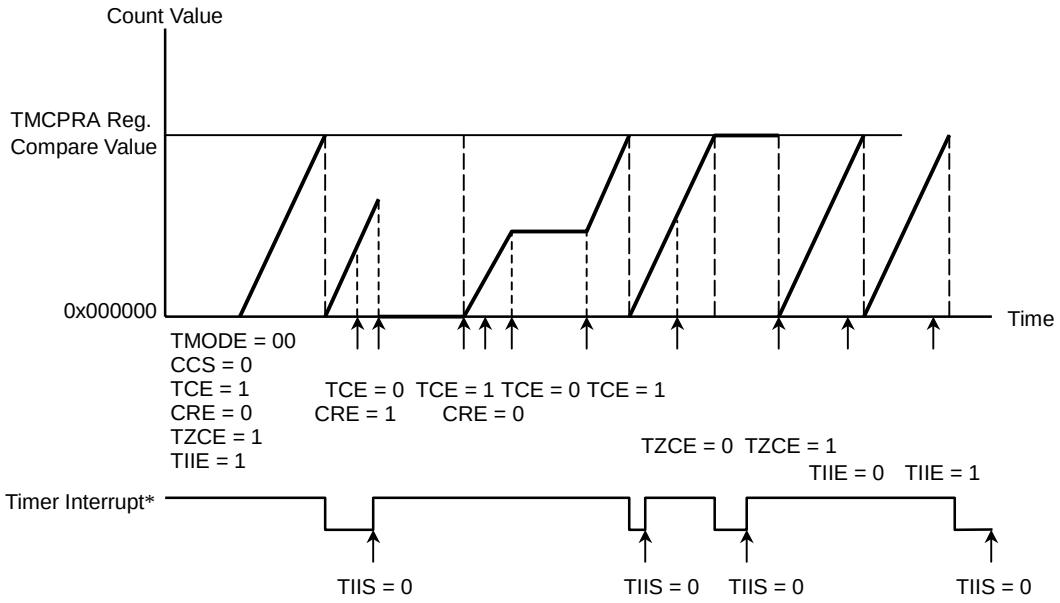


Figure 14.4.1 Example Interval Timer Mode Operation (with Internal Clock)

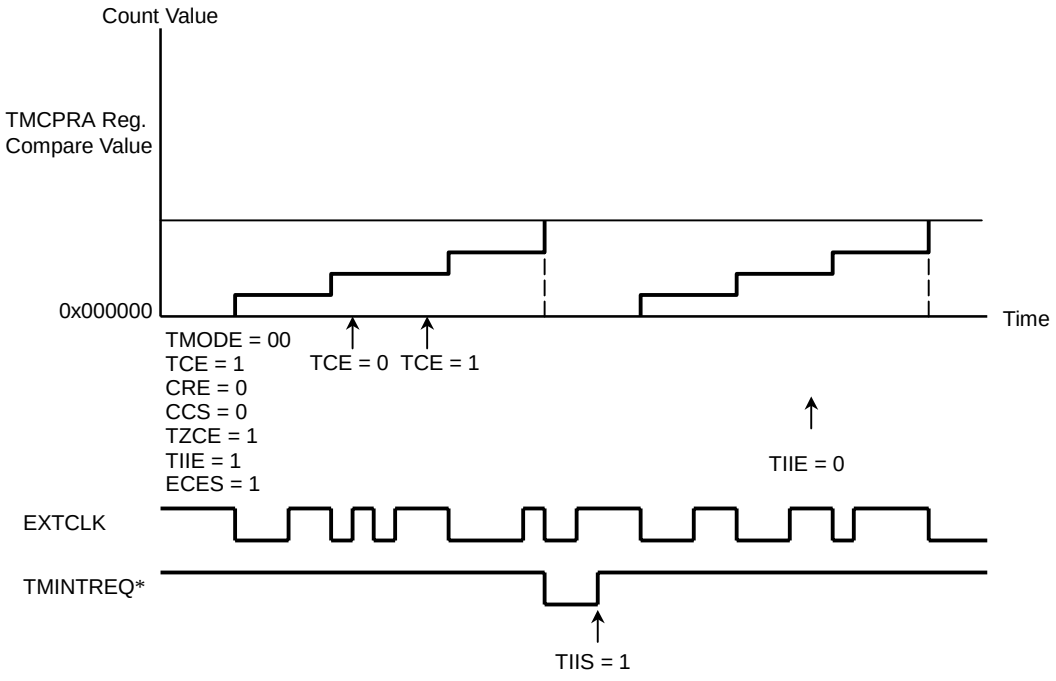


Figure 14.4.2 Example Interval Timer Mode Operation (with External Clock Input, Falling Edge Detected)

Table 14.4.2 shows the maximum count duration when the internal clock (IMCLK) is used. The assumption is that the CPU frequency is 133 MHz and the internal clock (IMCLK) frequency is 33 MHz.

Table 14.4.2 Divisors and Count Values

Divisor			System Clock = 33 MHz		
Decimal	TMCCDR. CCD	Frequency (Hz)	Resolution (Seconds)	24-bit Maximum Duration (Seconds)	Count of 1s
2	000	16.50E + 6	60.61E – 9	1.02	16500000
4	001	8.25E + 6	121.21E – 9	2.03	8250000
8	010	4.13E + 6	242.42E – 9	4.07	4125000
16	011	2.06E + 6	484.85E – 9	8.1	2062500
32	100	1.03E + 6	969.70E – 9	16.3	1031250
64	101	515.63E + 3	1.94E – 6	32.5	515625
128	110	257.81E + 3	3.88E – 6	65.1	257813
256	111	128.91E + 3	7.76E – 6	130.2	128906

14.4.2 Pulse Generator Mode

Pulse generator mode is supported by Timers 0 and 1, but not by Timer 2.

The timer is configured for pulse generator mode by setting the TMODE field of the TMTCR register to 01. In this mode, two compare registers, TMCPRA and TMCPRB, are used to generated a square wave with variable frequency and duty cycle.

Setting the TCE bit of the TMTCR enables counting. When the count value equals the value programmed in the TMCPRA, the timer flip-flop toggles. The output of the timer flip-flop is driven onto the TIMER[0] or TIMER[1] pin. The timer continues to count up. When the count value equals the value programmed in the TMCPRB, the timer flip-flop toggles again and the counter is reset. The TMCPRA value must be smaller than the TMCPRB value. The timer flip-flop can be initially set to 1 or 0 via the FFI bit of the TMPGMR register.

When the count value has reached the TMCPRA value, the TPIAS flag in the TMTISR is set.

The timer interrupt request (TMINTREQ*) signal is asserted if the TPIAE bit of the TMPGMR is set. Otherwise, no interrupt request is issued. Clearing the TPIAS bit of the TMTISR negates the TMINTREQ* signal.

If the TPIBE bit of the TMPGMR is set, TMINTREQ* is also asserted when the count value has reached the TMCPRB values, setting the TPIBS flag in the TMTISR. Clearing the TPIBS bit negates the TMINTREQ* signal.

Regardless of the setting of the TPIAE (TPIBE) bit, when the count value matches the TMCPRA (TMCPRB) value, the TPIAS (TPIBS) bit is set. The TPIAS (TPIB) bit, when set, indicates a pending interrupt; thus a timer interrupt request is issued if the TPIAE (TPIBE) bit is then set. To prevent this, the TPIAS (TPIBS) bit must be cleared before setting the TPIAE (TPIBE) bit.

The CCS bit of the TMTCR selects either the internal clock (IMCLK) or an external clock input.

When the internal clock (IMCLK) is selected, the clock divider divides IMCLK by the value programmed into the CCD field of the Clock Divider register (TMCCDR). The division factor can be 2^1

to 2^8. Counting occurs at the rising edge of the clock.

When the external pin (TCLK) is selected as the clock source, the active clock edge can be selected with the ECES bit of the TMTCR.

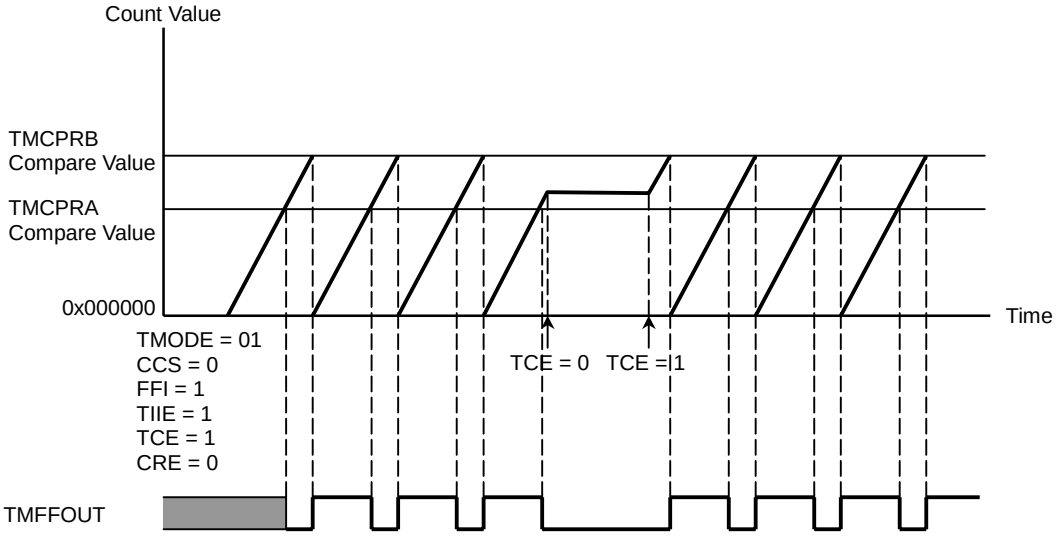


Figure 14.4.3 Example in Pulse Generator Mode Operation

14.4.3 Watchdog Timer Mode

Watchdog timer mode is supported by Timer 2.

A watchdog timer can be configured to cause either and nonmaskable interrupt or a system reset upon time-out. The WR bit of the CCFG determines whether the watchdog timer interrupt request signal is connected to the internal NMI* pin or the reset logic. If the CCFG.WR bit is set, the TX3927 is reset. If the CCFG.WR bit is cleared, NMI* is raised.

Watchdog timer mode is selected when the TMODE field of the TMTCR is set to 10. Setting the TCE bit of the TMTCR enables counting. When the count value equals the value programmed in the TMCPRA, the TWIS flag in the TMTISR is set. If the TWIE bit of the TMWTMR2 is set, a watchdog timer interrupt is generated. The watchdog timer interrupt generation is disabled by clearing the TWIE bit. If the watchdog timer interrupt request signal has been asserted, it is negated when the TWIS bit of the TMTISR is cleared. A write of 1 has no effect on this bit.

Regardless of the setting of the TWIE bit, when the count value matches the Compare Register B value, the TWIS bit is set. The TWIS bit, when set, indicates a pending interrupt; thus a watchdog timer interrupt request is issued if the TWIE bit is then set. To prevent this, the TWIS bit must be cleared before setting the TWIE bit.

The 24-bit counter can be reset to 0 by setting the TWC bit of the TMWTMR2. This bit is self-clearing.

If the WDIS bit of the TMWTMR2 is set, clearing the TCE bit halts the watchdog timer. If the WDIS bit is cleared, clearing the TCE bit does not disable counting operation. If the WDIS bit is set, clearing the TWIE bit of the TMWTMR2 can also disable the watchdog timer (and interrupt generation). When the WDIS bit is cleared, TWIE bit cannot be cleared. Once the watchdog timer is disabled, the WDIS bit is automatically cleared.

The TMTRR follows the contents of the 24-bit counter. A read of this register causes the current value of the counter to be read.

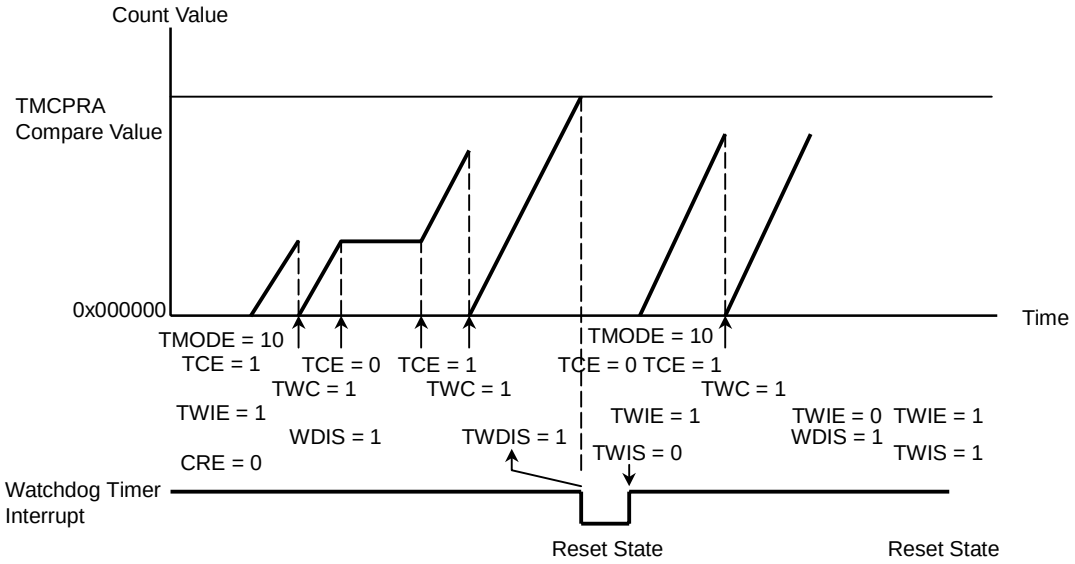


Figure 14.4.4 Example Watchdog Timer Mode operation

14.5 Timing Diagrams

14.5.1 Interrupt Timing in Interval Timer Mode

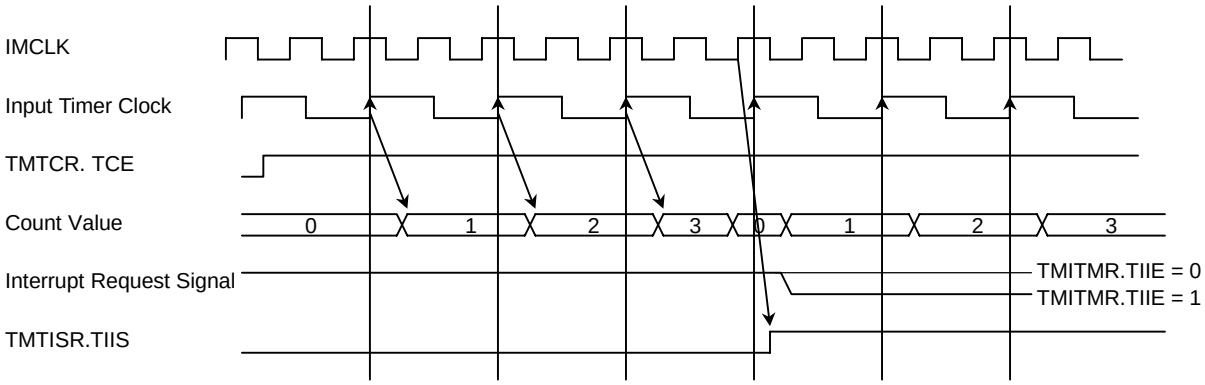


Figure 14.5.1 Interval Timer Timing (Internal Clock)

The above diagram is an example when TMCPRA=3 and the counter clock is IMCLK/2. After the count value has reached the TMCPRA value, the TMTISR.TIIS bit is set at the next rising edge of IMCLK, and TMINTREQ* is asserted synchronously with the internal system clock (IMCLK). At the same time, the counter is reset to 0 (if TMITMR.TZCE = 1).

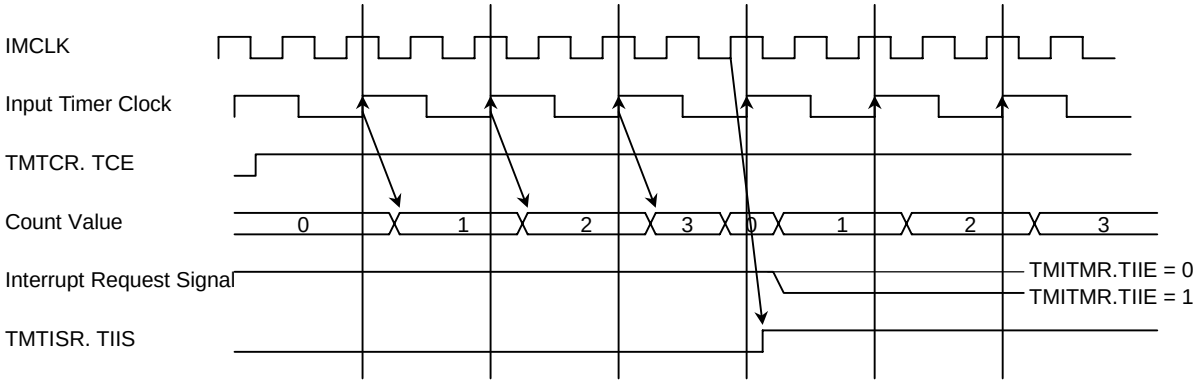


Figure 14.5.2 Interval Timer Timing (External Input Clock)

The above diagram is an example when TMCPRA=3 and the external input clock is selected as the counter clock source. After the count value has reached the TMCPRA value, the TMTISR.TIIS bit is set at the next rising edge of IMCLK, and TMINTREQ* is asserted. At the same time, the counter is reset to 0 (if TMITMR.TZCE = 1).

14.5.2 Output Flip-Flop Timing in Pulse Generator Mode

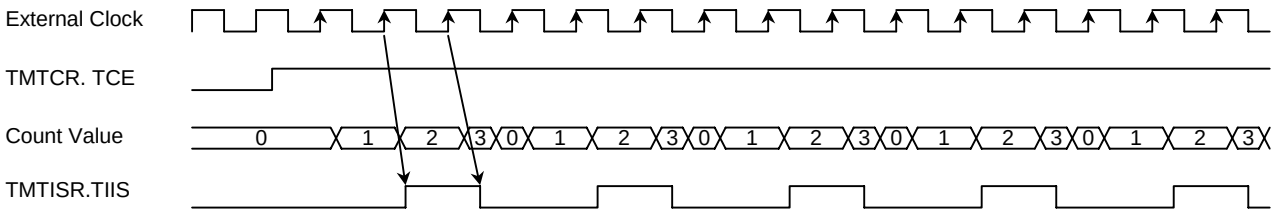


Figure 14.5.3 Pulse Generator Timing

The above diagram is an example when TMCPRA=2 and TMCPRB=3 in pulse generator mode. The initial value of the timer flip-flop is 0. The timer flip-flop is reset when the TMPGMR register is written.

14.5.3 Interrupt Timing in Watchdog Timer Mode

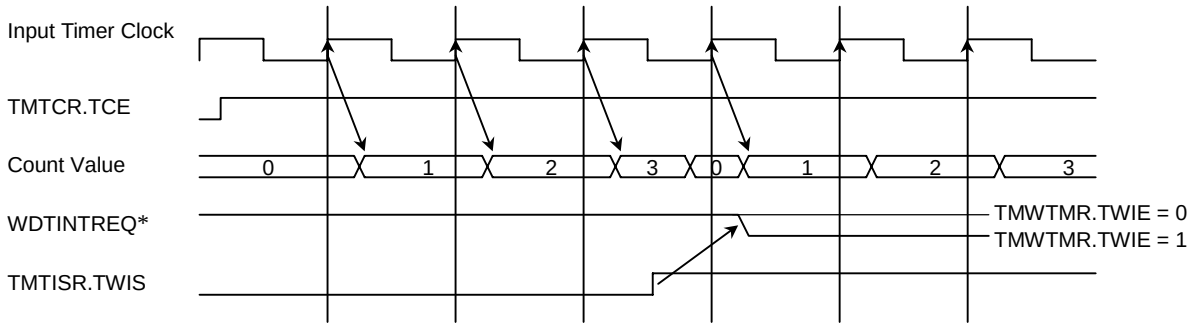


Figure 14.5.4 Watchdog Timer Timing

15. Parallel I/O Port (PIO)

15.1 Features

The TX3927 has a 16-bit general-purpose parallel I/O port (PIO).

The PIO provides the following features:

- Of the 16 I/O port pins, PIO1 to PIO15 are multiplexed with other pin functions on the Serial I/O (SIO), the Timer/Counter, or the DMAC. Each pin's function is individually selectable.
- The direction of each I/O pin can be individually configured for input or output.
- Each I/O pin can be individually configured for totem-pole or open-drain output.
- All 16 I/O pins can be read at all times, regardless of their direction or function.
- A 16-bit read/write flag register can be used to provide either general-purpose or special-purpose flags for software control.
- The flag bits can be used to trigger an interrupt request; a 16-bit polarity control register defines the polarity of each flag.
- All 16 flag interrupts can be independently masked by the corresponding bits of a 16-bit interrupt mask register.
- A CPU interrupt and a PCI interrupt are available. Any flag bit, when set to a programmed level, can produce an interrupt request. The CPU interrupt request signal is delivered to a CPU internal interrupt pin through the Interrupt Controller (IRC). The PCI interrupt request signal is presented via the PCI Controller (PCIC) as an external interrupt request. The PCI interrupt is valid only when the PCIC is programmed to operate in external arbiter mode. There are two mask registers, one each for the CPU interrupt and the PCI interrupt. The polarity control register specifies the logic conditions required for generating an interrupt.

15.2 Registers

15.2.1 Register Map

All the registers in the PIO should be accessed as a word quantity. For the bits other than those defined in this section, the values shown in the figures must be written.

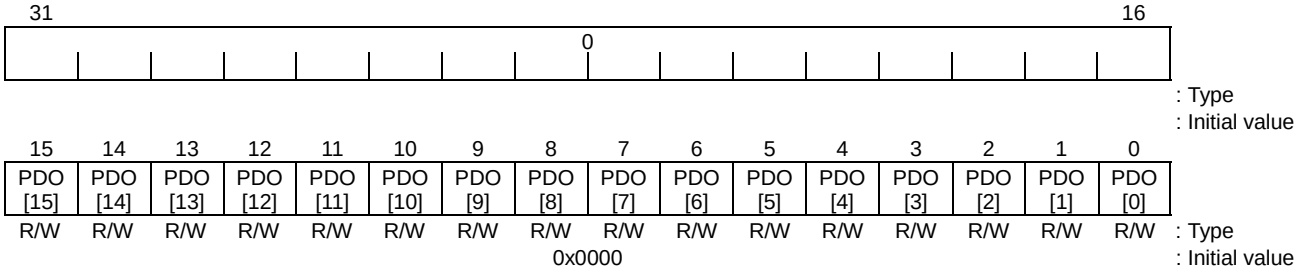
Table 15.2.1 PIO Registers

Address	Register Mnemonic	Register Name
0xFFFE_F524	XPIOMASKEXT	External Interrupt Mask Register
0xFFFE_F520	XPIOMASKCPU	CPU Interrupt Mask Register
0xFFFE_F51C	XPIOINT	Interrupt Control Register
0xFFFE_F518	XPIOPOL	Flag Polarity Control Register
0xFFFE_F514	XPIOFLAG1	Flag Register 1
0xFFFE_F510	XPIOFLAG0	Flag Register 0
0xFFFE_F50C	XPIOOD	Open-Drain Control Register
0xFFFE_F508	XPIODIR	Direction Control Register
0xFFFE_F504	XPIODI	Data Input Register
0xFFFE_F500	XPIODO	Data Output Register

Note 1: All registers are readable.

Note 2: All registers can only be accessed as words.

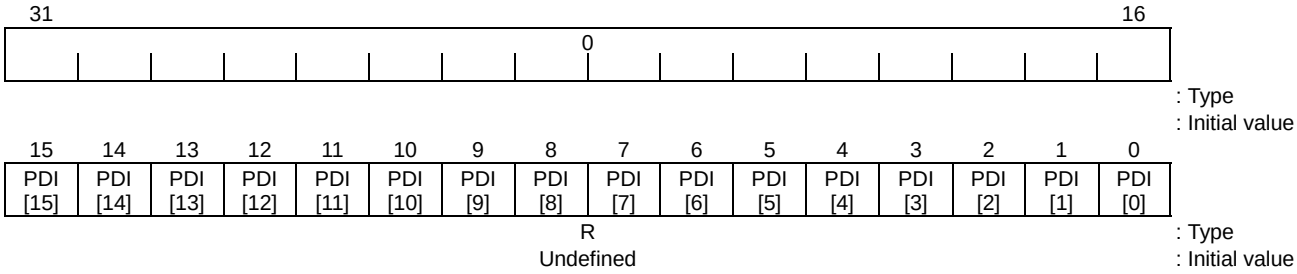
15.2.2 PIO Data Output Register (XPIODO) 0xFFFE_F500



Bits	Mnemonic	Field Name	Description
15:0	PDO[15:0]	Data Out	Port Data Output [15:0] (initial value: 0x0000) Holds data to be driven out from the PIO[15:0] pins.

Figure 15.2.1 PIO Output Data Register

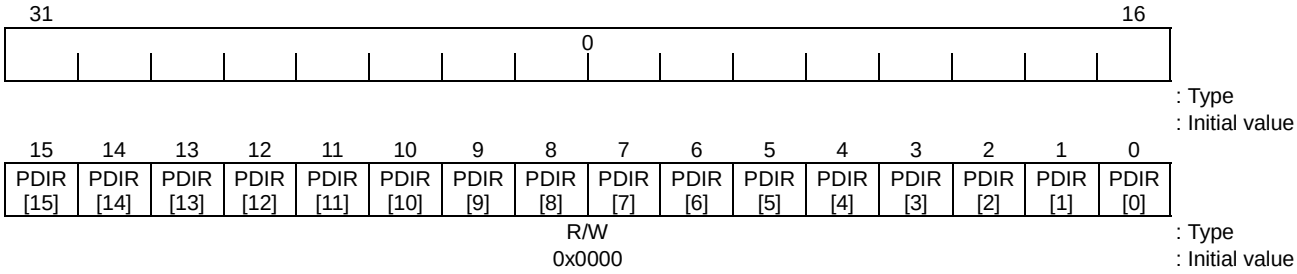
15.2.3 PIO Data Input Register (XPIODI) 0xFFFE_F504



Bits	Mnemonic	Field Name	Description
15:0	PDI[15:0]	Data In	Port Data Input [15:0] (initial value: undefined) Holds data read from the PIO[15:0] pins.

Figure 15.2.2 PIO Input Data Register

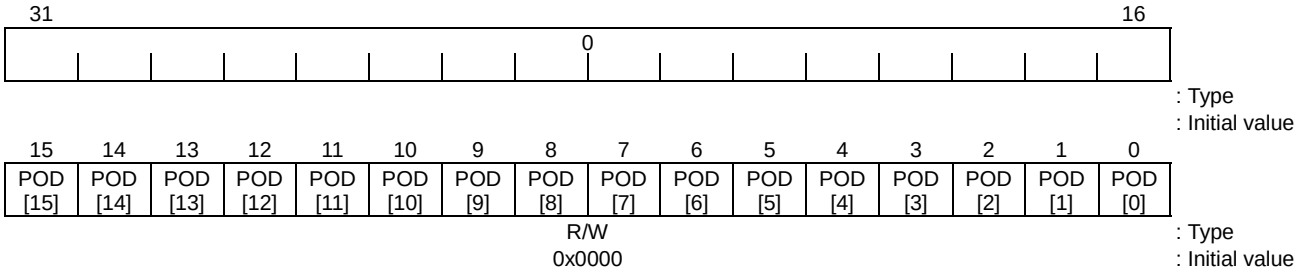
15.2.4 PIO Direction Control Register (XPIODIR) 0xFFFE_F508



Bits	Mnemonic	Field Name	Description
15:0	PDIR[15:0]	Direction Control	Port Direction Control [15:0] (initial value: 0x0000) Specifies the direction of each of the PIO[15:0] pins. 0: Input (Reset) 1: Output

Figure 15.2.3 PIO Direction Control Register

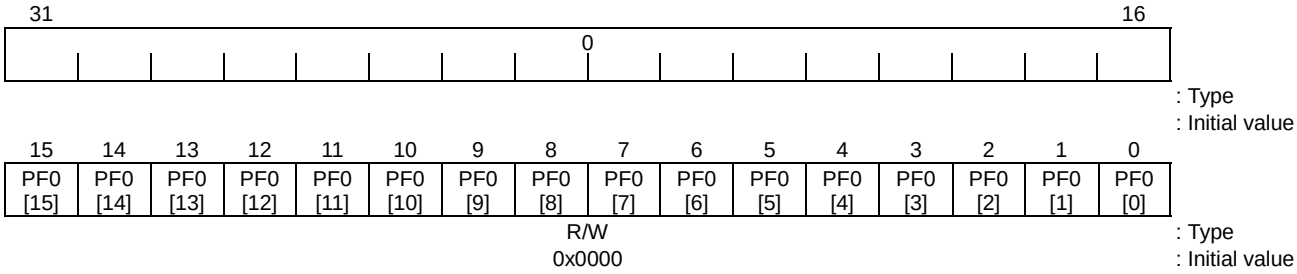
15.2.5 PIO Open-Drain Control Register (XPIOOD) 0xFFFE_E50C



Bits	Mnemonic	Field Name	Description
15:0	POD[15:0]	Open-Drain Control	Port Open-Drain Control [15:0] (initial value: 0x0000) Specifies whether to configure the PIO[15:0] pins as open-drain outputs or totem-pole outputs. 0: Open-drain (Reset) 1: Totem-pole

Figure 15.2.4 PIO Open-Drain Control Register

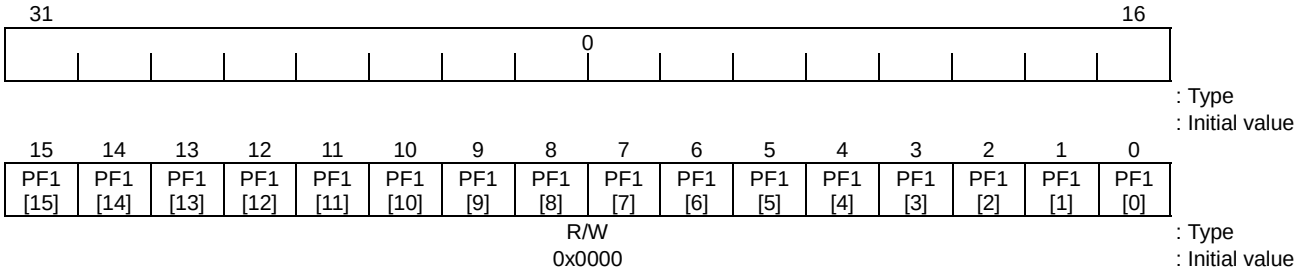
15.2.6 PIO Flag Register 0 (XPIOFLAG0) 0xFFFE_F510



Bits	Mnemonic	Field Name	Description
15:0	PF0[15:0]	Flag 0	PIO Flag 0 [15:0] (initial value: 0x0000) This is a general-purpose flag register. Flag registers 0 and 1 share the same storage elements. Flag register 0 allows the writing of both 1s and 0s in all its bits, whereas Flag register 1 has restrictions on writes.

Figure 15.2.5 PIO Flag Register 0

15.2.7 PIO Flag Register 1 (XPIOFLAG1) 0xFFFE_F514

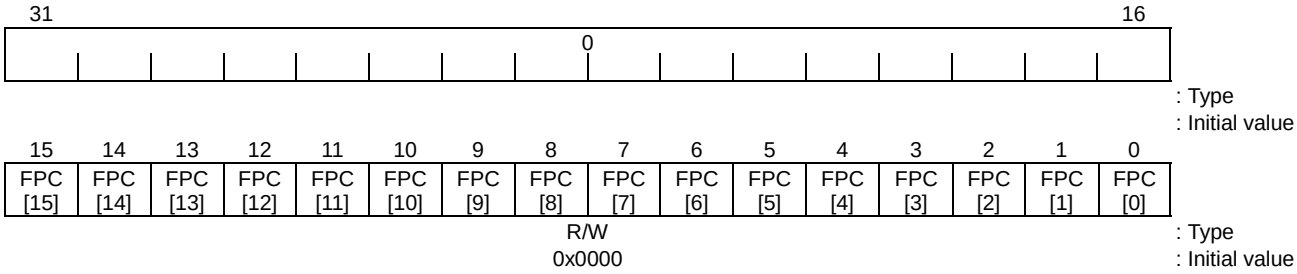


Bits	Mnemonic	Field Name	Description
15:0	PF1[15:0]	Flag 1	PIO Flag 1 [15:0] (initial value: 0x0000) This is a special-purpose flag register. Flag registers 0 and 1 share the same storage elements. Flag register 1 has the following restrictions. - On writes - Writes by the CPU 1: A write of 1 sets a flag bit. 0: A write of 0 has no effect on a flag bit. - Write by other devices (i.e., DMAC and PCIC) 1: A write of 1 clears a flag bit. 0: A write of 0 has no effect on a flag bit. - On reads: There is no restriction on reads. The bit value is read.

Note: A write of 0 has no effect on the flag bits (PF1[15:0]) of Flag register 1. A write of 1 by the CPU to a flag bit sets that bit, while a write of 1 by other device (bus master) to a flag bit clears that bit. Flag register 1 can be read by all devices.

Figure 15.2.6 PIO Flag Register 1

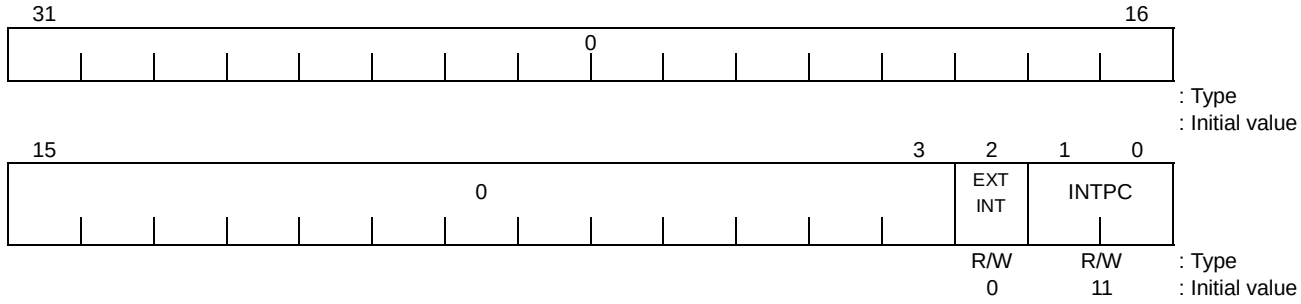
15.2.8 PIO Flag Polarity Control Register (XPIOPOL) 0xFFFE_F518



Bits	Mnemonic	Field Name	Description
15:0	FPC[15:0]	Flag Polarity Control	Flag Polarity Control [15:0] (initial value: 0x0000) Determines the logic conditions required for a flag bit to generate an interrupt request. The FPC bit is XORed with the current value of the flag bit to form an interrupt request. Flag bits FPC bits Interrupt request 0 0 Not issued 0 1 Issued 1 0 Issued 1 1 Not issued

Figure 15.2.7 PIO Flag Polarity Control Register

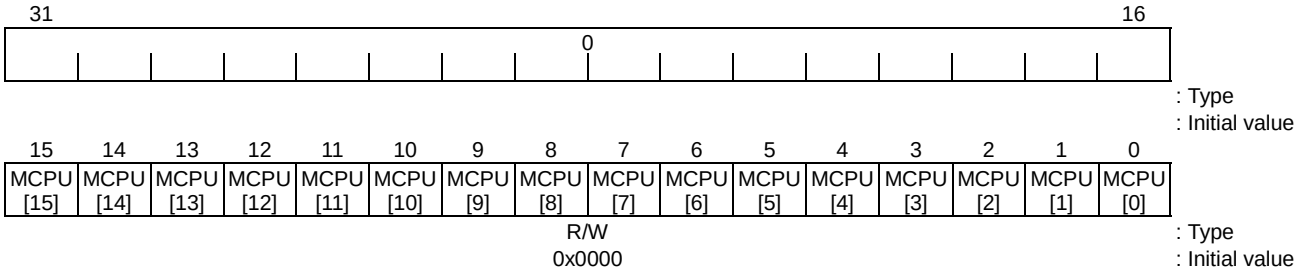
15.2.9 PIO Interrupt Control Register (XPIOINT) 0xFFFE_F51C



Bits	Mnemonic	Field Name	Description
2	EXT INT	EXT Interrupt OD Control	EXT Interrupt OD Control (initial value: 0) Specifies whether the external interrupt signal is configured for open-drain or totem-pole operation. 0: Open-drain (Reset) 1: Totem-pole
1:0	INTPC[1:0]	Interrupt Polarity Control	Interrupt Polarity Control (initial value: 11) Determines the logic conditions required to generate an interrupt. The INTPC bit is XORed with each interrupt request signal, which is connected to the XOR of the flag bit and the FPC bit. An interrupt is generated if the result is 1. Bit 0 controls the interrupt to the CPU; bit 1 controls the interrupt to the external bus master. Interrupt request / INTPC bit / Interrupt 0 / 0 / Not generated 0 / 1 / Generated 1 / 0 / Generated 1 / 1 / Not generated

Figure 15.2.8 PIO Interrupt Control Register

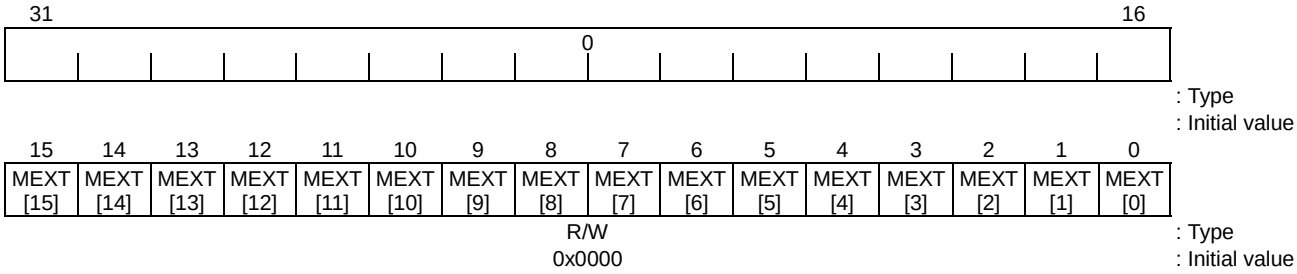
15.2.10 CPU Interrupt Mask Register (XPIOMASKCPU) 0xFFFE_F520



Bits	Mnemonic	Field Name	Description
15:0	MCPU[15:0]	Mask Bits	Mask CPU [15:0] (initial value: 0x0000) Allows any flag bits to be masked off as CPU interrupt sources. Clearing a bit in this register masks the corresponding interrupt source. 0: Mask the interrupt (Reset). 1: Don't mask the interrupt.

Figure 15.2.9 CPU Interrupt Mask Register

15.2.11 External Interrupt Mask Register (XPIOMASKEXT) 0xFFFE_F524



Bits	Mnemonic	Field Name	Description
15:0	MEXT[15:0]	Mask Bits	Mask EXT [15:0] (initial value: 0x0000) Allows any flag bits to be masked off as an external bus master interrupt sources. Clearing a bit in this register masks the corresponding interrupt source. 0: Mask the interrupt (Reset). 1: Don't mask the interrupt.

Figure 15.2.10 External Interrupt Mask Register

15.3 Operation

15.3.1 Assigning PIO Pin Functions

The TX3927 has a 16-bit PIO channel. The PIO[0] pin is dedicated to the PIO function while PIO1 to PIO15 are shared with other pin functions on the SIO, the Timer/Counter and the DMAC. The Pin Configuration register is used to select the desired pin functions. When PIO is selected for a pin, that pin's PIO function is determined by the settings in the PIO registers.

See "3.3 Pin Multiplexing" for shared pin functions.

15.3.2 General-Purpose Parallel Port

The PIO Direction Control register determines the direction of each PIO pin, input or output. The PIO Input Data register holds the data read from the PIO pins configured as inputs. The PIO Output Data register holds the data to be written to the PIO pins configured as outputs.

15.3.3 Interrupt Requests

The PIO provides two interrupt request signals: a CPU interrupt request and an external interrupt request. A polarity control register, two mask registers, an interrupt control register and a flag register are used to control the generation of interrupts. The following equations represent the requirements for an interrupt to be generated:

CPU interrupt request = |((XPIOFLAG[15:0] ^ XPIOPOL[15:0]) & XPIOMASKCPU[15:0]) ^ XPIOINT[0]

External interrupt request = |((XPIOFLAG[15:0] ^ XPIOPOL[15:0]) & XPIOMASKEXT[15:0]) ^ XPIOINT[1]

Note: "^" is the exclusive-OR operator. "|" is the reduction OR operator, which takes the OR value of all the bits of the operand and returns a 1-bit result. Upon power-on reset, both the interrupt request signals default to 1.

The external interrupt request is valid only when the PCI Controller (PCIC) is programmed to operate in PCI external arbiter mode. The REQ[1] pin is used to deliver the interrupt signal.

The CPU interrupt request is presented as a PIO interrupt, which is assigned to interrupt number 9 in the integrated Interrupt Controller (IRC).

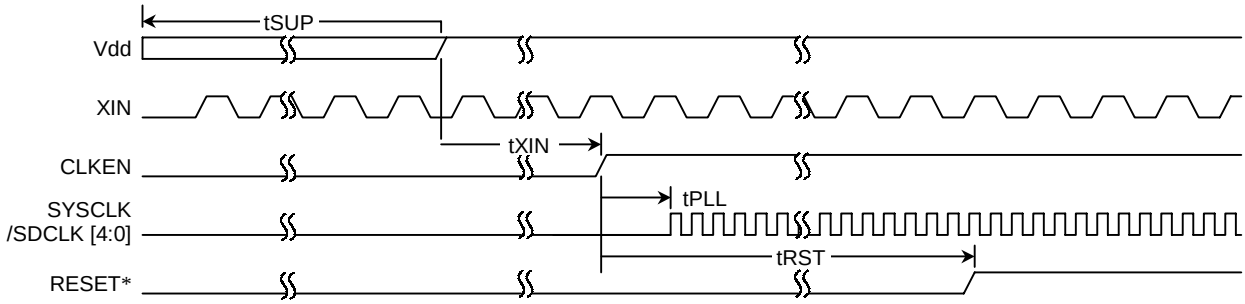
There are two flag registers, Flag register 0 and Flag register 1. They have different addresses, but share the same storage elements. As such, both of these registers always return an identical value on reads. However, they function differently during write operation. While Flag register 0 allows both 1s and 0s to be written to all its bits, Flag register 1 accepts only a write of 1 from the CPU. An attempt by the DMAC or PCIC to write a 1 to a Flag register 1's bit clears that bit; a write of 0 by the DMAC or PCIC has no effect on the bit.

15.3.4 Accessing PIO Pins

All 16 PIO pins can be read at all times, regardless of their function or direction.

16. Power-On Sequence

Figure 16.1 is a functional timing diagram for the power-on sequence, illustrating the relationships among Vdd, CLKEN, RESET* and other signals. Unless CLKEN or RESET* meets the specified requirements, the PLL will not lock or the initialization of the TX3927 will not complete properly.



Note 1: tSUP is the time after power-on required for Vdd to stabilize

Note 2: tXIN is the oscillation settling time for the XIN clock input.

Note 3: tPLL is the lock time for the on-chip PLL.
tPLL varies with the PLL multiplication factor, as follows:

PLLM	1	0	n
	0	1	16401
	1	1	2051

Note 4: tRST is the interval between the assertion of CLKEN and the negation of RESET*.
tPLL + 256 SDCLK cycles (SDCLK is half the CPU frequency.)

Figure 16.1 Power-On Initialization Sequence

The CLKEN signal is used to initialize the on-chip clock generator of the TX3927. After power-on, CLKEN must be held low until Vdd and XIN stabilize. Asserting CLKEN causes the on-chip PLL to begin oscillation. The TX3927 contains an autonomous counter/timer used to allow the PLL to settle; thereafter, the internal clock, the SYSCLK output and the SDCLK output start free-running. This counter/timer uses XIN as a clock source. The count value varies with the PLL multiplication factor defined by boot signals PLLM[1:0], as shown in Table 16.1.

Table 16.1 PLL Multiplication Factors and Count Values

PLL Multiplication Factor	PLLM[1:0] Pin Value	Count Value
2	01	16401
16	11	2051

17. Electrical Characteristics

17.1 Absolute Maximum Ratings (*1)

Parameter	Symbol	Rating	Units
Supply voltage	V_{DDS} V_{DD2}*	−0.3 ~ 4.5 −0.3 ~ 3.6	V V
Input voltage for RXD [1:0], CTS* [1:0], PCIAD [31:0], PCICLK [3:0], GNT [3:0], REQ [3:0], C_BE [3:0], IDSEL, FRAME*, IRDY*, TRDY*, DEVSEL*, STOP*, PERR*, SERR*, PAR	V_{IN1}	−0.3 ~ 6.7	V
Input voltage for all other inputs	V_{IN2}	−0.3 ~ V_{DDS} + 0.3V	V
Storage temperature	T_{STG}	−40 ~ 125	°C
Maximum power dissipation	P_D	2.0	W

(*1) Don't use the device under conditions in which any one of its absolute maximum ratings is exceeded. Otherwise, the device may break down or its performance may be degraded, causing it to catch fire or explode, resulting in injury to the user. Thus, when designing a product which includes this device, ensure that no absolute maximum rating value will ever be exceeded.

*: Including PLLVDD.

17.2 Recommended Operating Conditions (*2)

Parameter		Symbol	Conditions	Min	Max	Units
Supply voltage	I/O	V_{DDS}		3.0	3.6	V
	Internal logic	V_{DD2}*	TLB not used	2.3	2.7	V
			TLB used	2.4	2.7	
Operating temperature (case temperature)		T_c		0	70	°C

Note: This product is designed principally for use in office equipment. If you intend to use it for any other type of application, please contact Toshiba engineering staff.

(*2) The recommended operating conditions for the device are those necessary to guarantee that the device will operate as specified. If the recommended operating conditions (supply voltage, operating temperature range, specified AC and DC values etc.) are exceeded, malfunction may occur. Thus, when designing a product which includes this device, ensure that the recommended operating conditions for the device are adhered to.

*: Including PLLVDD.

17.3 DC Characteristics

17.3.1 DC Characteristics – Non-PCI Interface Pins

(Tc = 0 ~ 70°C, V_{DDS} = 3.3V ± 0.3V, V_{DD2} = 2.5V ± 0.2V, V_{SS} = 0V)

Parameter		Symbol	Conditions	Min	Max	Units
Low-level input voltage		V_{IL1} V_{IL2}	Other than RXD[1:0] and CTS*[1:0] RXD[1:0] and CTS*[1:0]		V_{DDS}× 0.2 0.8	V
High-level input voltage		V_{IH1} V_{IH2}	Other than RXD[1:0] and CTS*[1:0] RXD[1:0] and CTS*[1:0]	V_{DDS}×0.8 2.0	V_{DDS} + 0.3 5.5	V
Low-level output current		I_{OL1} I_{OL2}	(Note 1) V_{OL} = 0.4 V (Note 2) V_{OL} = 0.4 V		8 16	mA mA
High-level output current		I_{OH1} I_{OH2}	(Note 1) V_{OH} = 2.4 V (Note 2) V_{OH} = 2.4 V	−8 −16		mA mA
Operating current	I/O	I_{DDS}	f = 133 MHz, V_{DDS} = 3.6 V		120	mA
	Internal logic	I_{DD2}	f = 133 MHz, V_{DD2} = 2.7 V		420	
Input leakage current		I_{IH} I_{IL}		−10 −10	10 10	μA μA
Pull-up resistance		RST		50	300	kΩ

Note 1: Signals other than those shown in Note 2, below.

Note 2: ADDR[19:5], SDCLK[4:0], DQM[3:0], DATA[31:0], CAS*, RAS*, CKE, WE*, OE*, SYSCLK, GDCLK, ACK*

Note 3: f = CPU core operating frequency

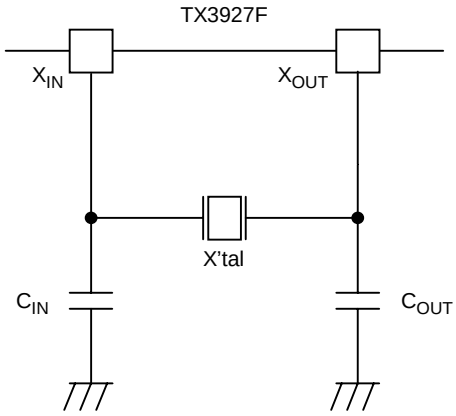
17.3.2 DC Characteristics – PCI Interface Pins

(Tc = 0 ~ 70°C, V_{DDS} = 3.3V ± 0.3V, V_{DD2} = 2.5V ± 0.2V, V_{SS} = 0V)

Parameter	Symbol	Conditions	Min	Max	Units
Low-level input voltage	V_{IL3}		−0.5	V_{DDS} × 0.3	V
High-level input voltage	V_{IH3}		V_{DDS}×0.5	5.5	V
High-level output voltage	V_{OH}	I_{OUT} = −500 μA	V_{DDS}×0.9		V
Low-level output voltage	V_{OL}	I_{OUT} = 1500 μA		V_{DDS}×0.1	V
Input leakage current	I_{IH} I_{IL}	0 < V_{IN} < 5 V	−10 −10	10 10	μA μA

17.4 Crystal Oscillator Characteristics

17.4.1 Recommended Oscillator Conditions (with a PLL Multiplication Factor of 16)



Parameter	Symbol	Recommended value	Units
Crystal oscillator frequency	f_{IN}	6.25 ~ 8.33	MHz
External capacitor	C_{IN}, C_{OUT}	T.B.D.	pF
Crystal oscillator Rise time Fall time	 t_r t_f	 5(1) 5(1)	 ns ns

(1) These are reference values. Refer to the latest data provided by the manufacturer of the crystal oscillator.

17.4.2 Recommended Input Clock Conditions (with a PLL Multiplication Factor of 2)

When using a multiplication factor of 2, an external clock should be supplied via the XIN pin, with XOUT left open.

Parameter	Symbol	Recommended value	Units
Input clock frequency	f_{IN}	50 ~ 66.67	MHz

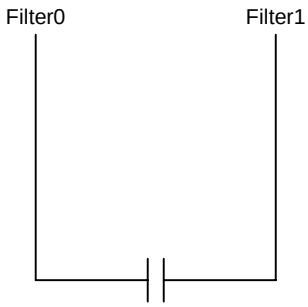
17.4.3 Electrical Characteristics

($Tc = 0 \sim 70°C$, $V_{DDS} = 3.3V \pm 0.3V$, $V_{DD2} = 2.5V \pm 0.2V$, $V_{SS} = 0V$)

Parameter	Symbol	Conditions	Min	Typ.	Max	Units
Crystal oscillation start time	t_{STA}	f=6.25~8.33 MHz	—	1	10	ms

17.5 PLL Filter Circuit

An external filter capacitor is required between the Filter[1] and Filter[0] pins to configure the PLL filter.



Parameter	Symbol	Recommended value	Units
External capacitor	C_{Filter}	1800 (with a PLL multiplication factor of 16) 220 (with a PLL multiplication factor of 2)	pF

17.6 AC Characteristics – Non-PCI Interface Pins

17.6.1 AC Characteristics

(Tc = 0 ~ 70°C, V_{DDS} = 3.3 ± 0.3 V, V_{DD2} = 2.5 ± 0.2 V, V_{SS} = 0 V, CL = 50 pF)

Parameter	Signals	Description	Min	Max	Units
t_{sys}	SYSCLK/SDCLK[4:0]	Cycle Time (Full-speed bus mode)	15		ns
t_{sysh}	SYSCLK	Cycle Time (Half-speed bus mode)	30		ns
t_{sysm}	SYSCLK/SDCLK[4:0]	Min High/Low Level	5		ns
t_{sysmh}	SYSCLK	Min Half-Speed High/Low Level	12		ns
t_d	(1)	Output Delay		7	ns
t_{oh}	(1)	Output Hold	1		ns
t_{su}	(2)	Input Setup	7		ns
t_{ih}	(2)	Input Hold	0		ns
t_{daz}	DATA[31:0], ACK*	Data Active to Hi-Z		7	ns
t_{dza}	DATA[31:0], ACK*	Data Hi-Z to Active	1		ns

(1) ACK*, DATA[31:0], ROMCE[7:0]*, OE*, ACE*, SWE*, BWE[3:0]*, ADDR[19:2], DMAACK[3:0], DMADONE*, PIO[15:0], TIMER[1:0]

(2) ACK*, DATA[31:0], NMI*, INT[5:0], DMAREQ[3:0], DMADONE*, PIO[15:0]

17.6.2 SDRAM Interface AC Characteristics

(Tc = 0 ~ 70°C, V_{DDS} = 3.3 ± 0.3 V, V_{DD2} = 2.5 ± 0.2 V, V_{SS} = 0 V, CL = 50 pF for SDCLK[4:0])

Parameter	Signals	Description	50 pF		100 pF		150 pF		Units
			Min	Max	Min	Max	Min	Max	
t_{sdclk}	SDCLK[4:0]/SYSCLK	Cycle Time	15		15		15		ns
t_{sdclkm}	SDCLK[4:0]/SYSCLK	Minimum High/Low Level	5		5		5		ns
t_{sd}	(3)	Output Delay		7		8		9	ns
t_{sdd}	DATA[31:0]	Output Delay		8		10		12	ns
t_{soh}	(4)	Output Hold	1		1		1		ns
t_{ssu1}	DATA[31:0]	Input Setup (Internal clock)	7		7		7		ns
t_{ssu2}	DATA[31:0]	Input Setup (Pin feedback clock)	2		2		2		ns
t_{sih}	DATA[31:0]	Input Hold	0		0		0		ns
t_{sdaz}	DATA[31:0]	Data Active to Hi-Z		7		7		7	ns
t_{sdza}	DATA[31:0]	Data Hi-Z to Active	1		1		1		ns

(3) SDCS[7:0], RAS*, CAS*, WE*, CKE, OE*, DSF, ADDR[19:5], DQM[3:0]

(4) SDCS[7:0], RAS*, CAS*, WE*, CKE, OE*, DSF, ADDR[19:5], DQM[3:0], DATA[31:0]

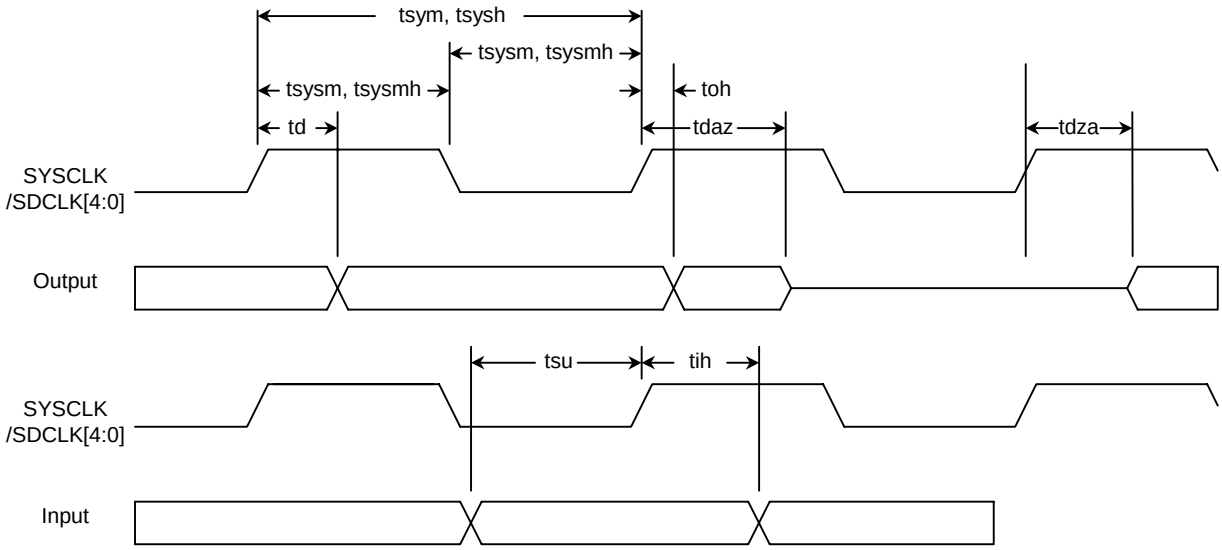
17.7 AC Characteristics – PCI Interface Pins

17.7.1 AC Characteristics

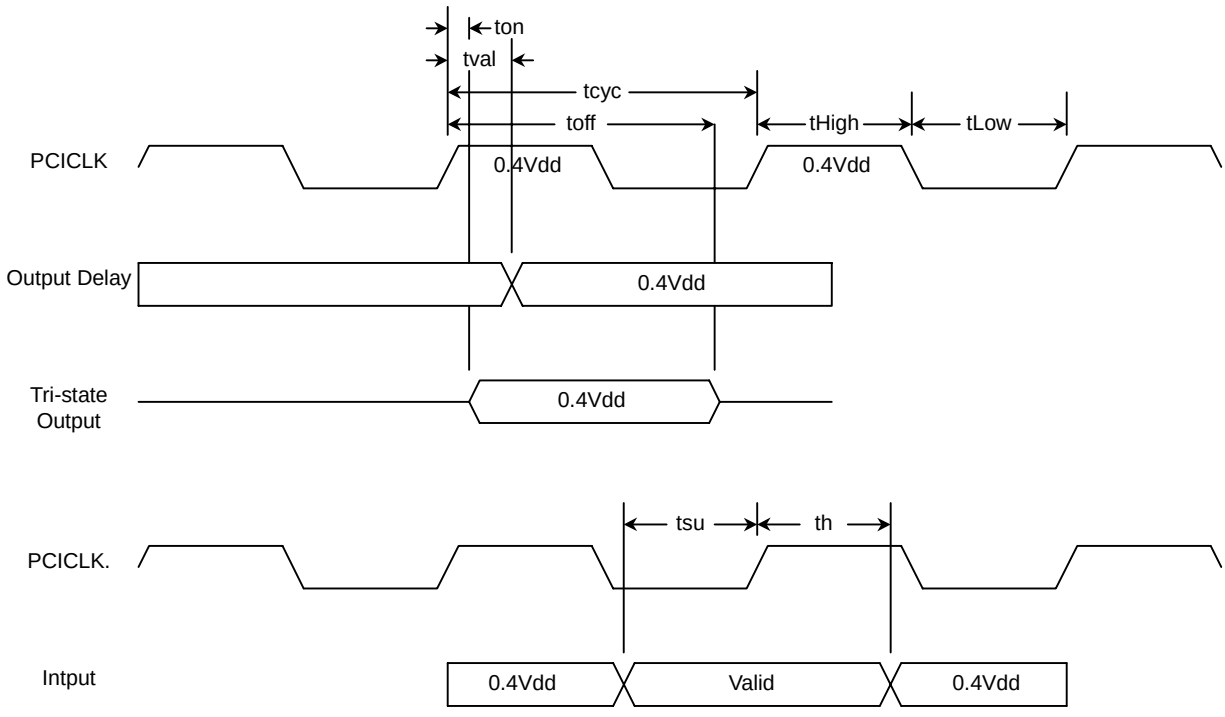
(PCI_CLK speed = 33 MHz, Tc = 0 ~ 70°C, V_{DDS} = 3.3 ± 0.3 V, V_{DD2} = 2.5 ± 0.2 V, V_{SS} = 0 V, CL = 50 pF)

Parameter	Description	Min	Max	Units
t_{cyc}	PCI_CLK cycle time	30		ns
t_{high}	PCI_CLK High time	11		ns
t_{low}	PCI_CLK Low time	11		ns
—	PCI_CLK slew rate	1	4	V/ns
t_{val}	PCI_CLK to signal valid delay (bus signals)	2	11	ns
t_{val}(ptp)	PCI_CLK to signal valid delay (point-to-point signals)	2	12	ns
t_{on}	Hi-Z to active delay	2		ns
t_{off}	Active to Hi-Z delay		28	ns
t_{su}	Input setup time to PCI_CLK (bus signals)	7		ns
t_{su}(ptp)	Input setup time to PCI_CLK (point-to-point signals)	12		ns
t_h	Input hold time from PCI_CLK	0		ns
t_{rst}	Reset active time after power stable	1		ms
t_{rst}-clk	Reset active time after PCI_CLK stable	100		us
t_{rst}-off	Reset active to output Hi-Z delay		40	ns

17.7.2 Timing Diagram for SDRAMC and ROMC Interface Pins



17.7.3 Timing Diagram for PCI Interface Pins



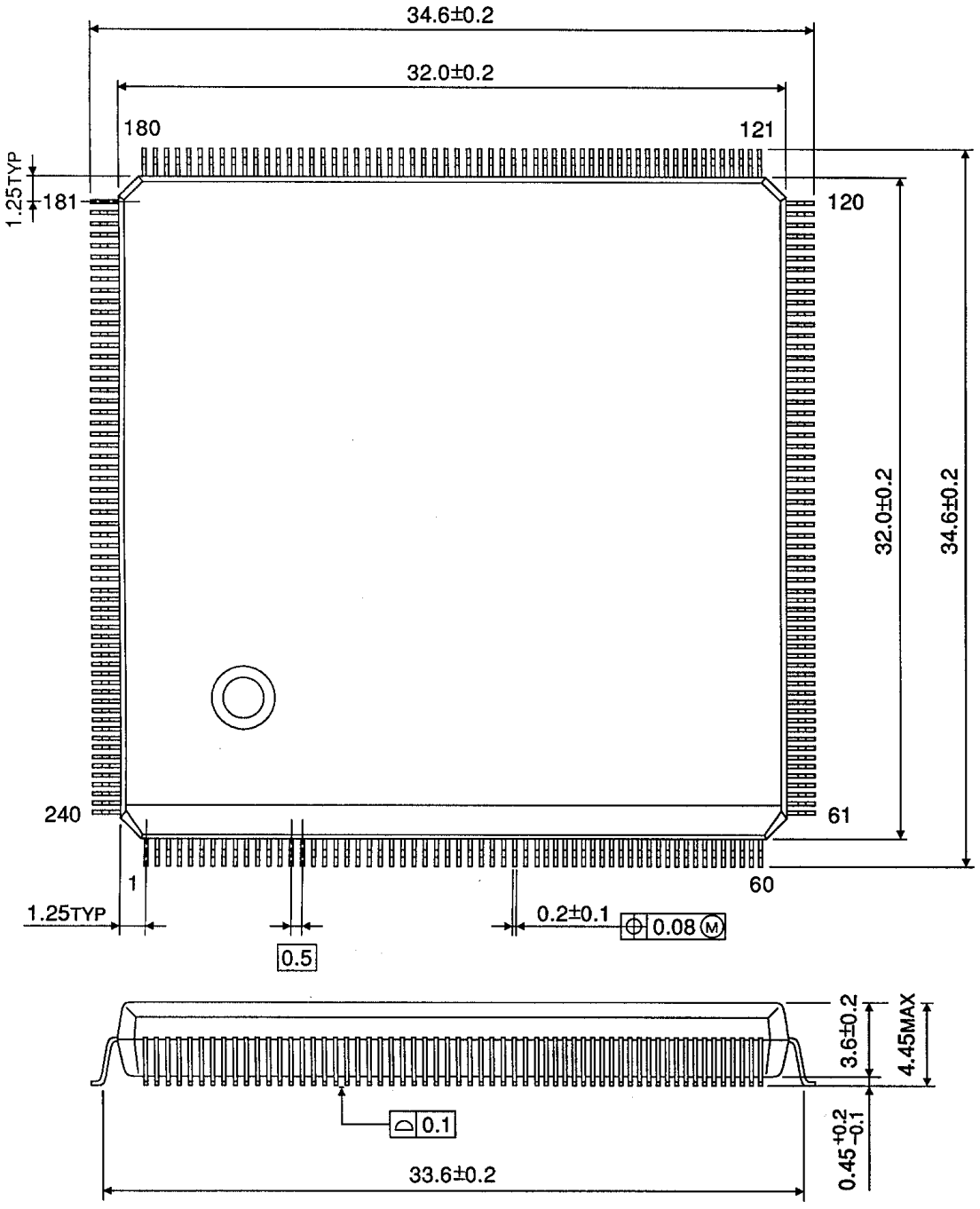
17.8 Serial Input Clock

Parameter	Description	Equation		CPU Operating Frequency = 133 MHz		Units
		Min	Max	Min	Max	
t_{SCY}	SCLK period	8X + 25 ns		85		ns
t_{SCYL}	SCLK low-level width	8X + 10 ns		40		ns
t_{SCYH}	SCLK high-level width	8X + 10 ns		40		ns

Note: X is the CPU operating clock period.

18. Package Dimensions

P-QFP240-3232-0.50 Units: mm



19. Notes on Use

19.1 Programming Restrictions for the TMPR3927A

The TX39 processor core contains cache memory and a write buffer, which may cause the sequence in which bus operations (for example, writes to registers in the SDRAMC, ROMC, IRC, etc.) take place to differ from that in program code.

(1) Programs residing in uncacheable and cacheable spaces

Programs that reside in an uncacheable space (see "4.1 Memory Mapping") execute instructions and bus operations in the exact order in which they are coded. Programs that reside in a cacheable space, in the case of cache hits, may execute instructions or bus operations in an unorderly fashion (read operations may take precedence).

(2) sync instruction

The sync instruction stalls the pipeline until the bus operation performed prior to the present instruction is completed. As for writes, however, it only stalls the write from the CPU to the write buffer; it does not stall the write from the write buffer to the target peripheral register.

(3) Write buffer

The write operation takes priority over the read operation for the following two cases:

- When the processor issues a read request to the target address of one of the write buffer entries.
- When the processor issues an uncacheable read reference while the write buffer has uncacheable write data.

Following are code examples that enable interrupts by setting interrupt masks in the IRC:

```
sw      r8, IRC_IRIMR_ADDR      # Set IRIMR interrupt mask
mfc0    r9, r12                 # Read CP0 Status register
or      r9, r9, r10             # IP[4:0], IEc interrupt enable: r10 = 0x00007c01
mtc0    r9, r12                 # Enable interrupts
```

Figure 19.1.1 Program that Can Accept an Unintended Interrupt

```
sw      r8, IRC_IRIMR_ADDR      # Set IRIMR interrupt mask
lw      r8, IRC_IRSSR_ADDR      # Read uncacheable space to complete previous write to
                                # uncacheable space
mfc0    r9, r12                 # Read CP0 Status register
or      r9, r9, r10             # IP[4:0], IEc interrupt enable: r10 = 0x00007c01
mtc0    r9, r12                 # Enable interrupts
```

Figure 19.1.2 Program that Accepts an IMR-Masked Interrupt

The program in Figure 19.1.1, if located in a cacheable space with cache enabled, might enable interrupts prior to the write to the IRIMR, accepting an unexpected interrupt. On the other hand, the program in Figure 19.1.2 first intentionally reads the IRSSR, which is an uncacheable space, so that the previous write to the IRIMR is certainly completed before interrupts are enabled.

Appendix A. TX3927 Programming Samples

Note 1: Toshiba does not guarantee that the sample programs shown in this appendix function properly in your operating environment.

Note 2: The program source codes are provided for explanation and cannot be compiled and executed as is. Some examples assume the use of UDEOS, a feeware μITRON-compliant operating system from Toshiba Information Systems Corporation.

A.1 Programming Tips for Beginners

A.1.1 Memory-Mapped I/O

The MIPS architecture employs memory-mapped I/O. Programmers who don't have experience with many types of microprocessors seem to get puzzled first by this scheme, because there are no input/output instructions or functions to access peripheral registers, (i.e., those equivalent to the x86 in/out instructions and inp/outp functions).

In a memory-mapped I/O system, I/O and control registers are acccessed in the same way as memory locaitons. Any instruction that can be used to access memory such as Store Byte or Store Word can also be used to access I/O registers. Those registers are located at fixed addresses as described in this manual. For example, to write a value of 0x0000f000 to the PIODIR register (at address 0xfffef508), first define the following:

```
volatile int *pdir;
pdir = (volatile int*) 0xfffef508;
```

Then, execute:

```
*pdir = 0x0000f000;
```

Remember that "volatile" is required. Without "volatile," the register access might be optimized out of the program by the compiler.

A.1.2 Accessing Coprocessor 0 Registers

With the MIPS architecture, System Control Coprocessor 0 (CP0) registers are used to write system programs, such as interrupt handling routines. Since C does not support CP0 instructions, the compiler library does not provide functions to access CP0 registers. CP0 registers, therefore, entail programming in assembly language. If you want to code software in C, you must create an additional library of your own or use inline assembly code programs. Following is a sample CP0 register access routine which can be called from a C program:

- File name: cp0ins.S

```
.macro      getcp0reg       name,regno      # getcp0_xx();
   .globl   ¥name
   .ent ¥name
¥name:
   mfc0     $2,¥regno       # reg->$2
   jr       $31
   nop
   .end     ¥name
 .endm

 .macro     putcp0reg       name,regno      # put cp0_xx(val);
   .globl   ¥name
   .ent ¥name
```

```
¥name:
   mtc0      $4,¥regno      # reg->$2
   jr        $31
   nop
   .end      ¥name
 .endm

 .macro      setcp0reg      name,regno       # setcp0_xx(mask,val);
   .globl    ¥name
   .ent ¥name
¥name:
   mfc0      $2,¥regno
   nop
   and       $1,$2,$4
   or        $1,$1,$5
   mtc0      $1,¥regno      # val->CP reg
   jr        $31
   nop
   .end       ¥name
 .endm

   .set noat
   getcp0reg           getcp0_0 $0
   putcp0reg           putcp0_0 $0
   setcp0reg           setcp0_0 $0
   getcp0reg           getcp0_1 $1
   putcp0reg           putcp0_1 $1
   setcp0reg           setcp0_1 $1
   getcp0reg           etcp0_2 $2
   putcp0reg           putcp0_2 $2
   setcp0reg           setcp0_2 $2
   getcp0reg           getcp0_3 $3
   putcp0reg           putcp0_3 $3
   setcp0reg           setcp0_3 $3
   getcp0reg           getcp0_4 $4
   putcp0reg           putcp0_4 $4
   setcp0reg           setcp0_4 $4
   getcp0reg           getcp0_5 $0
   putcp0reg           putcp0_5 $0
   setcp0reg           setcp0_5 $0
   getcp0reg           getcp0_6 $6
   putcp0reg           putcp0_6 $6
   setcp0reg           setcp0_6 $6
   getcp0reg           getcp0_7 $7
   putcp0reg           utcp0_7 $7
   setcp0reg           setcp0_7 $7
   getcp0reg           getcp0_8 $8
   putcp0reg           putcp0_8 $8
   setcp0reg           setcp0_8 $8
   getcp0reg           getcp0_9 $9
   putcp0reg           putcp0_9 $9
   setcp0reg           setcp0_9 $9
   getcp0reg           getcp0_10 $10
   putcp0reg           putcp0_10 $10
   setcp0reg           setcp0_10 $10
   getcp0reg           getcp0_11 $11
   putcp0reg           putcp0_11 $11
   setcp0reg           setcp0_11 $11
   getcp0reg           getstatus $12
   putcp0reg           putstatus $12
   setcp0reg           setstatus $12
   getcp0reg           getcp0_13 $13
   putcp0reg           putcp0_13 $13
   setcp0reg           setcp0_13 $13
   getcp0reg           getcp0_14 $14
```

```
   putcp0reg          putcp0_14 $14
   setcp0reg          setcp0_14 $14
   getcp0reg          getprid $15
   putcp0reg          putcp0_15 $15
   setcp0reg          setcp0_15 $15
   getcp0reg          getcp0_16 $16
   putcp0reg          putcp0_16 $16
   setcp0reg          setcp0_16 $16
   getcp0reg          getcp0_17 $17
   putcp0reg          putcp0_17 $17
   setcp0reg          setcp0_17 $17
   getcp0reg          getcp0_18 $18
   putcp0reg          putcp0_18 $18
   setcp0reg          setcp0_18 $18
   getcp0reg          getcp0_19 $19
   putcp0reg          putcp0_19 $19
   setcp0reg          setcp0_19 $19
   getcp0reg          getcp0_20 $20
   putcp0reg          putcp0_20 $20
   setcp0reg          setcp0_20 $20
   getcp0reg          getcp0_21 $21
   putcp0reg          putcp0_21 $21
   setcp0reg          setcp0_21 $21
   getcp0reg          getcp0_22 $22
   putcp0reg          putcp0_22 $22
   setcp0reg          setcp0_22 $22
   getcp0reg          getcp0_23 $23
   putcp0reg          putcp0_23 $23
   setcp0reg          setcp0_23 $23
   getcp0reg          getcp0_24 $24
   putcp0reg          putcp0_24 $24
   setcp0reg          setcp0_24 $24
   getcp0reg          getcp0_25 $20
   putcp0reg          putcp0_25 $20
   setcp0reg          setcp0_25 $20
   getcp0reg          getcp0_26 $26
   putcp0reg          putcp0_26 $26
   setcp0reg          setcp0_26 $26
   getcp0reg          getcp0_27 $27
   putcp0reg          putcp0_27 $27
   setcp0reg          setcp0_27 $27
   getcp0reg          getcp0_28 $28
   putcp0reg          putcp0_28 $28
   setcp0reg          setcp0_28 $28
   getcp0reg          getcp0_29 $29
   putcp0reg          putcp0_29 $29
   setcp0reg          setcp0_29 $29
   getcp0reg          getcp0_30 $30
   putcp0reg          putcp0_30 $30
   setcp0reg          setcp0_30 $30
   getcp0reg          getcp0_31 $31
   putcp0reg          putcp0_31 $31
   setcp0reg          setcp0_31 $31

   .globl   getstackp
   .ent getstackp
getstackp:
   addu     $2,$0,$29
   jr       $31
   nop
   .end     getstackp
```

A.2 Basic Operation

A.2.1 Header File

- File name: tx3927.h

```
/************************************************
 *  MODULE NAME: tx3927.h  TX 3927 REG. INFO.       *
 *  FUNCTION   : tx 3927 header                     *
 *  UPDATE     : 1998.07.15                         *
 ************************************************/
#ifndef __TX3927__
#define __TX3912
#include    "cosbd_27.h"

/************
 *  SDRAM   *
 ************/
/* ***** Channel Control Register ***** */
                                    /* Memory Type */
#define SDM_SDRAM    0x00000000     /* 0x00000000...SDRAM      */
#define SDM_DIMM     0x00040000     /* 0x00040000...DIMM FLASH */
#define SDM_SMROM    0x00080000     /* 0x00080000...SMROM      */
#define SDM_SGRAM    0x000C0000     /* 0x000C0000...SGRAM      */
                                    /* SDRAM Enable */
#define SDE_ENA      0x00020000     /* Enable */
#define SDE_DIS      0x00000000     /* Disable */
                                    /* BANK# */
#define SDB_0        0x00000000     /* Type 0 */
#define SDB_1        0x00010000     /* Type 1 */
                                    /* Address Mask Register */
#define SDAM         0x0000ffe0     /* */
                                    /* SDRAM Row Size */
#define SDRS_2048    0x00000000     /* 2048 Rows */
#define SDRS_4096    0x00000008     /* 4096 Rows */
#define SDRS_8192    0x00000010     /* 8192 Rows */
                                    /* DRAM Colum Size */
#define SDCS_256     0x00000000     /*  256 Word */
#define SDCS_512     0x00000002     /*  512 Word */
#define SDCS_1024    0x00000004     /* 1024 Word */
#define SDCS_2048    0x00000006     /* 2048 Word */
                                    /* SDRAM Memory Width */
#define SDMW_32      0x00000000     /* 0:32 Bit */
#define SDMW_16      0x00000001     /* 1:16 Bit */

/* ***** SDRAM Shard Timing Register ***** */
                                    /* BANK Cycle Time */
#define SDBC_5TCK    0x00000000     /*  5 tck(Reset) */
#define SDBC_6TCK    0x20000000     /*  6 tck */
#define SDBC_7TCK    0x40000000     /*  7 tck */
#define SDBC_8TCK    0x60000000     /*  8 tck */
#define SDBC_9TCK    0x80000000     /*  9 tck */
#define SDBC_10TCK   0xa0000000     /* 10 tck */
                                    /* SDACP Time */
#define SDACP_3TCK   0x00000000     /* 3 tck(Reset) */
#define SDACP_4TCK   0x08000000     /* 4 tck */
#define SDACP_5TCK   0x10000000     /* 5 tck */
#define SDACP_6TCK   0x18000000     /* 6 tck */
                                    /* Precharge Time */
#define SDP_2TCK     0x00000000     /* 2 tck(Reset) */
#define SDP_3TCK     0x04000000     /* 3 tck */
                                    /* RAS to CAS Delay */
#define SDCD_2TCK    0x00000000     /* 2 tck(Reset) */
#define SDCD_3TCK    0x02000000     /* 3 tck */
                                    /* Refresh Counter */
```

```
#define SDRC(cnt) ((cnt & 0x0000003F)  << 18)
                                        /* CAS Latency */
#define CASL_2TCK    0x00000000         /* 2 tck(Reset) */
#define CASL_3TCK    0x00020000         /* 3 tck */
                                        /* Data Read Bypass */
#define DRB_REG      0x00000000         /* (Reset) */
#define DRB_NORMAL   0x00010000         /* */
                                        /* Slow Write Burst */
                                        /* SGRAM Black Write */
                                        /* SGRAM Write Pre Bit */
                                        /* Refresh Period */
/* ***** FLASH Shard Timing Register ***** */
/* ***** SMROM Shard Timing Register ***** */
/* ***** SDRAM Command Register ***** */
                                        /* Channel Mask */
#define SDCMSK_CH0   0x00000010         /* Channel #0 */
#define SDCMSK_CH1   0x00000020         /* Channel #1 */
#define SDCMSK_CH2   0x00000040         /* Channel #2 */
#define SDCMSK_CH3   0x00000080         /* Channel #3 */
#define SDCMSK_CH4   0x00000100         /* Channel #4 */
#define SDCMSK_CH5   0x00000200         /* Channel #5 */
#define SDCMSK_CH6   0x00000400         /* Channel #6 */
#define SDCMSK_CH7   0x00000800         /* Channel #7 */
                                        /* Command */
#define SDC_NOP      0x00000000         /* NOP Command */
#define SDC_SDMOD    0x00000001         /* Set SDRAM Mode Register */
#define SDC_SMMOD    0x00000002         /* Set SMROM Mode Register */
#define SDC_PRE      0x00000003         /* Precharge All SDRAM Banks */
#define SDC_LPMOD    0x00000004         /* Enter Low Power Mode */
#define SDC_PDMOD    0x00000005         /* Enter Power Down Mode */
#define SDC_POWEXT   0x00000006         /* Exit Low Power/Power Down Mode */

/* ***** SGRAM Load Mask Register ***** */
/* ***** SGRAM Load Color Register ***** */
/* ************ *
 *  ROMC        *
 * ************ */
typedef struct{
   volatile int RCCR;                   /* ***** Channel Control Register ***** */
}ROMCC;

#define S_RCCR0      0x1fc3e200         /* Initial Data */
                                        /* ROM control Page Mode ROM Page Size */
#define RPS_NON      0x00000000         /* 4-Word */
#define RPS_4 W      0x00040000         /* 4-Word */
#define RPS_8 W      0x00080000         /* 8-Word */
#define RPS_16 W     0x000c0000         /* 16-Word */
                                        /* ROM control Page Read Mode Wait Time on
                                           channel0 */
#define RPWT_0       0x00000000         /* 0 Wait */
#define RPWT_1       0x00010000         /* 1 Wait */
#define RPWT_2       0x00020000         /* 2 Wait */
#define RPWT_3       0x00030000         /* 3 Wait */
                                        /* ROM control Wait Time on channel0 */
#define RWT(wc)      ((wc & 0xf)<< 12) /* wc : Wait Count */
                                        /* ROM control Channel Size on channel0 */
#define RCS_1 M      0x00000000         /* 1 Mbyte */
#define RCS_2 M      0x00000100         /* 2 Mbyte */
#define RCS_4 M      0x00000200         /* 4 Mbyte */
#define RCS_8 M      0x00000300         /* 8 Mbyte */
#define RCS_16 M     0x00000400         /* 16 Mbyte */
#define RCS_32 M     0x00000500         /* 32 Mbyte */
#define RCS_64 M     0x00000600         /* 64 Mbyte */
#define RCS_128 M    0x00000700         /* 128 Mbyte */
#define RCS_256 M    0x00000800         /* 256 Mbyte */
#define RCS_512 M    0x00000900         /* 512 Mbyte */
```

```
                                        /* ROM control BUS Size */
#define BUS16        0x00000080         /* ROM control 16bit width bus size on channel0
                                        */
#define BUS32        0x00000000         /* ROM control 32bit width bus size on channel0
                                        */
#define RPM          0x00000001         /* ROM control Page Mode on channel0 */


/************
 *  DMA     *
 ************/
/* ***** Master Control Register ***** */
#define S_MCR        0x00000000                /* Initial Data */
#define FIFOVC(vc) ((wc & 0x0002c000) << 14)   /* FIFO Valid Entry Count */
#define FIFWP(wp)  ((wp & 0x00003800) << 11)   /* FIFO Write Pointer */
#define FIFRP(rp)  ((rp & 0x00000700) << 8)    /* FIFO Read Pointer */
#define RSFIF        0x00000080                /* Reset FIFO */
#define FIFUM        0x00000078                /* FIFO Use Mask */
#define LE           0x00000004                /* Little Endian */
#define RRPT         0x00000002                /* Round Robin Priority */
#define MSTEN        0x00000001                /* Master Enable */


/* ***** Channel Control Register ***** */
#define S_CNTL       0x00000000                /* Initial Data */
#define CH_reset     0x01000000                /* Channel Reset */
#define RVBYTE       0x00800000                /* Reverse Byte */
#define ACKPOL       0x00400000                /* Acknowledge Polarity */
#define REQPL        0x00200000                /* Request  Polarity */
#define EGREQ        0x00100000                /* Edge Request */
#define CHDN         0x00080000                /* Chain Done */
                                               /* Done Control */
                                               /* External Request */
                                               /* Internal Request Delay   */
                                               /* Interrupt Enable on Error */
                                               /* Interrupt Enable on Chain Done */
                                               /* Interrupt Enable on Transfer Done */
                                               /* Chain Enable   */
                                               /* Transfer Active */
                                               /* Snop */
                                               /* Mixed Destination Increment */
                                               /* Mixed Source Increment */
                                               /* Transfer Size */
                                               /* Memory to I/O */
                                               /* One Address */


/* ***** Channel Status Register ***** */
#define S_STS        0x00000000                /* Initial Data */
#define WAITC(wc) ((wc & 0x0000ffc000) << 14)  /* Internal Wait Counter */
#define CHNACT       0x00000100                /* Channel Active */
#define ABCHC        0x00000080                /* Abnormal Chain Completion */
#define NCHNC        0x00000040                /* Normal Chain Completion */
#define NTRNFC       0x00000020                /* Normal Transfer Completion */
#define EXTDN        0x00000010                /* External Done Asserted */
#define CFERR        0x00000008                /* Configuration Error */
#define CHERR        0x00000004                /* Chain Bus Error */
#define DESERR       0x00000002                /* Destination Bus Error */
#define SORERR       0x00000001                /* Source Bus Error */

/* ***** Source Address Register ***** */
/* ***** Destination Address Register ***** */
/* ***** Chain Address Register ***** */
/* ***** Source & Destination Address Increment ***** */
/* ***** Count Register ***** */
```

```
/************
 * TIMMER   *
 ************/
/* ***** Timer Control Register ***** */
#define S_TMTCR      0x00000000        /* Initial Data */
#define TCE          0x00000080        /* Timer Count Enable */
#define CCDE         0x00000040        /* Counter Clock Divide Enable */
#define CRE          0x00000020        /* Counter Reset Enable */
#define ECES         0x00000008        /* External Clock Edge Select */
#define CCS          0x00000004        /* Counter Clock Select */
/* Timer Mode */
#define TMODE_WT     0x00000002        /* Watchdog Timer Mode */
#define TMODE_PG     0x00000001        /* Pulse Generator Mode */
#define TMODE_IT     0x00000000        /* Interval Timer Mode */
/* ***** Timer Interrupt Status Register ***** */
#define S_TMTISR     0x00000000        /* Initial Data */
#define TWIS         0x00000008        /* Timer Watchdog Interrupt Status */
#define TPIBS        0x00000004        /* Timer Pulse Generator Interrupt by CPRB
                                          Status */
#define TPIAS        0x00000002        /* Timer Pulse Generator Interrupt by CPRA
                                          Status */
#define TIIS         0x00000001        /* Timer Interval Interrupt Status */
/* ***** Compare Register A ***** */
#define S_TMCPRA     0x00000000        /* Initial Data */
/* ***** Compare Register B ***** */
#define S_TMCPRB     0x00000000        /* Initial Data */
/* ***** Interval Timer Mode Register ***** */
#define S_TMITMR     0x00000000        /* Initial Data */
#define TIIE         0x00008000        /* Timer Interval Interrupt Enable */
#define TZCE         0x00000001        /* Interval Timer Zero Clear Enable */
/* ***** Divider Register ***** */
#define S_TMCCDR     0x00000000        /* Initial Data */
#define CCD_1        0x00000000        /* Counter clock Divided by 1st power of 2 */
#define CCD_2        0x00000001        /* Counter clock Divided by 2nd power of 2 */
#define CCD_3        0x00000002        /* Counter clock Divided by 3rd power of 2 */
#define CCD_4        0x00000003        /* Counter clock Divided by 4th power of 2 */
#define CCD_5        0x00000004        /* Counter clock Divided by 5th power of 2 */
#define CCD_6        0x00000005        /* Counter clock Divided by 6th power of 2 */
#define CCD_7        0x00000006        /* Counter clock Divided by 7th power of 2 */
#define CCD_8        0x00000007        /* Counter clock Divided by 8th power of 2 */
/* ***** Pulse Generator Mode Register ***** */
#define S_TMPGMR     0x00000000        /* Initial Data */
#define TPIBE        0x00008000        /* Timer Pulse Generator Interrupt by CPRB
                                          Enable */
#define TPIAE        0x00004000        /* Timer Pulse Generator Interrupt by CPRA
                                          Enable */
#define FFI          0x00000001        /* Timer Flip-Flop Initial */
/* ***** Watchdog Timer Mode Register ***** */
#define S_TMWTMR     0x00000000        /* Initial Data */
#define TWIE         0x00008000        /* Timer Watchdog Interrupt Enable */
#define WDIS         0x00000080        /* Watchdog Timer Disable */
#define TWC          0x00000001        /* Timer Watchdog Clear */
/* *** Timer Read Register *** */
#define S_TMTRR      0x00000000        /* Initial Data */

/****************
 *  CHIP CONFIG *
 ****************/
/* *** Chip Configuration Register *** */
/* #define S_CCFG    0x0000            /* Initial Data */
#define CCFG_GHA     0x00040000        /* GBus Half Speed */
#define CCFG_TOF     0x00020000        /* TLB Off(On) */
#define CCFG_BEW     0x00010000        /* Bus Error on Write */
#define CCFG_WRS     0x00008000        /* Watchdog Timer for Reset/NMI */
#define CCFG_TOE     0x00004000        /* Timeout Enable for Bus Error */
#define CCFG_PAB     0x00002000        /* Internal or External PCI arbiter */
```

```
#define CCFG_PCI     0x00001000        /* PCI Clock Divider Control(1/2) */
#define CCFG_PSN     0x00000800        /* PCI Bus Request Snoop(Ena. Cache) */
#define CCFG_PRI     0x00000400        /* Select PCI Arbitration Priority */
                                       /* PLL Offset */
#define CCFG_PLO(off) ((off & 0x00000003) << 8)
                                       /* PLL Gain */
#define CCFG_PLG(gn) ((gn & 0x00000003) << 6)
                                       /* PLL Multiplier */
#define CCFG_PLM(adr) ((adr & 0x00000003) << 4)
#define CCFG_POF     0x00000008        /* PLL Off(ON) */
#define CCFG_END     0x00000004        /* Current Endian Setting of G-Bus */
#define CCFG_HLF     0x00000002        /* System Clock Half-Speed Mode */
#define CCFG_HLD     0x00000001        /* ACE Address Hold */
/* *** Pin Configuration Register *** */
#define S_PINCFG     0x00000000        /* Initial Data */
#define S_PIN_ENA    0x0fffffff        /* Enable All Pins */
#define PINC_SCE     0x08000000        /* System Clock Enable */
/* SDRAM Clock Enable */
#define SDCLK_4      0x04000000        /* SDCLK[4] Enable */
#define SDCLK_3      0x02000000        /* SDCLK[3] Enable */
#define SDCLK_2      0x01000000        /* SDCLK[2] Enable */
#define SDCLK_1      0x00800000        /* SDCLK[1] Enable */
#define SDCLK_0      0x00400000        /* SDCLK[0] Enable */
/* SDRAM Clock Enable */
#define PCICLK_3     0x00200000        /* PCICLK[3] Enable */
#define PCICLK_2     0x00100000        /* PCICLK[2] Enable */
#define PCICLK_1     0x00080000        /* PCICLK[1] Enable */
#define PCICLK_0     0x00040000        /* PCICLK[0] Enable */
#define PINC_SCS     0x00020000        /* Select DMA/SDCS_CE Function */
#define PINC_SDF     0x00010000        /* Select DSF Function */
/* Select SIO Control Pins */
#define SELSIOC_1    0x00008000        /* Select the Function CTS[1]/PIO[2]
                                          and RTS[1]/PIO[1]/DSF. */
#define SELSIOC_0    0x00004000        /* Select the Function CTS[0]/INT[5]
                                          and RTS[0]/INT[4] */
/* Select SIO/PIO function Select */
#define SELSIO_1     0x00002000        /* Select the Function RXD[1]/PIO[6]
                                          and TXD[1]/PIO[5] */
#define SELSIO_0     0x00001000        /* Select the Function RXD[0]/PIO[4]
                                          and TXD[0]/PIO[3] */
/* Select Timer PIO function Select */
#define SELTMR_2     0x00000800        /* Select the Function */
#define SELTMR_1     0x00000400        /* Select the Function */
#define SELTMR_0     0x00000200        /* Select the Function */
/* Select DMADONE/PIO */
#define SELDONE      0x00000100        /* Select the Function */
/* Internal/External DMA Source Connection */
#define INTDMA_3     0x00000080        /* DMAREQ/ACK[3] connect to SIO[1] TXREQ/ACK */
#define INTDMA_2     0x00000040        /* DMAREQ/ACK[3] connect to SIO[0] TXREQ/ACK */
#define INTDMA_1     0x00000020        /* DMAREQ/ACK[3] connect to SIO[1] RXREQ/ACK */
#define INTDMA_0     0x00000010        /* DMAREQ/ACK[3] connect to SIO[1] RXREQ/ACK */
/* Select DMA/PIO/TIMER Function */
#define DMAREQ_3     0x00000008        /* Used to select the function of
                                          DMAREQ[3]/PIO[15]/TIMER[1]
                                          DMAACK[3]/PIO[14]/TIMER[0] */
#define DMAREQ_2     0x00000004        /* Used to select the function of
                                          DMAREQ[2]/PIO[13]
                                          DMAACK[2]/PIO[12] */
#define DMAREQ_1     0x00000002        /* Used to select the function of
                                          DMAREQ[1]/PIO[11]
                                          DMAACK[1]/PIO[10] */
#define DMAREQ_0     0x00000001        /* Used to select the function of
                                          DMAREQ[0]/PIO[9] DMAACK[0]/PIO[8] */
/* *** Power Down Control Register *** */
/* interrupt signals wake from power down mode */
#define PDNMSK_NMI   0x00800000
```

```
#define PDNMSK_INT5  0x00400000
#define PDNMSK_INT4  0x00200000
#define PDNMSK_INT3  0x00100000
#define PDNMSK_INT2  0x00080000
#define PDNMSK_INT1  0x00040000
#define PDNMSK_INT0  0x00020000
/* Power Down Trigger */
#define PDN          0x00010000
/* PIO Output Direction */
#define PIO_0        0x00000001
#define PIO_1        0x00000002
#define PIO_2        0x00000004
#define PIO_3        0x00000008
#define PIO_4        0x00000010
#define PIO_5        0x00000020
#define PIO_6        0x00000040
#define PIO_7        0x00000080
#define PIO_8        0x00000100
#define PIO_9        0x00000200
#define PIO_10       0x00000400
#define PIO_11       0x00000800
#define PIO_12       0x00001000
#define PIO_13       0x00002000
#define PIO_14       0x00004000
#define PIO_15       0x00008000
/* PIO Open Drain Mode */
#define ALL_TOTEM    0x0000ffff        /* PIO All Totem-pole Mode */

/************
 *  SIO     *
 ************/
/* *** Line Control Register *** */
#define S_SILCR      0x4000            /* Initial Data */
#define SILC_RWUB    0x8000            /* Wake Up Bit for Receive */
#define SILC_TWUB    0x4000            /* Wake Up Bit for Transmit */
#define SILC_UODE    0x2000            /* SOUT Open Drain Enable */
#define SILC_HS_CTS  0x0100            /* Hand Shake Enable(CTS) */
#define SILC_SCS_I   0x0000            /* SIO Clock Select:
                                          T0 (Internal System Clock) */
#define SILC_SCS_B   0x0020            /* SIO Clock Select: Baud Rate Generator */
#define SILC_SCS_E   0x0040            /* SIO Clock Select: External Clock (SCLK) */
#define SILC_UPE_ODD 0x0010            /* UART Parity Odd */
#define SILC_UPE_EVN 0x0008            /* UART Parity Enable */
#define SILC_UPE_NON 0x0000            /* UART Parity Non */
#define SILC_U2STOP  0x0004            /* UART 2 Stop Bit */
#define SILC_UM_8    0x0000            /* UART MODE: 8-bit Data Length */
#define SILC_UM_7    0x0001            /* UART MODE: 7-bit Data Length */
#define SILC_UM_M8   0x0002            /* UART MODE: Multidrop 8-bit Data Length */
#define SILC_UM_M7   0x0003            /* UART MODE: Multidrop 7-bit Data Length */
/* *** DMA/Interrupt Control Register *** */
#define S_SILSR      0x0000            /* Initial Data */
#define TDR          0x8000            /* Tran. DMA Request(Enable) */
#define RDR          0x4000            /* Rsv.  DMA Request(Enable) */
#define TIR          0x2000            /* Tran. Interrupt Request(Enable) */
#define RIR          0x1000            /* Rsv.  Interrupt Request(Enable) */
#define SPIR         0x0800            /* Sp.   Interrupt Request(Enable) */
/* CTSS Status active condition */
#define CTS_DIS      0x0000            /* Disable */
#define CTS_UEG      0x0200            /* Rising Edge od CTS */
#define CTS_DEG      0x0400            /* Falling Edge of CTS */
#define CTS_BEG      0x0600            /* Both Edge of CTS */
/* Status Change Interrupt Enable */
#define STIE_OVE     0x0020            /* Overrun Error Status */
#define STIE_CTS     0x0010            /* CTS Terminal Status */
#define STIE_RBK     0x0008            /* Receive Break */
#define STIE_TRD     0x0004            /* Transmit Ready */
```

```
#define STIE_TXA     0x0002            /* Transmit All Sent */
#define STIE_UBK     0x0001            /* UART Break Detect */
/* *** DMA/Interrupt Status Register *** */
#define S_SIDISR     0x4100            /* Initial Data */
#define SIDI_UBK     0x8000            /* UART Break Detect */
#define SIDI_UVA     0x4000            /* UART Available Data */
#define SIDI_UFE     0x2000            /* UART Frame Error */
#define SIDI_UPE     0x1000            /* UART Parity Error */
#define SIDI_UOE     0x0800            /* UART Overrun Error */
#define SIDI_ERI     0x0400            /* Error Interrupt */
#define SIDI_TOU     0x0200            /* Time Out */
#define SIDI_TEMP    0x0100            /* Transmit DMA/Interrupt Status */
#define SIDI_RFUL    0x0080            /* Receive DMA/Interrupt Status */
#define SIDI_STI     0x0040            /* Status Change Interrupt Status */
/* *** Status Change Interrupt Status Register *** */
#define S_SICISR     0x0006            /* Initial Data */
#define SICI_OVE     0x0020            /* Overrun Error Detect */
#define SICI_CTS     0x0010            /* CTS Terminal Status(Signal High) */
#define SICI_RBK     0x0008            /* Receive Break */
#define SICI_TRD     0x0004            /* Transmit Ready */
#define SICI_TXA     0x0002            /* Transmit All Sent */
#define SICI_UBK     0x0001            /* UART Break Detect */
/* *** FIFO Control Register *** */
#define S_SIFCR      0x0000            /* Initial Data */
#define SISF_RDL     0x0010            /* Receive FIFO DMA Request Trigger Level */
#define SISF_TDL     0x0008            /* Transfer FIFO DMA Request Trigger Level */
#define SISF_TFR     0x0004            /* Transmit FIFO Reset */
#define SISF_RFR     0x0002            /* Receive FIFO Reset */
#define SISF_FRS     0x0001            /* FIFO Reset Enable */
/* *** FLOW Control Register *** */
#define S_SIFLCR     0x0182            /* Initial Data */
#define SIFL_RCS     0x1000            /* RTS Control Select */
#define SIFL_TES     0x0800            /* Transmit Enable Select */
#define SIFL_RTS     0x0200            /* RTS Software Control */
#define SIFL_RSE     0x0100            /* Receive Serial Data Enable */
#define SIFL_TSE     0x0080            /* Transmit Serial Data Enable */
                                       /* RTS Trigger Level (1 to 15)*/
#define SIFL_RTSTL(lvl) ((lvl & 0xf) << 1)
#define SIFL_TBK     0x0001            /* Break Transmit(Enable) */
/* *** Baud Rate Generator Clock *** */
#define S_SIBGR      0x03FF            /* Initial Data */
#define BCLK_FC4(n)  (0x0000|(n&0xff)) /* Select Prescalar Output T0(fc/4) */
#define BCLK_FC16(n) (0x0100|(n&0xff)) /* Select Prescalar Output 2(fc/16) */
#define BCLK_FC64(n) (0x0200|(n&0xff)) /* Select Prescalar Output T4(fc/64) */
#define BCLK_FC256(n) (0x0300|(n&0xff)) /* Select Prescalar Output T6(fc/256) */

/************************
 *  Data Format Table   *
 ************************/
/**************
 * DMA        *
 **************/
typedef struct {
   volatile int CHAR;       /* ***** Chained Address Register ***** */
   volatile int SAR;        /* ***** Source Address Register ***** */
   volatile int DAR;        /* ***** Destination Address Register ***** */
   volatile int CTR;        /* ***** Count Register ***** */
   volatile int SAI;        /* ***** Source Address Increment Register ***** */
   volatile int DAI;        /* ***** Destination Address Increment Register ***** */
   volatile int CCR;        /* ***** Control Register ***** */
   volatile int CSR;        /* ***** Status Register ***** */
 } DMA;
```

```
/**************
 *  INTR      *
 **************/
typedef struct {
   volatile int IRDER;      /* ***** Interrupt Detect Enable Register ***** */
   volatile int IRDMR[2];   /* ***** Interrupt Detect Mode Register ***** */
   int      tmp1;
   volatile int IRILR[8];   /* ***** Interrupt Level Register ***** */
   volatile int dummy23[4];
   volatile int IRIMR;      /* ***** Interrupt Mask Register ***** */
   volatile int dummy24[7];
   volatile int IRSCR;
                            /* ***** Interrupt Status Control Register ***** */
   volatile int dummy25[7];
   volatile int IRSSR;      /* ***** Interrupt Source Register ***** */
   volatile int dummy26[7];
   volatile int IRCSR;      /* ***** Interrupt Current Register ***** */
   char     tmp9[0xd000-0xc0a4];
 } INTR;

/**************
 *  PCI       *
 **************/
typedef struct{ /* PCI Configuration Hader */
   int      DID;
   int      PCISTAT; /* Status reg.*/
#define DECPE        0x80000000        /*Detected Parity Error(write clear)*/
#define SIGSE        0x40000000        /*Signaled System Error(write clear)*/
#define RECMA        0x20000000        /*Received Master Abort(write clear)*/
#define RECTA        0x10000000        /*Received Target Abort(write clear)*/
#define SIGTA        0x08000000        /*Signaled Target Abort(write clear)*/
#define PERPT        0x01000000        /*Parity Error Reported(write clear)*/
#define FBBCP        0x00800000        /*Fast Back-to-Back Capabe*/
#define USPCP        0x00200000        /*USPCP bit field*/
/* Command reg.*/
#define FBBEN        0x00000200        /*Fast Back-to-Back Enable*/
#define SEEN         0x00000100        /*SERR Enable*/
#define PEREN        0x00000040        /*PERR Enable*/
#define MWIEN        0x00000010        /*Memory Write and Invalidate Enable*/
#define SCREC        0x00000008        /*Special Cycle Recognition*/
#define MEN          0x00000004        /*Master Enable*/
#define MACEN        0x00000002        /*Memory Access Enable*/
#define IACEN        0x00000001        /*I/O Access Enable*/
   int      CC;
   int      INF;
   int      IOBA,MBA,BA2,BA3,BA4,BA5;  /* chenge MBA<->IOBA tanka 990429 */
   int      non[2];
   int      EXTIOBA;
   int      non2[2];
   int      ML;
}PCI_CONF;

typedef struct{ /* PCI initiator Configuration */
   int      IC;             /* Initiator Control register */
   int      ISTAT;          /* initiator Status register */
   int      IIM;            /* Initiator Interrupt Mask register */
   int      RRT;            /* Retry/Reconnect Timer Register */
   int      tmp1[3];
   int      IPBMMAR;        /* Initiator Local bus IO Mapping register */
   int      IPBIOMAR;       /* Initiator Local bus Memory Mapping register */
   int      ILBMMAR;        /* Initiator PCI bus IO Mapping register */
   int      ILBIOMAR;       /* Initiator PCI bus Memory Mapping register */
   char     tmp2[0x90-0x6c];
 }PCI_ICONF;

typedef struct{ /* PCI target Configuration */
```

```
   int      TC;             /* Target Control register */
   int      TSTAT;          /* Target status register */
   int      TIM;            /* Target Interrupt Mask register */
   int      TCCMD;          /* Target Current Command register */
   int      PCIRRT;         /* PCI Read Retry Tag register */
   int      PCIRRT_CMD;     /* PCI Read Retry Timer Command Register */
   int      PCIRRDT;        /* PCI Read Retry Discard Timer register */
   int      tmp1[3];
   int      TLBOAP;         /* Target Local bus Output FIFO Address Pointer */
   int      TLBIAP;         /* Target Local bus Input FIFO Address Pointer */
   int      TLBMMA;         /* Target Local bus Memory Mapping Address register */
   int      TLBIOMA;        /* Target Local bus IO Mapping Address register */
   int      SC_MSG;         /* Special Cycle Message register */
   int      SC_BE;          /* Special Cycle Byte Enable register */
   int      TBL;            /* Target Burst Length */
   char     tmp2[0x100-0xd4];
}PCI_TCONF;

typedef struct{ /* PCI bus Arbiter/Park */
   int      REQ_TRACE;      /* Request Trace register */
   int      PBAPMC;         /* PCI Bus Arbiter/Park Master Control register */
#define     BARST    0x04   /* Reset Bus Arbiter */
#define     BAENA    0x02   /* Enable Bus Arbiter */
#define     MBCENA   0x01   /* Broken Master Check Enable */

   int      PBAPMS;         /* PCI Bud Arbiter/Park Master Status register */
   int      PBAPMIM;        /* PCI Bus Arbiter/Park Master Interrupt Mask register */
   int      BM;             /* Broken Master register */
   int      CPCIBRS;        /* Current PCI bus Request register */
   int      CPCIBGS;        /* Current PCI bus Grant Status register */
   int      PBACS;          /* Current PCI bus Arbiter Status register */
}PCI_EXT;

typedef struct{/* PCI local bus */
   int      IOBAS;          /* Target IO Base Address Register */
   int      MBAS;           /* Target Memory Base Address Register */

   int      LBC;            /* Local bus Control register */
#define HRST         0x00000800        /*Hard Reset */
#define SRST         0x00000400        /*Soft Reset */
#define EPCAD        0x00000200        /*External PCI Configuration Access Disable */
#define MSDSE        0x00000100        /*Memory Space Dynamic Swap Enable */
#define CRR          0x00000080        /*Configuration Registers Ready for Access */
#define ILMDE        0x00000040        /*Initiator Local Bus Memory Address Space
                                         Decoder Enable */
#define ILIDE        0x00000020        /*Initiator Local Bus I/O Address Space Decoder
                                         Enable */
#define TPIIC        0x00000010        /*Test PCI I/O Buffer Idd Current */

   int      LBSTAT;         /* Local bus Status register */
   int      LBIM;           /* Interrupt Mask register */
   int      PCISTATIM;      /* Interrupt Mask Status register */
#define     LS_PERR  0x20
#define     LS_SERR  0x10
#define     LS_GERR  0x08
#define     LS_IAS   0x04
#define     LS_RST   0x02

   int      ICAR;           /* Initiator Configuration Address register */
   int      ICDR;           /* Initiator Configuration Data register */

   int      IIADP;          /* Initiator Interrupt Acknowledge Data Port register */
   int      ISCDP;          /* Initiator Special Cycle Data Port register */

   int      MMAS;           /* Initiator Memory Mapping Address register */
   int      IOMAS;          /* Initiator IO Mapping Address register */
```

```
   int      IPCIADDR;       /* Initiator Indirect Address register */
   int      IPCIDATA;       /* Initiator Indirect Data register */
   int      IPCICBE;        /* initiator Indirect Command/Byte Enable register */
}PCI_LSP;


/**************
 *  TMR       *
 * ************/
typedef struct {
   volatile int TMTCR;      /* Timer Control Register */
   volatile int TMTISR;     /* Timer Interrupt Status Register */
   volatile int TMCPRA;     /* Compare Register A */
   volatile int TMCPRB;     /* Compare Register B */
   volatile int TMITMR;     /* Interval Timer Mode Register */
   volatile int dummy1[3];
   volatile int TMCCDR;     /* Clock Divider Register */
   volatile int dummy2[3];
   volatile int TMPGMR;     /* Pulse Generator Mode Register */
   volatile int dummy3[3];
   volatile int TMWTMR;     /* Watchdog Timer Mode Register */
   volatile int dummy4[43];
   volatile int TMTRR;      /* Timer Read Register */
   volatile int dummy5[3];
} TMR;

/* *************
 *  SIO        *
 * *************/
typedef struct { /* little */
   volatile int SILCR;
   volatile int SIDICR;
   volatile int SIDISR;
   volatile int SISCISR;
   volatile int SIFCR;
   volatile int SIFLCR;
   volatile int SIBGR;
   volatile int SITFIFO;
   volatile int SIRFIFO;
   char dumy[0x100-0x24];
} SIO;

#if 0
/* 0xfffe8000  * SDRAM Channel Control Register */
   volatile int SDRAMC[8];  /* 0x1fc00000 (0x00000000)...Base Address */
#endif

/***********************
 * TX3927 Register Map *
 ***********************/
typedef struct {
/* 0xfffe8000  * SDRAM Channel Control Register */
   volatile int SDRAMC[8];
   volatile int SDCTR1;     /* SDRAM Shard Timing Register */
   volatile int SDCTR2;     /* FLASH Shard Timing Register */
   volatile int SDCTR3;     /* SMROM Shard Timing Register */
   volatile int SDCCMD;     /* SDRAM Command Register */
   volatile int SDCSMRS1;   /* SGRAM Load Mask Register    */
   volatile int SDCSMRS2;   /* SGRAM Load Color Register */
   char tmp1[0x9000-0x8000-0x38];
/* 0xfffe9000  * ROM Channel Control Register *** */
   ROMCC ROMC[8];
   Char     tmp2[0xb000-0x9000-sizeof(ROMCC)*8];
/* 0xfffeb000  * DMA Register *** */
   DMA DREG[4];             /* Data Buffer Register */
```

```
   volatile int DBR[8];     /* Temporary Data Holding Register    */
   volatile int TDHR;       /* Muster Control Register */
   volatile int DMACR;
   char     tmp3[0xc000-0xb0a8];
/* 0xfffec000 <<Interrupt Controller>>*/
   INTR IREG;

/* 0xfffed000  * << PCI Controller>> */
   PCI_CONF     pci_conf;
   PCI_ICONF    pci_iconf;
   PCI_TCONF    pci_tconf;
   PCI_EXT      pci_ext;
   PCI_LSP      pci_lsp;
   Char         tmp10[0xe000-0xd15c];

/* 0xfffee000   Chip Configuration Register *** */
   volatile int CCFG;       /* Chip Revision ID Register    */
   volatile int CREVID;     /* Pin Configuration Register    */
   volatile int PINCFG;     /* Timeout Error Register       */
   volatile int TMOUTERR;   /* Power Down Control Register    */
   volatile int PDNCTL;
   volatile int dummy29[1019];
/* 0xfffef000   Timer Register *** */
   TMR TREG[3];
/* 0xfffef300   SIO Register *** */
   SIO SREG[2];
/* 0xfffef500   PIO Register *** */
   volatile int PIODO;      /* PIO Output Register */
   volatile int PIODI;      /* PIO Input Register */
   volatile int PIODIR;     /* PIO Direction Control Register */
   volatile int PIOOD;      /* PIO Open Drain Control Register */
   volatile int PIOFLAG0;   /* PIO Flag Register */
   volatile int PIOFLAG1;   /* PIO Flag Register */
   volatile int PIOPOL;     /* PIO Flag Polarity Control Register */
   volatile int PIOINT;     /* PIO Interrupt Control Register */
   volatile int PIOMASKCPU; /* CPU Interrupt Mask Register */
   volatile int PIOMASKEXT; /* External Interrupt Mask Register */
   char     tmp12[0x10000-0xf528];
}TX3927;

/* *** Physical Address *** */
#define CPUReg ((TX3927 *)0xfffe8000)

/* ***** End Of File "tx3927.h" ***** */
#endif
```

A.2.2 Start Routine

This section shows a simple start routine for the TX3904, a member of the TX39 family, that initializes the processor following power-on reset and then transfers control to the user's main routine. The start routine is placed at the Reset exception vector address when your application is ROMed.

The start routine performs the following tasks:

- Determines whether the exception was caused by NMI or RESET*.
- Initializes the memory controller (boot_initreg).
- Clears the Bss section (using boot_memset).
- Copies the initial program data in the data section from ROM to RAM (using boot_memcpy).
  * Memory writes are followed by a routine which ensures that the external memory is synchronized with the internal caches (boot_synccache).
- Enables the caches.
- Initializes the gp, sp and pid_base registers.
- Jumps to the main routine.

The exception vectors other than the reset vector are implemented as dummy (dummy software loop).

The TX39 does not require any settings for the processor core to become operational. (The COLDRESET input disables the caches and TLB.) The above tasks are most commonly used in applications comprised of ELF object files; the coding style largely depends on the compiler used.

In TX3927, memory controller settings and cache operations have been changed from the TX3904. Refer to the appropriate chapters in this manual.

Note: The following start routine example was generated using a GHS compiler; so the resulting code uses symbols and tables that have been automatically generated by the GHS linker. Specifically, the following symbols starting with __ghs are used:

__ghsbinfo_clear, __ghseinfo_clear:	Start and end addresses of initialization section table
__ghsbinfo_copy, __ghseinfo_copy:	Start and end addresses of initial data section table
__ghsbegin_sdabase:	Lowest address of the SDA section
__ghsend_stack:	Highest address of the stack space

You cannot use these symbols with other compilers, and other versions of the GHS compilers may use symbols differently. It is recommended to refer to the crt0 source program that comes with the GHS compiler.

Cygnus' GNUPro does not generate the above symbols automatically. The user can, however, write a linker script file to define arbitrary symbols and use them in the same way.

- File name: Boot.mip

```
# $Id$
#
# ROM Boot Routine for TX3904
# Copyright(c) 1998 TOSHIBA Corp.
#
# Depend on GHS Cross MIPS Compiler ver.1.8.8
```

```
 #
   .file "boot.mip"
   .section ".boot",.text
   .set noreorder

$status=$12
$config=$3
/* $k0=$26 */
/* $k1=$27 */
 # ------------------------------------------------------------------------
 # Exception Vector
 # ------------------------------------------------------------------------
   .globl ResetVector
ResetVector:                # Reset and NMI vector
   J       boot_main
   nop
   .word  Revision  /* Required by Test Monitor on JMR-TX3904 */
Revision:
   .byte  "ApplicationName ver.0.01"

   .offset 0x100
UtlbExcVector:
1: b       1b
   nop

   .offset 0x180
GeneralExcVector:
1: b       1b
   nop

   .offset 0x200
DebugExcVector:
1: b       1b
   nop

   .offset 0x300
 # ------------------------------------------------------------------------
 # Boot Main Routine
 # ------------------------------------------------------------------------
boot_main:
    # Check NMI
   mfc0    $k1,$status
   nop
   srl     $k1,$k1,16
   andi    $k1,$k1,0x1       # select NMI bit
   beq     $k1,$0,1f         # go ahead if not NMI
   nop
   jal     NMIHandler
   nop
1:
    # Initialize ROMC and RAMC and Etc.
   jal     boot_initreg
   nop

    # Copy Rom Image into Ram and Sync Cache
    # Clear bss area with zero
   lui     $16,%hi(__ghsbinfo_clear)
   addiu   $16,$16,%lo(__ghsbinfo_clear)
   lui     $17,%hi(__ghseinfo_clear)
   addiu   $17,$17,%lo(__ghseinfo_clear)
   b       2f
   nop                       # delay slot
1: addi    $16,$16,4
   lw      $5,0($16)
   addi    $16,$16,4
   lw      $6,0($16)
```

```
    jal     boot_memset
    addi    $16,$16,-8        # delay slot
    lw      $4,0($16)
    addi    $16,$16,8
    lw      $5,0($16)
    jal     boot_synccache
    addi    $16,$16,4         # delay slot
2:  bne     $16,$17,1b
    lw      $4,0($16)         # delay slot

     # Copy Rom to Ram
    lui     $16,%hi(__ghsbinfo_copy)
    addiu   $16,$16,%lo(__ghsbinfo_copy)
    lui     $17,%hi(__ghseinfo_copy)
    addiu   $17,$17,%lo(__ghseinfo_copy)
    b       2f
    nop                       # delay slot
1:  addi    $16,$16,4
    lw      $5,0($16)
    addi    $16,$16,4
    lw      $6,0($16)
    jal     boot_memcpy
    addi    $16,$16,-8        # delay slot
    lw      $4,0($16)
    addi    $16,$16,8
    lw      $5,0($16)
    jal     boot_synccache
    addi    $16,$16,4         # delay slot
2:  bne     $16,$17,1b
    lw      $4,0($16)         # delay slot

     # Cache ON
    mfc0    $k1,$config
    nop
    ori     $k1,$k1,0x30
    mtc0    $k1,$config
    j       3f
    nop
3:
     # Set Global Pointer
    lui     $gp, %hi(__ghsbegin_sdabase)     # set gp
    addiu   $gp, $gp, %lo(__ghsbegin_sdabase)
    addiu   $gp, $gp, 0x4000  # Add 32K to $gp
    addiu   $gp, $gp, 0x4000

     # Clear PIC Pointer($23) with zero
    addi    $23,$0,$0

     # Set Stack Pointer
    lui     $sp, %hi(__ghsend_stack)         # set sp
    addiu   $sp, $sp, %lo(__ghsend_stack)

     # Jump into Entry Point of Program
    lui     $4,%hi(main)
    addiu   $4,$4,%lo(main)
    jr      $4
    nop

 # ------------------------------------------------------------------------
 # SyncCache Routine
 # void boot_synccache(void* addr, unsigned size)
 # ------------------------------------------------------------------------
boot_synccache:
    mfc0    $2,$config
     li     $3,0xffffffcf
    and     $3,$2,$3
```

```
   mtc0    $3,$config
   j       1f
   nop
1: add     $5,$5,$4
2: bge     $4,$5,3f
   nop
   .align 16
   cache   0,0($4)
   nop
   .align 16
   cache   1,0($4)
   j       2b
   addi    $4,$4,4
3: mtc0    $2,$config
   jr      $31
   nop

 # ------------------------------------------------------------------------
 # NMI handler
 # ------------------------------------------------------------------------
NMIHandler:
1: b       1b
   nop
```

A.2.3 Initializing the Memory Controller (SDRAMC)

This section demonstrates how to initialize the SDRAMC on the TX3927 evaluation board, the JMR-TX3927, from Toshiba Information Systems Corporation.

The first several instructions form a timer-loop in order to wait for the SDRAM to be ready by way of precaution.

The 32-bit SDRAM bank is formed using two TC59S6416BFTL-80 1Mx16-bitx4-bank SDRAMs for a total of 16 MB.

The SDRAM memory space begins at physical address 0.

Following are excerpts from the TC59S6416BFTL-80 datasheet that require initialization in the SDRAMC:

Page1: 64 ms, 4K-cycle refresh (refreshing 4096 rows) ---------- (a)
Page3: 4096 rows × 256 colums × 16 bits ---------- (b)
Page6: tRC 64 ns (min) ---------- (c)
tRAS 48 ns (min), 100000 ns (max) ---------- (d)
tRCD 20 ns (min) ---------- (e)
tRP 20 ns (min) ---------- (f)
tWR 10 ns (min) (CL=2) ---------- (g)
Page 29 Address inputs (Row addressing: A0-A11, column addressing: A0-A7) ---------- (b)

When the TX3927 operates at 133 MHz, the SDRAM clock frequency is 66 MHz (with a 15-ns clock cycle).

The SDRAMC registers are programmed as follows in the sample code:

SDCCR0
SDBA0 = 0x0000
SDM0 = 0 (SDRAM)
SDE0 = 1 (Enable)
SDBS0 = 1 (4 banks)
SDAM0 = 0x000e (16 MB)
SDRS0 = 1 (4096 rows) ------- (b)
SDCS0 = 0 (256 columns) ------- (b)
SDMW0 = 0 (32-bit width)

SDCTR1
SDBC1 = 0 (5 tCK = 75 ns) ------- (c)
SDACP1 = 1 (4 tCK = 60 ns) ------- (d)
SDP1 = 0 (2 tCK = 30 ns) -------- (f)
SDCD1 = 0 (2 tCK = 30 ns) ------- (e)
WRT1 = 0 (1 tCK = 15 ns) ------- (g)
SDRC1 = 0 (Counter not used)
CASL1 = 0 (2 tck) A CAS latency of 2 is sufficient because the TX3927 bus clock runs at 66 MHz.
DRB1 = 1 (TX3927) SDRAMC-specific setting. In principle, set this bit when the AC timing is tight. SDRAMs that don't function

properly with DRB1 = 0 could function with DRB1 = 1, but the opposite is not possible; so a DRB1 value of 1 is used here.

SWB1 = 0 (Slow write burst is usually not used.)

BW1 = 0 This bit is not used for SDRAM.

WpB1 = 0 This bit is not used for SDRAM.

SDRP1 = 0x400 (15.5 μs)

64 ms / 4096 = 15.625 μs (time required to refresh a row)

15.625 μs / 15 ns = 1041.7 = 0x411 (CLK)

* Refresh cycles have sufficient timing slack. No strict setting is required.

SDCCMD

Command #1

SDCMSK3 = 1 (Channel 0)

SDCMD3 = 3 (Precharge all SDRAM banks)

Command #2

SDCMSK3 = 1 (Channel 0)

SDCMD3 = 1 (Write to SDRAM Mode register)

- File name: sdramc.s

```
/*Wait for about 200us to make SDRAM ready when core speed 200Mhz */
/*      li     $9, 4000000          /*TX3927*/
        li     $9, 800000
        li     $8, 0x00000000
sdram:
        bne    $9, $8, sdram
        addiu  $8, $8, 1

        /*initialize SDRAMC*/
        la     $8, 0xfffe8000       /*SDRAM0*/
        li     $9, 0x000300e8
        sw     $9, 0x00($8)

        la     $8, 0xfffe8020       /*shared Timing Reg*/
        li     $9, 0x08010400
        sw     $9, 0x00($8)

        la     $8, 0xfffe802c       /*command Reg*/
        li     $9, 0x00000013       /*all refresh */
        sw     $9, 0x00($8)

        li     $9, 0x00000011       /* mode set */
        sw     $9, 0x00($8)
```

A.2.4 Interrupt Handling Routines

This section shows a simple routine for interrupt handling. This routine is not for any specific device; it is intended to demonstrate how to determine the cause of interrupts and control branches.

The TX3927 interrupt handling consists of two stage: the TX39 core (CPU) exception/interrupt processing and the Interrupt Controller processing. Refer to the TX39/H2 Core Architecture manual and the Interrupt Controller (IRC) chapter in this manual.

The following routine assumes that an array named swIntVector already contains the addresses for each exception/interrupt handler (written in C).

The swIntVector array must be initialized as follows:

swIntVector[0]-[12]: Corresponds to the ExCode values of 0 to 12 in the Cause register.

swIntVector[13]-[14]: Corresponds to software interrupts.

swIntVector[15]-[30]: Corresponds to the IP[4:0] field (0-15) of the Cause register encoded in the Interrupt Controller.

The following shows an example of an swIntVector-setting routine and an interrupt-enabling function:

- File name: intttx3927.c

```
/*********************************************************/
/* TX3927 Interrupt information                          */
/*********************************************************/

typedef int (*VINTFUNC)(void);

#define     CPUEXC_BASE       0
#define      INTNO_Int        0
#define      INTNO_Mod        1
#define      INTNO_TLBL       2
#define      INTNO_TLBS       3
#define      INTNO_AdEL       4
#define      INTNO_AdES       5
#define      INTNO_IBE        6
#define      INTNO_DEB        7
#define      INTNO_Sys        8
#define      INTNO_Bp         9
#define      INTNO_RI         10
#define      INTNO_CpU        11
#define      INTNO_Ov         12
#define      INTNO_SW0        13
#define      INTNO_SW1        14
#define     TX3927INT_BASE    (CPUEXC_BASE+15)

#define      INTNO_TMR2       (TX3927INT_BASE+15)
#define      INTNO_TMR1       (TX3927INT_BASE+14)
#define      INTNO_TMR0       (TX3927INT_BASE+13)
#define      INTNO_NU0        (TX3927INT_BASE+12)
#define      INTNO_NU1        (TX3927INT_BASE+11)
#define      INTNO_PCI        (TX3927INT_BASE+10)
#define      INTNO_PIO        (TX3927INT_BASE+9)
#define      INTNO_DMA        (TX3927INT_BASE+8)
#define      INTNO_SIO1       (TX3927INT_BASE+7)
#define      INTNO_SIO0       (TX3927INT_BASE+6)
#define      INTNO_INT5       (TX3927INT_BASE+5)     /* JMR-TX3927: m pin          */
#define      INTNO_INT4       (TX3927INT_BASE+4)     /*           : m pin          */
```

```
#define      INTNO_INT3       (TX3927INT_BASE+3)     /*          : 100M,10M ether  */
#define      INTNO_INT2       (TX3927INT_BASE+2)     /*          : ISA             */
#define      INTNO_INT1       (TX3927INT_BASE+1)     /*          : IOC             */
#define      INTNO_INT0       (TX3927INT_BASE+0)     /*          : PCI INTA,C 10M? */

#define     MAXINT_TABLE     (TX3927INT_BASE+16)

VINTFUNC     swIntVector[MAXINT_TABLE];

/*************************************
 * Register Virtual Interrupt Vector     *
 *************************************/
VINTFUNC setIntVect(int no,VINTFUNC func)
{
   VINTFUNC ret;

   ret = swIntVector[no];
   swIntVector[no] = func;
   return ret;
}



/*************************************
 * Initialize Interrupt Processing       *
 *************************************/
#define      GINT_VECT        0x80000080             /* Interrupt vector address */
extern int org_GINT_VECT(void);    /* First program for interrupt vector */
                                   /* (Assembly language routine) */
extern int org_EXT_INT_VECT(void); /* External interrupt (Excode=0) Handler */
                                   /* (Assembly language routine) */
extern int size_org_GINT_VECT ;
int size = (int)(&size_org_GINT_VECT) ;

int intInt(void)
{

   memcpy((void *)GINT_VECT,org_GINT_VECT,size);

   setIntVect(INTNO_Int,org_EXT_INT_VECT);

   CPUReg->IREG.IRDER = 1;             /* Enable interrupts in Interrupt Controller */
   CPUReg->IREG.IRIMR = 1;             /* INT0 interrupt = High level */
   setstatus(~SR_BEV,SR_IE | 0xff00);  /* Enable interrupts in Status register */

   return 0;
}
```

The following shows an example of an interrupt handling routine written in assembly language:

- File name: aintcosmp.S

```
/*****************************************************************
   TX3927 Interrupt Handling Assembler Module

org_GINT_VECT:
   Entry point for general exception handling. Use the table to pass control to a handler.
 # go swIntVector[C0_CAUSE]

return_GINTVECT:
 Handler called from org_GINT_VECT returns control to instruction at which interrupt
 occurred
 # retuen GINT
```

```
org_EXT_INT_VECT:
 Branch to external interrupt handler (org_GINT_VECT processing for 0)
 Destination depends on TX3927 external interrupt status (PI).
 If return value is 0, control is returned to address at which interrupt occurred. If not
0, control is returned to original interrupt vector. Nested interrupts are supported.
 # go &(swIntVector[TX3927INT_BASE])[IP]



*****************************************************************/

   .text
   .set     mips1
   .set     noreorder
   .set     noat

/* ------------------------------------------------------------------------------*/
/* Entry point for TX3927 general interrupt handling (Branch, using Cause register */
/* excCode)                                                                      */
/* ------------------------------------------------------------------------------*/
   .globl org_GINT_VECT
   .globl size_org_GINT_VECT
   .ent org_GINT_VECT
org_GINT_VECT:                          # go swIntVector[C0_CAUSE]
   mfc0       k0,C0_CAUSE
   nop
   andi       k0,0x7c                  # isolate exception code
   la         k1,swIntVector
tablejmp:
   add        k0,k1,k0                 # offset of VSR entry
tablejmp2:
   lw         k0,0(k0)                 # ke = pointer to vsr
   jr         k0                       # jump into virtual vector handler
   nop

/* ------------------------------------------------------------------------*/
/* Return from branch destination to instruction at which interrupt occurred */
/* ------------------------------------------------------------------------*/
   .globl return_GINT_VECT
return_GINTVECT:                        # return GINT
   mfc0       k1, C0_EPC
   nop
   jr         k1
   rfe
   nop
end_org_GINT_VECT:
   .equ    size_org_GINT_VECT,end_org_GINT_VECT-org_GINT_VECT
   .end org_GINT_VECT

/* ---------------------------------------------------------------*/
/* Interrupt stack definitions                                     */
/* ---------------------------------------------------------------*/

#define       GINT_STACKSIZE           4096    /* allocate 4K bootstack */
   .globl     gint_stack_end
gint_stack_end:
   .space     GINT_STACKSIZE                   /* allocate the exception stack */
   .globl     gint_stack
gint_stack:                                    /* stack top here (stack grows DOWN) */
   .space     8                                /* allocate dummy stack */

#define       ik0     k0
#define       ik1     k1
#define       GISTACK_SIZE    REG_SIZE*(26+1)
   /*         17 1-15,24-25(at,t0-1,a0-3,t0-9)
              5 16(s0),23(s7),29(sp),28(gp),31(ra)
```

```
                4 cp0*4(hi,lo,status,ecp)
                26 */
#define         MAXINT  8

/* ------------------------------------------------------------------*/
/* Branch for external interrupt                                     */
/*        Set interrupt stack                                        */
/*        Save registers                                             */
/*        Branch using Cause register IP                             */
/* ------------------------------------------------------------------*/

   .globl org_EXT_INT_VECT
   .ent org_EXT_INT_VECT
org_EXT_INT_VECT:
#if 1
   la      ik0,gint_stack    /* normal stack hi address */
   sltu    ik1,ik0,sp
   bne     ik1,zero,1f
   addiu   ik1,ik0,-GINT_STACKSIZE
   sltu    ik1,ik1,sp
   beq     ik1,zero,1f
   nop
#else
   la      ik0,gint_stack    /* normal stack Low address */
   addiu   ik1,ik0,-GINT_STACKSIZE
   sltu    ik1,ik1,sp
   beq     ik1,zero,1f
   sltu    ik1,ik0,sp
   bne     ik1,zero,1f
   nop
#endif
   add     ik0,zero,sp
1:                           /* ik1 = temp stack */
   addiu   ik0,ik0,-(GISTACK_SIZE)   /* Save Registers */
   sw              at,REG_SIZE*1(ik0)
   sw              v0,REG_SIZE*2(ik0)
   sw              v1,REG_SIZE*3(ik0)
   sw              a0,REG_SIZE*4(ik0)
   sw              a1,REG_SIZE*5(ik0)
   sw              a2,REG_SIZE*6(ik0)
   sw              a3,REG_SIZE*7(ik0)
   sw              t0,REG_SIZE*8(ik0)
   sw              t1,REG_SIZE*9(ik0)
   sw              t2,REG_SIZE*10(ik0)
   sw              t3,REG_SIZE*11(ik0)
   sw              t4,REG_SIZE*12(ik0)
   sw              t5,REG_SIZE*13(ik0)
   sw              t6,REG_SIZE*14(ik0)
   sw              t7,REG_SIZE*15(ik0)
   sw              t8,REG_SIZE*16(ik0)
   sw              t9,REG_SIZE*17(ik0)
   sw              gp,REG_SIZE*18(ik0)
   sw              ra,REG_SIZE*19(ik0)
   sw              s0,REG_SIZE*20(ik0)
   sw              s7,REG_SIZE*21(ik0)
   mflo    t0
   sw              t0,REG_SIZE*22(ik0)
   mfhi            t0
   sw              t0,REG_SIZE*23(ik0)
   mfc0            t0,C0_STATUS
   sw              t0,REG_SIZE*24(ik0)
   mfc0            t0,C0_EPC
   sw              t0,REG_SIZE*25(ik0)
   sw              sp,REG_SIZE*26(ik0)
   addu            sp,zero,ik0
```

```
    lui             s0,%hi(_gint_count)     # Increment Interrupt counter
    lw              a1,%lo(_gint_count)(s0)
    addiu   a1,a1,1
    sw              a1,%lo(_gint_count)(s0)

    la              gp,_gp                  # set the global data pointer

    mfc0    k0,C0_CAUSE
    nop
    andi    k1,k0,0x0300
    bne             k1,zero,1f              # sw interrupt
    srl             k0,k0,8
    andi    k0,0x3c
    la              k1,swIntVector+15*4     # go &(swIntVector[TX3927INT_BASE])[]
    j               3f
    addu    k0,k0,k1
1:
    andi    k1,k1,0x0100
    beq             k1,zero,2f
    la              k0,swIntVector+13*4     # go sw0
    j               3f
    nop
2:
    la              k0,swIntVector+14*4     # go sw1
3:
    lw              k0,0(k0)                # swIntVector read
    jalr    k0                              # call
    nop

    lw              a1,%lo(_gint_count)(s0)
    addiu   at,a1,-MAXINT
    bgtz    at,99f              /* over multi interrupt */
    addiu   a1,a1,-1
    sw              a1,%lo(_gint_count)(s0)

    lw              at,REG_SIZE*1(sp)   /* return target */
    lw              v1,REG_SIZE*3(sp)
    lw              a0,REG_SIZE*4(sp)
    lw              a1,REG_SIZE*5(sp)
    lw              a2,REG_SIZE*6(sp)
    lw              a3,REG_SIZE*7(sp)
    lw              t1,REG_SIZE*9(sp)
    lw              t2,REG_SIZE*10(sp)
    lw              t3,REG_SIZE*11(sp)
    lw              t4,REG_SIZE*12(sp)
    lw              t5,REG_SIZE*13(sp)
    lw              t6,REG_SIZE*14(sp)
    lw              t7,REG_SIZE*15(sp)
    lw              t8,REG_SIZE*16(sp)
    lw              t9,REG_SIZE*17(sp)
    lw              gp,REG_SIZE*18(sp)
    lw              ra,REG_SIZE*19(sp)
    lw              s0,REG_SIZE*20(sp)
    lw              s7,REG_SIZE*21(sp)
    lw              t0,REG_SIZE*22(sp)
    mtlo    t0
    lw              t0,REG_SIZE*23(sp)
    mthi    t0
    lw              t0,REG_SIZE*24(sp)
    mtc0    t0,C0_STATUS
    lw              k1,REG_SIZE*25(sp)
    mtc0            k1,C0_EPC
    lw      t0,REG_SIZE*8(sp)

    lw      v0,REG_SIZE*2(sp)
    lw      sp,REG_SIZE*26(sp)
```

```
    sync                                /* Flush the write buffer to memory. */
    jr              k1                  /* Return to the user program. */
    rfe
    nop

99:     /* stack overflow */
    b       99b
    nop
```

Interrupt handling flow:

(1) Examine the ExCode field of the Cause register and pass control to the appropriate exception handler (swIntVect[0-12]). (In this example, only the external interrupt (ExCode = 0) handling routine is provided.)

(2) For an external interrupt (swIntVect[0]), read the IP field of the Cause register and pass control to the appropriate handler (swIntVect[13-30]).
This routine sets the interrupt stack, and saves and restores registers. It does not save all registers because the program assumes that the body of interrupt handling is written in C. Those registers not saved here will be saved and restored, as required, in the C function.
When this example is used, the actual interrupt handling section is written in C, as follows:

```
int C_int_timer0(void)
{
   static int count = 0;

   count++;                     /* Count number of interrupts */
   CPUReg->TREG[0].TMTISR = 0;  /* Negate interrupt */
   return 0;
}
```

The C function should include the following processes:

- Appropriately handles the interrupt.
- Clears the interrupt condition.
- (Optionally) allows nested interrupts.

A.2.5 Manipulating the Caches

The following sample functions are provided for data and instruction caches:

- FlushDCache — If the cache block contains data of specified size from the specified address, write back the data and invalidate the cache block.
- FlushDCacheAll — Write back and invalidate all blocks in data cache
- InvalidateICache — If cache contains an instruction of specified size from the specified address, invalidate the cache block.
- InvalidateICacheAll — Invalidate all instructions in the instruction cache

Note that any instruction cache operation requires invalidating the cache block first.

- File name: cache.c

```
/*==========================================================================
 * $Id$
 *--------------------------------------------------------------------------
 * Copyright(C) 1991-1998 TOSHIBA CORPORATION  All rights reserved.
 *==========================================================================
 * TX39/H2 cache control routines
 */

/* TX39H2 Core Cache Size */
#define ICACHE_SIZE 0x2000    /* 8KB */
#define DCACHE_SIZE 0x1000    /* 4KB */

void
FlushDCache(unsigned int address, int size)
{
    __asm(".set noreorder");
    __asm("add   $5, $4");
    __asm("li    $6, ~0xf");           /* TX39/H2 line size 16 bytes */
    __asm("and   $4, $4, $6");
    __asm("1:");
    __asm("bge   $4, $5, 2f");
    __asm("nop");
    __asm("cache 21, 0($4)");          /* Hit_Writeback_Invalidate */
    __asm("b     1b");
    __asm("addi   $4, $4, 16");
    __asm("2:");
}

void
FlushDCacheAll()
{
    __asm(".set noreorder");
    __asm("lui   $4, 0x8000");         /* start address */
    __asm("addi  $5, $4, 0x800");      /* end address   */
    __asm("1:");
    __asm("bge   $4, $5, 2f");
    __asm("nop");
    __asm("cache 1, 0($4)");           /* Index_Writeback_Inv_D way 0 */
    __asm("cache 1, 1($4)");           /* Index_Writeback_Inv_D way 1 */
    __asm("b     1b");
    __asm("addi  $4, $4, 16");
    __asm("2:");
}

void
InvalidateICache(unsigned int address, int size)
{
```

```
    __asm(".set noreorder");
    __asm("add   $5, $4");
    __asm("li    $6, ~0xf");           /* line size 16 bytes */
    __asm("and   $4, $4, $6");
    __asm("mfc0  $6, $3");             /* get C0_Config */
    __asm("nop");
    __asm("andi  $7, $6, 0xffdf");     /* ICE OFF   */
    __asm("mtc0  $7, $3");             /* set C0_Config */
    __asm("1:");
    __asm("bge   $4, $5, 2f");
    __asm("nop");
    __asm("cache 16, 0($4)");          /* Hit_Invalidate_I */
    __asm("b     1b");
    __asm("addi  $4, $4, 16");
    __asm("2:");
    __asm("mtc0  $6, $3");             /* set C0_Config */

}

void
InvalidateICacheAll()
{
    __asm(".set noreorder");
    __asm("lui   $4, 0x8000");         /* start address */
    __asm("addi  $5, $4, 0x1000");     /* end address   */
    __asm("mfc0  $6, $3");             /* get C0_Config */
    __asm("nop");
    __asm("andi  $7, $6, 0xffdf");     /* ICE OFF   */
    __asm("mtc0  $7, $3");             /* set C0_Config */
    __asm("1:");
    __asm("bge    $4, $5, 2f");
    __asm("nop");
    __asm("cache 0, 0($4)");           /* Index_Invalidate_I way 0 */
    __asm("cache 0, 1($4)");           /* Index_Invalidate_I way 1 */
    __asm("b     1b");
    __asm("addi  $4, $4, 16");
    __asm("2:");
    __asm("mtc0  $6, $3");             /* set C0_Config */
}
```

A.3 Examples of Using On-Chip Peripherals

A.3.1 Timer/Counter

The following sample program uses Timer 0 in interval timer mode to generate interrupts:

- File name: TimerInt.c

```
#include    "tx3927.h"
#include    "intcosmp27.h"

/* Sample program. Initialize interrupt handler and generate timer interrupts */
int test_timerInt0(void)
{
   initInt();                                /* Initialize interrupt handler */

   setIntVect(INTNO_TMR0,C_int_timer0);      /* Register interrupt handler */

   CPUReg->TREG[0].TMTCR = TCE | CCDE;       /* Start timer */
   CPUReg->TREG[0].TMITMR = TIIE | TZCE;

   CPUReg->IREG.IRILR[6] = (CPUReg->IREG.IRILR[6] & 0x000f)  | 0x0300;
/* Set interrupt level */

   CPUReg->IREG.IRDER = 1;             /* Enable interrupts in interrupt controller */
   CPUReg->IREG.IRIMR = 1;
   setstatus(~SR_BEV,SR_IE | 0xff00);  /* Enable interrupts in CPU's Status register*/
   return 0;
}

int off_timerInt0(void)
{
   CPUReg->TREG[0].TMTCR = 0;          /* Stop timer */
   CPUReg->TREG[0].TMITMR = 0;
   return 0;
}

/* Timer interrupt handling routine */
int C_int_timer0(void)
{
   static int count = 0;

   count++;                            /* Count number of interrupts */
   CPUReg->TREG[0].TMTISR = 0;         /* Negate interrupt */
   return 0;
}
```

A.3.2 SIO

This sample contains the following operational settings:

send-polling & receive-polling mode
send-interrupt & receive-interrupt mode
send-polling & receive-interrupt mode
send-interrupt & receive-polling mode
DMA-SEND & polling-RECEIVE mode
DMA-SEND & interrupt-RECEIVE mode
polling-send & DMA-RECEIVE mode
interrupt-send & DMA-RECEIVE mode
DMA-SEND & DMA-RECEIVE mode
polling loopback mode
interrupt loopback mode
dma loopback mode
receive overflow test

- File name: 3927sio.c

```
/*==========================================================================
 * $Id$
 *--------------------------------------------------------------------------
 * Copyright(C) 1998-1999 TOSHIBA CORPORATION All rights reserved.
 *==========================================================================
 */
/*
 *  TX3927 SIO Control Routine
 *
 * This is a sample routine to control the TX3927 SIO.
 *
 * Notes on using the TX3927 SIO:
 *
 * Macro:
 * USE_UDEOS  Required when used as an application program for
 *            UDEOS/r39, a µITRON 3.0-compliant operating system from Toshiba
 *            Information Systems Corp.
 *            This macro will affect configurations; Be sure to add the definition to
 *            the cfo files.

 * Operation tested with:
 *  • COSMP2-TX3927 evaluation board
 *  • GHS C Compiler & MULTI Debugger ver.1.8.8 for Win32
 *  • GHS Monitor Server RS232C connection or HP.ProcessorProbe
 *
 * Reference material:
 *  • 32-bit RISC Microprocessor Family TX39 Family
 * TMPR3901F User's Manual
 *  • 32-bit RISC Microprocessor Family TX39 Family
 * TMPR3927F User's Manual Rev.0.5
 *
 * History:
 * 1999/05/06 ver.0.21 yasui based on r3904sio.c ver.1.08
 ****************************************************************************/

#include <itron.h>
#include "3927sio.h"
```

```
/* vertiul address to physical address */
#define Vadrs2Padrs(adrs)  ( (unsigned int)(adrs) & 0x5FFFFFFF )
/* I/O Base Addr */
#define REG3927SIO_BASE 0xfffef300

/* Semaphore */
#ifdef USE_UDEOS
#define SIGNAL_SEMAPHORE(x) isig_sem(SIO_SEMID0+x)
#define WAIT_SEMAPHORE(x) wai_sem(SIO_SEMID0+x)
#define SIGNAL_SEMAPHORE_TX(x) isig_sem(SIO_SEMID0+2+x)
#define WAIT_SEMAPHORE_TX(x) wai_sem(SIO_SEMID0+2+x)
#endif

/* Etc */
#define WAIT_TIME  (CPU_CLOCK/500)
#define INCIDX(a) ((a+1)&(SIO_RCVBUFSZ-1))

/* Structure */

typedef struct {
   int type;
   int (*getc)();
   int (*putc)();
   volatile int sndbuf_beg;
   volatile int sndbuf_end;
   volatile int rcvbuf_beg;
   volatile int rcvbuf_end;
   int sndbuf_ovf;
   int rcvbuf_ovf;
   char sndbuf[SIO_SNDBUFSZ];
   char rcvbuf[SIO_RCVBUFSZ];
} type3927sio;

typedef struct {
   unsigned int silcr;
   unsigned int sidicr;
   unsigned int sidisr;
   unsigned int sicisr;
   unsigned int sifcr;
   unsigned int siflcr;
   unsigned int sibgr;
   unsigned int sitfifo;
   unsigned int sirfifo;
   int pad[55];
} reg3927sio;

#define SILCR_SCS_BGIMCLK   0x20
#define SILCR_SCS_EXTSCLK   0x40
#define SILCR_SCS_BGSCLK    0x60
#define SILCR_UMODE         0x00000001
#define SILCR_USBL          0x00000004
#define SILCR_UPEN          0x00000008
#define SILCR_UEPS          0x00000010
#define SIDISR_ERRMASK      0x0000b800
#define SIDISR_RDIS         0x00000080
#define SIDISR_TDIS         0x00000100
#define SIDISR_TOUT         0x00000200
#define SIDISR_ERI          0x00000400
#define SIDISR_UOER         0x00000800
#define SIDISR_UPER         0x00001000
#define SIDISR_UFER         0x00002000
#define SIDISR_UBRK         0x00008000
#define SIDISR_UVALID       0x00004000
#define SICISR_TRDY         0x00000004
#define SICISR_TXALS        0x00000002
#define SIFCR_FRSTE         0x00000001
```

```
#define SIFCR_RFRST         0x00000002
#define SIFCR_TFRST         0x00000004
#define SIFCR_TDIL1         0x00000000
#define SIFCR_TDIL4         0x00000008
#define SIFCR_TDIL8         0x00000010
#define SIFCR_RDIL1         0x00000000
#define SIFCR_RDIL4         0x00000080
#define SIFCR_RDIL8         0x00000100
#define SIFCR_RDIL12        0x00000180
#define SIFLCR_TSDE         0x00000080
#define SIFLCR_RSDE         0x00000100
#define SIFLCR_RTSTL(x)   ((x&0xf)<<1)

#define SIDICR_TDE          0x00008000
#define SIDICR_RDE          0x00004000
#define SIDICR_TIE          0x00002000
#define SIDICR_RIE          0x00001000
#define SIDICR_SPIE         0x00000800

#define SIFCR_RDIL_MASK     0x00000180
#define SIFCR_TDIL_MASK     0x00000018


/* Data */

static type3927sio siotbl[2];
static unsigned int dma_req_flag, dma_end_flag;

/*********************************************************
 *       Wait loop
 *********************************************************/
static int
noop (int i){
   int a=10;

   while (i--)
            while(a)
                    a--;
   return a;
}

/*************************************************************
 * Initialize SIO
 *
 * siono is 0 or 1. baud is baud-rate(ie.9600,38400,etc.).
 * type is
 *   SIO_TXPOL   output is polling.
 *   SIO_TXINT   output is interrupt.
 *   SIO_TXDMA   output is DMA.
 *   SIO_RXPOL   input is polling.
 *   SIO_RXINT   input is interrupt.
 *   SIO_RXDMA   input is DMA
 *************************************************************/
int
tx3927sio_init(int siono,int type,int baud)
{
   reg3927sio *sioreg = (reg3927sio *)REG3927SIO_BASE;
   type3927sio* sio;
   int bclk,regbaud,silcr;
   unsigned int *dma_adrs;



   /* check parameter (0 or 1) */
   if(siono & ~0x1)
            return -1;              /* ERROR: illegal siono */
```

```
   /* preparing */
   sioreg+=siono;
   sio = &siotbl[siono];
   sio->type = type;

   /* calcualte baud value */
   bclk=0;
#ifdef CPU133M
   regbaud = CPU_CLOCK/128/baud;
#else
   regbaud = CPU_CLOCK/64/baud;
#endif
   while(regbaud>=256){
            bclk++;
            regbaud /= 4;
   }
   if(bclk>3)
            return -2;              /* ERROR: illegal baud */
   /* general setting */
            silcr = 0;
   if(type & SIO_PARITY_ODD)
            silcr |= SILCR_UPEN;
   if(type & SIO_PARITY_EVEN)
            silcr |= SILCR_UPEN|SILCR_UEPS;
   if(type & SIO_STOP2)
            silcr |= SILCR_USBL;
   if(type & SIO_7BIT)
            silcr |= SILCR_UMODE;

   sioreg->silcr  = silcr|SILCR_SCS_BGIMCLK;                  /* BRG-IMCLK */
   sioreg->siflcr = SIFLCR_TSDE|SIFLCR_RSDE|SIFLCR_RTSTL(1);
   sioreg->sidicr = 0x0;
   sioreg->sifcr  = SIFCR_TFRST|SIFCR_RFRST|SIFCR_FRSTE;      /* Reset FIFO */
   sioreg->sifcr  = 0x0;
   sioreg->sibgr  = (bclk<<8)|regbaud;                        /* Set baudrate */
   sioreg->sidisr = 0x0;
   sioreg->sicisr = 0x0;




   if(type & SIO_TXPOL){
      sio->putc = (int(*)())tx3927sio_pol_putc;
      sioreg->siflcr &= ~SIFLCR_TSDE;                         /* enable TX */
      sioreg->siflcr |= 0x200;

   }


   if(type & SIO_RXPOL){
      sio->getc = (int(*)())tx3927sio_pol_getc;
      sioreg->siflcr &= ~SIFLCR_RSDE;                         /* enable RX */
   }


   if(type & SIO_TXINT){
      sio->sndbuf_beg = sio->sndbuf_end = sio->sndbuf_ovf = 0;
      sio->putc = (int(*)())tx3927sio_int_putc;
      if (type & SIO_TFIFO_1) {                               /* trigger byte */
               sioreg->sifcr |= SIFCR_TDIL1;
      }else if (type & SIO_TFIFO_4) {
               sioreg->sifcr |= SIFCR_TDIL4;
      } else if (type & SIO_TFIFO_8) {
               sioreg->sifcr |= SIFCR_TDIL8;
      } else {
```

```
            sioreg->sifcr |= SIFCR_TDIL8;                /* default 8byte */
   }

            sioreg->siflcr &= ~SIFLCR_TSDE;              /* enable TX */
            chg_ilv(INTNO_SIO0+siono,2);                 /* set int level */
}


if(type & SIO_RXINT){
   sio->rcvbuf_beg = sio->rcvbuf_end = sio->rcvbuf_ovf = 0;
   sio->getc = (int(*)())tx3927sio_int_getc;
   if (type & SIO_RFIFO_1) {                             /* trigger byte */
            sioreg->sifcr |= SIFCR_RDIL1;
   } else if (type & SIO_RFIFO_4) {
            sioreg->sifcr |= SIFCR_RDIL4;
   } else if (type & SIO_RFIFO_8) {
            sioreg->sifcr |= SIFCR_RDIL8;
   } else if (type & SIO_RFIFO_12) {
            sioreg->sifcr |= SIFCR_RDIL12;
   } else {
            sioreg->sifcr |= SIFCR_RDIL12;               /* default */
   }
   sioreg->sidicr |= SIDICR_RIE+SIDICR_SPIE;             /* enable RX int */
   sioreg->siflcr &= ~SIFLCR_RSDE;                       /* enable RX */
   chg_ilv(INTNO_SIO0+siono,2);                          /* set int level */
}


if(type & SIO_TXDMA){
   sio->sndbuf_beg = sio->sndbuf_end = sio->sndbuf_ovf = 0;
   if (type & SIO_TFIFO_1) {                             /* trigger byte */
            sioreg->sifcr |= SIFCR_TDIL1;
   } else if (type & SIO_TFIFO_4) {
            sioreg->sifcr |= SIFCR_TDIL4;
   } else if (type & SIO_TFIFO_8) {
            sioreg->sifcr |= SIFCR_TDIL8;
   } else {
            sioreg->sifcr |= SIFCR_TDIL8;      /* default 8byte */
   }
   sioreg->siflcr &= ~SIFLCR_TSDE;             /* enable TX */
   dma_adrs  = (unsigned int *)( 0xfffeb0a4 ) ;
                                               /* MCR(Master Control Register) */
   *dma_adrs = 0x00000000;                     /* MASTEN(bit0)=off */
   chg_ilv(INTNO_DMA,2);                       /* set int level */


}
if(type & SIO_RXDMA){
   sio->rcvbuf_beg = sio->rcvbuf_end = sio->rcvbuf_ovf = 0;
   if (type & SIO_RFIFO_1) {         /* trigger byte */
            sioreg->sifcr |= SIFCR_RDIL1;
   } else if (type & SIO_RFIFO_4) {
            sioreg->sifcr |= SIFCR_RDIL4;
   } else if (type & SIO_RFIFO_8) {
            sioreg->sifcr |= SIFCR_RDIL8;
   } else if (type & SIO_RFIFO_12) {
            sioreg->sifcr |= SIFCR_RDIL12;
   } else {
            sioreg->sifcr |= SIFCR_RDIL12;     /* default */
   }
            sioreg->sidicr |= SIDICR_SPIE;     /* enable RX int */
            sioreg->siflcr &= ~SIFLCR_RSDE;             /* enable RX */
            dma_adrs  = (unsigned int *)( 0xfffeb0a4 ) ;
                                                  /* MCR(Master Control Register) */
            *dma_adrs = 0x00000000;                     /* MASTEN(bit0)=off */
            chg_ilv(INTNO_SIO0+siono,2);                /* set int level */
```

```
                chg_ilv(INTNO_DMA,2);                       /* set int level */

   }


   return 0;               /* Successful */
}

/*****************************************************************
 * Finish SIO
 *
 * siono is 0 or 1.
 * clear register. reset fifo. disable interrupt.
 *****************************************************************/
int
tx3927sio_fini(int siono){
   reg3927sio *sioreg = (reg3927sio *)REG3927SIO_BASE;
   type3927sio* sio;
   int bclk,regbaud;

   /* check parameter */
   if(siono!=0 && siono!=1)
      return -1;                                            /* ERROR: illegal siono */

   tx3927sio_all_sent(siono);


   /* initialize */
   chg_ilv(INTNO_SIO0+siono,0);                             /* clear int level */
   sioreg->siflcr |= SIFLCR_RSDE;                           /* disable RX */
   sioreg->siflcr |= SIFLCR_TSDE;                           /* disable TX */
   sioreg->sidicr &= ~SIDICR_TIE;                           /* disable TX int */
   sioreg->sidicr &= ~SIDICR_RIE;                           /* disable TX int */
   sio = &siotbl[siono];
   sio->putc = 0;
   sio->getc = 0;
   sio->type = 0;
}

/*********************************************************************
 * SIO interrupt handler
 *
 * clear interrupt. read data and put into buffer.
 *********************************************************************/
static void
tx3927sio_int(int siono)
{
   reg3927sio  *sioreg = (reg3927sio *)REG3927SIO_BASE;
   type3927sio *sio    = &siotbl[siono];
   int sidisr,next,err,c;

   sioreg += siono;
   sidisr  = sioreg->sidisr;
   if(sidisr & 0xbc00){
           printf("error1(sidisr=%x) ¥n", sidisr);
           if(sidisr & SIDISR_UBRK)
                   printf("break during int mode ¥n");
           if(sidisr & SIDISR_UFER)
                   printf("frame error during int mode ¥n");
           if(sidisr & SIDISR_UPER)
                   printf("parity error during int mode ¥n");
           if(sidisr & SIDISR_UOER)
                   printf("overflow during int mode ¥n");
           if(sidisr & 0x0400)
                   printf("error interrupt during int mode ¥n");
           c = sioreg->sirfifo;
```

```
    }

    sidisr  = sioreg->sidisr;
    if(sidisr & 0xbc00){
            printf("error1(sidisr=%x) ¥n", sidisr);
            if(sidisr & SIDISR_UBRK)
                    printf("break during int mode ¥n");
            if(sidisr & SIDISR_UFER)
                    printf("frame error during int mode ¥n");
            if(sidisr & SIDISR_UPER)
                    printf("parity error during int mode ¥n");
            if(sidisr & SIDISR_UOER)
                    printf("overflow during int mode ¥n");
            if(sidisr & 0x0400)
                    printf("error interrupt during int mode ¥n");
            c = sioreg->sirfifo;
    }



    if(sidisr & (SIDISR_RDIS | SIDISR_TOUT)){
            while(!(sidisr & SIDISR_UVALID) && !(sidisr & SIDISR_ERRMASK)){
                    c = sioreg->sirfifo;
                    if(sio->rcvbuf_beg != (next=INCIDX(sio->rcvbuf_end))){
                            if(sio->rcvbuf_beg == sio->rcvbuf_end)
                                    SIGNAL_SEMAPHORE(siono);
                            sio->rcvbuf[sio->rcvbuf_end] = c;
                            sio->rcvbuf_end = next;
                    }
                    else{
                            sio->rcvbuf_ovf++;   /* Overflow */
                            printf("receive buffer overflow ¥n");
                    }
                    sidisr = sioreg->sidisr;
                    if( sidisr & 0xbc00 )
                            printf("error2(sidisr=%x) ¥n", sidisr);
            }
            sioreg->sidisr = ~(SIDISR_RDIS|SIDISR_TOUT);
    }
    if(sidisr & SIDISR_ERI){
            sioreg->sidisr = ~SIDISR_ERI;         /* Clear Error Status */
            /* routine after error */
            sioreg->sirfifo;                      /* waiste */
    }
    if(sidisr & SIDISR_TDIS){
            while((sio->sndbuf_beg != sio->sndbuf_end) &&
            (sioreg->sicisr & SICISR_TRDY)){
                    sioreg->sitfifo = sio->sndbuf[sio->sndbuf_beg];
                    sio->sndbuf_beg = INCIDX(sio->sndbuf_beg);
            }
            if(sio->sndbuf_beg != sio->sndbuf_end){
                    sioreg->sidisr &= ~SIDISR_TDIS;
            }
            else{
                    sioreg->sidicr &= ~SIDICR_TIE;
                    if(sio->sndbuf_ovf){
                            SIGNAL_SEMAPHORE_TX(siono);
                    }
            }
    }
    return;
}

void tx3927sio0_int()
{
    tx3927sio_int(0);
```

```
}
void tx3927sio1_int()
{
   tx3927sio_int(1);
}

void tx3927dma_int()
{
   unsigned int *dma_adrs, *dma0_adrs, *dma2_adrs, dma_status;


#if 0
   dma_adrs   = (unsigned int *)( 0xfffeb0a4 ) ;    /* MCR(Master Control Register) */
   *dma_adrs  = 0x00000000;                         /* MASTEN(bit0)=off */
#endif

   if( dma_req_flag == 2 ){
           dma2_adrs  = (unsigned int *)( 0xfffeb05c );
                                                    /* Channel Status Register(CSR2) */
           dma_status = *dma2_adrs;                 /* read status register */
           if( dma_status != 0x60 )   /* NCHNC & NTRNFC(Normal Transfer Completion) */
                   printf(" dma2 complete status error=%x ¥n", dma_status);
           *dma2_adrs = 0xffffffff;         /* clear CSR2 */
           dma_end_flag = 1;

   }
   if( dma_req_flag == 0 ){
           dma0_adrs  = (unsigned int *)( 0xfffeb01c );
                                                    /* Channel Status Register(CSR0) */
           dma_status = *dma0_adrs;                 /* read status register */
           if( dma_status != 0x60 )   /* NCHNC & NTRNFC(Normal Transfer Completion) */
                   printf(" dma0 complete status error=%x ¥n", dma_status);
           *dma0_adrs = 0xffffffff;                 /* clear CSR0 */
           dma_end_flag = 1;
   }

   return;

}


/*****************************************************************************
 * Get & Put one byte data in interrupt handler
 *
 * if data is bufferd, return soon. otherwise sleep.
 * use message box.
 *****************************************************************************/
int
tx3927sio_int_getc(int siono)
{
   type3927sio *sio = &siotbl[siono];
   int c;

   if(sio->rcvbuf_beg == sio->rcvbuf_end)
      WAIT_SEMAPHORE(siono);
   if(sio->rcvbuf_beg != sio->rcvbuf_end){
      c = sio->rcvbuf[sio->rcvbuf_beg] & 0xff;
      sio->rcvbuf_beg = INCIDX(sio->rcvbuf_beg);
      return c;
   }
   else
      return -1;
}

int
tx3927sio_int_putc(int siono, char c)
```

```
{
   reg3927sio *sioreg = (reg3927sio *)REG3927SIO_BASE;
   type3927sio *sio = &siotbl[siono];
   int next=INCIDX(sio->sndbuf_end);
   /* check console's sio */
   if(siono!=0 && siono!=1)
            return -1;        /* ERROR:illegal siono */
   sioreg+=siono;
  retry_tx:
   if((sio->sndbuf_beg == sio->sndbuf_end) && (sioreg->sicisr & SICISR_TRDY)){
       sioreg->sitfifo = c;
       return 0;
   }
   else
            if(sio->sndbuf_beg != next){
                     sio->sndbuf[sio->sndbuf_end] = c;
                     sio->sndbuf_end = next;
                     sioreg->sidisr &= ~SIDISR_TDIS;
                     sioreg->sidicr |= SIDICR_TIE;
                     return 0;
            }
            else{
                     sio->sndbuf_ovf++;      /* Overflow */
                     WAIT_SEMAPHORE_TX(siono);
                     sio->sndbuf_ovf--;
                     goto retry_tx;
            }
   return -1;
}


/*****************************************************************************
 * Get & Put data block dma handler
 *
 * if data is bufferd, return soon. otherwise sleep.
 * use message box.
 *****************************************************************************/
int
tx3927sio_dma_getc(int siono, char *rcv_buf, int buf_size, int *rcvd_size)
{
   reg3927sio *sioreg = (reg3927sio *)REG3927SIO_BASE;
   type3927sio *sio = &siotbl[siono];
   int c,i,sidisr,work;
   unsigned int *dma0_adrs, *pcr, dma_status;


   pcr   = (unsigned int *)( 0xfffee008 ) ;      /* Pin Configuration Register */
   *pcr |= 0x0000f0f0;

   dma0_adrs  = (unsigned int *)( 0xfffeb0a4 ) ; /* MCR(Master Control Register)  */
   *dma0_adrs = 0x00000001;                      /* MASTEN(bit0)=on */
   dma0_adrs  = (unsigned int *)( 0xfffeb018 ) ; /* Channel Control Register(CCR0) */
   *dma0_adrs = 0x01000000;                      /* CHRST(bit24)=on(reset channel) */
   dma0_adrs  = (unsigned int *)( 0xfffeb018 ) ; /* Channel Control Register(CCR0) */
   *dma0_adrs = 0x00000000;                      /* CHRST(bit24)=off(enable channel) */
   dma0_adrs  = (unsigned int *)( 0xfffeb004 ) ; /* Source Address Register(SAR0) */
   *dma0_adrs = 0xfffef320+3;                    /* Receive FIFO buffer 0 */
   dma0_adrs  = (unsigned int *)( 0xfffeb008 ) ;
                                                 /* Destination Address Register(DAR0) */
   *dma0_adrs = Vadrs2Padrs(rcv_buf);            /* receive buffer physical address */
   dma0_adrs  = (unsigned int *)( 0xfffeb00c ) ; /* Count Register(CNAR0) */
   *dma0_adrs = buf_size;                        /* receive buffer size */
   dma0_adrs  = (unsigned int *)( 0xfffeb010 ) ;
                                          /* Source Address Incremnet Register(SAI0) */
   *dma0_adrs = 0x0000000;                       /* no increment */
   dma0_adrs  = (unsigned int *)( 0xfffeb014 ) ;
```

```
                                        /* Destination Address Incremnet Register(DAI0) */
    *dma0_adrs = 0x0000001;                     /* transfer size one byte */
    dma0_adrs  = (unsigned int *)( 0xfffeb018 ) ; /* Channel Control Register(CCR0) */
    *dma0_adrs = 0x00011500;                    /* EXTRQ,INTENE,INTENT,XTACT,XFSZ=0 */

    dma_req_flag = 0;                           /* dma channel number 0 */
    dma_end_flag = 0;
    sioreg->sidicr |= SIDICR_RDE;               /* enable receive DMA */

    do{
            for(i=0;i<100;i++);                 /* don't disturb DMA(memory) bus */
    }while( dma_end_flag == 0 );

    sidisr  = sioreg->sidisr;

    if(sidisr & 0xbc00)
            printf("error3(sidisr=%x) ¥n", sidisr);

    dma0_adrs  = (unsigned int *)( 0xfffeb01c ) ; /* Channel Status Register(CSR0) */
    *dma0_adrs = 0xffffffff;                      /* clear CSR0 */
    dma0_adrs  = (unsigned int *)( 0xfffeb018 ) ; /* Channel Control Register(CCR0) */
    *dma0_adrs = 0x01000000;                      /* CHRST(bit24)=on(reset channel) */

    dma0_adrs  = (unsigned int *)( 0xfffeb008 ) ;
                                                /* Destination Address Register(DAR0) */
    work = *dma0_adrs;                          /* end buffer pointer */
    work = work - Vadrs2Padrs(rcv_buf);         /* receive buffer physical address */
    *rcvd_size = work;                          /* return received size */

    return 0;


}


int
tx3927sio_dma_putc(int siono, char *send_buf, int send_size)
{
    unsigned int *dma2_adrs, *pcr, dma_status;
    int i, c=0x12345678;
    reg3927sio *sioreg = (reg3927sio *)REG3927SIO_BASE;
    type3927sio *sio = &siotbl[siono];
    int next=INCIDX(sio->sndbuf_end);
                                                /* check console's sio */
    if(siono!=0 && siono!=1)
             return -1;                         /* ERROR:illegal siono */
    sioreg += siono;

    pcr   = (unsigned int *)( 0xfffee008 ) ;   /* Pin Configuration Register */
    *pcr |= 0x0000f0f0;

    dma2_adrs  = (unsigned int *)( 0xfffeb0a4 ) ; /* MCR(Master Control Register) */
     *dma2_adrs = 0x00000001;                     /* MASTEN(bit0)=on */
    dma2_adrs  = (unsigned int *)( 0xfffeb058 ) ; /* Channel Control Register(CCR2) */
    *dma2_adrs = 0x01000000;                      /* CHRST(bit24)=on(reset channel) */
    dma2_adrs  = (unsigned int *)( 0xfffeb058 ) ; /* Channel Control Register(CCR2) */
    *dma2_adrs = 0x00000000;                     /* CHRST(bit24)=off(enable channel) */
    dma2_adrs  = (unsigned int *)( 0xfffeb044 ) ; /* Source Address Register(SAR2) */
    *dma2_adrs = Vadrs2Padrs(send_buf);           /* physical memory address */
    dma2_adrs  = (unsigned int *)( 0xfffeb048 ) ;
                                                /* Destination Address Register(DAR2) */
    *dma2_adrs = 0xfffef31c+3;                  /* Transmit FIFO Channel-0(SITFIFO0) */
    dma2_adrs  = (unsigned int *)( 0xfffeb04c ) ; /* Count Register(CNAR2) */
    *dma2_adrs = send_size;                  /* transfer size(force short alignment) */
    dma2_adrs  = (unsigned int *)( 0xfffeb050 ) ;
                                          /* Source Address Incremnet Register(SAI2) */
```

```
   *dma2_adrs = 0x00000001;                    /* transfer size = 1byte */
   dma2_adrs  = (unsigned int *)( 0xfffeb054 ) ;
                                  /* Destination Address Incremnet Register(DAI2) */
   *dma2_adrs = 0x0000000;          /* no increment */
   dma2_adrs  = (unsigned int *)( 0xfffeb058 ) ; /* Channel Control Register(CCR2) */
   *dma2_adrs = 0x00011500;                    /* EXTRQ,INTENE,INTENT,XTACT,XFSZ=0*/

   dma_req_flag = 2;                           /* dma channel number 2 */
   dma_end_flag = 0;
   sioreg->sidicr |= SIDICR_TDE;               /* enable transmit DMA */

   do{
      for(i=0;i<100;i++);                      /* don't disturb DMA(memory) bus */
   }while( dma_end_flag == 0 );

   return 0;

}




int
tx3927sio_dma_putc_getc(int siono, char *send_buf, int send_size, char *rcv_buf, int
*rcvd_size)
{
   reg3927sio *sioreg = (reg3927sio *)REG3927SIO_BASE;
   type3927sio *sio = &siotbl[siono];
   int c,i,sidisr,work;
   unsigned int *dma0_adrs, *pcr, dma_status, *dma2_adrs;


   pcr   = (unsigned int *)( 0xfffee008 ) ;      /* Pin Configuration Register */
   *pcr |= 0x0000f0f0;

   dma0_adrs  = (unsigned int *)( 0xfffeb0a4 ) ; /* MCR(Master Control Register) */
   *dma0_adrs = 0x00000001;         /* MASTEN(bit0)=on */
   dma0_adrs  = (unsigned int *)( 0xfffeb018 ) ; /* Channel Control Register(CCR0) */
   *dma0_adrs = 0x01000000;         /* CHRST(bit24)=on(reset channel) */
   dma0_adrs  = (unsigned int *)( 0xfffeb018 ) ; /* Channel Control Register(CCR0) */
   *dma0_adrs = 0x00000000;         /* CHRST(bit24)=off(enable channel) */
   dma0_adrs  = (unsigned int *)( 0xfffeb004 ) ; /* Source Address Register(SAR0) */
   *dma0_adrs = 0xfffef320+3;                  /* Receive FIFO buffer 0 */
   dma0_adrs  = (unsigned int *)( 0xfffeb008 ) ;
                                        /* Destination Address Register(DAR0) */
   *dma0_adrs = Vadrs2Padrs(rcv_buf) ;         /* receive buffer physical address */
   dma0_adrs  = (unsigned int *)( 0xfffeb00c ) ; /* Count Register(CNAR0) */
   *dma0_adrs = send_size;                     /* receive buffer size */
   dma0_adrs  = (unsigned int *)( 0xfffeb010 ) ;
                                     /* Source Address Increment Register(SAI0) */
   *dma0_adrs = 0x0000000;                     /* no increment */
   dma0_adrs  = (unsigned int *)( 0xfffeb014 ) ;
                                   /* Destination Address Increment Register(DAI0)*/
   *dma0_adrs = 0x0000001;                     /* transfer size one byte */
   dma0_adrs  = (unsigned int *)( 0xfffeb018 ) ; /* Channel Control Register(CCR0) */
   *dma0_adrs = 0x00011500;                    /* EXTRQ,INTENE,INTENT,XTACT,XFSZ=0*/

   sioreg->sidicr |= SIDICR_RDE;               /* enable receive DMA */



   dma2_adrs  = (unsigned int *)( 0xfffeb0a4 ) ; /* MCR(Master Control Register) */
   *dma2_adrs = 0x00000001;         /* MASTEN(bit0)=on */
   dma2_adrs  = (unsigned int *)( 0xfffeb058 ) ; /* Channel Control Register(CCR2) */
   *dma2_adrs = 0x01000000;         /* CHRST(bit24)=on(reset channel) */
```

```
   dma2_adrs  = (unsigned int *)( 0xfffeb058 ) ; /* Channel Control Register(CCR2) */
   *dma2_adrs = 0x00000000;         /* CHRST(bit24)=off(enable channel)    */
   dma2_adrs  = (unsigned int *)( 0xfffeb044 ) ; /* Source Address Register(SAR2) */
   *dma2_adrs = Vadrs2Padrs(send_buf);           /* physical memory address */
   dma2_adrs  = (unsigned int *)( 0xfffeb048 ) ; /* Destination Address Register(DAR2)
                                                 */
   *dma2_adrs = 0xfffef31c+3;                    /* Transmit FIFO Channel-0(SITFIFO0)
                                                 */
   dma2_adrs  = (unsigned int *)( 0xfffeb04c ) ; /* Count Register(CNAR2) */
   *dma2_adrs = send_size;                       /* transfer size(force short
                                                    alignment) */
   dma2_adrs  = (unsigned int *)( 0xfffeb050 ) ;
                                        /* Source Address Incremnet Register(SAI2) */
   *dma2_adrs = 0x00000001;                      /* transfer size = 1byte */
   dma2_adrs  = (unsigned int *)( 0xfffeb054 ) ;
                                      /* Destination Address Incremnet Register(DAI2) */
   *dma2_adrs = 0x0000000;                       /* no increment */
   dma2_adrs  = (unsigned int *)( 0xfffeb058 ) ; /* Channel Control Register(CCR2) */
   *dma2_adrs = 0x00011500;                      /* EXTRQ,INTENE,INTENT,XTACT,XFSZ=0*/

   dma_req_flag = 2;                             /* dma channel number 2 */
   dma_end_flag = 0;
   sioreg->sidicr |= SIDICR_TDE;                 /* enable transmit DMA */
   do{
      for(i=0;i<100;i++);                        /* don't disturb DMA(memory) bus */
   }while( dma_end_flag == 0 );


   dma_req_flag = 0;                             /* dma channel number 0 */
   dma_end_flag = 0;
   do{
      for(i=0;i<100;i++);                        /* don't disturb DMA(memory) bus */
   }while( dma_end_flag == 0 );

   sidisr  = sioreg->sidisr;

   if(sidisr & 0xbc00)
      printf("error3(sidisr=%x) ¥n", sidisr);

   dma0_adrs  = (unsigned int *)( 0xfffeb01c ) ; /* Channel Status Register(CSR0) */
   *dma0_adrs = 0xffffffff;                      /* clear CSR0 */
   dma0_adrs  = (unsigned int *)( 0xfffeb018 ) ; /* Channel Control Register(CCR0) */
   *dma0_adrs = 0x01000000;                      /* CHRST(bit24)=on(reset channel) */

dma0_adrs  = (unsigned int *)( 0xfffeb008 ) ;
/* Destination Address Register(DAR0) */
   work = *dma0_adrs;                            /* end buffer pointer */
   work = work - Vadrs2Padrs(rcv_buf);           /* receive buffer physical address */
   *rcvd_size = work;                            /* return received size */



   return 0;


}


/*****************************************************************************
 * Get & Put one byte data with polling
 *
 * if successful, return charactor.
 * if timeout, return -1.
 * interruptible.
 * not return until get a charactor.
 *****************************************************************************/
```

```
int
tx3927sio_pol_getc(int siono)
{
   reg3927sio *sioreg = (reg3927sio *)REG3927SIO_BASE;
   type3927sio *sio = &siotbl[siono];
   int t = WAIT_TIME;
   int     c, silsr, sidisr;

   /* check console's sio */
   if(siono!=0 && siono!=1)
            return -1;                          /* ERROR:illegal siono */
   sioreg+=siono;
   while (t--) {
      if((sidisr=sioreg->sidisr) & SIDISR_RDIS){ /* Check status */
               c = sioreg->sirfifo;             /* Read data */
               sioreg->sidisr = ~SIDISR_RDIS;

               if( sidisr & SIDISR_ERI){
                        sioreg->sidisr = ~SIDISR_ERI;
               }

               if(!(sidisr & SIDISR_ERRMASK)){
                        return c;
               }

               if(sidisr & SIDISR_UBRK){        /* Break signal */
                       c = c | 0xfffb0000;
                       return c;
               }

               if(sidisr & SIDISR_UFER){        /* Frame Error */
                       c = c | 0xfffe0000;
                       return c;
               }

               if(sidisr & SIDISR_UPER){        /* Parity Error */
                       c = c | 0xfffd0000;
                       return c;
               }

               if(sidisr & SIDISR_UOER){        /* Over Run Error */
                       sioreg->sifcr = 0x00008000;
                       c = c | 0xfffc0000;
                       return c;
               }
      }
      noop(2);
   }
   return -1;                                   /* timed out */
}


int
tx3927sio_pol_putc(int siono, char c)
{
   reg3927sio *sioreg = (reg3927sio *)REG3927SIO_BASE;
   type3927sio *sio = &siotbl[siono];
   int t = WAIT_TIME;
   int silsr, sidisr;

   /* check console's sio */
   if(siono!=0 && siono!=1)
            return -1;                          /* ERROR:illegal siono */
   sioreg+=siono;
   while (t--) {
            if (sioreg->sidisr & SIDISR_TDIS) {
```

```
                    sioreg->sitfifo = c;
                    sioreg->sidisr = ~SIDISR_TDIS;
                    return 0;
            }
            noop(2);
   }
   return -1;                                   /* timeout */
}



int
tx3927sio_getc(int siono){
   int(*func)() = siotbl[siono].getc;
   if(func)
   return func(siono);
   return -1;
}
int
tx3927sio_putc(int siono, char c){
   int(*func)() = siotbl[siono].putc;
   if(func)
   return func(siono,c);
   return -1;
}



/*********************************************************************
 * Wait until all data is sent
 *********************************************************************/
int
tx3927sio_all_sent(int siono)
{
   reg3927sio *sioreg = (reg3927sio *)REG3927SIO_BASE;
   /* check parameter */
   if(siono!=0 && siono!=1)
      return -1;                                /* ERROR: illegal siono */
   sioreg+=siono;
   while(!(sioreg->sicisr & SICISR_TXALS));
}
```

- File name: 3927sio.h

```
/*=========================================================================
 * $Id$
 *-------------------------------------------------------------------------
 * Copyright(C) 1998-1999 TOSHIBA CORPORATION  All rights reserved.
 *=========================================================================
 */

#ifndef __TX3927SIO_H
#define __TX3927SIO_H

#define CPU133M
/* imclock */
#ifndef CPU_CLOCK
#ifdef CPU133M
#define CPU_CLOCK 132710400
#else
#define CPU_CLOCK 58982400
#endif
#endif


/* buffer size between interrupt and task. multiple of 2 */
#define SIO_RCVBUFSZ 4096
#define SIO_SNDBUFSZ 4096

#ifdef USE_UDEOS

#ifndef SIO_SEMID_BASE
#define SIO_SEMID_BASE 2
#endif
#define SIO_SEMID0   (SIO_SEMID_BASE)
#define SIO_SEMID1   (SIO_SEMID_BASE+1)
#define SIO_SEMID0TX   (SIO_SEMID_BASE+2)
#define SIO_SEMID1TX   (SIO_SEMID_BASE+3)

CRE_SEM(SIO_SEMID0,"sem_sio0",TA_TFIFO,0)
CRE_SEM(SIO_SEMID1,"sem_sio1",TA_TFIFO,0)
CRE_SEM(SIO_SEMID0TX,"sem_sio0tx",TA_TFIFO,0)
CRE_SEM(SIO_SEMID1TX,"sem_sio1tx",TA_TFIFO,0)

/* define for interrupt handler */
/* not required if you use only polling */
DEF_INT(INTNO_SIO0,TA_HLNG,tx3927sio0_int)
/* DEF_INT(INTNO_SIO1,TA_HLNG,tx3927sio1_int) */
#endif

/* type */
#define SIO_DISABLE      0x00000000
#define SIO_TXPOL        0x00000001
#define SIO_TXINT        0x00000002
#define SIO_TXDMA        0x00000004
#define SIO_RXPOL        0x00000010
#define SIO_RXINT        0x00000020
#define SIO_RXDMA        0x00000040

#define SIO_PARITY_ODD   0x00000100
#define SIO_PARITY_EVEN  0x00000200
#define SIO_STOP2        0x00000400
#define SIO_7BIT         0x00000800

#define SIO_HWFLOW       0x00001000

#define SIO_RFIFO_1      0x00002000
```

```
#define SIO_RFIFO_4      0x00004000
#define SIO_RFIFO_8      0x00008000
#define SIO_RFIFO_12     0x00010000

#define SIO_TFIFO_1      0x00020000
#define SIO_TFIFO_4      0x00040000
#define SIO_TFIFO_8      0x00080000

extern int tx3927sio_init(int siono,int type,int baud);
extern int tx3927sio_pol_getc(int siono);
extern int tx3927sio_pol_putc(int siono, char c);
extern int tx3927sio_getc(int siono);
extern int tx3927sio_putc(int siono, char c);
extern int tx3927sio_int_getc(int siono);
extern int tx3927sio_int_putc(int siono, char c);
extern int tx3927sio_all_sent(int siono);

#endif /* __TX3927SIO_H */
```

A.3.3 DMA Controller

The following sample program performs memory-to-memory transfer. It is used for an Ethernet driver to copy the contents of a receive buffer. The sample shown in "A.3.2 SIO" contains a DMA example using on-chip serial interface.

- File name: tx3927dma.c

```
/*---------------------------------------------------------------------------
  Function name: buff_copy
  Function: Copy receive data buffer
  Input:
  Output: None
---------------------------------------------------------------------------*/
typedef Uint32 unsigned int
void buff_copy(char *userbuff8, char *tc35815buff8, Uint32 recvsize)
{
   short       *tc35815buff16,*userbuff16;
   Uint32      *tc35815buff32,*userbuff32,*endbuff,*endbuff1,*endbuff2;
   Uint32      *dma3_adrs,dma_status;
   register unsigned int work32,work33,short_count,char_count,i;

   dma3_adrs  = (Uint32*)( 0xfffeb0a4 ) ;     /* MCR(Master Control Register)  */
   *dma3_adrs = 0x00000001;                   /* MASTEN(bit0)=on */
   dma3_adrs  = (Uint32*)( 0xfffeb078 ) ;     /* Channel Control Register(CCR3) */
   *dma3_adrs = 0x01000000;                   /* CHRST(bit24)=on(reset channel) */
   dma3_adrs  = (Uint32*)( 0xfffeb078 ) ;     /* Channel Control Register(CCR3) */
   *dma3_adrs = 0x00000000;         /* CHRST(bit24)=off(enable channel) */
   dma3_adrs  = (Uint32*)( 0xfffeb064 ) ;     /* Source Address Register(SAR3) */
   *dma3_adrs = Vadrs2Padrs(tc35815buff8);    /* physical address */
   dma3_adrs  = (Uint32*)( 0xfffeb068 ) ;     /* Destination Address Register(DAR3) */
   *dma3_adrs = Vadrs2Padrs(userbuff8+NE_ETHER_ALIGN);   /* physical address */
   dma3_adrs  = (Uint32*)( 0xfffeb06c ) ;     /* Count Register(CNAR3) */
   *dma3_adrs = (recvsize+1) & 0xffe;         /* transfer size(force short
   alignment) */
   dma3_adrs  = (Uint32*)( 0xfffeb070 ) ;     /* Source Address Inclement
   Register(SAI3) */
   *dma3_adrs = 0x00000002;                   /* 2byte */
   dma3_adrs  = (Uint32*)( 0xfffeb074 ) ;     /* Destination Address Increment
   Register(DAI3)*/
   *dma3_adrs = 0x00000002;                   /* 2byte */
   dma3_adrs  = (Uint32*)( 0xfffeb078 ) ;     /* Channel Control Register(CCR3) */
   *dma3_adrs = 0x00000104;
                         /* EXTRQ=0,INTRQD=0,INTENT=0,XTACT=1,XFSZ=001,ONEAD=0 */
   do{
        for(i=0;i<125;i++);                   /* don't disturb DMA(memory) bus */
       dma3_adrs  = (Uint32*)( 0xfffeb07c );  /* Channel Status Register(CSR3) */
       dma_status = *dma3_adrs;               /* read status register */
       }while( dma_status != 0x60 );          /* NCHNC & NTRNFC(Normal Transfer
                                                 Completion) */
       *dma3_adrs = 0xffffffff;               /* clear CSR3 */


   return;

}
```

A.3.4 PIO

This is a simple example to turn on the LEDs on the JMR-TX3927. You must set the Pin Configuration register (PCFG), Open-Drain Control register (XPIOOD) and Direction Control register (XPIODIR) to enable the PIO.

- File name: pio.c

```
#include    <stdio.h>
#include    <setjmp.h>
#include    <string.h>
#include    "console.h"
#include    "menu.h"


/* Example of using TX3927 PIO switch and LEDs */

/*
   PIO Direction Control register address     0xfffef508
   PIO Data Out register Address              0xfffef500
   PIO Data 15,14,13,12bit DIP Switches
            1 -> LED off
            0 -> LED on
   PIO Data 11,10bit LEDs
            Switch ON  -> 0
            Switch OFF -> 1
*/

/* LED 12bit off */

int initPIO(void){

   volatile int *pcfg,*xpiood;

    pcfg   = (volatile int *)0xfffee008;
    xpiood = (volatile int *)0xfffef50c;

    *pcfg   = 0x08fc3131;
    *xpiood = 0x00000000;
   return 0;

}

int led1(void)
{
   volatile int *pdir,*pdo;

    pdir = (volatile int *)0xfffef508;
    pdo  = (volatile int *)0xfffef500;

    *pdir = 0x0000f000;
    *pdo  = 0x00001000;
   return 0;
}
```

A.4 PCI Controller

A.4.1 Initializing the PCI Controller

Following is a sample for initializing the PCI Controller. The sample program performs the following using the JMR-TX3927:

- Detects PCI devices connected to the bus.
- Sets memory addressing and interrupts.
- Sets the address space for master and slave accesses from the TX3927.

This sample code does not allow you to configure a PCI bridge device because it only supports configuration type 0. It can detect multifunction devices, but does not perform any actual processing for them. The JMR-TX3927 board does not have a dynamic interrupt routing function; it uses fixed interrupt numbers that are assigned when the board is designed.

The program structure is as follows:

```
Init_PCI                      : Initialize the PCI Controller.
    +-- find_device           : Detect devices connected to the TX3927 PCI bus.
    |      +--- set_pci_irq       : Create an interrupt table.
    |      +--- get_addr_size     : Detect the requested address space size.
    +-- pci_mem_space_mapping     : Allocate address space.
           +--- sort_table        : Sort by address size.
                   +--- swap_table: Swap resource maps.
```

The sample also uses the following subroutines:

```
get_pci_config          : PCI bus configuration (write access)
put_pci_config PCI      : Bus configuration (read access)
master_abort_check      : Master abort detection
```

- File name: pci3927.c

```
/************************************************************
   Copyright (C) 1999 TOSHIBA Corporation

   TX3927 PCI Initialize function
   module PCI3927.c

   1999.05.05 Akira.Tanaka
   1999.07.19 chenge C file
$Id: PCI3927.c 1.2 1999/07/19 01:29:45 tanaka Exp $
************************************************************/

#include          <stdio.h>
#include          "cosbd_27.h"
#include          "intcosmp27.h"
#include          "tx3927.h"

/************************************************************/
#define TC35815         0xd
#define max_device      22

#define TX3927_IO_PA    0x08000000     /* G-Bus I/O Base Address */
#define TX3927_MEM_PA   0x04000000     /* G-Bus Memory Base Address */
#define TX3927_END_PA   0x03FFFFFF     /* 64MB space */
#define PCI_IO_PA       0x08000000     /* PCI I/O Base Address */
#define PCI_MEM_PA      0x04000000     /* PCI Memory Base Address */
```

```
#define PCI_END_PA      0x03FFFFFF     /* 64MB space */
#define TX3927_VA       0xa0000000     /* Virtual Address Offset */
#define START_PA        0x00000000     /* target start address       */
#define MEM_OFFSET      0x02000000     /* target space offset(4 i/o) */
/* Address Mapping Image with Above Definitions */
/*                                                                   */
/* TX3927   p-address                    PCI Bus                     */
/* v-address    MEMORY   (GBus addr)                p-address      */
/* ac00_0000 |---------| 0c00_0000                 |---------|     */
/*           |  pcic   |                           | PCI bus |     */
/*    <64MB> | IO space|  -------------->   <64MB> | IO space|     */
/* a800_0000 |---------| 0800_0000        0800_0000|---------|     */
/*           |   pcic  |<TX3927_IO_PA> <PCI_IO_PA> | PCI bus |     */
/*    <64MB> |MEM space|  -------------->   <64MB> |MEM space|     */
/* a400_0000 |---------| 0400_0000        0400_0000|_________|     */
/*           |         |<TX3927_MEM_PA><PCI_MEM_PA>|         |     */
/*           |         |                           |         |     */
/*           |         |                           |         |     */
/* a200_0000 |---------| 0200_0000        0200_0000|---------|     */
/*           | SDRAM   |        <MEM_OFFSET>       |   PCI   |     */
/*    <32MB> | space   |   <--------------- <32MB> |  Target |     */
/* a000_0000 |         | 0000_0000        0000_0000|  Space  |     */
/*           ~~~~~~~~~~          <START_PA>        ~~~~~~~~~~     */


/*************************************************************/
/* TX3927 PCIC (Internal) Interrupt Signal (Fixed)           */
/*************************************************************/
#define PCI_INT10       0xa

/*************************************************************/
/* TX3927 PCI Resource Table Definitions                     */
/* Declare a Structure of type device                        */
/*************************************************************/
struct device {
   int             id;                /* Equal to IDSEL */
   int             bus_num;           /* Bus Number, 0 Only */
   unsigned short  vender_id;         /* Vendor ID */
   unsigned short  device_id;         /* Device ID */
   unsigned short  subsvid;           /* Subsystem Vendor ID */
   unsigned short  subsid;            /* Subsystem ID */
   int             io_addr_size;      /* I/O Request Address Size */
   int             mem_addr_size;     /* Memory Request Address Size */
   int             io_base_addr;      /* Assigned I/O Base Address */
   int             mem_base_addr;     /* Assigned Memory Base Address */
   int             int_num;           /* Interrupt Signal Number (INT of TX3927) */
   int             flg;               /* Used Flag ("1" = Used) */
};

/*************************************************************/
/*************************************************************/
struct IRQtable {
   int             bus_num;           /* Bus Number */
   int             dev_num;           /* Device Number */
   int             int_pin;           /* Interrupt Pin 0=INTA..3=INTD */
   int             int_line;          /* Interrupt Line 0x0=INT0..0xF=INT15 */
   int             slot_num;          /* Slot Number */
};

/*************************************************************/
/* TX3927 board PCI Interrupt Resouce Table                  */
/*************************************************************/
struct IntResTbl {
   int             INTR_NUM;          /* Interrupt Signal Number (TX3927) */
   int             DEV_ID;            /* Device Number Connected to This Interrupt
                                         Signal */
```

```
   int             SLOT_NUM;           /* Slot Number (0 = Platform) */
};

struct device   sd[max_device];        /* Declare Global Variables */
struct device   tmp_sd[max_device];    /* Sort Resource Table */
struct IRQtable irq[max_device];       /* IRQ Resource Table */

/* TX3927 board PCI Interrupt Resource Table  */
/* Set 0xff to ID in Last Table as Identifier */
struct IntResTbl inttbl[8] = {
   {0, 0x0f, 0},/* INT[0] PCI Card CN (INTA) or PCI Card Edge (INTC) *
   {1, 0x12, 3},
   {1, 0x13, 2},
   {1, 0x14, 1},
   {3, 0x0d, 0},/* INT[3] TC35815 or 10M Ether(from I/O board)       */
   {0, 0xff, 0}
};
   /* Set Interrupt Connection Information for TX3927 Board */
   /* Connection of TX3927 External Interrupt int[3:0]  */
   /* TX3927        Connected Device                    */
   /* INT[0] PCI Card CN (INTA) or PCI Card Edge (INTC) */
   /* INT[1] IO-C                                       */
   /* INT[2] ISA-C (from I/O board)                     */
   /* INT[3] TC35815 or 10M Ether(from I/O board)       */
   /* INT[10]TX3927 PCIC (INTA<- Mistake 05/11/99)      */
   /* With TX3927, set interrupt pins that do not have INTA to 0 */
   /* INT[2:0] need not be set                          */
   /* INT[3] setting                                    */

/************************************************************/
/* TX3927 PCIC Initialization                               */
/************************************************************/
void
Init_PCI(int mode)
{
   /* TX3927 PCIC Initialization */
   CPUReg->pci_conf.PCISTAT = MEN;     /* Master Enable */
   CPUReg->pci_lsp.IOMAS = 0xfc000000; /* PCI I/O Size 64MB */
   CPUReg->pci_lsp.MMAS  = 0xfc000000; /* PCI Memory Size 64MB */
   CPUReg->pci_lsp.LBC   = EPCAD;      /* Disable TX3927 Target Configuration */

   /* INT[10] setting (TX3927 as a target, set separately from PCI device) */
   CPUReg->pci_conf.ML   = 0xffff010a;

   /* Detect Devices Connected to PCI */
   find_device(mode);

   CPUReg->pci_iconf.IPBMMAR  = PCI_MEM_PA;    /* PCI Memory Base Address */
   CPUReg->pci_iconf.ILBMMAR  = TX3927_MEM_PA; /* G-Bus Memory Base Address */
   CPUReg->pci_iconf.IPBIOMAR = PCI_IO_PA;     /* PCI I/O Base Address */
   CPUReg->pci_iconf.ILBIOMAR = TX3927_IO_PA;  /* G-Bus I/O Base Address */

   /* Assignment of TX3927 target devices requires a consideration */
   /* I/O space is 256 bytes (from 0200_0000)              */
   /* Note that this is set outside TX3927 board memory space */

   CPUReg->pci_lsp.MBAS     = 0xfe000000;      /* Target Memory Size 32MB */
   CPUReg->pci_lsp.IOBAS    = 0xffffff00;      /* Target I/O Size 256 bytes */
   CPUReg->pci_conf.MBA     = START_PA;        /* Target Memory Base Address */
   CPUReg->pci_conf.IOBA    = START_PA + MEM_OFFSET;  /* Target I/O Base Address */
   CPUReg->pci_tconf.TLBMMA  = START_PA;              /* Target G-Bus Memory Address*/
   CPUReg->pci_tconf.TLBIOMA = START_PA + MEM_OFFSET; /* Target G-Bus I/O Address */

   /* Enable PCIC Memory and I/O Access, Disable Configuration */
   CPUReg->pci_lsp.LBC = EPCAD | ILMDE | ILIDE;
```

```
    /* Assign Addresses to Detected Devices */
    pci_mem_space_mapping();
    if (master_abort_check() != 0) {
            printf("¥n Config Register Access Error!! after mapping");
            CPUReg->pci_conf.PCISTAT = RECMA | MEN;
    }

    /* Arbiter Settings */
    /* CPUReg->pci_ext.REQ_TRACE = 0x73737373; */
    CPUReg->pci_ext.REQ_TRACE = 0x73210731;
    CPUReg->pci_ext.BM = 0x00000000;  /* clear */
    /* Enable Arbiter */
    CPUReg->pci_ext.PBAPMC = 0x00000002;
}

/*****************************************************************/
/* Detecting devices connected to TX3927 PCI bus                 */
/*    If no device detected, set sd[idsel].id to 0xff.           */
/*    If device detected, perform multifunction device detection. */
/*       If multifunction device is detected, do nothing.        */
/*    Collect interrupt information. Add INT number to           */
/*    resource table sd.                                         */
/*    Information table entries created when devices are         */
/*    detected should be arranged sequentially without any gap   */
/*    (not necessary to match IDSEL number).                     */
/*    Add SubsystemVenderID and Subsystem ID.                    */
/*****************************************************************/
int
find_device(int mode)
{
    int             idsel, i;
    int             vid, htype;
    int             intp, intl;
    int             num;
    int             base_addr;

    num = 0;

    for (i = 0; i < max_device; i++) {
            idsel = i;

            /* read vender ID */
            vid = 0xffff & get_pci_config(idsel, 0, 0x00);

            if ((vid != 0xffff) && (master_abort_check() == 0)) {
                  sd[num].id        = idsel;
                  sd[num].vender_id = 0xffff &  get_pci_config(idsel, 0, 0x00);
                  sd[num].device_id = 0xffff & (get_pci_config(idsel, 0, 0x00) >> 16);
                  sd[num].subsvid   = 0xffff &  get_pci_config(idsel, 0, 0x2c);
                  sd[num].subsid    = 0xffff & (get_pci_config(idsel, 0, 0x2c) >> 16);
                  sd[num].bus_num   = 0;
                  sd[num].flg       = 0;

                  /* read header type */
                  htype = 0x80 & (get_pci_config(idsel, 0, 0x08) >> 16);
                  if (htype != 0)
                          printf("¥n this device is multi device ");

                  /* Read Interrupt Pin. Set INT Number Which Matched ID
                     (*InterruptLine) */
                  /* Read int_num */
                     set_pci_irq(num, mode);

                  /* read Interrupt line */
                  intl = 0x0f & get_pci_config(idsel, 0, 0x3c);
                  sd[num].int_num = intl;
```

```
                        /* base address size check */
                        /* check 0x10 may be I/O base address register */
                        base_addr = get_addr_size(idsel, 0x10);
                        if ((base_addr & 0x00000001) && (base_addr != 0))
                                sd[num].io_addr_size = base_addr;
                        else {
                                if (base_addr != 0)
                                        sd[num].mem_addr_size = base_addr;
                                else {
                                        sd[num].mem_addr_size = 0xffffffff;
                                        sd[num].io_addr_size  = 0xffffffff;
                                }
                        }

                        /* check 0x14 may be memory base address register */
                        base_addr = get_addr_size(idsel, 0x14);
                        if ((base_addr & 0x00000001) && (base_addr != 0))
                                sd[num].io_addr_size = base_addr;
                        else {
                                if (base_addr != 0)
                                        sd[num].mem_addr_size = base_addr;
                                else {
                                        sd[num].mem_addr_size = 0xffffffff;
                                        sd[num].io_addr_size  = 0xffffffff;
                                }
                        }

                        num = num + 1;
                } else {
                        /* Restore Status If No Device Found */
                        CPUReg->pci_conf.PCISTAT = RECMA | MEN;
                        if (master_abort_check() != 0)     /* Recheck */
                                printf("¥n Ireagal status !!");
                }
        }

        /* Assign 0xff to ID of Last Table (Identifier) */
        sd[num].id = 0xff;
}

/************************************************************
 * Definition of Interrupt Resource Table */                 *
 ************************************************************/
void
set_pci_irq(int num, int mode)
{
        int             i;
        int             past_data; /* Status Before Change */
        int             undef;     /* When Interrupt Resource Cannot Be Defined: 1 */

        i = 0;
        undef = 0;

        while (inttbl[i].DEV_ID != sd[num].id) { /* Search Table for Matching Device Number
                                                    */
                if (inttbl[i].DEV_ID == 0xff) { /* End Search at End of Table */
                        undef = 1;
                        if (mode != 0)
                                printf("¥n %02x is not defined! please check Interrupt
Resource Table ", sd[num].id);
                        break;     /* Warning if No Match Was Found in Resource Table */
                }
                i++;
        }
```

```
    /* Set Interrupt Resource Table with Relevant Device Number */
    if (undef == 0) {
           irq[i].bus_num  = 0;
           irq[i].dev_num  = inttbl[i].DEV_ID;
           irq[i].int_pin  = (get_pci_config(irq[i].dev_num, 0, 0x3c) >> 8) & 0x07;
           irq[i].int_line = 0x0f & inttbl[i].INTR_NUM;
           irq[i].slot_num = inttbl[i].SLOT_NUM;
           /* Write Interrupt Signal to Interrupt_Line Register */
           /* Word Write: Mask Other Registers (Read and Write) */
           past_data = get_pci_config(irq[i].dev_num, 0, 0x3c) | (0x000000ff &
           irq[i].int_line);
           /* printf("¥n put_pci_config reg03h"); */
           put_pci_config(irq[i].dev_num, 0, 0x3c, past_data);
    }
}

/*************************************************************************/
/* Detect requested size for address space using base address register.     */
/* Write 0xffffffff to base address register. Detect requested size         */
/* according to successfully written bits. If all zeros, reserved register. */
/* If bit 0 is 1, I/O base register. If bit 0 is 0, memory base register.   */
/* xx  Search registers for 10h, 14h, 18h, 1ch, 20h, and 24h xx              */
/* Return value is size.                                                    */
/*************************************************************************/
int
get_addr_size(int idsel, int reg)
{
    int             ret;
    /* int reg[6]={0x10,0x14,0x18,0x1c,0x20,0x24}; */

    put_pci_config(idsel, 0, reg, 0xffffffff);
    ret = get_pci_config(idsel, 0, reg);

    return ret;

}

/*************************************************************/
/* Allocate Address Space According to Requested Space Size   */
/*************************************************************/
int
pci_mem_space_mapping(void)
{
    int             mem_space;     /* Required Memory Space */
    int             io_space;      /* Required for I/O Space */
    int             next_io_addr;  /* Lowest I/O Address Available After Mapping */
    int             next_mem_addr; /* Lowest Memory Address Available After Mapping */
    int             i;
    int             ret;

    i = 0;
    next_io_addr = PCI_IO_PA;
    next_mem_addr = PCI_MEM_PA;

    /* Preprocessing for Mapping: Sort Table */
    /* IO space size sort */
    sort_table(0);           /* Sort Completed */
    /* IO space mapping */
    while (sd[i].id != 0xff) {
             if (sd[i].io_addr_size != 0xffffffff)
                   io_space = ~(0xfffffffc & sd[i].io_addr_size) + 1;
             sd[i].io_base_addr = next_io_addr;
             next_io_addr = next_io_addr + io_space;
             if (next_io_addr > PCI_END_PA + PCI_IO_PA)
                      printf("¥n IO mapping failed!! ");
             i++;
```

```
    }

    /* MEM space size sort */
    sort_table(1);   /* Sort Completed */
    i = 0;
    /* MEM space mapping */
    while (sd[i].id != 0xff) {
            if (sd[i].mem_addr_size != 0xffffffff)
            mem_space = ~(0xfffffff0 & sd[i].mem_addr_size) + 1;
            sd[i].mem_base_addr = next_mem_addr;
            next_mem_addr = next_mem_addr + mem_space;
            if (next_mem_addr > PCI_END_PA + PCI_MEM_PA)
                    printf("¥n memory mapping failed!! ");
            i++;
    }
}


/*****************************************************************/
/* Sort by resource information address size in descending order */
/* Toggle between memory and I/O using switch                    */
/*****************************************************************/
int
sort_table(int sw)
{
    int             i, j;

    i = 0;
    if (sw == 0) { /* IO space */

            while (sd[i].id != 0xff) {
                    if (i != 0) {
                            j = 0;
                            while (j < i) {
                                    if (~sd[i].io_addr_size > ~sd[j].io_addr_size) {
                                            swap_table(j, i);
                                    } else
                                            j++;
                            }
                    }
                    i++;
            }

    } else {        /* MEM space */
            while (sd[i].id != 0xff) {
                    if (i != 0) {
                            j = 0;
                            while (j < i) {
                                    if (~sd[i].mem_addr_size > ~sd[j].mem_addr_size) {
                                            swap_table(j, i);
                                    } else
                                            j++;
                            }
                    } i++;
            }

    }
}

/*************************************************************/
/* Swap resource maps                                        */
/*************************************************************/
void
swap_table(int p, int c)
{
    struct device   tmp;
```

```
   tmp = sd[p];
   sd[p] = sd[c];
   sd[c] = tmp;

}


/*************************************************************/
/* PCI bus configuration (write access)                      */
/* Device number, function number, register number, write data */
/*************************************************************/
int
put_pci_config(int id, int func, int reg, int buf)
{
   int             confreg;

   confreg = 0x00000000 | ((id & 0x1f) << 11) | ((func & 0x7) << 8) | reg;
   CPUReg->pci_lsp.ICAR = confreg;
   CPUReg->pci_lsp.ICDR = buf;

}

/*************************************************************/
/* PCI bus configuration (read access)                       */
/* Device number , function number, register number          */
/* Return value: read data                                   */
/*************************************************************/
int
get_pci_config(int id, int func, int reg)
{
   int             ret, confreg;

   confreg = 0x00000000 | ((id & 0x1f) << 11) | ((func & 0x7) << 8) | reg;
   CPUReg->pci_lsp.ICAR = confreg;
   ret = CPUReg->pci_lsp.ICDR;
   return ret;
}

/*************************************************************/
/* Detect master abort caused by access to nonexistent addres  */
/*************************************************************/
int
master_abort_check(void)
{
   int             ret;
   ret = CPUReg->pci_conf.PCISTAT & RECMA;
   return ret;
}
```

Appendix B. Thermal Characteristics

B.1 Outline of Thermal Resistance of Packages with Fins

The thermal resistance of a package having fins is generally defined as:

$$\theta ja = \theta jc + \theta cf + \theta fa$$

where:

θja: Thermal resistance between the junction and the ambient

θjc: Thermal resistance between the junction and the package case surface

θjf: Thermal resistance between the package case surface and the fin surface

θfa: Thermal resistance between the fin surface and the ambient

Package thermal resistance is calculated as follows:

$$\theta ja: (Tj - Ta)/P$$

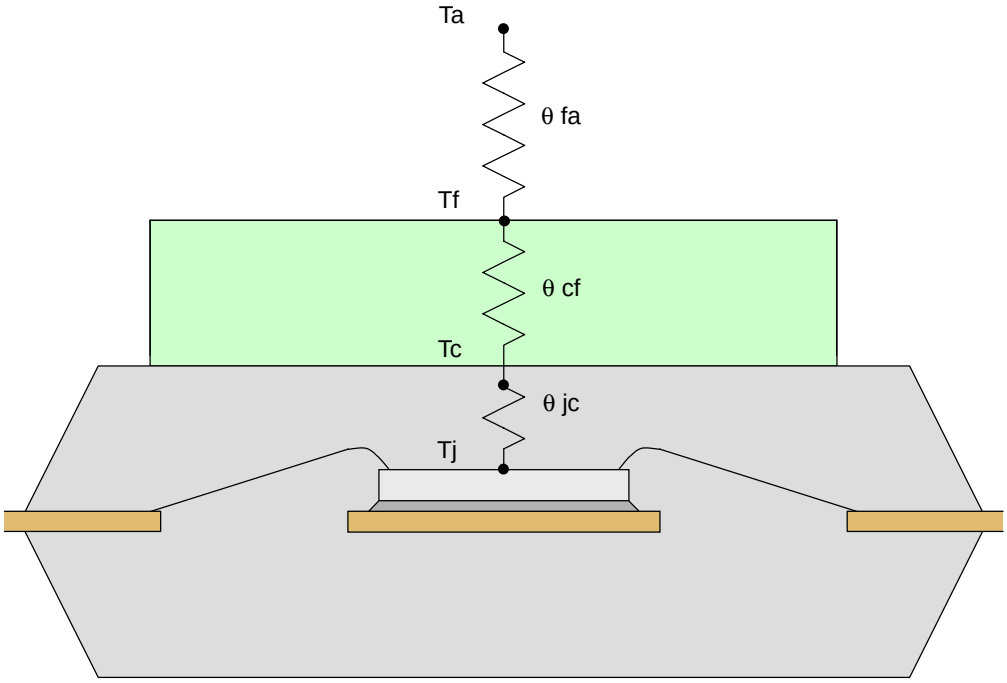


Figure B.1.1 Package with Fins

B.2 Outline of Thermal Resistance Measurement

Thermal resistance is the ability of the package to dissipate internally generated heat out of the package. Lower thermal resistance indicates better heat removal performance; i.e., packages with lower thermal resistance permits higher die power dissipation. The unit of measure for thermal resistance is usually °C/W.

Three thermal resistances are used: θja, θjc and θca. Historically, the junction-to-ambient thermal resistance (θja) is expressed as the sum of a junction-to-case thermal resistance (θjc) and a case-to-ambient (θca) thermal resistance.

$$\theta ja = \theta jc + \theta ca \quad \ldots (1)$$

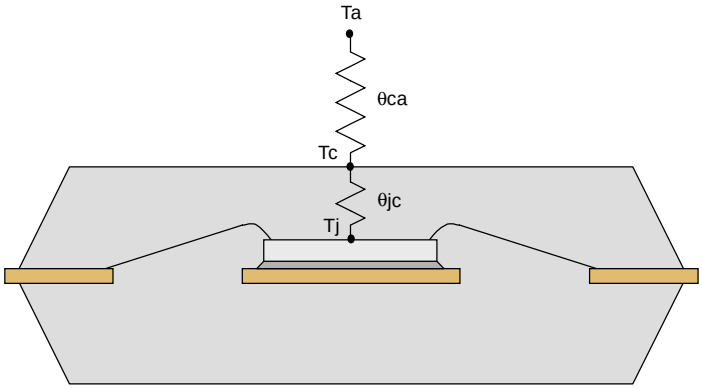


Figure B.2.1 Package Thermal Resistance

To obtain θja and θjc characteristics of a package, the chip's junction temperature (Tj) must be determined. It is impossible, however, to directly measure the junction temperature. The simplest and most common parameter used for temperature sensing within a device is the junction voltage across a forward-biased temperature-sensitive diode. Figure B.2.2 shows the slope of a typical calibration line for such a diode. [Refer to SEMI G46-88]

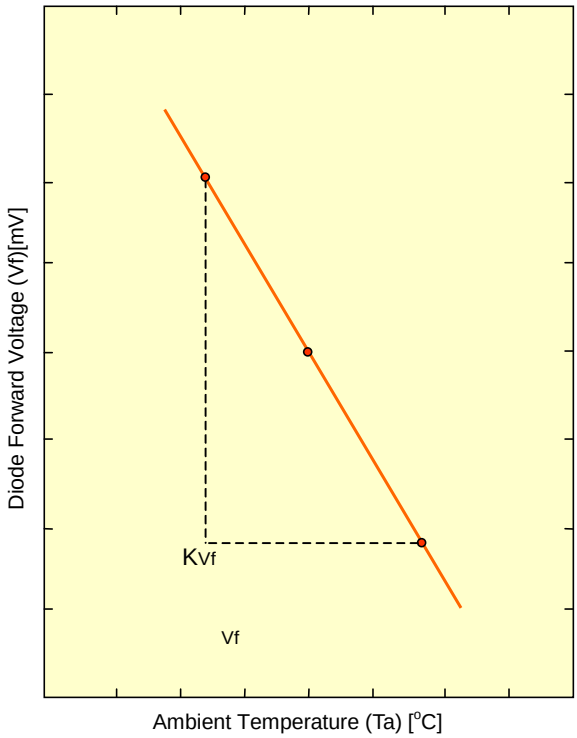


Figure B.2.2 Temperature-Sensing Diode Calibration Line

To create a calibration line, the device is put in a constant temperature bath and heated for a specified period of time. Then a fixed forward current (Im) is applied to the diode, and the diode forward voltage (Vf) is measured. This voltage usually varies linearly with temperature over a range suitable for making thermal measurements. The device is initially heated to ensure that a steady-state condition is produced in which Ta equals Tj. After measurements are taken at several (at least two) temperatures, a relational constant (Kvf) is calculated which defines the temperature dependency of the sensing diode forward voltage (Δv/Δt). Once Kvf is known, the chip's junction temperature (Tj) can be easily determined by making Vf measurements.

Junction-to-ambient thermal resistance (θja) is calculated by dividing the difference in temperature between the junction (Tj) and the ambient atomosphere (Ta) by the chip's power dissipation: (P)

$$\theta ja = \frac{Tj - Ta}{P} \quad ...(2)$$

Likewise junction-to-case thermal resistance (θjc) is calculated y dividing the difference in temperature between the junction, and the case (package) top surface (Tc) by the chip's power dissipation (P).

When making thermal measurements on active devices (as opposed to specially designed thermal test die), input protection diodes are used both as heating source and temperature sensor. Thermal measurements consist of three steps: 1) Heating power (P) is applied to the diode under test (DUT); 2) a fixed Measurement Current (Im) is applied to the DUT; and 3) the forward-biased voltage across the diode connection (Vf) is measured as the temperature-sensitive parameter. The use of electronic switching allows the power-turn-off-to-measurement completion time to be very short. See Figure B.2.3 and Figure B.2.4. [Refer to SEMI G46-88.]

The TH-256 ΔmV tester, manufactured by Kuwano Denki, is used for measurements.

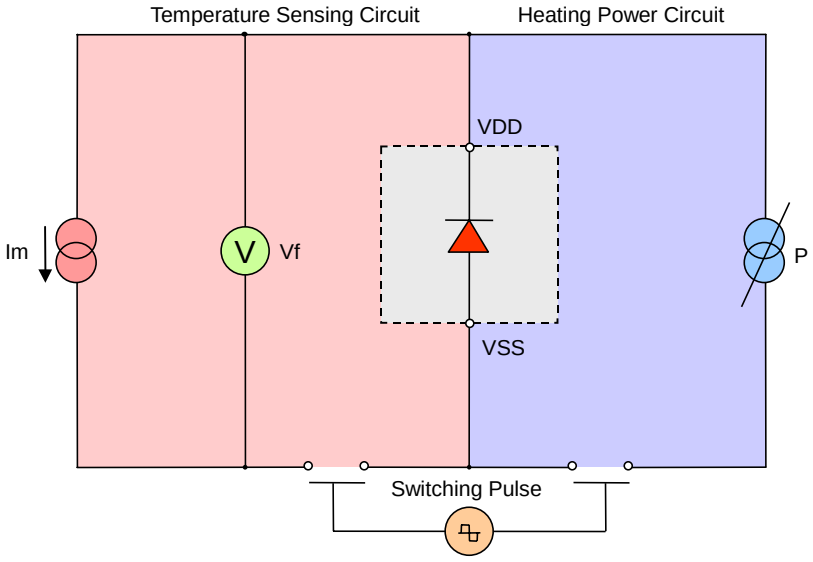


Figure B.2.3 Measuring circuit

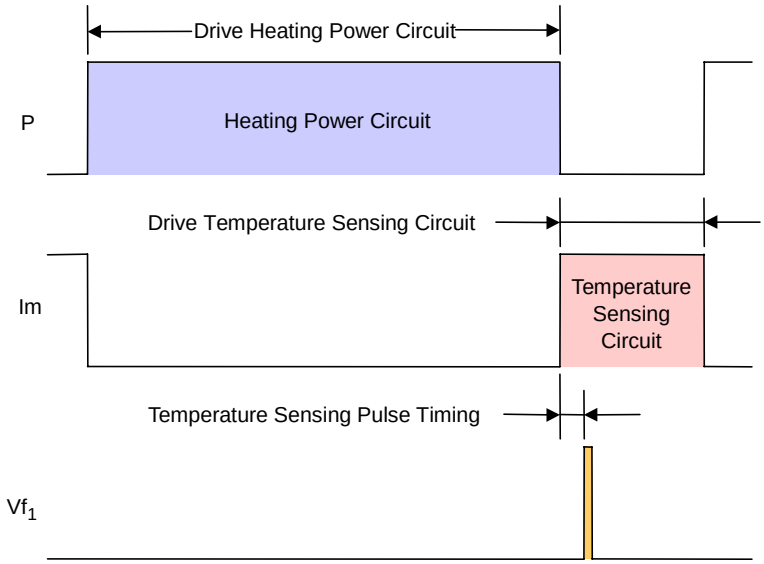


Figure B.2.4 Switching Timing

Thermal measurements are based on the change in Vf; thus the initial temperature-sensing diode junction voltage must be determined, with no heating power applied (i.e., at room temperature). Since this is used as a reference point, it is important to wait for an appropriate amount of time for a steady-state condition (Vf_0 = Ta) to occur before taking measurements.

Next, power is applied to the DUT. Thermal measurements require careful attention to the amount of time that power is applied with heating pulses, because the heat generated at the junction takes a finite amount of time to propagate outward to the surrounding environment and Vf_1 = Tj is reached. The temperature change is calculated from the difference between the initial voltage (Vf_0) and the value after heating, using the diode's Kvf constant. Thermal resistance is defined as the change in temperature divided by the power dissipation (P) that caused the temperature change.

$$\theta ja = \frac{Tj - Ta}{P} = \frac{(Vf_1 - Vf_0) / K_{Vf}}{P} \quad \text{.... (3)}$$

Figure B.2.5 shows an outline of the apparatus used to make junction-to-case thermal resistance (θjc) measurements using a fluid bath. A thermocouple is attached to the package surface. The package is submerged in the fluid bath, which is stirred constantly in order to keep the fluid temperature uniform. The fluid temperature is also kept constant while heating power is applied. Other measuring methods and equipment, including application of heating power, are the same as for θja measurements. [Refer to SEMI G30-88 and G43-87.]

Because θjc measurements require a large-sized apparatus including a fluid bath, thermal measurements may be performed in the air, with a thermocouple attached to the package surface. The resulting thermal resistance may loosely be referred to as θjc, although it is not actually θjc. SEMI defines it as φjt. Toshiba measures the thermal resistance in the air, that is, only measures φjt. [Refer to SEMI G69-96.]

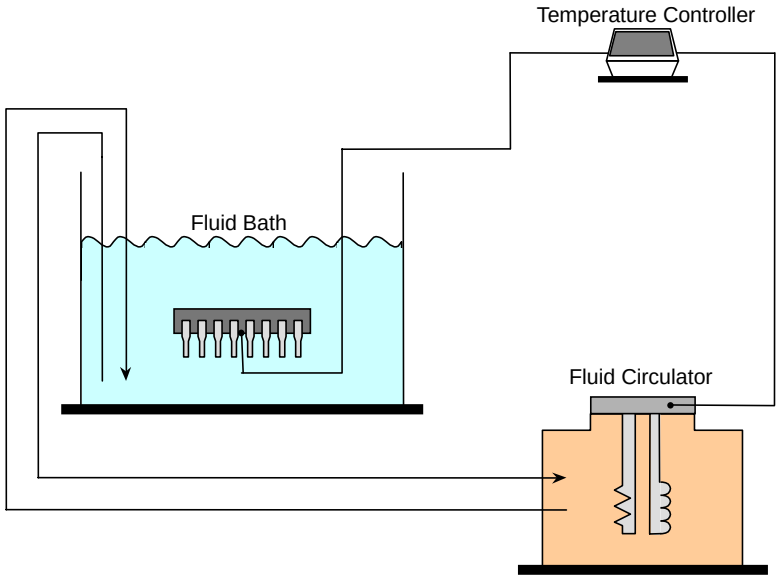


Figure B.2.5 Fluid Bath Testing Environment for Junction-to-Case Thermal Resistance (θjc) Measurements

B.3 Example Calculations for Designing Fins

The following formulas are used to select fins for a given chip, package, ambient temperature, and power dissipation:

Package temperature (Tc): 70 °C

Ambient temperature (Ta): 60 °C

Power dissipation (P): 1 W

$\theta ca = (Tc - Ta) / P$

$= (70 - 60) / 1$

$= 10$ °C/W

Fin thermal resistance:

$\theta cf = 0.17$°C/W (from a catalog of Mizutani Denki Kogyo)

$\theta fa = \theta ca - \theta c - \theta cf$

$= 10 - 0.17$

$= 9.83$ °C/W

Hence, you can use fins having thermal resistance of 9.83°C/W or less.

* Toshiba cannot attach fins to devices for you because reliability issues are involved.