- **Dual-core System Integrating an ARM7TDMI® ARM® Thumb® Processor Core and a mAgic DSP for Audio, Communication and Beam-forming Applications**
- **High-performance DSP Operating at 100 MHz**
  - **1 GFLOPS - 1.5 Gops**
  - **10 Arithmetic Operations per Cycle (4 Multiply, 2 Add/subtract, 1 Add, 1 Subtract Floating and Fixed Point) Allowing Single Cycle FFT Butterfly**
  - **Native Support for Complex Arithmetic and Vectorial SIMD Operations: One Complex Multiply with Dual Add/sub per Clock Cycle or Two Real Multiply and Two Add/sub or Simple Scalar Operations**
  - **32-bit Integer and IEEE 40-bit Extended Precision Floating Point Numeric Format**
  - **Large Multi-port Data Register File: 512 Registers Organized in Two 4-input 4-output 256-register Banks**
  - **Orthogonal VLIW Architecture, Code Compression for Code Size Reduction**
  - **Flexible Addressing Capability: 2 Independent Address Generation Units Operating on a 16-register Address Register File Supporting Programmable Stride, Circular Pointers and Bit Reversal**
  - **1.7 Mbits of On-chip SRAM**
    - **17K x 40-bit Data Memory Locations**
    - **8K x 128-bit Program Memory Location, Equivalent to 24K Instructions**
  - **DMA Access to the External Program and Data Memory**
  - **Two Main Operating Modes: Run and System Mode**
  - **Efficient Optimizing Assembler: Allows Easy Exploitation of the Available Hardware Resources Parallelism**
- **Utilizes the ARM7TDMI Processor Core with 32K Bytes of Integrated SRAM, Operating at 50 MHz**
  - **Fully-programmable External Bus Interface (EBI)**
    - **Maximum External Address Space of 4M Bytes**
    - **Up to 4 Chip Selects**
    - **Software-programmable 8/16-bit External Data Bus**
  - **8-channel Peripheral Data Controller (PDC)**
  - **8-level Priority, Individually Maskable Vectored Interrupt Controller**
    - **4 External, 20 Internal Interrupt Sources, Including a High-priority, Low-latency Interrupt Request**
  - **28 Programmable I/O Lines**
  - **8-channel 11-bit Programmable Clock Prescaler Feeding the Timer, Watchdog, USARTs, SPIs**
  - **3-channel 16-bit Timer/Counter**
    - **5 Internal Clock Sources and 3 Configurable Sources (External Source or Cascaded Timer Configuration)**
    - **2 Multi-purpose Output Pins plus 1 Output Dedicated to the ADDA Interface plus 3 Outputs Dedicated to the mAgic DSP**
  - **2 USARTs**
    - **2 Dedicated Peripheral Data Controller (PDC) Channels per USART**
    - **1 USART Supporting Full Modem Interface**
  - **2 Master/Slave SPI Interfaces**
    - **2 Dedicated Peripheral Data Controller (PDC) Channels per SPI**
    - **8- to 16-bit Programmable Data Length**
    - **4 External Slave Chip Selects for each SPI**
  - **Programmable Watchdog Timer**
  - **ADDA (A/D and D/A converters) Interface Supporting up to 4 Analog-to-digital and 4 Digital-to-analog, Stereo 24-bit Converters**
  - **IEEE 1149.1 JTAG Boundary Scan on all Active Pins**
- **Efficient ARM - DSP Interface Based on 1K x 40-bit Dual Ported Shared Memory, Memory Mapped Register Access and Interrupt Lines**
- **1.8 V Core Operating Voltage, 3.3 V I/O Operating Voltage**
- **On-chip PLL for 100 MHz Operation from 25 MHz Reference Clock**
- **352-ball PBGA Package**

DIOPSIS 740 Dual-core DSP

AT572D740

7001A–DPS–09/04

Description

DIOPSIS™ 740 (also named D740) is a dual CPU processor integrating a mAgic DSP and an ARM7TDMI RISC MCU, plus a total of 245 Kbytes of SRAM. The system combines the flexibility of the ARM7TDMI RISC controller with the very high performance of the DSP.

mAgic is a high-performance VLIW DSP delivering 1 Giga floating-point operations per second (GFLOPS) at a clock rate of 100 MHz. It has 512 data registers, 16 address registers, 10 independent operating units and 2 independent address generation units. As an example, by activating all the computing units, mAgic can produce one complete FFT butterfly per cycle. mAgic operates on 32-bit fixed-point and IEEE 754 40-bit extended precision floating-point numeric format. It also has on-chip 17K x 40-bit data memory locations and 8K x 128-bit program memory locations. Efficient usage of the internal program memory is achieved through a code compression mechanism.

An optimizing assembler frees the user from dealing with the parallelism of the processor resources and greatly simplifies the code development.

The ARM7TDMI embedded microcontroller core is a member of the Advanced RISC Machines (ARM) family of general-purpose 32-bit microprocessors, which offer high performance and very low power consumption. The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles, and the instruction set and the related decode mechanism are much simpler than those of micro-programmed Complex Instruction Set Computers.

This simplicity results in a high instruction throughput and impressive real-time interrupt response. The ARM7TDMI supports 16-bit Thumb subset of the most commonly used 32-bit instructions. These are expanded at run time with no degradation of system performance. This gives 16-bit code density (saving memory area and cost) coupled with 32-bit processor performance.

A rich set of peripherals and a 32 Kbyte internal memory provide a highly flexible and integrated system solution.

Pin Configuration

Table 1. D740 Ball Assignment (243 I/O)

Name	Ball	Name	Ball	Name	Ball	Name	Ball
ADDA_BRCK	C21	ARM_D[4]	W24	PIO[4]	AC25	SCAN_EN (dnc)	G2
ADDA0_IN	B21	ARM_D[5]	V23	PIO[5]	AD26	SCAN_TEST (dnc)	F1
ADDA1_IN	A22	ARM_D[6]	W25	PIO[6]	AD25	SINGLE	AE16
ADDA2_IN	C22	ARM_D[7]	Y24	PIO[7]	AE26	SPI0_MISO	C20
ADDA3_IN	D22	ARM_D[8]	Y26	PIO[8]	AD23	SPI0_MOSI	B20
ADDA0_OUT	B22	ARM_D[9]	Y25	PIO[9]	AE24	SPI0_NSS	C17
ADDA1_OUT	A23	ARM_D[10]	AA26	PIO[10]	AD22	SPI0_NSS[1]	A17
ADDA2_OUT	C23	ARM_D[11]	AA24	PIO[11]	AC22	SPI0_NSS[2]	D17
ADDA3_OUT	B23	ARM_D[12]	Y23	PIO[12]	AE23	SPI0_NSS[3]	B16
ADDA_TOPLL	A24	ARM_D[13]	AA25	PIO[13]	AD21	SPI0_SCK	D18
ADDA_WCK	B24	ARM_D[14]	AB26	PIO[14]	AF22	SPI1_MISO	B19
ARM_A[0]	A25	ARM_D[15]	AB24	PIO[15]	AE22	SPI1_MOSI	A20
ARM_A[1]	D24	ARM_NCS0	H25	PIO[16]	AD20	SPI1_NSS	C18
ARM_A[2]	C25	ARM_NCS1	J26	PIO[17]	AF21	SPI1_NSS [1]	C19
ARM_A[3]	E24	ARM_NCS2	K24	PIO[18]	AC20	SPI1_NSS [2]	A18
ARM_A[4]	D26	ARM_NCS3	J25	PIO[19]	AE21	SPI1_NSS [3]	B17
ARM_A[5]	D25	ARM_NRD	K23	PIO[20]	AD19	SPI1_SCK	A19
ARM_A[6]	F24	ARM_NWEB0	K26	PIO[21]	AF20	TEST_CLK (dnc)	M25
ARM_A[7]	E26	ARM_NWEB1	L24	PIO[22]	AC19	USART0_RXD	AE17
ARM_A[8]	E25	BIST_RES (dnc)	H1	PIO[23]	AE20	USART0_SCK	AF17
ARM_A[9]	G24	BIST_RUN (dnc)	H3	PIO[24]	AD18	USART0_TXD	AE18
ARM_A[10]	F26	FPU_EXC	AD15	PIO[25]	AE19	USART1_CTS	AD12
ARM_A[11]	G23	FPU_HALT	AD13	PIO[26]	AF18	USART1_DCD	AE14
ARM_A[12]	F25	FPU_MODE	AE15	PIO[27]	AD17	USART1_DSR	AC14
ARM_A[13]	H24	ICE_NTRST	K25	PLL_CLKIN	N24	USART1_DTR	AF14
ARM_A[14]	G26	ICE_TCK	M23	PLL_CLKOUT	N25	USART1_RI	AF15
ARM_A[15]	H23	ICE_TDI	L26	PLL_DIV (dnc)	P24	USART1_RTS	AF16
ARM_A[16]	G25	ICE_TDO	N23	PLL_DN (dnc)	T25	USART1_RXD	AC15
ARM_A[17]	J24	ICE_TMS	M24	PLL_EN	L25	USART1_SCK	AD16
ARM_A[18]	H26	JCFG	M26	PLL_LFT	T24	USART1_TXD	AC17
ARM_D[0]	V24	PIO[0]	AB23	PLL_LOCK	R24	XM_A[0]	AC12
ARM_D[1]	U25	PIO[1]	AB25	PLL_TST (dnc)	N26	XM_A[1]	AE13
ARM_D[2]	V26	PIO[2]	AC26	PLL_UP (dnc)	U23	XM_A[2]	AD11
ARM_D[3]	V25	PIO[3]	AC24	RESET	AD14	XM_A[3]	AD10

Table 1. D740 Ball Assignment (243 I/O) (Continued)

Name	Ball	Name	Ball	Name	Ball	Name	Ball
XM_A[4]	AE11	XM_D[4]	AB1	XM_D[31]	D5	XM_D[58]	C7
XM_A[5]	AC10	XM_D[5]	AB2	XM_D[32]	C11	XM_D[59]	A6
XM_A[6]	AD9	XM_D[6]	AA1	XM_D[33]	D12	XM_D[60]	D7
XM_A[7]	AE10	XM_D[7]	Y4	XM_D[34]	A11	XM_D[61]	C8
XM_A[8]	AF9	XM_D[8]	AA2	XM_D[35]	C12	XM_D[62]	A7
XM_A[9]	AE9	XM_D[9]	Y1	XM_D[36]	B11	XM_D[63]	D8
XM_A[10]	AD8	XM_D[10]	W4	XM_D[37]	A12	XM_D[64]	B7
XM_A[11]	AF8	XM_D[11]	Y2	XM_D[38]	C13	XM_D[65]	C9
XM_A[12]	AC9	XM_D[12]	W1	XM_D[39]	C14	XM_D[66]	A8
XM_A[13]	AE8	XM_D[13]	V1	XM_D[40]	U4	XM_D[67]	A9
XM_A[14]	AD7	XM_D[14]	U3	XM_D[41]	U1	XM_D[68]	C10
XM_A[15]	AF7	XM_D[15]	V2	XM_D[42]	T3	XM_D[69]	B9
XM_A[16]	AE7	XM_D[16]	L1	XM_D[43]	U2	XM_D[70]	D10
XM_A[17]	AF6	XM_D[17]	K3	XM_D[44]	R4	XM_D[71]	A10
XM_A[18]	AC7	XM_D[18]	L2	XM_D[45]	R3	XM_D[72]	A13
XM_A[19]	AE6	XM_D[19]	K4	XM_D[46]	T2	XM_D[73]	B13
XM_A[20]	AF5	XM_D[20]	K1	XM_D[47]	R1	XM_D[74]	A14
XM_A[21]	AD5	XM_D[21]	K2	XM_D[48]	P3	XM_D[75]	D15
XM_A[22]	AC5	XM_D[22]	J1	XM_D[49]	R2	XM_D[76]	B14
XM_A[23]	AE5	XM_D[23]	J2	XM_D[50]	N3	XM_D[77]	A15
XM_CLKOUT[0]	J4	XM_D[24]	E3	XM_D[51]	P1	XM_D[78]	B15
XM_CLKOUT[1]	H2	XM_D[25]	E4	XM_D[52]	N1	XM_D[79]	A16
XM_CLKOUT[2]	G1	XM_D[26]	E2	XM_D[53]	M4	XM_GNT	F2
XM_D[0]	AD2	XM_D[27]	D1	XM_D[54]	N2	XM_NCS	E1
XM_D[1]	AB3	XM_D[28]	D3	XM_D[55]	M2	XM_NWE	F3
XM_D[2]	AC1	XM_D[29]	D2	XM_D[56]	C6	XM_REQ	G4
XM_D[3]	AA3	XM_D[30]	C1	XM_D[57]	A5		

Note: dnc = do not connect pins. These pins are reserved for test use only and are not described in Table 6 on page 6.

Table 2. D740 Ball Assignment (VDD = 3.3V)

D6	F4	L4	AC6	D11	F23	L23	AC11	D16	AA4
T4	D21	AC16	AA23	T23	AC21				

Table 3. D740 Ball Assignment (VDDI = 1.8V)

B18	B12	B6	T1	W3	AD6	AF11	AF19	AF23	W26
E23									

Table 4. D740 Ball Assignment (VDDPLL = 1.8V)

P25	R26								

Table 5. D740 Ball Assignment (GND)

A1	C3	D23	W23	AD3	AF25	A2	C24	AC4	AD24
H4	AF26	A26	D4	AC8	J23	AE1	B2	D9	N4
AC13	AE2	B25	AE25	AC18	P23	D14	B26	AF1	AC23
D19	V4								

All balls not comprised in Tables 1 to 5 are “not connected”.

Pin Name Conventions

Pin names are built using the following structure:

(functional block name) _ (activity level) (line name) (bus index)

where:

- functional block name = name of the functional block which the pin belongs to
- activity level = “n” for low active lines; blank for high active lines
- line name = name of the function of the pin line
- bus index = number (in [ ]) corresponding to the index when the pin line is an element of a bus

Pin Description

Table 6. D740 Pin Description

Module	Name	Function	Type	Active Level	Notes
ADDA	ADDA_BRCK	ADDA Bit rate clock	in		digital serial audio stream bit rate clock (64 x F sampling)
ADDA	ADDA0_IN	ADDA 0 input channel	in		24 bit left + 24 bit right digital serial stereo audio stream
ADDA	ADDA1_IN	ADDA 1 input channel	in		24 bit left + 24 bit right digital serial audio stream
ADDA	ADDA2_IN	ADDA 2 input channel	in		24 bit left + 24 bit right digital serial audio stream
ADDA	ADDA3_IN	ADDA 3 input channel	in		24 bit left + 24 bit right digital serial audio stream
ADDA	ADDA0_OUT	ADDA 0 output channel	in		24 bit left + 24 bit right digital serial audio stream
ADDA	ADDA1_OUT	ADDA 1 output channel	in		24 bit left + 24 bit right digital serial stereo audio stream
ADDA	ADDA2_ OUT	ADDA 2 output channel	in		24 bit left + 24 bit right digital serial audio stream
ADDA	ADDA3_ OUT	ADDA 3 output channel	out-02		24 bit left + 24 bit right digital serial audio stream
ADDA	ADDA_TOPLL	ADDA clock generator Strobe	out-02		F Sampling toward an external PLL for ADCs/DACs synchronism generation
ADDA	ADDA_WCK	ADDA Word clock	out-03		F Sampling clock toward ADCs/DACs
ARM	ARM_A[18:0]	ARM external memory address bus	out-02		
ARM	ARM_D[15:0]	ARM external memory data bus	bi-02		(internal Pull-Down)
ARM	ARM_NCS0	ARM external memory Chip select command 0	out-02	low	
ARM	ARM_NCS1	ARM external memory Chip select command 1	out-02	low	
ARM	ARM_NCS2	ARM external memory Chip select command 2	out-02	low	
ARM	ARM_NCS3	ARM external memory Chip select command 3	out-02	low	
ARM	ARM_NRD	ARM external Memory Read enable	bi-02	low	
ARM	ARM_NWEB0	ARM external memory Low Byte Write enable	bi-03	low	data byte d[7:0]
ARM	ARM_NWEB1	ARM external memory High Byte Write enable	bi-03	low	data byte d[15:8]

Table 6. D740 Pin Description (Continued)

Module	Name	Function	Type	Active Level	Notes
mAgic	FPU_HALT	ARM Fast IRQ from mAgic “halt”	out-02	high	to be used for monitoring
mAgic	FPU_EXC	ARM IRQ15 from mAgic “exception”	out-02	high	to be used for monitoring
mAgic	FPU_MODE	ARM IRQ25 from mAgic “mode”	out-02		to be used for monitoring 0 = mAgic in system mode 1 = mAgic in run mode
JTAG	ICE_NTRST	JTAG Test reset	in	low	(internal Pull-Up)
JTAG	ICE_TCK	JTAG Test clock	in		
JTAG	ICE_TDI	JTAG Test data input	in		(internal Pull-Up)
JTAG	ICE_TDO	JTAG Test data output	out-02		
JTAG	ICE_TMS	JTAG Test mode	in	low	(internal Pull-Up)
D740	JCFG	ARM JTAG / D740 Boundary Scan selection	in		0→ D740 Boundary Scan 1→ ARM JTAG
PIO	PIO[27:0]	Parallel Input/Output	bi-02		general purpose programmable I/Os or ARM peripheral I/Os
PLL	PLL_CLKIN	Reference clock	in		25MHz (max) if PLL_EN =1 100MHz (max) if PLL_EN =0
PLL	PLL_CLKOUT	PLL Clock output	out-02		100MHz (max) if PLL_EN =1 fixed low if PLL_EN = 0
PLL	PLL_EN	Pll enable (PLL_CLKIN x4 multiply)	in	high	1 → system clock = PLL_CLKIN x 4 0 → system clock = PLL_CLKIN
PLL	PLL_LFT	PLL lowpass filter input	in		
PLL	PLL_LOCK	PLL lock condition	out-02	high	to be used for monitoring
D740	RESET	System reset	in	low	asynchronous
mAgic	SINGLE	Single user on mAgic external memory	in	high	(internal Pull-Up) 0 → Not default user of shared XM 1 → Single user of not shared XM or default user of shared XM
SPI	SPI0_MOSI	SPI 0 Master Out/Slave In data	bi-02		SPI SLV → data input SPI MST → data output
SPI	SPI0_MISO	SPI 0 Master In/Slave Out data	bi-02		SPI SLV → data output SPI MST → data input
SPI	SPI0_NSS	SPI 0 Input/Output Chip select	bi-02		SPI SLV → CS Input SPI MST → CS 0 Output
SPI	SPI0_NSS[3:1]	SPI 0 Output Chip Selects	out-02		SPI SLV → n.a. SPI MST → CS 3, 2, 1 Outputs
SPI	SPI0_SCK	SPI 0 Serial clock	bi-03		SPI SLV → clock input SPI MST → clock output

Table 6. D740 Pin Description (Continued)

Module	Name	Function	Type	Active Level	Notes
SPI	SPI1_MOSI	SPI 1 Master Out/Slave In data	bi-02		SPI SLV → data input SPI MST → data output
SPI	SPI1_MISO	SPI 1 Master In/Slave Out data	bi-02		SPI SLV → data output SPI MST → data input
SPI	SPI1_NSS	SPI 1 Input/Output Chip select	bi-02		SPI SLV → CS input SPI MST → CS 0 output
SPI	SPI1_NSS[3:1]	SPI 1 Output Chip Selects	out-02		SPI SLV → n.a. SPI MST → CS 3, 2, 1 outputs
SPI	SPI1_SCK	SPI 1 Serial clock	bi-03		SPI SLV → clock input SPI MST → clock output
USART	USART0_RXD	USART 0 Data in	in		(internal Pull-Down)
USART	USART0_SCK	USART 0 Serial clock	bi-03		for synchronous mode only
USART	USART0_TXD	USART 0 Data out	bi-02		used as output
USART	USART1_CTS	USART 1 Clear to send	in		
USART	USART1_DCD	USART 1 Data carriage detect	in		
USART	USART1_DSR	USART 1 Data set ready	in		
USART	USART1_DTR	USART 1 Data terminal ready	out-02		
USART	USART1_RI	USART 1 Ring indicator	in		
USART	USART1_RTS	USART 1 Request to send	out-02		
USART	USART1_RXD	USART 1 Data in	in		(internal Pull-Down)
USART	USART1_SCK	USART 1 Serial clock	bi-03		for synchronous mode only
USART	USART1_TXD	USART 1 Data out	bi-02		used as output
mAgic	XM_A[23:0]	mAgic external Memory address bus	out-03		
mAgic	XM_CLKOUT[2:0]	mAgic external Memory clocks	out-03		100MHz (max) One line for up to three mAgic XM chip.
mAgic	XM_D[39:0]	mAgic external Memory data bus	bi-03		Right bank (internal Pull-Down)
mAgic	XM_D[79:40]	mAgic external Memory data bus	bi-03		Left bank (internal Pull-Down)
mAgic	XM_GNT	mAgic shared external memory bus grant	out-02	high	
mAgic	XM_NCS	mAgic external Memory Chip select	out-03	low	
mAgic	XM_NWE	mAgic external Memory Write enable	out-03	low	
Power	VDD	IO power supply	Power		3.3 nominal Supply

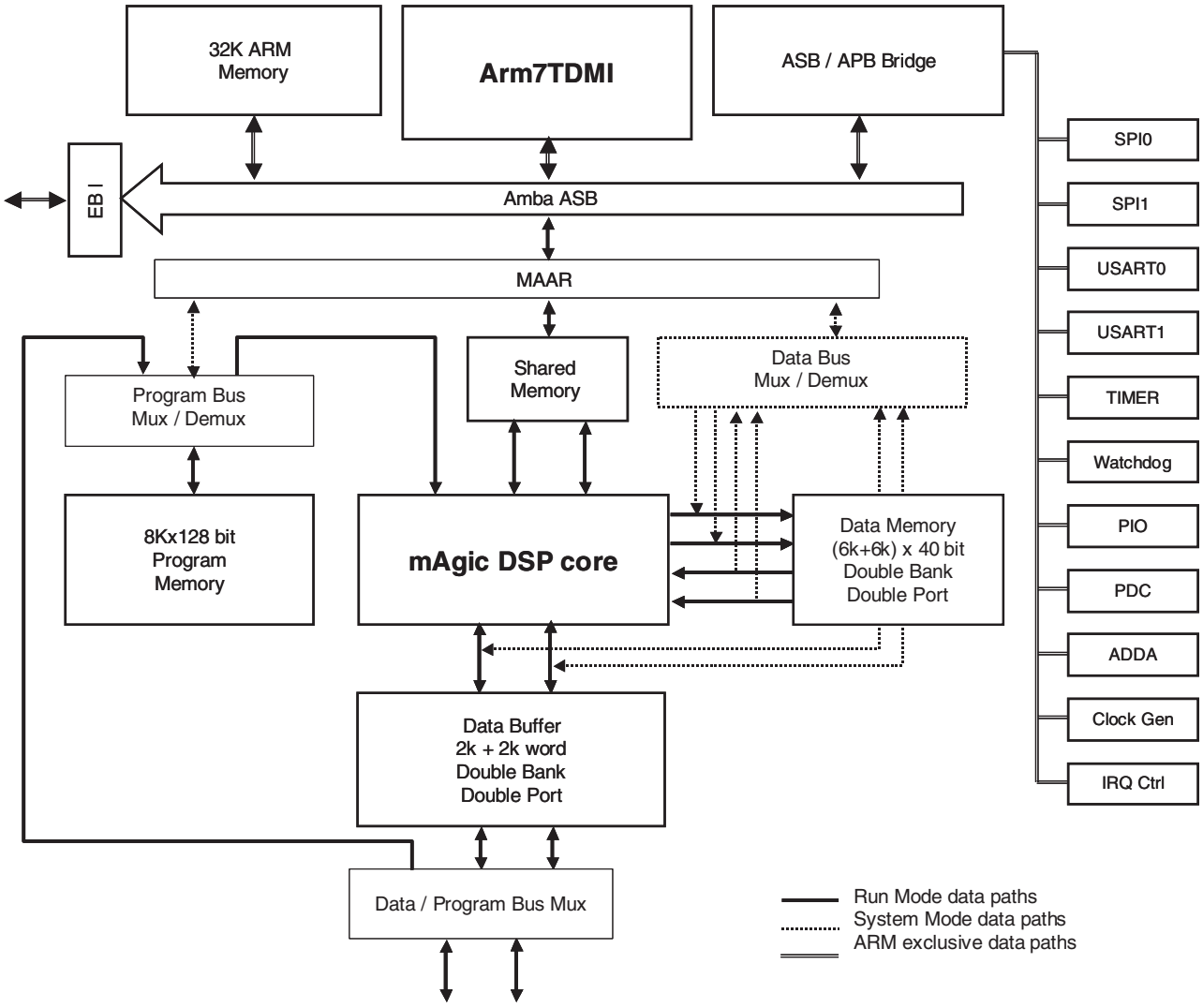
Table 6. D740 Pin Description (Continued)

Module	Name	Function	Type	Active Level	Notes
Power	VDDI	Core power supply	Power		1.8 nominal Supply
Power	VDDPLL	PLL power supply	Power		1.8 nominal Supply
Ground	GND	D740 ground reference	Ground		common to all Supplies

Note: The number code associated to the out and bi type pins represents the driver strength. See Table 63 on page 254.

Block Diagram

Figure 1. D740 Architecture



Architectural Overview

D740 is a high performance dual-core processing platform for audio, communication and beam-forming applications, integrating a floating-point DSP (mAgic DSP) and an ARM7TDMI Reduced Instruction Set Computer (RISC). The D740 is optimally suited for floating point applications with a significant need for complex domain computations like FFT and frequency domain phase-shift algorithms, requiring high dynamic range and maximum numerical precision.

The D740 combines the flexibility of the ARM7 RISC controller with the very high performance of the DSP oriented VLIW architecture of mAgic.

mAgic is an extended Harvard floating-point DSP natively supporting single cycle complex multiply and accumulate, FFT butterfly and vectorial operations. It has 2 banks of 256 data registers each, 10 independent arithmetic operating units, 2 independent address generation units with 16 address registers and an independent DMA unit for the transfer of program and data burst to/from the external memory. mAgic has 8.5K words of on-chip complex dual ported data RAM (17 real Kword). It operates on 32-bit fixed-point and IEEE 754 40-bit extended precision floating-point numeric format. mAgic supports a code compression technique allowing an efficient usage of the 8K-word 128-bit VLIW program memory. Compression factors greater than 3, without loss of performance, are reached on intensive numerical codes like FFTs and FIR filters. On these kinds of routines, which execute more than 10 operations per cycle, an equivalent code density greater than 4 bits per operation is reached. On average, a code density of 50 bits per cycle, without loss of performance, is typical at the level of a complete application. mAgic has two different states, Run Mode and System Mode. In System Mode, the VLIW processor halts and mAgic is memory mapped in the memory space of the RISC controller: the RISC controller can read and write the mAgic Local Data Memories and the configuration registers. In Run Mode mAgic works under direct control of its own VLIW program and the data is exchanged with the ARM through a dual port shared memory.

The ARM7TDMI embedded micro controller core is a member of the Advanced RISC Machines (ARM) family of general-purpose 32-bit microprocessors. The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles. The ARM7TDMI supports 16-bit Thumb subset of the most commonly used 32-bit instructions. These are expanded at run time with no degradation of system performance. This gives 16-bit code density coupled with 32-bit processor performance. The ARM system is equipped with a set of peripherals such as serial interfaces (SPI, USART), timers, watchdog and specialized interfaces for audio A/D and D/A converters. The ARM7 is equipped with 32 Kbytes of on-chip memory. In the D740 the ARM processor runs at half the clock speed of mAgic.

System Management

The availability of a standard RISC on-chip lowers software development effort for non critical and control segments of the application. ARM7TDMI supports the usage of light RTOS and has efficient interrupt management, leaving mAgic fully available for the numerically intensive part of the application. The synchronization between the two processors can be either based on software polling on semaphores or on interrupts.

The ARM is the D740 master processor. The bootstrap sequence of the D740 starts from the bootstrap of the ARM from its external non-volatile memory. The ARM then boots mAgic from a non-volatile memory. After bootstrap the D740 can start its normal operations. The DSP side of many applications can be implemented on the D740 using only the internal memory. In fact the program memory size of 8K by 128-bit coupled with the availability of the code compression, gives an equivalent on-chip program memory size of about 24K instructions (typical).

The ARM standard In-Circuit Emulation debug interface is supported via the ICE port.

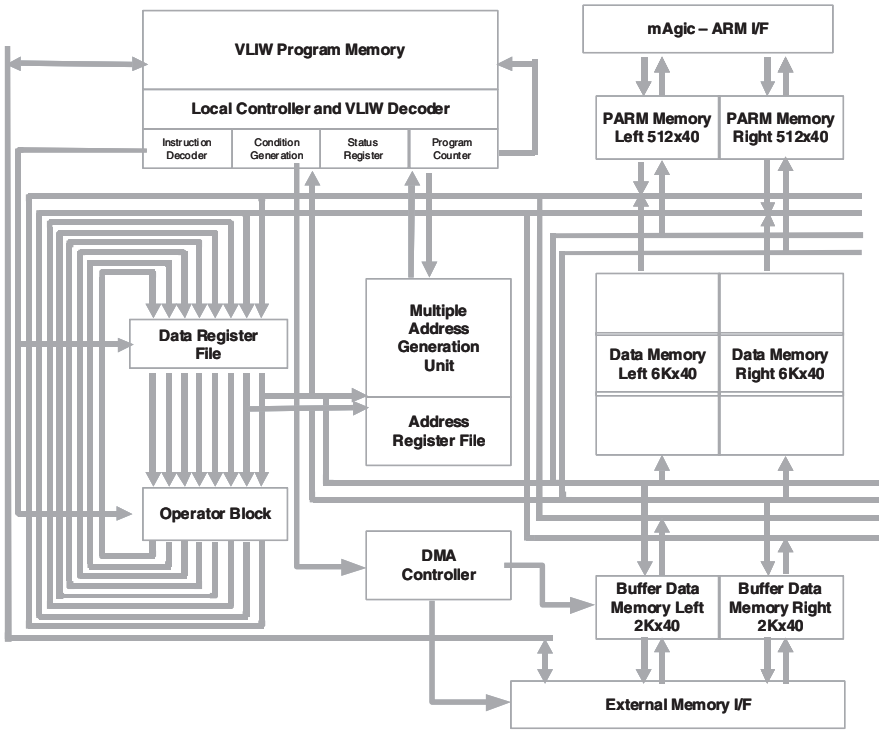
mAgic DSP Processor

The mAgic DSP is the VLIW numeric processor of the D740. It operates on IEEE 754 40-bit extended precision floating-point and 32-bit integer numeric format. The main components of the DSP subsystem are the core processor, the on-chip memories and the interfaces to and from the ARM subsystem. The operators block, the register file, the address generation unit and the program decoding and sequencing unit compose the core processor. A short description of each block is given in the next sections. For more detailed information see Section “mAgic DSP” on page 20.

Core Processor

mAgic is a VLIW engine, but from an user point of view, it works like a RISC machine by implementing triadic computing operations on data coming from the register file, and data move operations between the local memories and the register file. The operators are pipelined for maximum performance. The pipeline depth depends on the operator used. The operations scheduling and parallelism are automatically defined and managed at compile time by the assembler-optimizer, allowing efficient code execution. In order to give the best support to the RISC-like programming model, mAgic is equipped with a complex 256-entry register file. It can be used as a complex register file (real + imaginary part), or as a dual register file for vectorial operations. When performing single instructions the register file can be used as an ordinary 512 register file. Both the left and right side of the register file are 8-ported, making a total of 16 I/O port available for the data move to and from the operator block and the memory. The total data bandwidth between the register file and the operator block is 70 bytes per clock cycle, avoiding bottlenecks in the data flow between the two units.

Figure 2. mAgic DSP Block Diagram



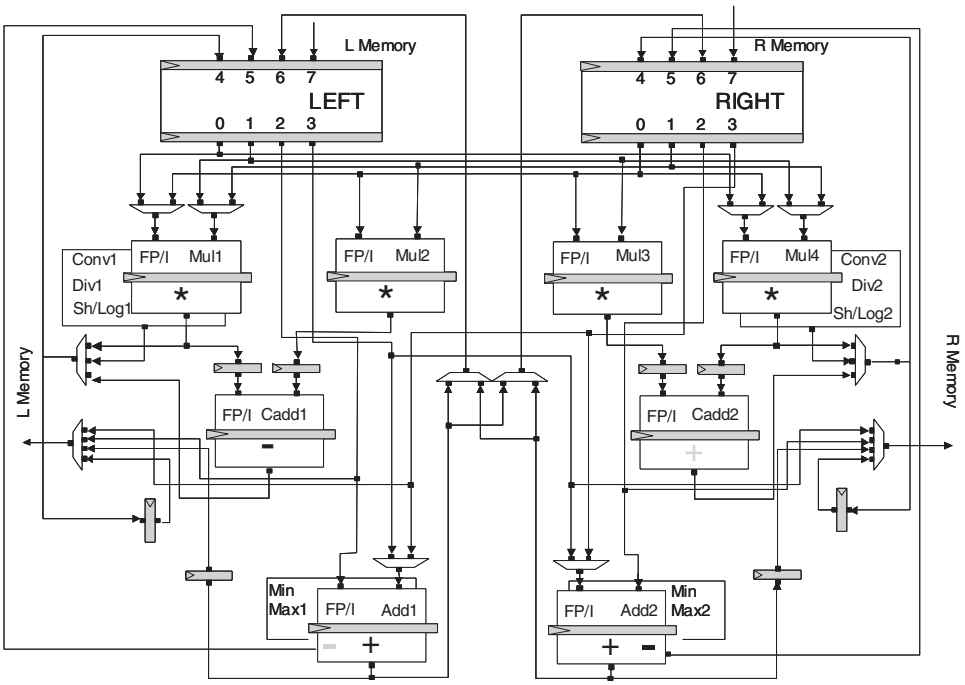
The operators block, the register file, the address generation unit and the program-sequencing unit compose the core processor. The Operators Block contains the hardware that performs arithmetical operations. It works on 32-bit integers and IEEE 754 extended precision 40-bit floating-point data.

The Operators Block is composed of four integer/floating point multipliers, an adder, a subtractor and two add-subtract integer/floating point units; moreover, it has two shift/logic units, a Min/Max operator and two seed generators for efficient division and inverse square root computation. The operators block is arranged in order to natively support complex arithmetic (single cycle complex multiply or multiply and add), fast FFT (single cycle butterfly computation) and vectorial computations. The peak performance of mAgic is achieved during single cycle FFT butterfly execution, when mAgic delivers 10 floating-point operations per clock cycle.

mAgic is equipped with two independent address generation units. It is able to generate up to two pairs of addresses, one to access the left and the right memory for reading and one to access the left and the right memory for writing. It is also used in the loop control to test if the end of a loop is reached. The Multiple Address Generation Unit (MAGU) supports linear addressing with stride, circular addressing and bit reversed addressing. The address generation unit has 16 registers.

The Program Address Generation Unit is devoted to control the correct Program Counter generation according to the program flow. It generates addresses for linear code execution as well as for non-sequential program flow. The Condition Generation Unit combines the flags generated by the operators to produce complex conditions flags used to control the program execution. Predicated instruction execution is supported for different groups of instructions: arithmetical instructions, memory write, immediate load, or all of them. The Program Address Generation Unit also allows to perform conditioned and unconditioned branch instructions, loops, call to subroutines and return from subroutines

Figure 3. Register Files and Operators Block



Internal Memories, External Memories and DMA

mAgic has four on-chip memory blocks: the Program Memory, the Data Memory, the Data Buffer, and the dual ported memory shared with the ARM processor. An External Memory Interface multiplexes the Data accesses and the Program accesses to and from the External Memory. The Program Memory stores the VLIW program to be exe-

cuted by mAgic. It is 8K words by 128-bit single port memory. When mAgic is in System Mode the ARM can modify the content of the mAgic Program Memory in two different ways. The ARM can directly write a Program Memory location by accessing the memory address space assigned to the mAgic Program Memory in the ARM memory map. In this access mode the writes four 32-bit words to four consecutive addresses at correct address boundaries, in order to properly complete a single VLIW word write cycle. The ARM can also modify the content of the mAgic Program Memory by initiating a DMA transfer from the External Memory to the mAgic Program Memory. In this access mode a single VLIW word is transferred from the mAgic External Memory to the mAgic Program Memory 64-bit per cycle, that is a complete word every two clock cycles. Due to the program compression scheme used (see later), which allows an average program compression between 2 and 3, the code accessing capability of mAgic from its External Memory is greater than an instruction per clock cycle. When mAgic is in Run Mode, the ARM cannot get access to the mAgic Program Memory. When in Run Mode mAgic can initiate a DMA transfer from the External Memory to the mAgic Program Memory to load a new code segment.

In order to optimize the internal Program Memory usage and the code bandwidth from the external Program Memory to the internal Program Memory, a code compression mechanism has been implemented. The code for mAgic can be generated and executed either in compressed or in uncompressed form. When the code stored in the Program Memory is compressed, the decompression is done “on flight” just after the Instruction Fetch. The current code compression scheme allows getting compression factors between 2 and 3, depending on the code structure without loss of performance.

Anyway, the classic DSP execution determinism is maintained: only the amount of Program Memory used can change as a function of the compression factor achieved, not the program execution timing. Thanks to the code compression, the code density obtained for mAgic is similar to the code density available on other non VLIW DSPs, while maintaining the advantage in terms of instruction level parallelism.

The mAgic internal Data Memory is made of three memory pages, 2K words by 40-bit for the left data memory and 2K words by 40-bit for the right data memory, giving a total of 6K words for the left and for the right memory banks (a total of 12K words). Each Data Memory bank is a dual port memory that allows four simultaneous accesses, two read and two write. The core can access vectorial and single data stored in the Data Memory. Accessing complex data is equivalent to accessing vectorial data. During simultaneous read and write memory accesses, the MAGU generates two independent read and write addresses common to both the left and the right memory banks. The total available bandwidth between the Register File and the Data Memory is 20 bytes per clock cycle, allowing full speed implementation of numerically intensive algorithms (e.g. complex FFT and FIR).

The Buffer Memory is 2K words by 40-bit for both the left and the right memory. The Buffer Memory is a dual port memory. A port is connected to the core processor. The MAGU generates the Buffer Memory addresses for transferring data to and from the core. The second port of the Buffer Memory is connected to the External Memory Interface. The Buffer Memory does not support dual read and write accesses neither from the core nor from the External Memory Interface. The available bandwidth between the core processor and the Buffer Memory is equal to the available bandwidth between the External Memory Interface and the Buffer Memory: 10 bytes per clock cycle. The maximum External Memory size of mAgic is 16 Mword Left and Right (equivalent to 32 Mword or 160 Mbytes; 24-bit address bus). A DMA controller manages the data transfer between the External Memory and the Buffer Memory. The DMA controller can generate accesses with stride for the External Memory. The DMA transfers to and from the Buffer

Memory can be executed in parallel with the full speed core instructions execution with zero-overhead and without the intervention of the core processor, except for initiating it.

Two kinds of DMA transfers are allowed: non-blocking transfers and blocking transfers. The first type (non-blocking transfer) consists of a transfer that is immediately launched if the DMA machine is idle. If the DMA machine is busy, the transfer request is queued into a FIFO. The second type of transfer (blocking transfer) consists of a transfer that is immediately launched if the DMA machine is idle. If the DMA machine is busy, the burst is started only after the current transfer has ended. In this case the execution of the core instruction is suspended until the requested transfer is started. The core can be synchronized with the DMA engine through the usage of specific synchronization instructions.

The last memory block in the address space of the mAgic DSP is the memory shared (PARM) between mAgic and the ARM processor. It is a dual port memory 512 words by 40- bit for both the left and the right bank (total 1K by 40-bit). This memory can be used to efficiently transfer data between the two processors. The available bandwidth between the core processor and the shared memory is 10 bytes per clock cycle. On the ARM side the available bandwidth is limited by the bus size of the ARM processor (32 bits) giving a bandwidth of 4 bytes per ARM clock cycle.

ARM Interface (mAAr)

The D740 master is the ARM7 RISC processor. mAgic behaves as a standard AMBA ASB slave device, allowing access to different resources depending on the operating mode (Run or System).

In System Mode, mAgic halts its execution and the ARM takes control of it. When mAgic is in System mode the ARM can access many mAgic internal devices. The ability of the ARM to access internal mAgic resources in System Mode can be used for initialization and debugging purposes. By accessing the Command Register, the ARM can change the operating status of the DSP (Run/System Mode), initiate DMA transactions, force single or multiple step execution, or simply read the DSP operating status.

In Run Mode, mAgic works under direct control of its own VLIW program and the ARM has access only to the 1K x 40-bit dual ported shared memory (PARM) and to the mAgic Command Register.

In order to allow a tight coupling between the operations of mAgic and the ARM at run time, they can exchange synchronization signals, based on interrupts.

VLIW Program Word

The mAgic VLIW program word can assume different configurations according to the kind of instructions it contains.

In the first configuration, that is also the most typical one, the VLIW is divided in four fields, corresponding to the building blocks of the VLIW core: Flow Control Unit, Multiple Address Generation Unit, Data Register File Addresses, and Operators Block. In this configuration each field directly drives the architectural blocks to which it's connected.

A second kind of mAgic instruction uses all the bits in the long instruction word to perform a single cycle, multiple loading of immediate data, multiple addressing initialization and looping set up.

A third kind of instruction contains the parameters for launching DMAs between the external memory interface and the local buffers. This instruction is passed to the DMA engine and is executed in complete parallelism with the activities of the VLIW core.

Instruction Set

The operands supported in the instruction set are different for each kind of instruction. The types of operands available are summarized in the Table 7.

Table 7. Operands Data Type

Complex (Float or Integer)
Complex Conjugated (Float or Integer)
Complex Double Conjugated (Float or Integer)
Complex with Real (Float or Integer)
Vectorial (Float or Integer)
Single Operand (Float or Integer)

mAgic treats complex numbers as pairs of 40-bit floating-point. The real part is stored in the left (L) memory bank and the imaginary part is stored at the same address of the right (R) memory bank. The Register File is also divided into real (L) and imaginary part (R).

mAgic instruction set supports the kinds of instructions summarized in Table 8.

Table 8. Summary of Instructions

Add-Sub Instructions
Address Register File Management Instructions
Branch Instructions
DMA (Burst Transfer) Instructions
Compare Instructions
Condition Code and Loop Instructions
Control and Miscellaneous Instructions
Conversion Instructions
Interrupt Management Instructions
Logical and Shift Instructions
Mathematical Seed Generation Instructions
Miscellaneous Arithmetic Instructions
Move Instructions
Multiply Instructions
Repeat Instructions

Some assembly instructions operate on complex conjugated numbers. They can be of two types: the CJ ones in which the first operand is a complex number while the second is conjugated before its use and the CJJ in which both operands are conjugated.

The execution of additions and multiplications between a complex number and a real number (40-bit floating point or 32-bit integer) is also possible. These kinds of instructions are obtained with complex additions or products in which the second complex operand has the imaginary part masked with zero.

The vectorial numbers are pairs of the same data type (40-bit floating point, or 32-bit integer). The first element of the pair must be in the L memory or registers; the second

element must be in the R memory or registers. On vectorial numbers, two operations of the same type (two additions, two products, etc.) are performed (Vectorial Operations). The operands for vectorial instructions are pairs of registers. Real numbers (40-bit floating point and 32-bit integer) can be placed either in the L or the R space. The single arithmetical operations are performed exclusively on one path (L or R, depending on the destination register). The input and the destination registers can be in any of the banks.

The combination of the available computing operations with the different kind of operands for the complex domain operations allows the implementation of many common signal-processing operations in a very natural way (e.g. a sampled correlation computation is simply a multiply with conjugate and add; Inverse FFT is a scaled FFT with conjugate coefficients). The operations scheduling and parallelization is automatically defined and managed by the assembler-optimizer at compile time, thus allowing efficient code execution and substantially simplifying the code development.

ARM System: ARM7TDMI Processor and Peripherals

The ARM7TDMI is a 32-bit RISC microprocessor; it is a member of the Advanced RISC Machines (ARM) family of general-purpose 32-bit microprocessors, offering high performance and very low power consumption. The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles, and the instruction set and related decode mechanism are much simpler than those of microprogrammed Complex Instruction Set Computers. This simplicity results in a high instruction throughput and a real-time interrupt response. Pipelining is employed so that all parts of the processing and memory systems can operate continuously. The typical operating scheme of the ARM7TDMI is the sequence fetch-decode-execute. The ARM7TDMI processor employs the architectural strategy known as THUMB. THUMB instructions operate with the standard ARM register configuration, allowing excellent interoperability between ARM and THUMB states. Each 16-bit THUMB instruction has a corresponding 32-bit ARM instruction with the same effect on the processor model. The 16-bit instructions are expanded at run time with no degradation of the system performance. This provides far better performance than a 16-bit architecture, with better code density than a 32-bit architecture.

The ARM7TDMI processor is built around a bank of 37 32-bit registers and six status registers. The ARM7TDMI supports seven operation modes:

1. User (usr): The normal ARM program execution state
2. FIQ (fiq): Fast Interrupt reQuest; it is connected to the mAgic Halt signal
3. IRQ (irq): Used for general-purpose interrupt handling
4. Supervisor (svc): Protected mode for the operating system
5. Abort mode (abt): Entered after data or instruction prefetch abort
6. System (sys): A privileged user mode for the operating system
7. Undefined (indentured when an undefined instruction is executed

Mode changes can be made under software control or can be brought about by external interrupts or exception processing. Most application programs execute in User mode. The non-user modes - known as privileged modes – are entered in order to service interrupts or exceptions, or to access protected resources. Each operating mode has dedicated banked registers for fast exception handling. The FIQ mode has five additional banked working registers, r8_fiq to r12_fiq, to enhance interrupt processing speed. The ARM7TDMI processor operates in little-endian mode.

For further reading on ARM7 architecture refer to the literature listed in Section “Related Documents” on page 268.

To speed-up critical routine execution or critical data segment access, the ARM7 is equipped with 32 Kbyte of zero wait states on-chip memory. The ARM system has two

buses. The main bus is the ASB (ARM System Bus). The APB (ARM Peripheral Bus) is designed for accesses to on-chip peripherals. The AMBA Bridge provides an interface between the ASB and the APB. The D740 is equipped with a set of peripherals controlled by the ARM. An on-chip Peripheral Data Controller (PDC) transfers data between the on-chip USARTs/SPI and the on- and off-chip memories in the DMA without the intervention of the processor. Most importantly, the PDC removes the processor interrupt handling overhead and significantly reduces the number of clock cycles required for data transfer. It can transfer up to 64K contiguous bytes without reprogramming the starting address. The ARM system peripherals are designed to be easily programmable with a minimum number of instructions. Each peripheral has a 16K-byte address space allocated in the upper 3M bytes of the 4Gbyte address space. Except for the interrupt controller, the peripheral base address is the lowest address of its memory space. The peripheral register set is composed of control, mode, data, status, and interrupt registers. To maximize the efficiency of bit manipulation, frequently written registers are mapped into three memory locations. The first address is used to set the individual register bits, the second resets the bits and the third address reads the value stored in the register. A bit can be set or reset by writing a one at the corresponding position at the appropriate address. Writing a zero has no effect. Individual bits can thus be modified without having to use costly read- modify-write and complex bit manipulation instructions.

A short description of the available peripherals is given:

- **EBI** (External Bus Interface): the EBI generates the signals that control the access to the External Memory or peripheral devices.
- **ADDA** (Analog-to-digital and Digital-to-analog interface): the ADDA provides 4 channel serial interface toward stereo audio 24-bit ADC and DAC.
- **PDC** (Peripheral Data Controller): The PDC provides 8 communication channels dedicated to the two USARTs and to the two SPIs. One PDC channel is connected to the receiving channel and the one to the transmitting channel of each peripheral.
- **USART** (Universal Synchronous / Asynchronous Receiver / Transmitter): two, full-duplex, universal synchronous/asynchronous receiver/transmitters provide a simple standard communication way managed by the Peripheral Data Controller.
- **SPI** (Serial Peripheral Interface): two four-wire serial interfaces provide a simple industry-standard communication way managed by the Peripheral Data Controller.
- **AIC** (Advanced Interrupt Controller): the AIC is an 8-level priority, individually-maskable, vectored interrupt controller. The interrupt controller is connected to the NFIQ (fast interrupt request) and the NIRQ (standard interrupt request) inputs of the ARM7TDMI processor.
- **PIO** (Parallel I/O Controller): The PIO features 32 programmable I/O lines, 28 PIO lines are available on D740 pads, while the remaining 4 are only internal.
- **TC** (Timer Counter): the TC contains three identical 16-bit timer/counter channels.
- **WD** (Watchdog Timer): the WD can be used to guard against system lock-up if the software becomes trapped in a deadlock. If an overflow occurs, the watchdog timer generates processor interrupts via the Advanced Interrupt Controller (AIC) and an external low pulse through the PIO.
- **CLKGEN** (Clock Generator): The clock generator provides divided clocks for several peripherals: the Timer Counter, the Watchdog, the USARTs and the SPIs.

System

Reset Logic

D740 can be externally reset through the RESET input pin. A 10 cycles (minimum) low pulse is required to reset the sequential logic.

Clock Logic

D740 provides and on-chip PLL that, when enabled (PLL_EN = high) allows x4 multiplication of an external reference input clock (PLL_CLKIN) to generate the magic_clock. The magic_clock feeds mAgic while ARM7TDMI and its peripherals use arm_clock = magic_clock / 2.

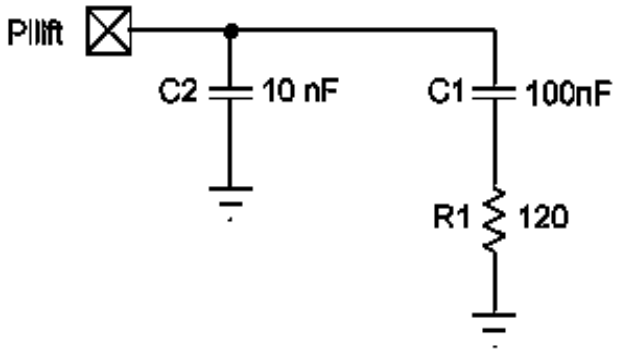
So summarizing:

if PLL_EN = high
- magic_clock = PLL_CLKIN × 4
- arm_clock = PLL_CLKIN × 2

if PLL_EN = low
- magic_clock = PLL_CLKIN
- arm_clock = PLL_CLKIN / 2

The PLL logic requires an external low pass filter circuitry, defined in the following figure:

Figure 4. PLL External RC Filter



The R1, C1 and C2 values depend on the selected system clock frequency. In the example these values are related to system clock = 100 MHz (PLL_CLKIN = 25 MHz).

For different system clock frequencies refer to [16] in Section “Related Documents” on page 268.

mAgic DSP

Overview

The mAgic DSP is the numeric processing part of the D740. It is a high performance integer and floating point processing unit. It is based on a Modular Very Long Instruction Word, built around a large multiport register file. The DSP is able to perform operations on 32-bit integer numbers and IEEE 754 40-bit extended precision floating-point numbers and is able to execute 10 FLOP per cycle and a total of 15 operations per cycle. For instance, in a single instruction four floating point multiplications, four floating point additions and two floating point subtractions, six memory accesses and six address updates can be executed simultaneously. mAgic supports floating point paired vector and complex arithmetic domain computations.

mAgic is designed for easy co-operation with the on-chip ARM processor.

The DSP is specialized in obtaining good performance in complex and vectorial operations. The assembly instruction set is supplied with complex and vectorial instructions, as well as with single instructions (with floating point and integer numbers).

mAgic DSP Features

The main features of the mAgic DSP are:

- Complex, real and vector floating point arithmetic handling
- Single and vector integer arithmetic, single and vector logic & shift capabilities;
- Autonomous flow control capability with dedicated instructions for loop management
- Autonomous Multiple Address Generation capability
- External Data/Program Memory access: up to two 16-Mw banks where each word is 40 bits long

The mAgic DSP contains:

- A VLIW core
- Core memories
- The mAgic- ARM interface (mAAr)
- The External Memory interface (XMIF)

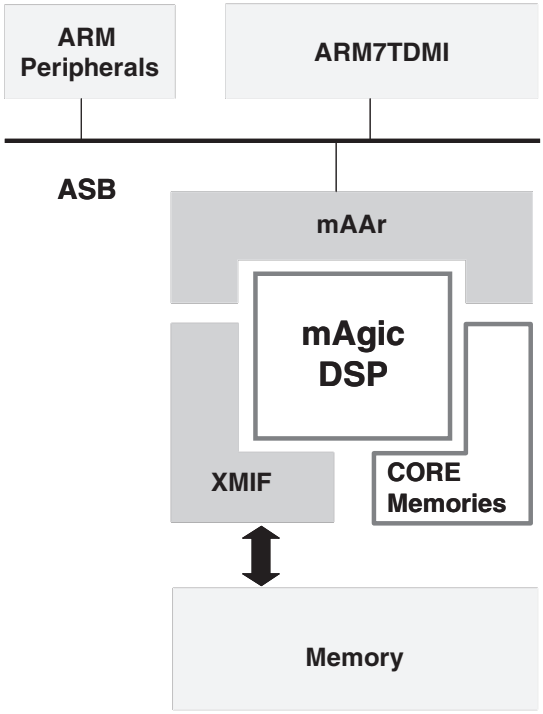
The VLIW Core consists of:

- Data Register File: 16 port, 512 register organized in two 4in-4out 256 reg × 40-bit. The two banks distribution ensures an optimal management in complex arithmetic
- Address Register File: 16 SLAMP registers each having 5 fields: S=11 bits, L=11 bits, A=11 bits, M=7 bits, P=9 bits
- The arithmetic operators block (4 multipliers, 2 adder/subtractors, 1 adder, 1 subtractor)
- The Multiple Address Generation Unit (MAGU)
- A Flow Control System equipped with:
  - a Condition Generation Unit
  - a set of logical operators acting on a Condition Code Stack
  - a Program Counter Generation Unit with a PC stack and a Status Unit
- the global controller which arbitrates the communication with the External Memory and with the ARM

The Core Memories consist of:

- On Core VLIW Program Memory: 8K × 128-bit, equivalent to 24K × 128-bit thanks to a VLIW compression mechanism
- On core Data Memory:
  - two 6K × 40-bit banks (left & right) for internal computation
  - two 2K × 40-bit banks (left & right) for data exchange with the External Memory
  - two 512 × 40-bit banks (PARM left & right) for the communication with the ARM

Figure 5. mAgic DSP General View



A general view of the mAgic DSP architecture is displayed in Figure 5. The VLIW core is surrounded by its interfaces, allowing the communications between mAgic and the ARM through the ASB of the AMBA bus, between mAgic and the External Memory and between mAgic and the other resources.

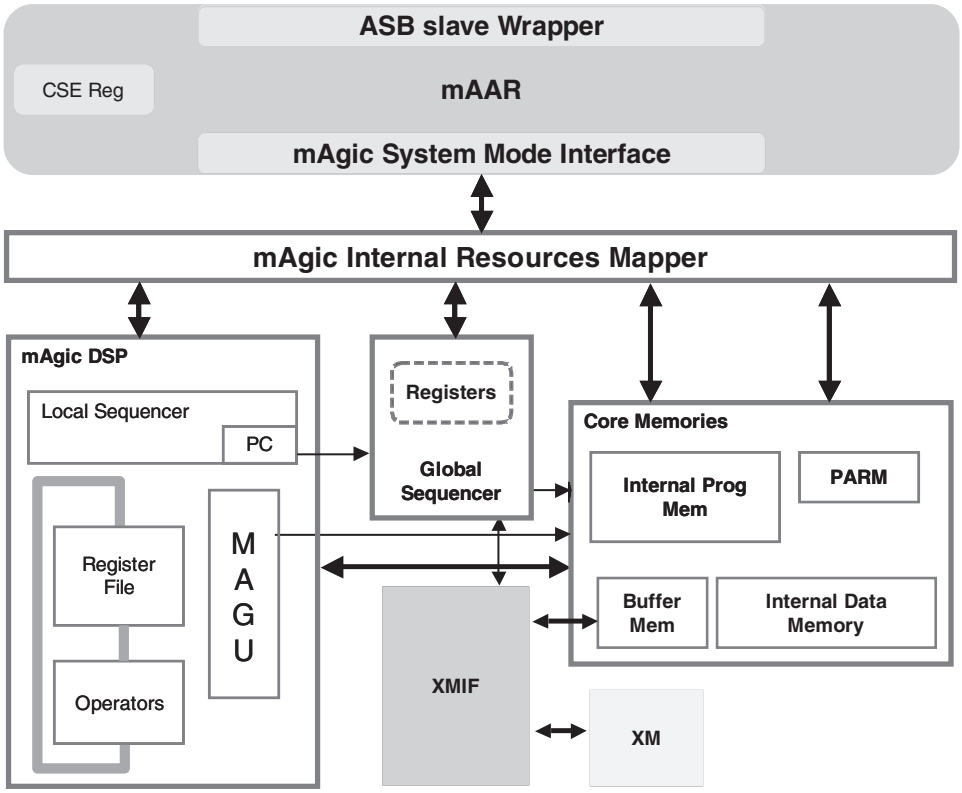
Figure 6 on page 22 shows the block diagram which displays the mAgic DSP architecture with more details: the VLIW core (composed of the Register File, of the arithmetical operators block, of the Multiple Address Generation Unit and of the Local Sequencer and the VLIW Decoder Unit), the on-chip and the external data memories, the on-chip program memory, the Global Sequencer, the mAAr interface between mAgic and the ARM and the XMIF interface between the internal and the external data memories.

The mAgic VLIW core is specialized in obtaining good performances in complex and vectorial operations. This is because the Register File is composed of two parallel banks that are connected to the operators block with two different path working either in parallel or jointly in order to perform complex multiplications.

The Register File is the logical center of the mAgic VLIW core. By reading and writing 16 data in a single cycle to and from the Register File, the DSP is able to perform several operations by launching, in the same cycle, a pair of read memory accesses and a paired-vector or complex arithmetic domain computation.

For instance, four floating point multiplications, four floating point additions and two floating point subtractions, six memory accesses and six address updates can be executed simultaneously in a single instruction.

Figure 6. mAgic DSP Schematic View



This capability can be used to calculate the scalar product between two vectors in a very powerful way. Suppose you have to calculate:

$$S = a_0 \times b_0 + a_1 \times b_1 + a_2 \times b_2 + a_3 \times b_3 + a_4 \times b_4 + ... + a_{50} \times b_{50}$$

Knowing that the Data Manipulation Operators are based on a pipeline of size 2, plus two more cycles necessary for the input/output data from/to the Register File, after 4 cycles the result of a real product or a real addition is available. In order to optimize the previous code execution in this machine it is more suitable to divide the previous scalar product in the following way:

$$S = \sum a_{j \times 4} \times b_{j \times 4} + \sum a_{j \times 4+1} \times b_{j \times 4+1} + \sum a_{j \times 4+2} \times b_{j \times 4+2} + \sum a_{j \times 4+3} \times b_{j \times 4+3}$$

The mAgic DSP high level parallelism allows to perform two memory accesses, a product and an addition in a single clock cycle. At steady state, when all the pipelines are filled, the instructions executed simultaneously are:

– Loading of a_i
– Loading of b_i

- Product of a_{i-1} by b_{i-1}
- Sum of the pre-computed $a_{i-5} \times b_{i-5}$ with one of the four partial sums ($a_{j-5} \times b_{j-5}+a_{j-9} \times b_{j-9}+a_{j-13} \times b_{j-13}+...$)

This powerful parallelism is possible because the VLIW instruction is able to drive simultaneously almost all the DSP devices in a single clock cycle.

With this architecture the maximum optimization is achieved by algorithms with a ratio of 2:2 between arithmetical operations and memory accesses.

Note that there is no need for the user to provide for the VLIW instruction parallelism. While the user writes a sequential code the instruction parallelism is automatically managed by the assembler/optimizer integrated into the visual development environment MADE (Multicore Application Development Environment).

Another basic element of the mAgic VLIW Architecture is the Multiple Address Generation Unit (MAGU). It is composed of an Address Register File (ARF) and a set of Address Generation Unit. These AGUs are able to simultaneously generate read addresses for both the right and the left local data memories (RAGU and LAGU) accesses and also to generate a write address in order to write, as well as to produce, a test condition controlling the program flow in case of loop cycles or sequential flow.

Moreover, an optimized support for loop sequencing is provided. Suppose it is necessary to execute the following code:

```
for i=1 to 20 step 2
begin
     ....
     ....
end
```

It can be translated in assembly code by using a particular instruction, INCCOMP, which uses MAGU and its SLAMP registers (See Section “MAGU Looping Control” on page 47) to compute the value of the index *i* and to test if the final bound is reached. The corresponding assembly code is:

```
IMM2A  6 S=0 L=20 A=1 M=2  ! ARF register initialization
LABEL loop
....
....
INCCOMP 6                  ! A6=A6+M6,
                           ! test if A6<L6
BRIF_NOT loop              ! branch to “loop” if test is false
```

So the use of LOOPGU allows to control the cycle with only two instructions: the INCCOMP (able to generate a condition) and a conditional branch.

Operating Modes

The mAgic DSP operates in two principal modes, in Run Mode and in System Mode. In System Mode, the VLIW processor halts and mAgic is memory mapped in the memory space of the RISC controller: the RISC controller can read and write the mAgic Local Data Memories and the configuration registers. In Run Mode mAgic works under direct

control of its own VLIW program and the data is exchanged with the ARM through a dual port shared memory. The two modes are detailed in the next sections.

Two schematics of the ARM+mAgic system are depicted in Figure 7 and in Figure 8 on page 24. They describe the interconnections through the AMBA Bus, respectively, in Run Mode and in System Mode.

Figure 7. ARM + mAgic in Run Mode

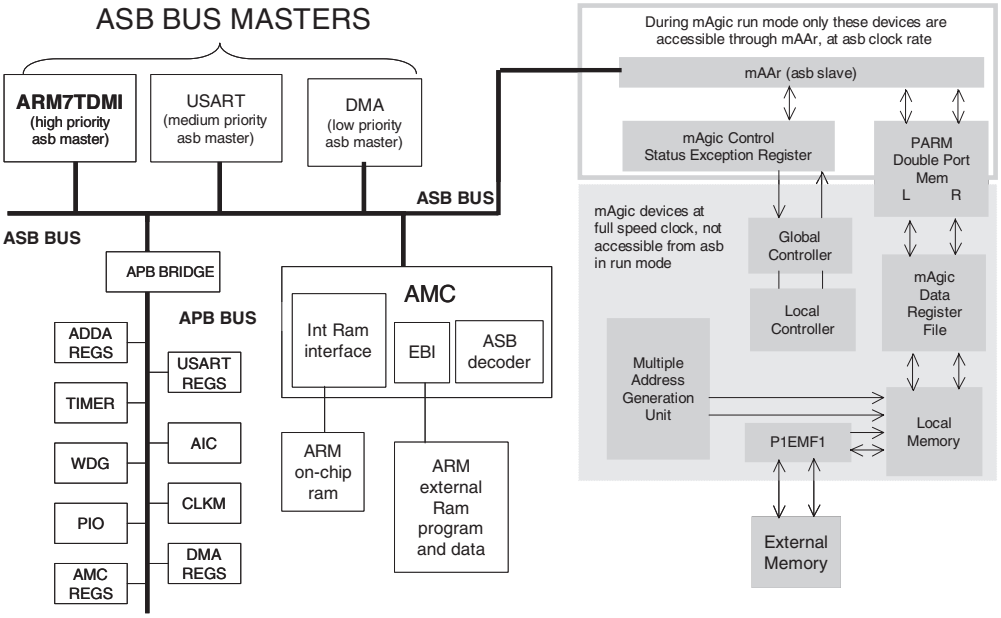
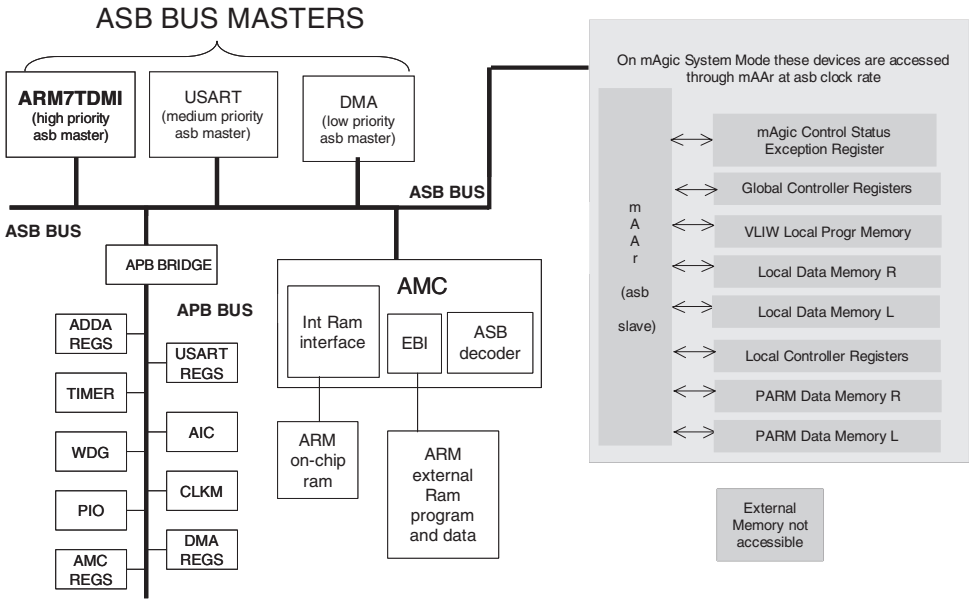


Figure 8. ARM + mAgic in System Mode



System Mode

In System Mode, the DSP halts and the ARM controller can read and write several mAgic internal resources, such as the program and the data memories and some internal control and status registers, acting as input/output buffers.

In this mode the ARM can read all the mAgic resources mapped in the memory space of the RISC controller. The full list of these resources with the corresponding addresses in the ARM Memory Space is shown in Table 9.

Table 9. mAgic FPU Resource Map in the ARM Address Space

mAgic Resource	Register Name	ARM Address Space
PARM Right	-	0x4A0000 to 0x4A0FFC
PARM Left	-	0x490000 to 0x490FFC
CSE	Exception Mask	0x46000C
	Exception	0x460008
	Status	0x460004
	Command	0x460000
GSR	Step_Mode	0x45003C
	DMA_Start	0x450038
	DMA_XMAdd	0x450034
	DMA_CAdd	0x450030
	DMA_Modifier	0x45002C
	DMA_Lenght	0x450028
	DMA_Type	0x450024
	S&SP	0x450020
	IF&CS	0x45001C
	QF&CS	0x450018
	PC	0x450014
	GSR_Exception_Mask	0x450010
	GSR_Exception	0x45000C
	GSR_Status	0x450008
	GSR_Configuration	0x450004
	SAR	0x450000
Program Memory	-	0x430000 to 0x44FFFC
Data Memory Right	-	0x420000 to 0x42FFFC
Data Memory Left	-	0x4100000x41FFFC

mAgic can move toward System Mode in three cases: after the D740 reset signal, when mAgic is in Run Mode and an HALT instruction is executed or if an exception occurs and when mAgic is in Run Mode and the ARM forces its state to System Mode (setting to the value “0” the bit 0 of the CSE Command register).

The access to mAgic internal resources during System Mode can be used for both the DSP initialization and the DSP debugging. The ARM can load mAgic Program memory with the VLIW code and the internal Data memories with the initial values moving data from its memories to the mAgic ones through the mAAr (mAgic-ARM interface). In order to start the DSP, the ARM must write some mAgic registers (SAR, Configuration, Exception Masks, both in GSR and in the CSE registers). Then it can change mAgic status by

writing the CSE Command register and forcing the Sys-to-Run transition (DSP start) or Run-to-Sys (DSP stop).

For debugging purpose, the ARM can read, always through the mAAr, all the mapped resources examining the content of memories and registers. For instance, it can read the PC register, the last DMA parameters or the DSP operating status.

In System Mode the ARM can also initiate mAgic DMA transactions by setting mAgic DMA control registers with the proper values. This way, even if the DSP is in System Mode, the controller can transfer data from the mAgic External Memory to the mAgic Internal Memory or vice versa. Alike, it can write the mAgic Program Memory by initializing a DMA transfer from the External Memory to the Program Memory. This initialization is an alternative to the Program and Data memories loaded through the mAAr.

Always for debugging purpose, the user can decide to execute a program on mAgic by using a Step modality instead of a real Run Mode execution. In this situation the DSP lying in System Mode, is forced by the ARM to execute a single cycle, by using a transition Sys-to-Run, then Run for a cycle, corresponding to a step, and then again a transition Run-to-Sys to return to System Mode. In this way the user has the possibility to check all the DSP resources available in System Mode at every step of the execution. For more details on the step mode see Section “Step Mode” on page 68.

Run Mode

Run Mode is the state in which the mAgic processor works under direct control of its own VLIW program. It fetches the code instruction from the Program Memory, then it decompresses and executes it.

In Run Mode the communications with the ARM are limited to data exchange through a dual port shared memory (PARM) plus some synchronization signals, based on interrupts. The ARM has also access to the mAgic CSE Command register with which, setting or clearing the bit 0, it can put the DSP respectively in Run or in System Mode.

At the beginning, after the D740 reset signal, mAgic can move from System to Run Mode only in one case: when the ARM sets equal to “1” the bit 0 of the CSE Command register. On the contrary, it can leave this state either when an HALT instruction is executed, or when an exception occurs, or when the ARM forces the same Command register to “0” to stop the DSP from the outside.

In Run Mode mAgic can initiate several kinds of DMA transfers between its own External Memory and the internal memories (Data or Program).

A first kind of DMA is from the External Memory to the mAgic Program Memory in order to load a new code segment. When such a DMA instruction is decoded, if the DMA controller is idle, the burst is activated and the core is stalled with a clock stretch. The Program Memory is a single port memory, so that while a program loading occurs, the program VLIW cannot be fetched from the core. During the whole DMA code transfer the core is stalled.

The DMA can also be started between mAgic External and Internal Memories (in both directions) for data load/download. In this case, immediately after the DMA activation, the core goes on with its program sequence and the burst proceeds in parallel with the core activity. The parallelism is granted from the architecture of the Data memory (double port memories with a Data Buffer memory dedicated to the DMA).

The parameters for the DMA are specified in apposite instructions and are stored in the GSR DMA registers. For more details on the DMA see “Direct Memory Access” on page 53 and for the DMA instructions see “DMA Instructions” on page 98 and refer to [4] in Section “Related Documents” on page 268.

When mAgic is in Run Mode, it can be forced through software into an idle state by using the WAITEXT instruction. When the DSP executes a WAITEXT instruction the following actions occur: the mAgic core is stopped with a clock stretch, while all the DMA, already launched or queued to be executed, are completed. The core waits for a signal, either external or generated by the ARM, which is able to reactivate it. When this kind of interrupt has been received, the mAgic core wakes-up and goes on with its program execution. This mechanism can be used every time the user wants to synchronize the DSP with the ARM or with an external device; moreover, it can be useful to reduce power dissipation.

Arithmetic Data Format and Operation Types

The mAgic DSP core is designed to manipulate, with its own operators, 40-bit floating point and 32-bit integer numbers performing complex, vectorial and single instructions. The arithmetic formats, as well as the three different kinds of instructions, will be detailed in the next sections.

Floating Point Format

mAgic supports a 40-bit floating point format, as an extension of the IEEE Standard 754 floating point, having 1-bit sign, 8-bit exponent and 31-bit mantissa instead of 23-bit mantissa:

S	Exponent		Mantissa	
39	38	31	30	0

Bit 39: sign;

Bits 38 downto 31: exponent;

Bits 30 downto 0: mantissa.

Note: The denormalized numbers are not implemented.

Integer Format

n mAgic supports a 32-bit integer format, with 31-bit integer and 1-bit sign.

S		
31	30	0

Bit 31: sign.

The number is in 2-complement format.

When 32-bit integer numbers are mapped on 40 bits format, MSB are aligned and additional LSB's are flushed to zero.

Complex Type Operations

Complex numbers consist of a pair of 40-bit floating point. In the Memory Space their real part is stored in the L bank and the imaginary part at the same address of the R bank. Their real and their imaginary parts are divided in the Register File as well: the first is in the left bank in an even register and the second is in the next odd register in the right bank.

Some assembly instructions operate on complex conjugated numbers. They can be of two types: the CJ and the CJJ. In the CJ the first operand is a complex number while the second is conjugated before its use. In the CJJ both operands are conjugated.

The execution of additions and multiplications between a complex number and a real number is also possible (40-bit floating point or 32-bit integer). These kinds of instruc-

tions are obtained from complex additions or products in which the second complex operand has the imaginary part masked with zero.

Vector Type Operations

Vectorial numbers are pairs of the same type of data (40-bit floating point, or 32-bit integer). They can be stored either in Memory Space (the first element of the pair in the L space and the second at the same address in the R space) or in the Register File (the first element in the left bank at an even register and the second in the next odd register in the right bank). On vectorial numbers, two operations of the same type (two additions, two products, etc.) are performed (Vectorial Operations). Pairs of registers are the operands for vectorial instructions.

Single Type Operations

Real numbers (40-bit floating point and 32-bit integer) are stored in both the Memory Space (L and R spaces) and in the Register File (left and right banks). The single arithmetical operations are performed exclusively on one path (left or right depending on the destination register). The input registers, on the contrary, can be in any bank, because of the mixing input mechanism.

Rounding

The mAgic DSP supports only the rounding mode round-toward-nearest. This rounding mode follows the IEEE 754 standard definitions: when the value in output from an operator cannot be exactly represented in the desired format, the result is rounded to the nearest representable value of the computed result.

This mode has the smallest errors associated with it because statistically rounding up and rounding down occur with the same frequency. As a result, rounding toward nearest is the most useful approach for most applications.

Register File

The 8-input/8-output ports Register File contains 512 registers of 40 bits organized in two banks, the left bank and right bank, each one of 256 registers. These registers can hold complex, vectorial and single values of floating point or integer type.

In complex operations one bank is used to contain the real part (left bank) and the other the imaginary part (right) of a complex number. In vector operations the banks contain couples of floating point or integer numbers.

In the left bank the registers have even identifiers starting from 0 to 510 whereas in the right bank they have odd identifiers starting from 1 to 511. A unique address is passed to the Register File in order to access both the real and the imaginary banks in complex and vectorial instructions.

It is also possible to perform single instructions, data movements within the Register File, from the Register File to the Address Register File and vice versa, from the Register File to the local data memories and vice versa.

For each Register File bank four input and four output ports grant connections toward the outside. These ports are used to connect the Register File to the operators and to the memory: the output ports are the inputs for the operators and the input ports are the outputs of the same operators.

In each Register File the ports are numbered as in the following (see Figure 9 on page 30):

- output port RF0L (RF0R): it is the first input for the multiplier (or for the shift/logic unit)
- output port RF1L (RF1R): it is the second input for the multiplier (or for the shift/logic unit)
- output port RF2L (RF2R): it is the first input for the adder

- output port RF3L (RF3R): it is the second input for the adder
- input port RF4L (RF4R): it is the output of the multiplier
- input port RF5L (RF5R): it is the “-” output of the adder in case of ADDSUB instructions
- input port RF6L (RF6R): it is the normal output of the adder and the “+” output in case of ADDSUB instructions
- input port RF7L (RF7R): it takes inputs from the Data Memory or immediate values from the Program Memory

The addresses, RF0ADD, RF1ADD, RF2ADD, RF3ADD, RF4ADD, RF5ADD, RF6ADD are specified directly by the subfields of the RFA VLIW word.

The RF7LADD and RF7RADD, are generated in almost all cases by the RFA7 VLIW field with the unique exception occurring in the double load immediate instruction, DIMM2RF. In this case the RF7QADD comes from the RFA6 subfield.

RFA VLIW FIELD																							
RFA7			RFA6			RFA5			RFA4			RFA3			RFA2			RFA1			RFA0		
106	...	99	98	...	91	90	...	83	82	...	75	74	...	67	66	...	59	58	...	51	50	...	43
8 bits			8 bits			8 bits			8 bits			8 bits			8 bits			8 bits			8 bits		

Operators

This section describes the operators block structure and functionality as well as their pipeline depth. Note that the pipeline management is done automatically and efficiently by the assembler optimizer.

The Operators Block contains all the operators performing arithmetical operations. It works on 32-bit integer data and 40-bit floating-point data (using the IEEE 754 standard but extending the mantissa to 31 bits and rejecting the denormalized numbers).

The Operators Block is composed of four integer/floating point multipliers, four integer/floating point adders, two shift/logic units and two seed generators (see Figure 9 on page 30), arranged in order to obtain the high performances in floating point paired vector and complex arithmetic domain computations.

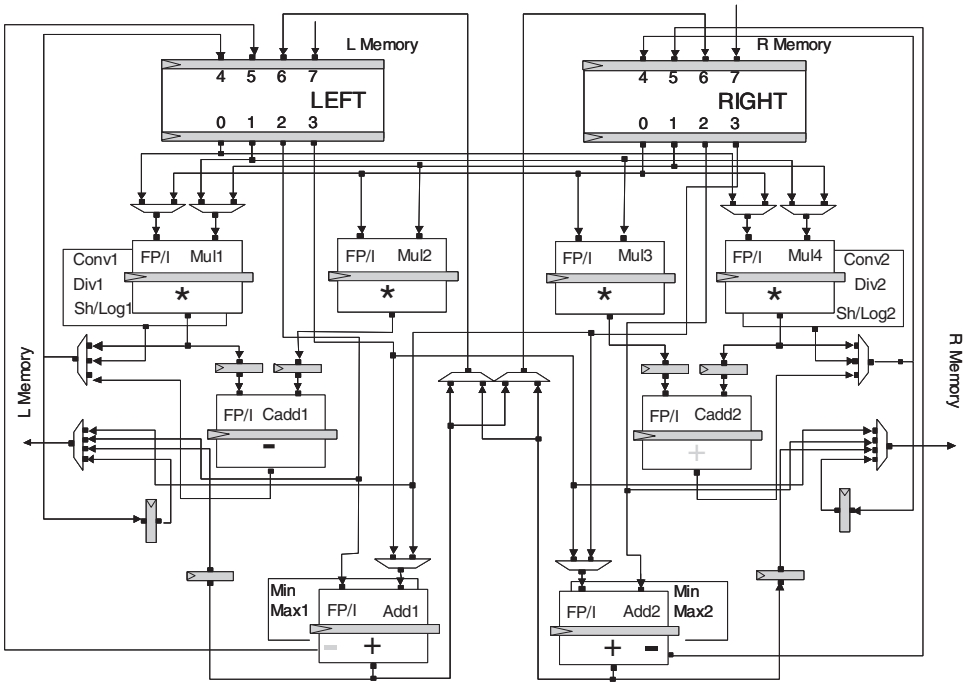
The complex product is hardwired in the operators block. This is obtained using four multipliers and two adders with integer and floating-point capabilities.

Vectorial operations are performed using two multipliers, two adders, two logic units and two seed generators with floating point and integer capabilities, on the left and right path simultaneously.

Single operations (floating or integer) also use the same operators block, but they operate only on one path (left or right depending on the registers involved).

The single operators are directly connected to the Register File ports, both in input and in output (See Section “Register File” on page 28). The inputs of the four multipliers are Port 0 and Port 1, while the input of the two independent adders are Port 2 and Port 3. In output, the adder returns its result in port 6, the multiplier (simple on in the hardwired complex implementation) returns it in port 4. Port 5 collects the “sub” output of the adder only in simultaneous add/sub instructions.

Figure 9. Operators Block



The Operators Block receives the DMO (Data Management Operators) VLIW field. The DMO directly corresponds to the arithmetic assembler instructions. Inside the Operators Block a decoder translates the DMO field into a code directed to the individual arithmetic blocks. The Operators Decoder also produces the Write Enable signals for the Register File Ports 4,5,6 which, depending on the assembler instruction, may be different on the left and on the right bank.

The DMO is a 23-bit word with the following structure:

DMO VLIW FIELD																						
MuxSel		**Add**							**Mul/Sh**							**Input/Output Selector**						
119	118	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Add Codes							Mul/Sh Codes							pt0	pt1	pt4	pt2	pt3	pt5	pt6

The Input/Output Selector subfield chooses, for each input and output of the single additions and multiplications, the banks of the source or the destination register. For example, if the Pt6 bit is 1 the input for the port 6 comes from the right bank, otherwise from the left bank. The same is for the bits Pt5, Pt4, Pt3, Pt2, Pt1 and Pt0, representing the remaining ports.

The Mul/Sh/Log/Conv/Div subfield contains the operative code for the multiplier, the shifter, the logical unit, the converter and the seed generator. Besides, the operative codes for Register File-to-Register File data movements and for Sticky Registers movements to Register File also take place in this subfield. As a consequence, all these kinds of operations cannot be performed in parallel (even because all the operators involved are connected to the same port of the Register File and this avoids the parallelism on them).

The Add subfield contains the operative codes for the independent adder (not for the adder used in hardwired complex product, that is driven directly with the complex multiplications codes).

The last subfield, the MuxSel, is used to drive the two multiplexes which connect the operators block to the left and the right internal data memory banks. With them it is possible to select different sources to be sent to the memory. It is possible to choose between a simple data transfer between the Register File and the memory (with the opportunity to send data allocated in the right RF bank to the left or the right memory bank, as well as data allocated on the left RF bank to the left or right memory bank), but a faster data move is also possible directly from the output of the add and mul operators (complex or real adder, complex or real multiplier) to the memory banks. Moreover, all these movements can be done with every kind of data, complex, vectorial and single data.

The use of instructions that directly move computed results from the operators to the memory space without the intermediate save in the Register File (identified with a suffix _OPOUT), produces a compactor code with a consequent gain in time execution and performances. For more details on these instructions, refer to [4] in Section “Related Documents” on page 268.

Add-Sub Operator

The add/sub operator allows computing different types of additions and subtractions, on integer and floating point data types: complex, vectorial and single additions. They also allow to explicitly execute an addition and a subtraction concurrently on the same data and a certain number of miscellaneous instructions such as absolute value computation, min/max, conditioned selection of input data, swap, etc. In complex additions, it is feasible to conjugate the second or both the operands as well as add a complex with a real number.

Add/Sub instructions include:

- Complex floating-point addition (simple, with operands conjugation, with a real number)
- Integer floating-point addition (simple, with operands conjugation, with a real number)
- Vectorial and single floating-point addition, subtraction, add/sub
- Vectorial and single integer addition, subtraction, add/sub
- Vectorial and single floating-point absolute value, min, max, min/max
- Vectorial and single integer absolute value, min, max, min/max
- Floating-point conditional instructions as selection between two input and addition with masked values
- Integer conditional instructions as a selection between two inputs and an addition with masked values
- Vectorial and single swap between input values

The add/sub floating-point device operates on 40-bit floating-point operands and output 40-bit floating-point results. The add/sub integer device operates on 32-bit integer operands and output 32-bit integer results. In the case of integer instructions, only 32 LSBs are treated, while the 8 MSBs remain unchanged.

The add/sub takes two input operands, which can be selected from the two Register File banks according to the input mixing mechanism. Usually, a result is returned to the Register File. Only in addsub operations two results are returned, while in compare operation no result is returned at all (only condition flags are updated).

Each add/sub instruction takes two pipeline cycles to be completed. During the first cycle the input data (read from the Register File at the cycle before) enters the adder and a first half of the instruction is executed. During the second cycle the last part of the instruction is executed and the result is ready to be written in the Register File. At the immediate following cycle, it is actually written in the Register File. From this cycle on the result can be read for the subsequent operations. So, the adder pipeline is two cycles long, plus two more cycles necessary for the input data read from and the output data write to the Register File.

The following example illustrates this pipeline scheme for a simple addition operation:

$Z = X + Y$

where X and Y are the adder input data already present in the Register File and Z is the result (adder output data) that will be written to the Register File.

Table 10. Adder Pipeline

Cycles	Operation	Note
0	Data X and Y are read from the Register File	Read from RF
1	Data X and Y enter the adder – First half instruction execution	Computation
2	Second half instruction execution – Data Z (result) output the adder	
3	Data Z written to the Register File	Write to RF
4	From this cycle on, data Z is available to be read from the Register File	

As shown in the above table, cycle 1 and cycle 2 are the two adder pipeline cycles (computation cycles), while cycle 0 and cycle 3 are used to access the Register File. From cycle 4 the result is available in the Register File.

Multiplier Operator

The multiplier operator executes products between integer and floating-point numbers. Products are allowed between single numbers, vectorial numbers and also complex numbers. In particular, the complex product is hardwired with the use of four multipliers, one adder and one subtractor, so that the complex multiply can be computed in a single cycle. The pipeline depth of the multiply operation is four cycles. Besides, in complex products it is feasible to conjugate the second or both the operands as well as to multiply a complex number with a real one.

Multiplier instructions include:

- Complex Floating-point product (simple, with operands conjugation, with a real number)
- Complex Integer product (simple, with operands conjugation, with a real number)
- Vectorial Floating-point product
- Vectorial Integer product
- Single Floating-point product
- Single Integer product

The floating-point multiplier operates on 40-bit floating-point operands and its output is 40-bit floating-point. The integer one operates on 32-bit integer operands and its output is 32-bit integer. In the case of integer instructions, only 32 LSBs are treated, while the 8 MSBs remain unchanged.

The multiplier takes two input operands, which can be selected from the two Register File banks according to the input mixing mechanism. A result is returned to the Register File after two pipeline cycles (vectorial and single instructions) or four pipeline cycles (complex instructions). So, the multiplier pipeline is two cycles long (vectorial and single instructions) or four cycles long (complex instructions), plus two more cycles in both cases which are necessary for the input data read from and the output data write to the Register File.

As an example of these two different pipeline cycles, two cases are described.

Vectorial or single product with two pipeline cycles

In this example a product between two real numbers is discussed.

The operation is

$Z = X \times Y$

where X and Y are the multiplier input data already present in the Register File and Z is the result (multiplier output data) that will be written to the Register File. X, Y and Z are real numbers (not complex numbers). The pipeline scheme is shown in Table 11.

Table 11. Real Product Pipeline

Cycles	Operation	Note
0	Data X and Y are read from the Register File	Read from RF
1	Data X and Y enter the multiplier – First half instruction execution	Computation
2	Second half instruction execution – Data Z (result) output the mult.	
3	Data Z written to the Register File	Write to RF
4	From this cycle on, data Z is available to be read from the Register File	

As shown in the above table, cycle 1 and cycle 2 are the two multiplier pipeline cycles (computation cycles), while cycle 0 and cycle 3 are used to access the Register File. From cycle 4 the result is available in the Register File.

Complex product with four pipeline cycles

This second example, instead, explains the case of a product between two complex numbers.

The operation is always the same:

$Z = X \times Y$

but in this case X, Y and Z are complex values. Note that when dealing with complex numbers, the product is equivalent to the following one:

$ZRe = (XRe \times YRe) - (XIm \times YIm)$

$ZIm = (XIm \times YRe) + (XRe \times YIm)$

where the subscript Re identifies the real part of the complex number and the suffix Im identifies the imaginary part of the complex number.

The pipeline scheme is shown in Table 12.

Table 12. Complex Product Pipeline

Cycles	Operation	Note
0	Data X_{Re}, X_{Im}, Y_{Re} and Y_{Im} are read from the Register File	Read from RF
1	Data X_{Re}, X_{Im}, Y_{Re} and Y_{Im} enter the multiplier – First half of the four multiplication instructions execution	Computation
2	Second half of the four multiplication instructions execution – Intermediate results output the four multipliers	
3	Intermediate results enter the subtractor and the adder – First half of the addition and of the subtraction instructions execution	
4	Second half of the addition and of the subtraction instructions execution – Data Z (Z_{Re} and Z_{Im}) outputs the adder and the subtractor	
5	Data Z (Z_{Re} and Z_{Im}) written to the Register File	Write to RF
6	From this cycle on, data Z is available to be read from the Register File	

Cycles 1, 2, 3 and 4 are the four-computation pipeline cycles (cycles 1 and 2 are used for the four products and cycles 3 and 4 are used for the addition and the subtraction), while cycle 0 and cycle 5 are used to access the Register File. From cycle 6 the result is available in the Register File.

Shifter Operator

The shifter operator works with integer operands performing single and vectorial instructions. Shifter operations include:

- Arithmetic left and right shifts
- Left rotates module 32-bit and module 40-bit
- Left and right shift with “1” insert
- Bit manipulation operations, including bit set, clear, toggle, and test
- Encode of “0” or “1” starting from either left or right

The shifter takes two input operands, which can be selected from the two Register File banks according to the input mixing mechanism. A result is returned to the Register File. The first of the two input operands is a 32-bit integer (only in rotate instructions module 40-bit it is a 40-bit value) contained in the Register File and it represents the data on which it is operated upon. The second input operand is a 40-bit value, which specifies all the parameters for the shift or for the bit manipulation instruction. The output is always a 32-bit value, except in rotate instructions module 40-bit where the output is a 40-bit value.

When the treated number is a 32-bit value, only 32 LSBs are manipulated, while the 8 MSBs remain unchanged.

Each shift or bit instruction takes two pipeline cycles to be completed. During the first cycle the input data (read from the Register File at the cycle before) enter the shifter and a first half of instruction is executed. During the second cycle the last part of the instruction is executed and the result is ready to be written in the Register File. At the immediate following cycle, it’s effectively written in the Register File, then the cycle right after it can be re-used. So, the shifter pipeline is two cycles long, plus two more cycles necessary for the input data read from and the output data write to the Register File.

The following example illustrates this pipeline scheme for a simple left shift operation:

Z = X << Y

where X and Y are the shifter input data already present in the Register File and Z is the result (shifter output data) that will be written to the Register File. As described before, the X data that has to be manipulated (in this case with a left shift) and the Y contains all the information about the shift (the exact kind of shift, how many positions, if it is left or right, etc.)

Table 13. Shifter Pipeline

Cycles	Operation	Note
0	Data X and Y are read from the Register File	Read from RF
1	Data X and Y enter the shifter – First half instruction execution	Computation
2	Second half instruction execution – Data Z (res.) outputs the shifter	
3	Data Z written to the Register File	Write to RF
4	From this cycle on, data Z is available to be read from the Register File	

As shown in the above table, cycle 1 and 2 are the two shifter pipeline cycles (computation cycles), while cycle 0 and cycle 3 are used to access the Register File. From cycle 4 the result is available in the Register File.

The two input operands format are described:

- The input operand SourceRegE is pure data to be manipulated. It is 40 bits long, but only 32 LSBs are used (except in rotate instructions module 40-bit where it is a 40-bit value).

The input operand SourceRegN contains the possible mask in the 32 LSB and other information for shift or BIT instruction in the 8 MSB. The bits are arranged as follows::

MS Byte (S2high)								32 LSB (S2low)									
39	38	37	36	35	34	33	32	31	30	...	...	...	...	...	...	1	0

- Bits 31-0 contain the mask for the logic operations (SHAND, SHOR, SHXOR). They are reserved in BIT instructions.
- Bits 39-32 as in the following tables:

In shift instructions

39	38	37	36	35	34	33	32	
0	0	0	N	N	N	N	N	Left arithmetic shift and N. of shifts (0 to +31)
0	0	1	N	N	N	N	N	Right arithmetic shift and N. of shifts (0 to +31)
0	1	N	N	N	N	N	N	Left rotate mod. 40 and N. of shifts (0 to +39)
1	0	0	N	N	N	N	N	"1" Insert Left Shift and N. of shifts (0 to 31)
1	0	1	N	N	N	N	N	"1" Insert Right Shift and N. of shifts (0 to 31)
1	1	0	N	N	N	N	N	Left rotate mod. 32 and N. of shifts (0 to 31)
1	1	1	-	-	-	L | R	0 | 1	Encode 0|1 starting from Left(0)|Right(1)

In BIT instructions

39	38	37	36	35	34	33	32	
-	-	N	N	N	N	N	N	N of manipulated bit (0 to 39)

Where the symbol "-" means "Reserved bit".

The shift operation, SH, shifts SourceRegE by the amount and the kind of shift (arithmetic, rotate, "0" insert, etc.) specified in the 8-MSB of SourceRegN and stores the result in DestRegF.

The instructions SH<log> DestRegF SourceRegE SourceRegN perform the following operation: F = (E <<>> Nhi) log Nlo, where <<>> is a kind of shift specified by 39-37 N bits of an amount specified by 36-32 N bits and log is AND, OR, XOR.

The arithmetic shift, either left or right, is a shift that maintains the sign of the operand SourceRegE, while the “1” insert left shift inserts a “1” in the least significant bit of the shifted data. On the contrary the “0” insert right shift inserts a 0 in the most significant bit.

The rotation module 40 involves all the 40 bits, while the module 32 involves only the 32 LSBs.

The encode operation returns a code specifying the first position in SourceRegE in which appears 0 (or 1) starting from left (or right).

The instructions BIT<log> DestRegF SourceRegE SourceRegN perform the following operation:

F = E log Nhi, where log is:

- SET: for bit set. It sets the Nhi-th bit of E
- CLR: for bit clear. It clears the Nhi-th bit of E
- TGL: for bit toggle. It toggles the Nhi-th bit of E
- TST: for bit test. It tests the Nhi-th bit of E returning the test result (0 if "False", 1 if "True") in F

Converter Operator

The converter operator performs single and vectorial conversion from 40-bit floating point to 32-bit integer and from 32-bit integer to 40-bit floating-point numbers. It takes one input operand from the Register File and returns one result to the Register File.

For integer-to-float conversions the input is a 32-bit integer value and the output a 40-bit float value. On the contrary, for float-to-integer conversions the input is a 40-bit float value and the output is a 32-bit integer value.

The conversion instruction takes two pipeline cycles to be completed. During the first cycle the input data (read from the Register File at the cycle before) enter the converter and a first half of instruction is executed. During the second cycle the last part of the instruction is executed and the result is ready to be written in the Register File. At the immediate following cycle, it is actually written in the Register File, then the cycle right after it can be re-used. So, the converter pipeline is two cycles long, plus two more cycles which are necessary for the input data read from and the output data write to the Register File.

The following example illustrates this pipeline scheme for a conversion from an integer to a floating-point operation:

F=I

where I is the integer input operand read from the Register File and F is the floating point conversion result that will be written to the Register File.

Table 14. Converter Pipeline

Cycles	Operation	Note
0	Data I read from the Register File	Read from RF
1	Data I enters the converter – First half instruction execution	Computation
2	Second half instruction execution – Data F outputs the converter	
3	Data F written to the Register File	Write to RF
4	From this cycle on, data F is available to be read from the Register File	

As shown in the above table, cycle 1 and 2 are the two converter pipeline cycles (computation cycles), while cycle 0 and cycle 3 are used to access the Register File. From cycle 4 the result is available in the Register File.

Mathematic Seed Generator

This device generates reciprocal and reciprocal square root 40-bit floating-point seeds that can be used in suited algorithms for the implementation of 1/x and 1/SQRT(x) functions. It takes one input floating- point operand from the Register File and returns one floating-point result (the appropriate seed read from a look-up table) to the Register File.

The seed generation instruction takes two pipeline cycles to be completed. During the first cycle the input data (read from the Register File at the cycle before) enter the operator and a first half of instruction is executed. During the second cycle the last part of the instruction is executed and the result is ready to be written in the Register File. At the immediate following cycle, it is actually written in the Register File, then the cycle right after it can be re-used. So, the seed generator pipeline is two cycles long, plus two more cycles which are necessary for the input data read from and the output data write to the Register File.

The following example illustrates this pipeline scheme for a seed generation for 1/SQRT(X). X is the input operand and S the corresponding generated seed.

Table 15. Mathematic Seed Generator Pipeline

Cycles	Operation	Note
0	Data X read from the Register File	Read from RF
1	Data X enters the converter – First half instruction execution	Computation
2	Second half instruction execution – Data S outputs the converter	
3	Data S written to the Register File	Write to RF
4	From this cycle on, data S is available to be read from the Register File	

As shown in the above table, cycle 1 and 2 are the two seed generator pipeline cycles (computation cycles), while cycle 0 and cycle 3 are used to access the Register File. From cycle 4 the result is available in the Register File.

Status Flags and Exceptions

The Magic DSP is IEEE754 compliant for most aspects. As a consequence, there are five types of exceptions:

- inexact result

- underflow
- overflow
- divide by zero
- invalid operation

There are however several differences from the IEEE 754 standard:

- No traps are implemented. When an exception is detected, a status flag is set.
- Note that in some cases the result would have been different if traps were implemented and enabled.
- Denormalized numbers are not implemented.
- Rounding mode is round to nearest (the number is rounded to the nearest representable value; this mode has the smallest errors associated with it because statistically rounding up and rounding down occur with the same frequency).
- Only 1/x and $1/\sqrt{x}$ are implemented in the DIV block.

Notes: 1. computed result - means the result which would have been computed if both exponent range and precision were unbounded (infinitely precise result).
2. rounded result - means the result after rounding to nearest.
3. delivered result - means the results that is returned by the operation and which fits with the format.

Invalid Operation (Inop)

Only invalid operands for the operation to be performed can cause invalid operation flag to be set:

- Signaling NaN (for any operation)
- $(+\infty) + (-\infty)$
- $0 \times \infty$
- $1/\sqrt{(\leq 0)}$
- ∞ or NaN input operand for floating-point to integer conversion

Delivered result is a quiet NaN when an invalid operation exception occurs.

Division by Zero (Div by Zero)

If input operand is zero, division by zero flag is set.

Delivered result is ∞ with correct sign when a division by zero exception occurs.

Overflow (Ovf)

If the magnitude of the rounded result exceeds 2^{128}, overflow exception is signaled.

Delivered result is ∞, with the sign of the computed result when an overflow exception occurs.

Underflow (Udf)

Two events can cause underflow: tininess and loss of accuracy.

As no traps are implemented, an underflow is signaled only when both tininess and loss of accuracy occur.

Tininess: computed result lies between $\pm 2^{-126}$ and is not zero

Ex: 2^{-130} (1.*x..x*)

Loss of accuracy: delivered result differs from computed result (which means the result is inexact)

Ex: computed result= 2^{-130} (1.*x*..*x*1100)

delivered result = 2^{-126} (0.0001*x*..*x*1)

As denormalized numbers are not implemented, delivered result is zero when an underflow exception occurs.

Inexact (Inex)

If rounded result differs from computed result or if an overflow exception occurs, inexact exception is signaled.

Delivered result is the rounded result or overflowed result (∞ with the right sign), or underflowed result (0 with the right sign).

Note that if underflow or overflow flag is high, inexact flag will be high.

Integer Flags

When input operands are in 32-bit-integer format the following rules apply:

- Flags Inop, Udf, Inex are cleared
- Flag overflow has the following meaning: if the result of the operation is not between 2^{31} and $-(2^{31}+1)$ the overflow flag is set. Delivered result is overflowed result
- There is one more flag for the ALU: the Carry flag can be used to extend the result with one more bit

Magic DSP Flags Summary

Each operator sets its own flag:

Mul: Inop, Ovf, Udf, Inex

Cadd: Inop, Ovf, Udf, Inex, Carry

Add: Inop, Ovf, Udf, Inex, Carry for Add output

Inop, Ovf, Udf, Inex, Carry for Sub output

Conv: Inex, Inop, Ovf

Integer to floating-point: (Inex, Inop, Ovf are all cleared)

Floating-point to integer: Inex, Inop, Ovf

Shift/log: No flag.

Min/max: Inop

Div: Inop, Ovf, Udf, Inex, Div by Zero

Taking into account that in the Operator block there are the following operators for the left path and the same list for the right path:

- 2 Mul
- 1 Cadd
- 1 Add
- 1 Conv

- 1 Min/max
- 1 Div

the total amount of flags is 64 (32 for the left path and 32 for the right path).

Flags Handling

Flags are accumulated in two **Sticky Status Registers** (STKYI for the left path and STKYQ for the right path) that are reset after reading.

The Sticky Status Register content is shown in Table 16:

Table 16. Sticky Status Register

Bit	Flag Name	Source Operator
0	Inex_Mul1	Primary Multiplier
1	Udf_Mul1	
2	Ovf_Mul1	
3	Inop_Mul1	
4	Inex_Mul2	Secondary Multiplier (used in complex products)
5	Udf_Mul2	
6	Ovf_Mul2	
7	Inop_Mul2	
8	Inex_Add	Secondary Adder (used in complex products)
9	Udf_Add	
10	Carry_Add	
11	Ovf_Add	
12	Inop_Add	
13	Inex_Sub	Primary Adder
14	Udf_Sub	
15	Carry_Sub	
16	Ovf_Sub	
17	Inop_Sub	
18	Inex_Add	
19	Udf_Add	
20	Carry_Add	
21	Ovf_Add	
22	Inop_Add	
23	Inex_Div	Div Operator
24	Udf_Div	
25	Ovf_Div	
26	DivByZero_Div	
27	Inop_Div	

Table 16. Sticky Status Register (Continued)

Bit	Flag Name	Source Operator
28	Inex_Conv	Converter
29	Ovf_Conv	
30	Inop_Conv	
31	Inop_MinMax	MinMax Operator

Note: Inop_Add, Ovf_Add, Carry_Add, Udf_Add, Inex_Add for Cadd nano-operator can be related either to add or sub nano-operations depending on the operation performed on Cadd.

Instructions VSTKY2RF (vectorial instruction) and STKY2RF (single instruction) allow the transfer of these registers from the Operators Block to the Register File, in order to execute computations and checks on their content.

Some examples of code using these instructions are shown:

Single instruction:

```
STKY2RF 100        ! STKYI reset
FADD 8 2 4         ! FADD flags are put in STKYI
STKY2RF 100        ! flags transfer from STKYI to RF and STKYI reset

!! Test of left path flags (Iflags)
SHAND 60 100 mask
IEQ 60 0           ! Istack contains Iflags (left path)
CALLIF_NOT label   ! jump to label if ICOND (Iflags)=false

STKY2RF 101        ! STKYQ reset
FADD 9 3 5         ! FADD flags are put in STKYQ
STKY2RF 101        ! flags transfer from STKYQ to RF and STKYQ reset

!! Test of right path flags (Qflags)
SHAND 60 101 mask
IEQ 60 0           ! Istack contains Qflags
CALLIF_NOT label   ! jump to label if ICOND (Qflags)=false
```

Vectorial instructions:

```
VSTKY2RF 100       ! STKYI/Q reset
VFADD 8 2 4        ! VFADD flags are put in STKYI/Q
VSTKY2RF 100       ! flags transfer from STKYI/Q to RF and
! STKYI/Q reset

!! Test of I/Q flags
VSHAND 60 100 mask
```

```
IEQ 60 0           ! Istack : Iflags x x
IEQ 61 0           ! Istack : Qflags Iflags x
FSOR               ! Istack : Qflags OR Iflags x
CALLIF_NOT label   ! jump to label if ICOND (Qflags OR Iflags)=false
```

Exceptions

Two Exception registers collect all the mAgic DSP exceptions. The first register is allocated in the CSE registers group (in the mAAr) and it is composed as shown in Table 17.

Table 17. CSE Exception Register

Spares	Core Exception	Decoder Exceptions
Bits (31:5)	Bit (4)	Bits (3:0)

Bits 3 to 0 contain the exceptions generated in the mAAr decoder when a wrong ARM access occurs, as detailed in Table 18:

Table 18. Decoder Exception Errors

Bits (3:0)	Decoder Exceptions
0000	No decoder error
0010	Write on read only registers
0100	Address error
1000	External access during run mode

Bit 4 is set to “1”when an exception occurs in the mAgic DSP core. If the user wants more details about the core exception, the GSR Exception register must be read.

The GSR Exception register (placed inside the mAgic Core) is composed as shown in Table 19:

Table 19. GSR Exception Register

Bit Number	Description
0	badinop
1	badoutop
2	divbyzero
3	fifo param full
4	exc from DMA
5	unknown opcode from Local Controller
9:6	unused (set to 0)
10	exc left write and read address equal
11	exc right write and read address equal
32:12	unused (set to 0)

The bits in the GSR Exception register have the following meaning:

- Bit 0: badinop exception is composed of the logical *or* of all the “inop” operators exception (invalid operand).

- Bit 1: badoutop exception is composed of the logical *or* of all the “overflow” operators exception.
- Bit 2: divbyzero exception is the “division by zero” exception of the Div operator.
- Bit 3: this exception is raised when the FIFO collecting the parameters of the pending DMA is full. The depth of this FIFO is 4. When the parameters for the forth DMA are written and the exception is raised.
- Bit 4: this exception is raised when an exception occurs during a DMA burst.
- Bit 5: this exception is raised when an unknown code is received from the Local Controller.
- Bit 10: this exception is set when the left Internal Data memory bank is accessed in read and in write mode at the same address.

Bit 11: this exception is set when the right Internal Data memory bank is accessed in read and in write mode at the same address.

When an exception is raised in the CSE Exception register (but also in the GSR exception register if the bit 4 is raised in the CSE Exception register) an interrupt signal is sent to the AIC (interrupt line 15); this signal is re-send to ARM according to its priority. When the micro-controller receives it, it starts an interrupt service routine including also the exceptions registers read. *NB: only with an ARM read these registers can be cleared.*

The CSE and the GSR Exception registers can be masked using the appropriate Mask Exception registers. In order to mask a bit in the Exception register the user must set to “1” the same bit in the corresponding Mask Exception register. Bit 6 in the CSE Mask Exception register is used to mask the HALT signal coming from the mAgic Core. As a result, the interrupt corresponding to this signal is masked and it is not sent to ARM.

Table 20. Mask Exception Register

Spares	Halt from Core	Not used	Core Exception	Decoder Exceptions Mask
Bit (31:7)	Bit (6)	Bit (5) stuck at 0	Bit (4)	Bit (3:0)

Multiple Address Generation Unit (MAGU)

The Multiple Address Generation Unit (MAGU) is the device designed for the address generation. It is able to generate up to two pairs of addresses to access the left and right memories, two in read and two in write mode. It is also used in the loop control to test if the end of a loop is reached.

The MAGU is composed of two main blocks:

- the RAGU for the right memory address generation; this unit is also used as the LOOPGU (Loop Generation Unit) supporting the dedicated loop control instruction INCCOMP
- the LAGU for the left memory address generation

MAGU has also an Address Register File that contains the SLAMP registers. These registers are used in order to generate addresses for the internal Data Memory, the Buffer Data Memory and the PARM interface.

The Address Register File stores 16 SLAMP registers. Each SLAMP register is divided into 5 fields with the following sizes and functions:

- S: 11 bits for the absolute base address of the array
- L: 11 bits holding the length of the array, used in the circular addressing mode
- A: 11 bits for the offset to the absolute base address

- M: 7 bits for the increment/decrement
- P: 9 bits wide, is the Page field and the 3 LSBs are used to access the different memory areas according to the following table. The other bits are not used

Table 21. "P" ARF Field Values

P value	Memory Area Addressed
0	Internal Data Memory from location 0 to 2047
1	Internal Data Memory from location 2048 to 4095
2	Internal Data Memory from location 4096 to 6143
3	Buffer Data Memory from location 0 to 2047
4	PARM Interface from location 0 to 511

Single data or arrays can be stored in the memories. The maximum length of the array is related to the memory involved, but in any case it can't be more than 2048 data. For the PARM memory, obviously, it can't be more than 512 data.

For what concern the Internal Data Memory, an array (less-equal 2048) can be allocated at every location inside the 6k words. If the array is allocated part in a section of memory with a certain P value and part in the subsequent section, the P field of the array pointer must contain the value of P where is located the first data. For instance, an array of length 256 is allocate at address 4040, the first 56 data are stored in P=1, the others in P=2. The pointer to the array has P=1.

Anyway, whenever the array is allocated, it's very important that the S field (and consequently the P field) points the first element of the array (i.e. the element in the lowest memory location). The offset A can be set wherever the user wants: at 0 (pointing to the first array location) if he wants to scan the array from the first to the last location (using the M field with a positive value); at the array length if he wants to scan the array from the last to the first location (using the M field with a negative value); at any other value for different purposes.

The format of each ARF register is described in the following diagram.

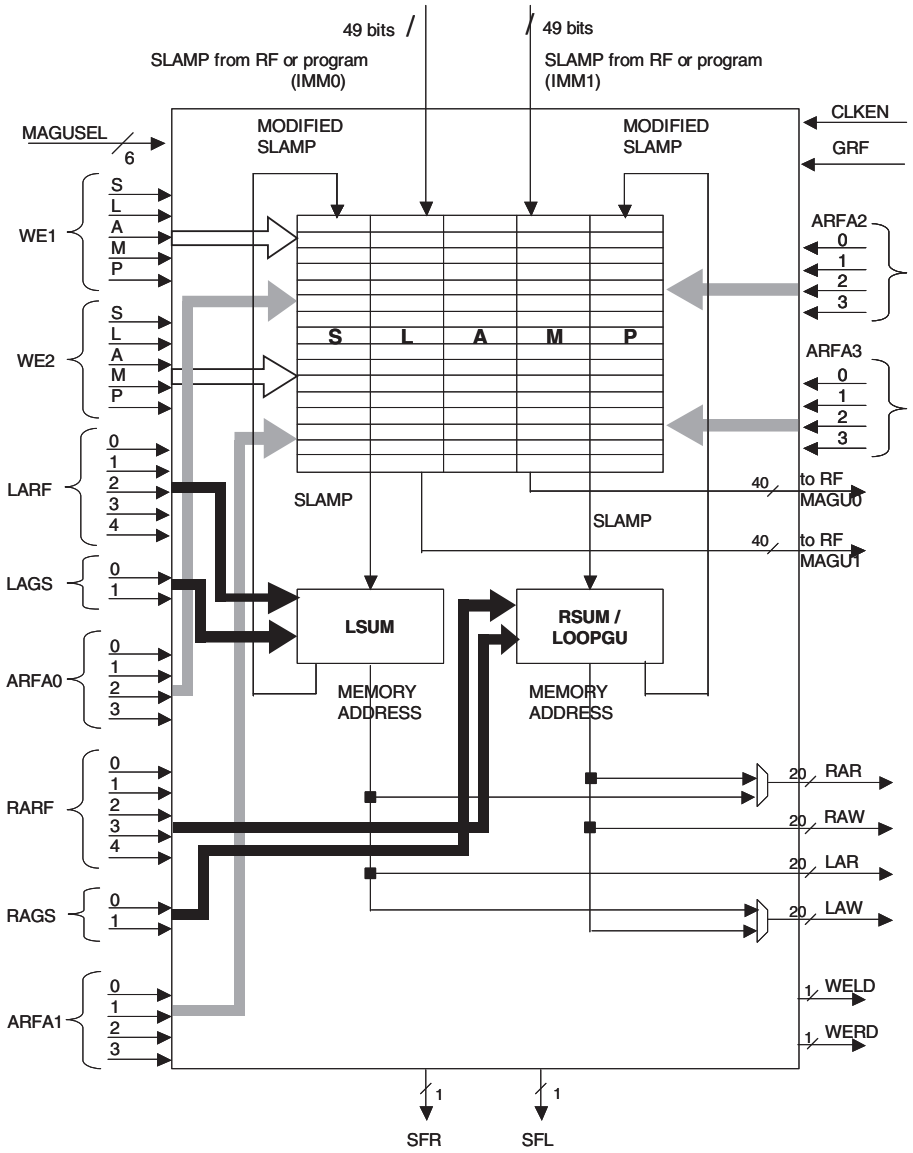
ARF										
P			S		L		A		M	
48	40	39		29	28	18	17	7	6	0

The MAGU Block receives the MAGU VLIW field. This field is a 22 bits word with the following structure: :

MAGU VLIW FIELD																					
LAGU											RAGU										
42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21
Address Mode		Register Group				Access Mode					Address Mode		Register Group				Access Mode				

The first 11 bits drive the LAGU, the other 11 bits the drive RAGU. For each LAGU/RAGU field, 2 bits specify the Addressing Mode (See Table 22 on page 46), 4 bits specify the number of the ARF registers used in the current address generation and 5 bits specify the Accessing Mode (See Table 23 on page 46).

Figure 10. Multiple Address Generation Unit (MAGU).



Address Generation

To access the Internal Data Memory, as well as the Buffer Data Memory and the PARM interface, two addressing modes are supported: circular (or modular) and non circular. In each addressing modes the value of the A field can be left unchanged or updated with the value A+M after or before the memory access, creating eight different situations for increment with or without update as summarized in Table 22. The assembly suffixes referred to in the table are used in the move instructions to select among all the possible situations for increment.

Table 22. Addressing Modes

	Addressing Mode Value	Assembly Suffix	Addressing Mode	Address Out	"A" Field After Update Execution
Non Circular	0	-	Indexed	S+A	A
L=0	1	M	Pre-modify without update	S+A+M	A
	2	U	Post-modify with update	S+A	A+M
	3	MU	Pre-modify with update	S+A+M	A+M
Circular	0	-	Indexed	S+A	A
	1	M	Pre-modify without update	S+(A+M)mod.L	A
	2	U	Post-modify with update	S+A	(A+M)mod.L
	3	MU	Pre-modify with update	S+(A+M)mod.L	(A+M)mod.L

Table 23. Accessing Modes

Access mode value	Name	Description
0	NOP	Nop
1	Reserved	Reserved
2	RD	Read
3	WR	Write
4	CLL	Clear L
5	CSL	Clear S & L
6	BTR	Bit Reversal Read
7	BTW	Bit Reversal Write
8	RDRW	Read & Write
9	BIRW	Bit Reversal Read & Write
A to 1F	Reserved	Reserved

Non Circular Addressing

In the non circular addressing mode the S field contains the base address of the vector, the A field contains the offset inside the vector and the M field contains the modifier. The L field is not used.

The address is computed using one of the following algorithms:

1. address = S+A: this is the simplest way to calculate the address adding an offset (A) to an absolute base address (S)
2. address = S+A+M: like the previous case using a modifier (M) to address an element starting from an absolute base address plus offset

Circular Addressing

In the circular (or modular) addressing mode the vector is considered closed on itself (i.e. the next element after the last is the first element of the vector). The S field contains

the base address of the vector, the L field contains the length of the vector, the A field contains the offset inside the vector and the M field contains the modifier.

The L field is used in address calculation to check whether a bound is exceeded (an address must follow this rule: S<=address<S+L) and in this case to compute the new address.

When using circular buffer addressing mode (L field in the ARF register not equal to 0) programmer must not:

1. Load the M field of the ARF register with a value greater than L, where L is the length of the buffer
2. Load the A field with a value greater than L
3. During run-time the program must not load in the A and M fields values that meet the condition:

   A + M > 2L.

   otherwise the result of addressing such data is not-predictable. This condition is automatically verified if the conditions 1. and 2. are met.

Bit Reversing

The BTR assembly suffix is used to instruct the MAGU to use the bit reversal access mode in reading, while the BTW in writing.

Note that when bit reversal modality is used, the following conditions must be satisfied:

1. The access mode must be one with “modify”: _M or _MU. Using the other access modes (without suffix, or _U), the bit reversal directive is ignored and the computed address is simply S+A
2. When the address is computed in bit reversal, the value contained in the M field is ignored and the increment is always performed with 1

MAGU Looping Control

The MAGU also supports the INCCOMP assembly instruction for the loop control. This instruction simultaneously performs both the loop counter increment by the specified step amount and the comparison between the new calculated index and the loop end value. The comparison sets the flags in the Flow Control Unit in order to generate the condition for the execution (or not) of the subsequent conditioned branch instruction.

In more detail, before a loop starting, an ARF register is loaded in the following way: the A field with the starting value of the loop counter, the L field with the loop end value and the M field with the step value. Any time an INCCOMP instruction is encountered in the flow control, the LOOPGU is activated. This unit performs the addition A'=A+M and simultaneously the check A'<L. The result of this check sets the condition flags used by the subsequent branch instruction.

Memory Organization

The mAgic DSP Internal Memories contains the Program Memory (8K words x 128 bits), the Internal Data Memory (a two-bank-memory of 16K words x 40 bits, 4K words of which are also used as Buffer Data Memory in DMA with External Memory) and the PARM Memory (a two-bank-memory of 1K words x 40 bits) used to connect mAgic to ARM during Run Mode.

Outwards, the mAgic DSP has an addressing capability of 24 bits, so that it can address up to 16M words 80-bit wide of External Memory.

Internal Program Memory

The Program Memory consists of 8K words with a 128-bit word. Thanks to the compression algorithm, up to on average 24 K VLIW instructions can be stored in it. A

decompressing device receives the fetched program word, decompresses it and produces the uncompressed code for the mAgic Core.

In Run Mode the Program Memory receives the address and the control signals from the mAgic Core, while in System Mode it is directly controlled by the mAAr thus allowing the program loading/downloading from the ARM.

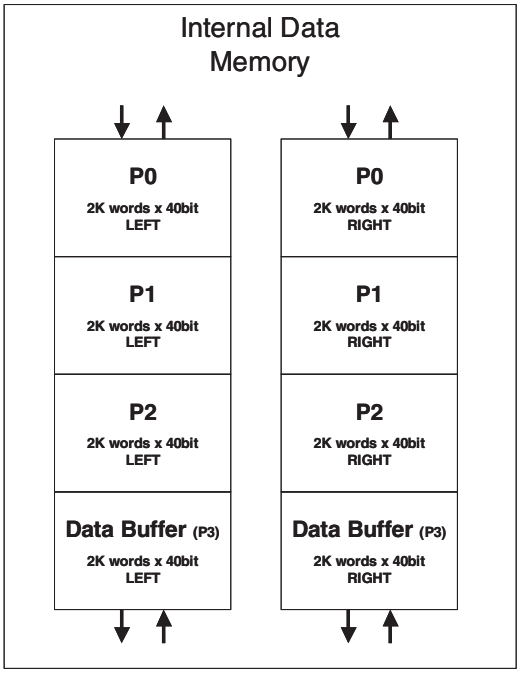
Internal Data Memory

The mAgic DSP is provided with a two-bank double port memory for a total amount of 16K words with a 40-bit word. The memory can store floating point and integer numbers. Pairs of floating point numbers, one in the left bank and the other in the right bank, are used in complex arithmetic: in this case the left memory bank is used to store the real part of a complex number, while the right bank is used to store the imaginary part.

The memory is used in two different ways depending on the processor state: in Run Mode as a Local Data RAM and in System Mode as an I/O Buffer.

In System Mode the Internal Data Memory is used as an I/O Buffer to exchange data with the controller through the mAAr interface. In Run Mode, it can be accessed by the MAGU and it is simultaneously connected in reading and writing mode with the Register File. In this modality, the memory can be seen as divided in pages, each one 2K words deep, four pages in the left bank and other four pages in the right bank (P0, P1, P2 and P3 both for left and for right). Note that page P3 will be also referred to as Internal Data Buffer.

Figure 11. Internal Data Memory Organization



P0, P1 and P2 Data Memory

mAgic accesses to these pages (either in reading or in writing mode) are performed through the MAGU addressing mechanism, preloading the P field of the used ARF with the correct value (See Table 21 on page 44). Note that there is no real barrier between a page and the subsequent one, so that it is also possible to allocate data vectors across two pages, for instance between P0 and P1 or P1 and P2 (but also between P2 and P3). The only limit for the user is that each vector cannot be longer than 2K words.

The pages P0, P1 and P2 are double port memory with both ports connected to the core. This means that every page can be simultaneously accessed both in reading and

in writing mode from the MAGU. The only limit is that the reading and the writing addresses must be different.

N.B.: no exception is raised when the last location of P3 page is reached. If the user is addressing the memory in a pre or post-increment modality, the address computed by the MAGU, immediately after the last address of P3, is the first address of P0, as in a circular addressing.

Internal Buffer Data Memory (P3 Data Memory)

The Buffer Data Memory, a two-bank double port memory of 4K words with a 40-bit word, matches with the last P3 page of the Internal Data memory (2K words for the left bank and other 2K words for the right bank). It provides a data exchange mechanism between the External Memory and the mAgic core.

The buffer can be accessed in System and in Run Mode. In System Mode the ARM can read or write data through the mAAr interface (mainly for debugging purpose). In Run Mode, instead, the Memory is connected on one side to the Register File (in reading or writing) and on the other side to the External Memory (in reading or writing). This way, for instance, the memory can be simultaneously written by the External Memory through a DMA transfer and read from the Register File through the usual mAgic core accesses performed with the MAGU. This allows data transfer from the External Memory to the Register File. The same happens on the opposite direction.

The Buffer Data memory can also be used as the Internal Data memory for storing floating point and integer numbers (with no need to send them to the External Memory). In this case the behavior is similar to that of the other 12K words of the Internal Data memory, referred to as P0, P1 and P2, except for the fact that being the mAgic core connected to only one port of the Buffer Data memory, the data bandwidth toward it is half the data bandwidth toward the other pages. This means that while the pages P0, P1 and P2 can be simultaneously accessed both in reading and in writing mode at different addresses, the Buffer Data memory can be accessed either in reading or in writing mode (not at the same time, because only one port is connected to the core, while the other port is connected to the External memory).

The Buffer Data memory is organized in a single page of 2K words for each bank (left and right) and the mAgic accesses (either in reading or in writing mode) are performed through the MAGU addressing mechanism, preloading the P field of the used ARF with the value “3” (See Table 21 on page 44).

PARM

The mAgic DSP is provided with a kind of mailbox mechanism for the dialog with the ARM, it consists of the PARM memory. The memory is divided in two banks (left and right) and each bank is 512 words deep, 40 bits wide.

The memory can be accessed in reading and writing mode both by mAgic and by the ARM in Run Mode and only by the ARM through the mAAr in System Mode. This interface is the only mechanism allowing data exchange between mAgic and the controller during Run Mode.

mAgic accesses (either in reading or in writing mode) are performed through the MAGU addressing mechanism, by preloading the P field of the used ARF with the value “4” (See Table 21 on page 44).

External Memory

The mAgic DSP has an external addressing capability of 24 bits, so that it is able to address up to 16M words with an 80-bit word of synchronous static RAM. As mAgic Internal Memory, the External Memory is also logically divided in two banks (left and right). Because of that it can be used to collect data organized in the same way as in the Internal Data memory.

Data can be stored in the External Memory in 32-bit integer and 40-bit floating-point formats. Complex values can be stored with the real part on the left bank and the imaginary part in the real bank. Vectorial values can be allocated in the same way as complex values, while real data can be stored both in the left and in the right bank.

The External Memory can also be used to collect VLIW program words, which can be transferred to the Program Memory with the apposite instruction (XM2PM) when a new section of program is needed. This allows loading from the external memory and executing the less frequently used routines in an overlay program memory area. When loading its code to the Internal Memory, mAgic transfers bursts of 64-bit word from the External Memory. The instruction format of mAgic is 128-bit, but the compression scheme implemented allows reducing the required number of bits per instruction by an average 3 factor. That leads to the fact that, on average, the code accessing capability of mAgic from its External Memory is greater than one instruction per clock cycle.

The read/write latencies, depending on the selected External Memory, can be set in the DMA controller using the GSR Configuration register. Bits 7-4 of the GRS_CONF register are used to set the address generation latency for the XM2DM instructions, bits 11-8 are used for the DM2XM instructions and bits 15-12 for the XM2PM instructions. This register must be written by the ARM during System Mode before the start of mAgic program. The default value, assigned at reset time, is “1” for XM2DM, “2” for DM2XM and “1” for XM2PM.

Because of this configuration mechanism of read/write latencies, the best results in terms of External Memory accesses can be obtained using ZBT (Zero Bus Turnaround) or NoBL (No Bus Latency) SSRAM. With these kinds of memories the pipelines of read and write accesses are the same length and can offer the best performances by configuring the accesses latencies with the proper values.

The XM data bus is 80 bits wide, so organized:

for integer data:

	Ext Mem Left		Ext Mem Right	
bit	79 72	71 40	39 32	31 0
addr n		32 bit left data		32 bit left data

for floating point data:

	Ext Mem Left		Ext Mem Right	
bit	79 72	71 40	39 32	31 0
addr n	8 bit exp. left data	32 bit mantissa left data	8 bit exp. right data	32 bit mantissa right data

for VLIW instructions:

	Ext Mem Left		Ext Mem Right	
bit	79 48	47 40	39 32	31 0
addr n	P1			P0
addr n+1	P3			P2

related to the 128 bit VLIW instruction word as follows:

VLIW WORD			
127			0
P3	P2	P1	P0

XM data bits that are not used can be left floating because internally pulldowned.

The XM bus for integer or floating point data can also assume the single bank configuration, limiting the number of external memory devices: for example for integer data a single 32-bit SSRAM can be connected to XM_D[31:0]. In single bank configuration internal P3 memory bank (left or write) that has the corresponding XM_D bits floating is unused.

Pins for the External Memory Interface

The External Memory is interfaced with the D740 chip through an 80-bit external data bus, a 24-bit address bus and some control signals. The pins used for the interface are the following:

- XM_D[39:0] – External Memory Data Bus Right – These pins are used as input/output for the data exchange between the External and the Internal Buffer memory. The exchange can be on data or on program. These pins connect the right Data Buffer bank to the right External Memory bank. At every external access all the 40 bits are transferred (read or written) together with the 40 bits of the left bank. Access to the right bank only is not allowed.
- XM_D[79:40] – External Memory Data Bus Left – These pins are used as input/output for the data exchange between the External and the Internal Buffer memory. The exchange can be on data or on program. These pins connect the left Data Buffer bank with the left External Memory bank. At every external access all the 40 bits are transferred (read or written) together with the 40 bits of the right bank. Access to the left bank only is not allowed.
- XM_A[23:0] – External Memory Address Bus – These pins are used to output addresses for the External memory. The same address is used for both the left and the right bank that are read/written simultaneously with a consequent move of 80 bits at each data memory access and 64 at each program memory access.
- XM_NCS – External Memory Chip Select signal – This line (active low) is used to select the External Memory. At every mAgic External memory access, either in reading or in writing mode, this signal is activated. When no External Memory access is required, the signal is inactive.
- XM_NWE – External Memory Write Enable signal – This line (active low) is used to select the External Memory data bus direction. When mAgic writes on the External Memory, the signal is active. When mAgic reads from the External Memory the signal is inactive.
- XM_CLKOUT[2:0] – External Memory Clock signal – This signal consists of three clock lines, one for each physical block of External Memory. The signals are generated from the PLL.

mAgic Shared External Memory Architecture

The mAgic external memory can be shared with other processors.

If the SINGLE input is asserted, mAgic is the default owner of the shared external memory (XM master).

mAgic has an internal XM bus arbiter that allows any other processor (XM slave) to access the shared external memory.

The XM bus slave processor in order to get the XM bus has to send a request on D740 XM_REQ input line a wait for the related bus grant from D740 XM_GNT output line.

mAgic (the XM master) in order to acknowledge a bus request must have the grant_enable (bit 2 in GSR_conf internal register) high (default low). This bit can be asserted/deasserted or by ARM or by mAgic itself (through the GNTON/GNTOFF instructions).

After completed the XM access the XM slave processor has to release the XM bus and then deassert the XM_REQ. Consequently D740 will deassert XM_GNT.

A second D740 can assume the role of the XM bus requesting processor if its SINGLE input is low (XM bus slave): its XM bus lines are in hi-impedance by default.

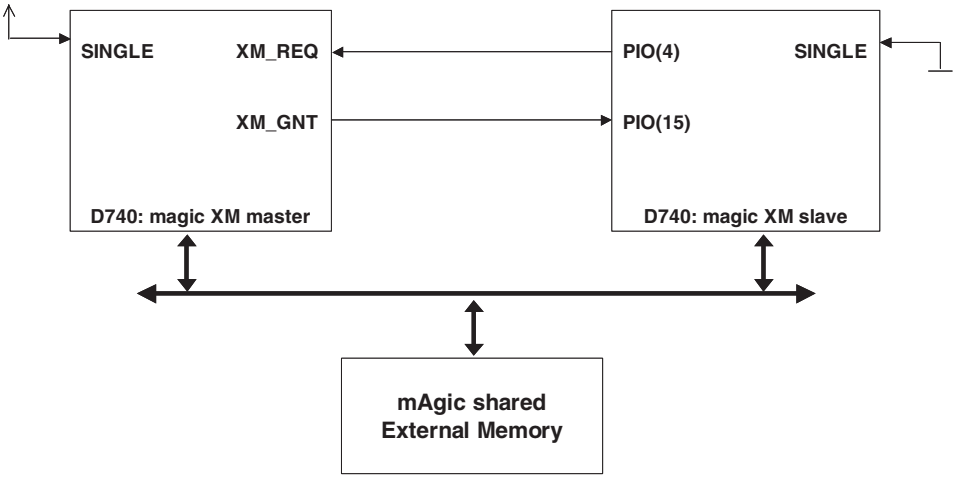
Two others pins are normally used to support dual D740 connection:

- PIO(15) to generate a magic interrupt that signal that the XM bus is granted
- PIO(4) to request the XM bus.

The request is asserted by the XM slave via software through the mAgic instruction SIRQ5 that asserts the PIO[4] line (configured as peripheral linked). After the completion of the XM access the XM slave mAgic has to remove its request through another SIRQ5 instruction that deasserts the PIO[4] line.

A typical dual D740 connection is described in Figure 12.

Figure 12. Dual D740 with Shared XM Architecture



Architectures with more than two D740 require an external arbitration (refer to the Application Note “Multi D740 Shared Memory Systems“).

Data Memory Hierarchy

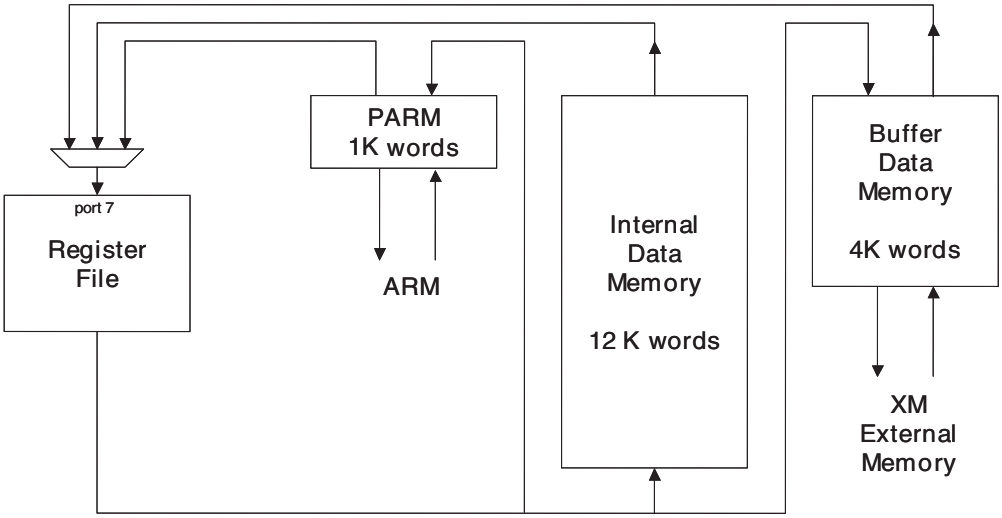
Data Memory hierarchy is organized as in Figure 13.

Data can be stored either in the Internal Data Memory (6K words x 40 bits for the left bank and 6K words x 40 bits for the right bank), or in the External Data Memory. When data must be used by the computational part of the DSP, it must be loaded either by the Internal Data Memory or by the External Data Memory to the Register File.

As regarding the data stored in the Internal Data Memory, it can be directly moved to the Register File by using core move instructions in one direction (VMS2RF or MEM2RF) or in the other (VRF2MS, RF2L or RF2R).

As for the data stored in the External Data Memory, an intermediate Buffer Data Memory (2K words x 40 bits for the left bank and 2K words x 40 bits for the right bank) is used for transfers from the External Memory to the Register File and vice versa. Data transfers are allowed from the External Memory to the Buffer Data Memory and a through core move instruction (VMS2RF or MEM2RF), each data can be moved from the Buffer to the Register File. The same happens in the opposite direction, from the Register File to the External Memory, through the Buffer, in order to store the computed data.

Figure 13. mAgic Data Memory Hierarchy.



A PARM memory (512 words x 40 bits for the left bank and 512 words x 40 bits for the right bank) is also provided for the data exchange with the ARM. The data can be written from the ARM in the PARM and mAgic moves it to the Register File using the core move instructions. On the other direction, mAgic writes the data in the PARM with the core move instructions and the ARM reads it on the other side of the memory.

The addresses used in all the core move instructions are computed by the Multiple Address Generation Unit (MAGU) as specified in the Section “Address Generation” on page 45.

Direct Memory Access

A DMA mechanism is provided for the data transfer between the External Memory and the Internal Data memory and vice versa. The external data is transferred with a burst to the mAgic internal Buffer Data Memory. From there, it can be read directly from the mAgic core using the MAGU addressing mechanism, preloading the P field of the used ARF with the value “3” (See Table 21 on page 44).

In the other direction, the mAgic core writes the data in the Buffer Data memory using the MAGU addressing mechanism, preloading the P field of the used ARF with the value “3” (See Table 21 on page 44). From there, it is transferred to the External Data Memory using the DMA mechanism.

The DMA is started by using two instructions: the first one is used for the parameter loading (IMM2BS and LDRB), while the second one for starting the burst (XM2DM, DM2XM, XM2DMB and DM2XMB). Detailed information on the syntax of each DMA instruction can be found in [4] (see Section “Related Documents” on page 268).

All the bursts are executed in parallel with the core instructions, while their activation depends on the kind of DMA they consist of. Two kinds of transfers are allowed: non-

blocking transfers (XM2DM and DM2XM) and blocking transfers (XM2DMB and DM2XMB).

The first type (non-blocking transfers) consists of a transfer that is immediately launched if the DMA machine is idle, while, on the contrary, if the DMA machine is busy, the transfer is queued into a FIFO containing all the other non-blocking transfers previously encountered, and it will be executed after all of them. In the meantime, the core execution goes on with the subsequent core instructions and it is absolutely not affected by the execution or by the FIFO encoding of the transfer. In this way the user does not have the exact cognition of “if” and “when” the transfer takes place. If the DMA machine is busy, or the total length of all the burst is longer than the total length of the core instructions, the FIFO may not be empty whenever the user expects it to be. To be sure that the transfer has been completed, the WAITTRX instruction must be used. When this instruction is decoded, the core goes on with the subsequent instructions only if there are no current transfers and the FIFO is empty, otherwise the core is stalled waiting for the condition becoming true. Because of this, when the user needs all the previously launched transfers to be completed, the WAITTRX instruction must be used. It is strongly recommended to use a WAITTRX instruction before the HALT instruction, in order to be sure that all the transfers are completed before the end of the program. It is also recommended to use a WAITTRX instruction every 4 non-blocking transfers, because the depth of this FIFO is 4, so that no more than 4 non-blocking transfers can be launched one after the other before a WAITTRX instruction.

The second type (blocking transfers) consists of a transfer that is immediately launched if the DMA machine is idle, while, if the DMA machine is busy, the end of the current transfer is reached and then the burst is started. In this case, the execution of the core instruction is suspended, and the core is stalled until the transfer is started. As soon as the transfer is activated, the core is restarted. Note that the core is re-activated when it is launched not when it is finished. In this case as well, to be sure that a transfer is ended, a WAITTRX instruction must be used.

With the blocking transfers the user knows exactly when the burst is activated. It happens immediately when the instruction is decoded if the DMA machine is idle, otherwise the entire core is stalled waiting for the end of the current transfer. In any case, when the instruction following the burst is executed, the preceding blocking transfer has been already started, but not concluded yet.

With blocking transfer instructions it is possible to synchronize a DMA transfer from the External Memory to the Buffer Data Memory with a subsequent core access by moving the same data from the Buffer Data Memory to the Register File. This synchronization can also be obtained in the opposite direction, when the data is moved from the Register File to the Buffer Data Memory, and then through a DMA, from the Buffer Data Memory to the External memory. The mAgic assembly provides the support to schedule the DMA transfer and the data usage by the core with the minimum latency, allowing the usage of the data by the core as soon as it is available in the Buffer Data Memory (the code is scheduled with the exact synchronization upon the first transferred data). The synchronization is obtained by establishing a relationship between the DMA transfer instruction and the ARF of the MAGU used by the core to access the data in the Buffer Data Memory. In order to obtain a correct assembly code, a PRAGMA_WAIT_ARF_ON/OFF assembler directive must be used. Note that, since the DMA accesses are sequential, the core accesses must be sequential as well, starting from the first location containing the first transferred data.

For example, suppose a data transfer must be done from the External Memory to the Register File: with the instruction XM2DMB the data is moved from the External Memory to the Buffer, then with the instruction VMS2RF, by using for instance ARF 7, the data

will be moved from the Buffer Memory to the Register File. In order to get the right schedule between the data transfer instruction from the External Memory to the Buffer Memory (XM2DMB) and the data transfer from the Buffer Memory to the Register File by using the ARF7, the assembler/optimizer must be instructed regarding the dependency (temporal correlation) between the two transfers. This is obtained by surrounding the instruction XM2DMB with a PRAGMA_WAIT_ARF_ON/OFF ARF7 directive. The PRAGMA_WAIT_ARF_ON/OFF ARF7 avoids the schedule of any instruction using the ARF7 in the N instructions immediately after the XM2DMB instruction (N is the XM2DMB instruction latency). In the example the instruction using the ARF7 is the data move from the Buffer Memory to the Register File using the ARF7 (VMS2RF_U <Coeff_Reg+2 × i> Arf7). The code will appear as in the following example:

```
\equ Arf7 7           ! ARF used to move data from Buffer to the
                      ! Register File
\equ Arf2 2           ! ARF not used in this example but necessary in
                      ! the DIMM2A instruction
\equ BufferAdd 128    ! Start Address of Buffer Memory for XM2DMB
\equ XM_Add 4000      ! Start Address of External Memory for XM2DMB
\equ Len 64           ! Length of DMA for XM2DMB
\equ Mod 1            ! Modifier of DMA for XM2DMB
\equ Coeff_Reg 16     ! Destination Register in the Register File

DIMM2A Arf7 S1=BufferAdd L1=64 A1=0 M1=1 P1=3 Arf2 S2=0 L2=0 A2=0
M2=0 P2=0
                      ! ARF loading for VMS2RF
IMM2A_P Arf7 P=3                    ! ARF loading for VMS2RF
IMM2BS BufferAdd XM_Add Len Mod 0x15 ! Parameter loading for XM2DMB
...
PRAGMA_WAIT_ARF_ON ARF7
XM2DMB                              ! DMA activation
PRAGMA_WAIT_ARF_OFF ARF7
...
\dup i=0 to Len-1
VMS2RF_U <Coeff_Reg+2×i> Arf7  ! Data move from Buffer to Register
File
\enddup
```

In this way the assembler/optimizer will schedule the VMS2RF instructions in the right order with respect to the XM2DMB instruction: they will be allocated at a sufficient distance from the DMA activation to ensure that the data moved using ARF7 is already present in the Buffer when the VMS2RF is executed.

Program Sequencing

The mAgic program flow is mostly linear: the VLIW instructions are fetched from the Program Memory and executed one after the other. The only exception to this sequential program flow is represented by calls to subroutines, branches and loops, which, in some cases, are executed only after a condition evaluation.

The unit who is in charge of the management of the program sequencing is the Local Controller. It provides the following features:

Program Counter increment

Program Counter Stack management

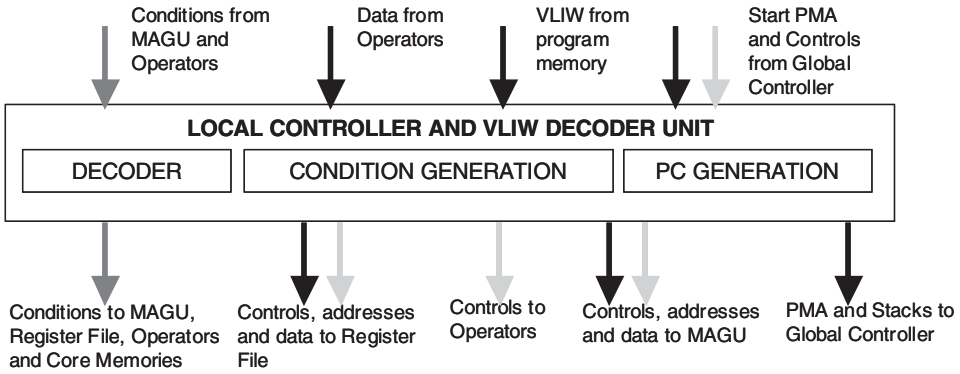
Condition Evaluation

Condition Stack management

VLIW fields decoding

In order to perform all these actions, the Local Controller Unit is divided into three devices: the Decoder, the Program Address Generation Unit and the Condition Generation Unit.

Figure 14. Local Global Controller High-level Block Diagram



Decoder Unit

The Decoder Unit dispatches all the signals necessary to activate the devices and to transfer immediate values. Its input is the VLIW word from the Program Memory. To decode the VLIW word, so to extract all the fields that drive the mAgic devices, the Decoder uses the Flow Control field (FLOC) in the VLIW word.

FLOC VLIW FIELD										
Conditioning				Flow Control						
117	116	115	114	113	112	111	110	109	108	107
imm	mem	mul	add	128 Flow Control Codes						

The FLOC VLIW field contains the operative codes for the Flow Control and the VLIW Decoder Unit. It consists of 11 bits. The 7 LSBS (107-113 bits) are used for the program flow control: they contain the code of all the instructions changing the program flow, the code of the immediate loading instructions and the code of some move instructions. Thanks to this field, the Decoder Unit is able to dispatch the correct VLIW subfields to the right devices. Despite the name, the Decoder Unit does not perform a real decoding of the program word, but only a correct division of its subfields and a correct routing of these subfields toward the mAgic internal devices.

The 4 MSBS (114-117 bits) are used for the program conditioning: the bit number 114 is connected to the addition instructions, the bit number 115 is connected to the multiply instructions, the bit number 116 is connected to the writing memory instructions and the bit number 117 is connected to the immediate loading instructions and to all the instructions that move data from the Register File to the ARF and from the ARF to the Register File. When these bits are equal to "1" the corresponding set of instructions coded in that VILW word are conditioned with the condition contained on the top of the stack (if the condition is equal to "1" the instructions are executed, otherwise not), while if the bits are equal to "0" the instructions are not conditioned and they are always executed.

These bits are activated through the PRAGMA instructions as in the following scheme:

- the PRAGMA CONDARIT_ON switch on the bit 114 and the bit 115, while the PRAGMA CONDARIT_OFF switch off the same bits
- the PRAGMA CONDMEM_ON switch on the bit 116, while the PRAGMA CONDMEM_OFF switch off the same bit
- the PRAGMA CONDIMM_ON switch on the bit 117, while the PRAGMA CONDIMM_OFF switch off the same bit
- the PRAGMA CONDALL_ON switch on all the bits 114, 115, 116 and 117, while the PRAGMA CONDALL_OFF switch off the same bits

Program Address Generation Unit

The Program Address Generation Unit is devoted to manage the correct Program Counter generation according to the program flow. It performs a simple increment by one on the Program Address in order to generate the address of the next instruction in linear code execution. Otherwise, in non-sequential program flow, it generates Program Counter for branch instructions, loops, calls to subroutines and returns from subroutine.

Inside the Program Address Generation Unit there is the Program Address Stack (PAST) holding the returning address after a call to a subroutine.

The generation of a new Program Counter is determined by five main sources, according to the program flow:

1. PINC is the previous Program Counter + 1 (in sequential program flow)
2. an immediate value from VLIW (when branch to an immediate value is required)
3. SAR (Start Address Register) from Global Controller device (at every mAgic transition from System Mode state to Run Mode state)
4. PAST that's the top of the internal Program Address Stack (in return form Subroutines)
5. ISVR, JMPA or StartBlock from mAgic Interrupt Controller or internal Jump register (when it is required respectively a branch to an Interrupt Service Routine, a branch with an offset contained in the dedicated Jump register or a branch inside a Repeat Instruction)

PC Stack

The PC Stack holds the return address after a subroutine call. It is 32 locations deep by 13 bits wide. The stack is popped when the program returns from a subroutine. Note that no exception is generated on PC Stack overflow or underflow. The effect is that when an overflow occurs, the Program Address allocated at the bottom location of the stack is lost. The user must prevent stack overflow by avoiding that the number of the nested calls instructions is greater than 32.

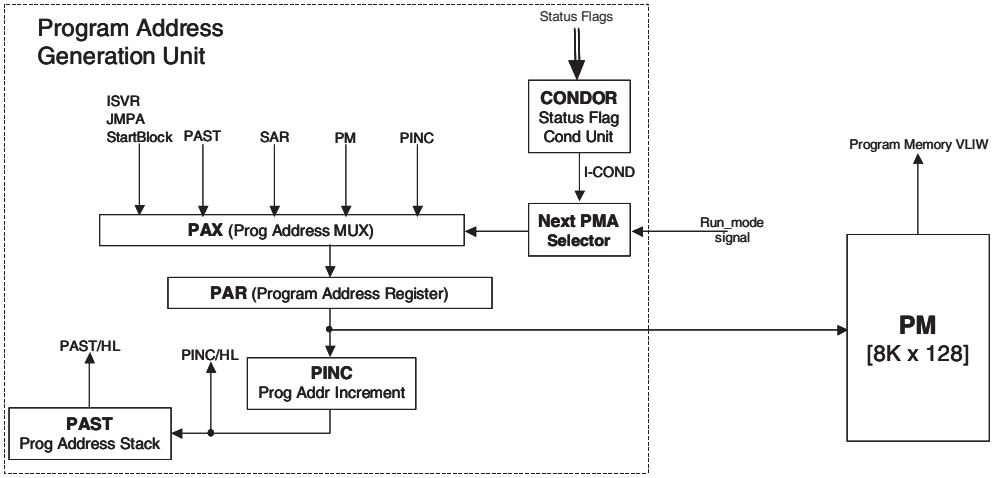
Condition Generation Unit

The program flow control capability of the mAgic DSP architecture is described in the current section. The Condition Generation Unit is the device generating the condition necessary to perform branches and to enable the writing of conditional operations results.

The Condition Generation Unit, depicted in Figure 16 on page 59, takes in input the left and the right conditions generated by the operators block, an external condition generated by the Multiple Address Generation Unit (exactly by the LOOPGU in the INCCOMP instruction) and the VLIW Flow Control Field that drives the device.

Two boolean conditions are generated in output: the LCOND conditioning the instructions on the left path and the RCOND conditioning the instructions on the right path.

Figure 15. Program Address Generation Unit



The main components of this unit are the two Condition Combiners, Left and Right, used for manipulating the top of the stack, two Flags Stacks, collecting the left and the right conditions, the Flasc unit, acting as a Flag Selector Combiner, and a Decoder of the VLIW Flow Control field.

Whenever a condition is generated, it is automatically pushed on the top of the corresponding Flag Stack (L or R), so that it can be inquired later on.

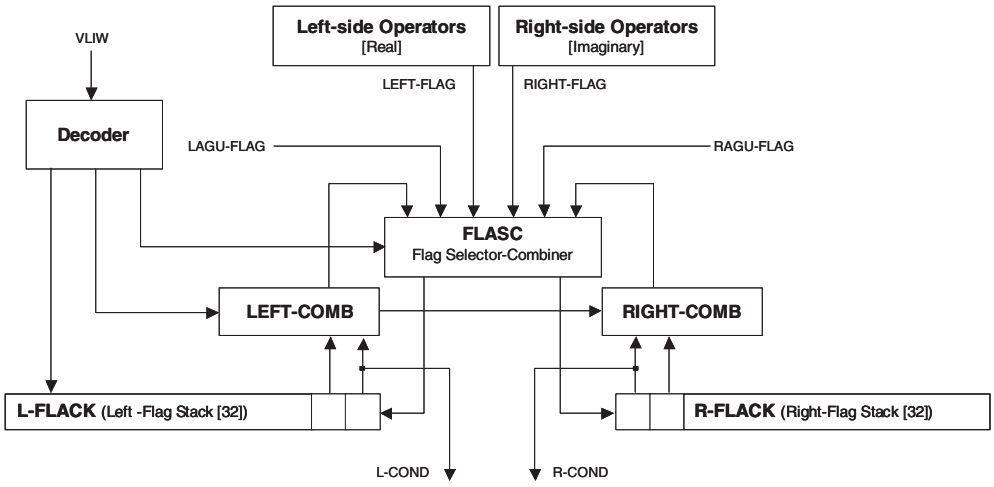
A set of logical operations are allowed on the top of the stack:

- AND, OR: they pop the first two elements and push the result of the corresponding logical operation
- NOT: it substitutes the first element with its negation
- REPUSH: it duplicates the top of the stack
- SWAP: it swaps the first two elements
- POP: it pops the top of the stack
- AND2: it pushes, on the top of the stack, the logical AND between the first two elements by popping only the element on the top
- FSI2IQ: it pushes the left condition on the Left and Right stacks
- FSQ2IQ: it pushes the right condition on the Left and Right stacks

In complex operations only equate and not equate conditions can be evaluated (CEQ and CNE). To obtain that all works properly, the condition generated from the operators block must be pushed on both Flag Stacks.

In vectorial operations the following conditions are evaluated, for floating point or integer operands: equate (VFEQ or VIEQ), not equate (VFNE or VINE), less than (VFLT or VILT), greater than (VFGT or VIGT), less-equal (VFLE or VILE), greater-equal (VFGE or VIGE). Two conditions are generated independently from the operators block on each path, LCOND and RCOND. They are pushed on the two Flag Stacks and used to condition the subsequent instructions on the two paths separately.

Figure 16. Condition Generation Unit



In single operations the following conditions are evaluated, for floating point or integer operands: equate (FEQ or IEQ), not equate (FNE or INE), less than (FLT or ILT), greater than (FGT or IGT), less-equal (FLE or ILE), greater-equal (FGE or IGE). As in complex cases, even if the condition is generated only from one path of the operators block, it is pushed on both Flag Stacks. This way, the condition evaluated in the left path is also available for the instructions conditioning on the right path and vice versa.

Internal Local Controller Registers

In the Local Controller there are 8 internal write only registers. These registers can be written via the LDAI or LDAR microcode. The full list of internal registers is detailed in Table 24.

Program Counter, Status register and L/R stack registers can be transferred to the RF via the following instructions: PC2RF, STAT2RF, STK2RF. A bit count instruction ONES can be used to calculate the number of ones into a Register File register. The list of these instructions is in Table 25.

Table 24. Internal Local Controller Registers

Register	Reset Value	Comment
JMPA[12:0]	0x0	Jump address register for BR_REG instruction
ISVR0[12:0]	0x0	Interrupt service routine address “0” (external interrupt)
ISVR1[12:0]	0x0	Interrupt service routine address “1” (internal OS)
ISVR2[12:0]	0x0	Interrupt service routine address “2” (internal USER)
ISVR3[12:0]	0x0	Interrupt service routine address “3” (internal USER)
IMR[3:0]	0xF (all masked)	Interrupt mask register active high
ISCR[3:0]	0x0	Interrupt Set/Clear corresponding interrupt request
IPR[7:0]	0xE4	Interrupt Priority Register set priority on the four interrupt lines, at reset the priority is: int[0]: 0 (highest) int[1]: 1 int[2]: 2 int[3]: 3 (lowest)

Table 25. Internal Local Controller Registers Transfer Instructions

Microcode	Data	We Enable
LDAI	Vliw[58:43]	vliw[90:83]
LDAR	Dinl[15:0]	vliw[90:83]
PC2RF	PC	RF 7I, 7Q
STAT2RF	Run machine status register	RF 7I, 7Q
STK2RF	L/R registers	RF 7I, 7Q
ONES	Dinl[31:0]	RF 7I, 7Q

Branch

The sequential program flow can be interrupted by branch instructions. A branch may consists of a simple jump, of a call to a subroutine, or of a return from a subroutine. In all cases the program flow is interrupted and transferred to a different Program Memory location.

A simple jump, identified as a “branch” in mAgic assembly language (BR), instructs the Local Controller to fetch and execute instructions from a Program Memory location that is different from the subsequent one. The new address for the branch is specified with the help of a label written at the beginning of the portion of the code where the user wants to jump.

A call instruction (CALL) branches to the address corresponding to the specified subroutine (in this case too the univocal identification occurs through a label) and pushes a return address in the Program Counter stack. That program address will be used when a return instruction will be fetched at the end of the subroutine.

A return instruction (RETURN) produces a branch to the address stored at the top of the Program Counter stack. At the same time that address is popped from the top of the PC stack.

Simple branch, call and return instructions can be conditional (BRIF_NOT, CALLIF_NOT and RETURNIF_NOT). In this case, the Local Controller evaluates the condition on the top of the Condition Code Stack and executes the branch only if the condition is true (it means that if the top of the stack contains “1” the branch is executed). Otherwise, the branch is not executed and the program flow goes on in a sequential way, ignoring the fetched branch instruction.

For branch and call (but non return) instructions it is possible to specify the jump address using the Local Controller register JMPA. The jump address can be preloaded in this register using an appropriate instruction (LDAI or LDAR) and then it can be used with the BR_REG and CALL_REG instructions.

Branch Instruction Scheduling

Every time a branch or a label is inserted in the assembly code for a pipelined machine, an overhead in code execution is generally introduced. As a matter of fact, these instructions represent a kind of barrier for the instruction scheduling mechanism of which the user must be aware of and be careful in optimizing the code exclusively between one barrier and the other without moving instructions behind them. Any instruction written in the sequential code by the user after a branch or a call instruction cannot be moved from the optimizer before that branch. Same for the label: all the instructions before a label cannot be moved afterwards, and all the instructions after a label cannot be moved before.

However, sometimes branches, as well as calls, returns and labels are obviously necessary in a code. mAgic DSP is provided with an advanced instruction scheduling mechanism that is able to minimize and often to annul the overhead introduced from these kinds of instructions.

The user writes branch instructions (branches, calls and returns) in the assembly code wherever their execution is wanted. The assembler-optimizer schedules them correctly in order to avoid loosing a cycle during their execution.

Because the pipeline between the program VLIW fetch and its execution is equal to two cycles, if the branch were scheduled exactly in the last VLIW before the jump, two cycles would be lost, as described in the following example.

Suppose that the VLIW n contains the branch to address j. The machine must execute VLIW n-2, then VLIW n-1 and finally VLIW n. At this point there is the branch, and VLIW n+1 and VLIW n+2, already in the fetch-execute pipeline, must not be executed: there must be two nops. The machine must wait until the VLIW j, the arrival VLIW after the jump, is decoded and decompressed (2 cycles), before executing any instruction again.

Code:

```
VLIW n-2
VLIW n-1
VLIW n                  (VLIW containing the jump to j)
VLIW nop
VLIW nop
....
....
....
VLIW j                  (arrival VLIW after the jump)
VLIW j+1
VLIW j+2
```

In cycles this case can be described as follows:

	Cycle k	**Cycle k+1**	**Cycle k+2**	**Cycle k+3**
Execute	VLIW n (BR to j)	VLIW nop	VLIW nop	VLIW j
Decompress	VLIW nop	VLIW nop	VLIW j	VLIW j+1
Fetch	VLIW nop	VLIW j	VLIW j+1	VLIW j+2

The table shows that, if the jump were scheduled in the exact moment in which it should be executed, two cycles would be spent in NOP instructions.

On the contrary, the assembler-optimizer is able to schedule the branch two cycles before it should be executed, so that no cycle is lost. The situation is the following: the branch is again contained in VLIW n, but this time the VLIWs following it do not contain two nops but two instructions (k and p) that in the sequential assembly flow precede (not follow as in the optimized code) the branch instruction. The two VILW k and p this time will be executed in the two cycles waiting for the arrival of the VLIW j (the arrival VLIW after the jump).

Optimized code:

VLIW n-2

VLIW n-1

VLIW n (VLIW containing the jump to j)

VLIW k (In sequential code the instructions here contained precede VLIW n)

VLIW p (In sequential code the instructions here contained precede VLIW n)

....

....

....

VLIW j (arrival VLIW after the jump)

VLIW j+1

VLIW j+2

In cycles:

	Cycle k	Cycle k+1	Cycle k+2	Cycle k+3
Execute	VLIW n (BR to j)	VLIW k	VLIW p	VLIW j
Decompress	VLIW k	VLIW p	VLIW j	VLIW j+1
Fetch	VLIW p	VLIW j	VLIW j+1	VLIW j+2

In this case, VLIW k and VLIW p, already in the fetch-execute pipeline when the branch (VLIW n) arrives, will be executed because the two VLIW logically precede the branch and do not follow it. This way no cycle is lost in nop instructions waiting for the arrival of the VLIW j.

Obviously, VLIW k and VLIW p must not contain any other branch instruction (branch, call or return), but this is automatically checked and resolved by the advanced scheduling mechanism of the assembler optimizer.

Note that all this optimization mechanism is absolutely transparent to the user. The code can be written in a sequential way, disregarding parallelism, optimization and scheduling problems.

Loops

In the mAgic assembler the loops are implemented by using a dedicated instruction (INCCOMP) and a conditional branch instruction (BRIF_NOT).

The INCCOMP is used to perform the loop counter increment/decrement, the condition evaluation and the loop counter update. The subsequent conditional branch performs the jump only if the condition is not verified (BRIF_NOT). When the loop-end is reached, the condition becomes true, then the jump is not executed anymore.

In the mAgic DSP a reserved register for the loop counter or for the loop-end value does not exist. An ARF register is used to this aim. The fields of this register are used in the following way: the L field contains the loop-end value, (i.e. the number of iteration of the loop - 1); the M field contains the step value; the A field contains the loop counter value. For a pure use of this register in an INCCOMP instruction, the S and P fields can be ignored. It is different if the same register is also used to address a vector in the memory space.

We can start describing the first situation: the ARF register used only for loop parameters. Suppose we have to implement the following high-level code:

```
for i=1 to 20 step 2
  begin
    ....
    ....
  end
```

with a loop-end value equal 20, a loop counter starting from 1 and a step equal 2. The ARF register must be loaded like this: L=20, A=1, M=2. S and P can be loaded with 0. The resulting assembly code is the following (the ARF 6 has been used):

```
IMM2A6 S=0 L=20 A=1 M=2       ! ARF register initialization
LABEL loop
....
....
INCCOMP 6                     ! A6=A6+M6 and test if A6<L6
BRIF_NOT loop                 ! branch to "loop" if test is false.
```

The ARF used in INCCOMP instruction can be used for memory addressing as well. Suppose that at every loop iteration the user wants to read data from a vector allocated in the memory. In this case everything proceeds as before but the user must set the proper value for the S and P fields in order to point to the vector with the ARF. Note that the number of read elements of the vector is equal to the loop length. In this case the assembly code is:

```
IMM2A   6 S=50 L=20 A=1 M=2   ! ARF register initialization
LABEL loop
....
VMS2RF 60 6                   ! read from memory
....
INCCOMP 6                     ! A6=A6+M6 and test if A6<L6
BRIF_NOT loop                 ! branch to "loop" if test is false.
```

At each iteration, the vector element at address 50+2×i (i = iteration index) is moved into the register 60 of the Register File. At the end of the loop, 20 elements of the vector have been read. This way, it is possible to save one ARF register, using the same both for addressing and for loop control.

The addressing with the ARF can be done in a linear way, as in the previous example, but also in bit-reversal. In this last case the code looks like as follows:

```
IMM2A   6 S=50 L=20 A=1 M=2   ! ARF register initialization
LABEL loop
....
VMS2RF 60 6 BTR               ! read from memory
....
INCCOMP 6                     ! A6=A6+M6 and test if A6<L6
BRIF_NOT loop                 ! branch to "loop" if test is false.
```

In this case also 20 elements are read from the Internal Data Memory with bit reversal accesses, and a loop of 20 iterations is also done.

Synchronization Barriers

Interrupt Control

An interrupt exchange mechanism on dedicated lines allows an efficient synchronization system between the DSP and the ARM System. mAgic is provided with six software interrupt output signals, while it can receive up to five different interrupt lines.

The software interrupts are sent by mAgic by using the assembly instruction SIRQx, where “x” is a number between 0 and 5. The interrupts are caught by the Advanced Interrupt Controller (AIC) and resent to the ARM according to their priority. When the ARM receives one of these interrupts, the appropriate Interrupt Service Routine is started. These signals are also driven to the Parallel I/O (PIO) lines.

Other four interrupts raised by mAgic are generated by:

- the end of a mAgic DMA transfer
- an exception in the mAAr during the ARM accesses to mAgic internal resources
- a transition of mAgic from Run Mode to System Mode (an HALT mAgic instruction has been executed)
- a transition of mAgic from System Mode to Run Mode

The interrupts sent to mAgic are detected by a mAgic polling on IRQ. WAITEXT and WATCHINT are the instructions used for this polling.

WAITEXT is used to synchronize the DSP with the interrupt Intm0 coming from PIO line 14. The source of this interrupt can be the ARM or an external pin, according to the PIO configuration. When this instruction is decoded, the core is stalled waiting for the arrival of the interrupt. This interrupt is used to wake the DSP up from idle state (See Section “Idle State” on page 70).

WATCHINT is used to synchronize the DSP with four interrupts Intm1, Intm2, Intm3 and Intm4 coming respectively from PIO line 15, 16, 17 and 18. According to the PIO configuration, the source of the interrupt Intm1 can be the ARM or an external pin, while the source of the interrupt Intm2, Intm3 and Intm4 can be the ARM or periodic signals generated by three different timer channels. The WATCHINT instruction looks for pending interrupts. If an interrupt has been received, it is served by starting an interrupt service routine. If no pending interrupt is found, the program flow goes on without any change. The WATCHIN instruction never stretches the core. If more than one pending interrupt is found, these are served according to their priority, as set in the Interrupt Priority Register (IPR). In this register, as depicted in Table 26, four different priority levels are assigned to the four interrupt input lines:

prioline0[1:0] is the int line with the highest priority

prioline1[1:0] is the int line with the medium-high priority

prioline2[1:0] is the int line with the medium-low priority

prioline3[1:0] is the int line with the lowest priority

The default is indicated in the same table.

The IPR register is written by using the mAgic instructions LDAI or LDAR. For more details on these instructions refer to [4] in Section “Related Documents” on page 268.

Note: The four IPR fields must assume different values. Otherwise the priorities are unpredictable.

Table 26. Interrupt Priority Register

prio3line[1:0]	prio2line[1:0]	prio1line[1:0]	prio0line[1:0]
default: 0×3	default: 0×2	default: 0×1	default: 0×0

External Memory Synchronization Mechanisms

The mAgic DSP can share its External Memory with another processor. A safe mechanism of bus request/bus grant allows a secure bus sharing between mAgic and the external processor. mAgic has the possibility of enabling or disabling the bus sharing through the use of GNTON/GNTOFF instructions. GNTON enables the mAgic DSP to accept access requests to its External Memory coming from an external device. This access enable can be removed through the instruction GNTOFF. When the enable is “on” and an external device requests access to the External Memory, the bus is granted as soon as the current mAgic transfer (if there is one) is finished. If the enable is “off” no grant is given to any request by an external device.

This simple mechanism allows a secure bus sharing system of mAgic External Memory.

mAgic-ARM Interface (mAAr)

This block is a standard ARM AMBA ASB slave device. It provides memory mapping of mAgic internal devices on the ASB memory space. This interface allows the ARM micro controller to control / initialize the mAgic DSP both in Run and System Mode.

In Run Mode, the mAgic DSP works under direct control of its own VLIW program and the ARM has access to the 1K × 40-bit dual ported shared memory (PARM) and to mAgic Command Register.

In System Mode, the mAgic DSP core halts its execution and the ARM takes control of it. In this mode ARM can access some DSP devices (Program Counters, Data Memory, Program Memory, etc.) The ability of the ARM to access internal mAgic resources in System Mode is especially useful for initialization and debugging purpose. By Accessing the Command Register, the ARM can change the operating status of the DSP (Run/System Mode), force single or multiple step executions, or simply read the DSP operating status.

System Mode Functions

In System Mode, the mAgic DSP core halts its execution and the micro-controller takes control of it. In this mode, the ARM interacts with some mAgic DSP internal resources (Data Memory, Program Memory, Internal Registers, etc.) for initialization and debugging.

The main features supported by the mAgic DSP in System Mode can be summarized as follows:

- The possibility for the ARM to read and write the mAgic DSP Local Data Memories and Program Memory. In this way the program code and the related data can be loaded into the mAgic memories
- The loading of the Program Memory Start Address into the Start Address Register. When the mAgic DSP changes into Run Mode, it will start the program execution from the specified Program Memory location
- The possibility for the ARM to activate the mAgic DSP in a debugging Run Mode, forcing the DSP to a step by step execution. The mAgic core executes instructions as in Run Mode, halting after every clock cycle

- The possibility to access the interface mAgic registers and the internal mAgic registers: the Command register, the Control and Status registers, the Exception registers, the DMA registers, the Condition Stack, the Program Counter and the Program Counter Stack

The interface provides a buffer mechanism that allows solving the problem of data type differences between the mAgic DSP (Data Memory 40-bit word, Program Memory 128-bit word) and the ARM (32-bit word).

During a program memory read operation (from the mAgic DSP to the ARM), the interface receives 128-bit data from the mAgic Core, but it sends only a 32-bit word at a time to the AMBA bus.

In the same way, during a program memory write operation (from the ARM to the mAgic DSP), the interface allows loading sequentially four 32-bit words and sends them in parallel to the Program Memory. This means that the internal device is selected and data sent to the mAgic DSP internal resource only when the most "significant" word to write is received from the interface,. The accesses (read or write) to the mAgic program memory must be done in atomic sequences composed of four accesses to consecutive locations aligned at the correct address boundaries (See Section “Memory Mapping of the mAgic DSP Internal Resources” on page 80).

This approach is also used during read/write operations from/to the Internal Data Memory: in this case the first 32-bit word from the wrapper will be registered as Buffer WORD0 while the second one as Buffer WORD1; WORD1 will contain the most significant eight bits of the 40-bit Data Memory word (See Table 27), while its remaining part will be pulled down.

Table 27. 40-bit Data

	40-bit mAgic word			
	39	**32**	**31**	**0**
Hardwired to ‘0’	8 MSB of 40-bit data word		32 LSB of 40-bit data word	
24 bits (WORD1)	8 bits (WORD1)		32 bits (WORD0)	
32-bit ARM WORD1			32-bit ARM WORD0	

Examples of the data protocol during program memory read/write operations, are shown in Figure 17 and Figure 18.

Figure 17. Write a Program Memory Word

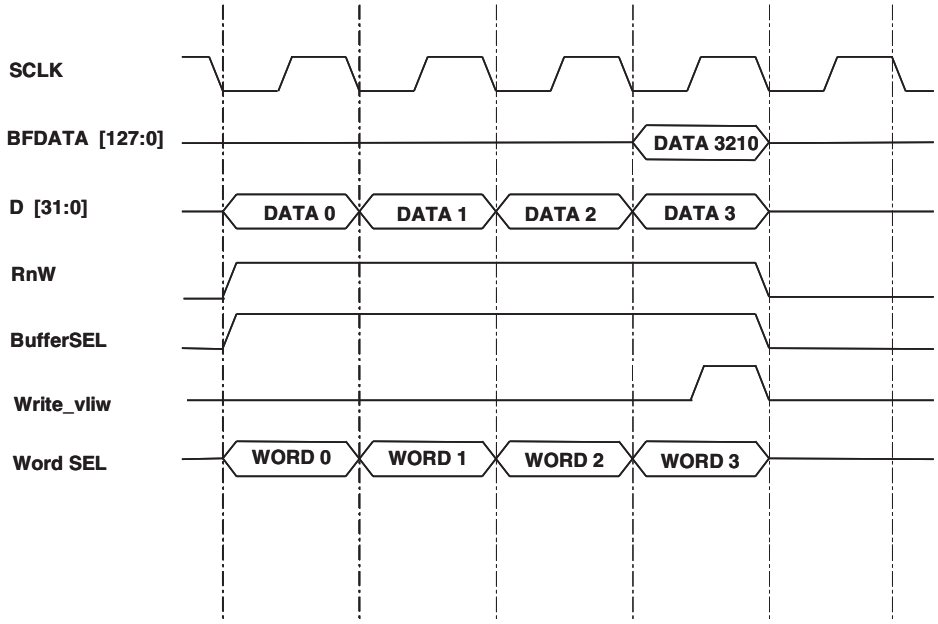
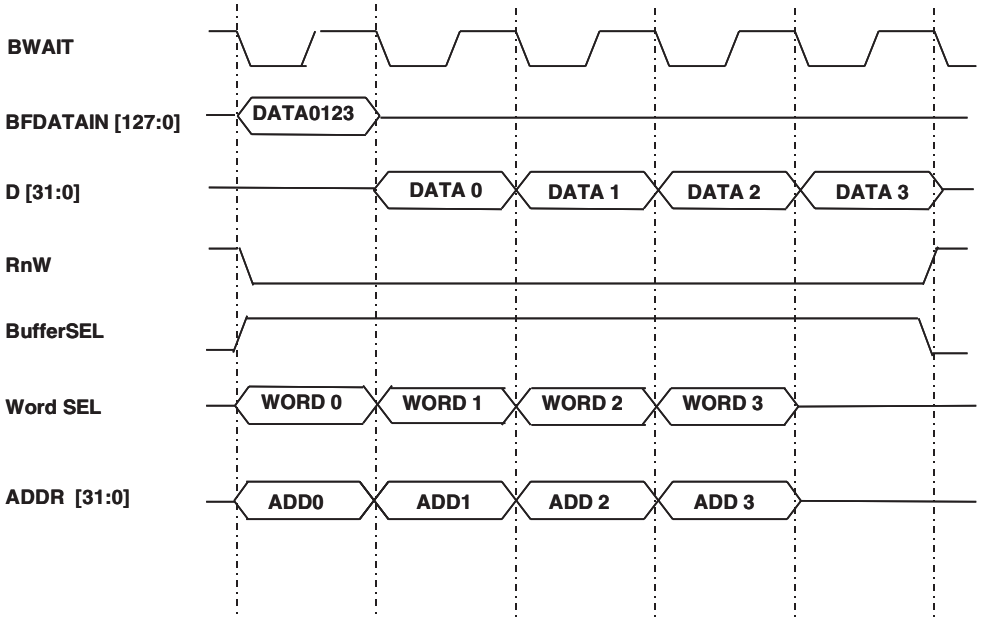


Figure 18. Read a Program Memory Word



Step Mode

When the mAgic DSP is in System Mode, the user can decide to run a mAgic program in Step Mode for debugging purpose. This means that the DSP is started in Run Mode for a single clock cycle, then mAgic returns in System Mode. In this way the user can fully debug the DSP status by reading all the mAgic memory mapped resources. The user can read (but also write) the Program and the Data Memories; and access to the Interface and to the Internal mAgic DSP registers is allowed. As a result, the user has the possibility to know what happened in the machine during the last clock cycle.

Then the user can decide to start mAgic again for a further cycle. Immediately after the execution of the cycle, the DSP will be stopped again and it will be fully debuggable as in the previous cycle. This modality can be continued until the end of the mAgic program. If the user is no more interested in the Step Mode, the machine can be put, whenever wanted, in the normal Run Mode, in order to complete the program execution.

The Step Mode is activated by writing the appropriate values in the Internal register named Step_Mode. The bits in the register must be set according to the following table.

Step_Mode Bits	Description	Reset Value	Note
0	step mode activation	0	“0” step mode disabled “1” step mode enabled
1	continue step mode	0	“0” next cycle step mode stop “1” next cycle step mode continue
2 to 31	Reserved	0	-

Run Mode Functions

In Run Mode, the mAgic DSP core executes its program and the ARM can check and control the DSP status by reading the mAgic registers. While writing the configuration register, the ARM processor can stop the DSP or let it run again; it can specify the execution of only one clock cycle (single step modality) by writing the Step_Mode register, or it can check the DSP status by simply reading the Status register.

The ARM can also read and write the PARM (512 40-bit words for the left PARM bank and 512 40-bit words for the right PARM bank), which is used as a shared double port memory with mAgic. This is the only way by which the two processors can exchange data during Run Mode.

mAgic is tightly coupled with both the ARM micro-controller and the ARM system through the use of synchronization signals based on interrupts. mAgic is provided with six software interrupt output signals, while it can receive up to five different interrupt lines. Besides, mAgic is able to send other four interrupts when well determined situations occur (a DMA end, an exception, the execution of the HALT instruction and the transition of mAgic from System Mode to Run Mode). The next section summarizes the interrupt scheme.

Signal from the ARM to mAgic

The interrupt sent to mAgic are summarized in the next tables. They are divided into two groups, according to the source which generates them.

- INTM(4:2) → interrupt request from ARM (through PIO configured as SW controlled) or from ARM System TIMERS (through PIO configured as peripheral linked) to mAgic

Sources		Destination
ARM (PIO)	**ARMSystem TIMERS**	**mAgic**
pio(16) internal	Timer B0	INTM(2)
pio(17) internal	Timer B1	INTM(3)
pio(18) internal	Timer B2	INTM(4)

Note: PIO[ ] pad and pio( ) internal bus indexes do not exactly coincide (See “PIO: Parallel I/O Controller” on page 143).

INTM(1:0) → interrupt request to mAgic and ARM (through PIO configured as Peripheral) from external PAD

Sources	Destination	
PAD	**ARM (AIC)**	**mAgic**
PIO[14]	AIC _IRQ(30)	INTM(0)
PIO[15]	AIC _IRQ(31)	INTM(1)

The interrupts sent to mAgic are detected by a mAgic polling on IRQ. WAITEXT and WATCHINT are the instructions used for this polling.

WAITEXT is used to synchronize the DSP with the interrupt Intm0 coming from PIO line 14. The source of this interrupt can be the ARM or an external pin, according to the PIO configuration. This interrupt is used to wake the DSP up from idle state (See Section “Idle State” on page 70).

WATCHINT is used to synchronize the DSP with four interrupts Intm1, Intm2, Intm3 and Intm4 coming respectively from PIO line 15, 16, 17 and 18. According to the PIO configuration, the source of the interrupt Intm1 can be the ARM or an external pin, while the source of the interrupt Intm2, Intm3 and Intm4 can be the ARM or periodic signals generated by three different timer channels. The WATCHINT instruction looks for pending interrupts. If an interrupt has been received, it is served by starting an interrupt service routine. If no pending interrupt is found, the program flow goes on without any change. If more than one pending interrupt is found, these are served according to their priority, as set in the Interrupt Priority Register (IPR).

Signal from mAgic to the ARM

Six software interrupts can be raised by mAgic. The interrupts are caught by the Advanced Interrupt Controller (AIC) and resent to the ARM according to their priority. When the ARM receives one of these interrupts, the appropriate Interrupt Service Routine is started. These signals are also driven to Parallel I/O (PIO) lines.

- FPU_SIRQ(6:0) → interrupt request from mAgic to ARM (through AIC) also available on external pads through PIO lines (if PIO are configured as peripheral).

Source	Description	Destinations	
mAgic		**ARM (AIC)**	**PAD**
FPU_SIRQ(0)	SW interrupt	AIC _IRQ(8)	PIO[8]
FPU_SIRQ (1)	SW interrupt	AIC _IRQ(9)	PIO[9]
FPU_SIRQ (2)	SW interrupt	AIC _IRQ(10)	PIO[10]
FPU_SIRQ (3)	SW interrupt	AIC _IRQ(11)	PIO[11]
FPU_SIRQ (4)	SW interrupt	AIC _IRQ(12)	PIO[16]
FPU_SIRQ (5)	SW interrupt	AIC _IRQ(13)	PIO[4]
FPU_SIRQ (6)	DMA end	AIC _IRQ(27)	PIO[17]

Other three interrupts are raised by mAgic. They are generated by:

  – a transition of mAgic from Run Mode to System Mode (an HALT mAgic instruction has been executed)
  – a transition of mAgic from System Mode to Run Mode
  – an exception in the mAAr during the ARM accesses to mAgic internal resources

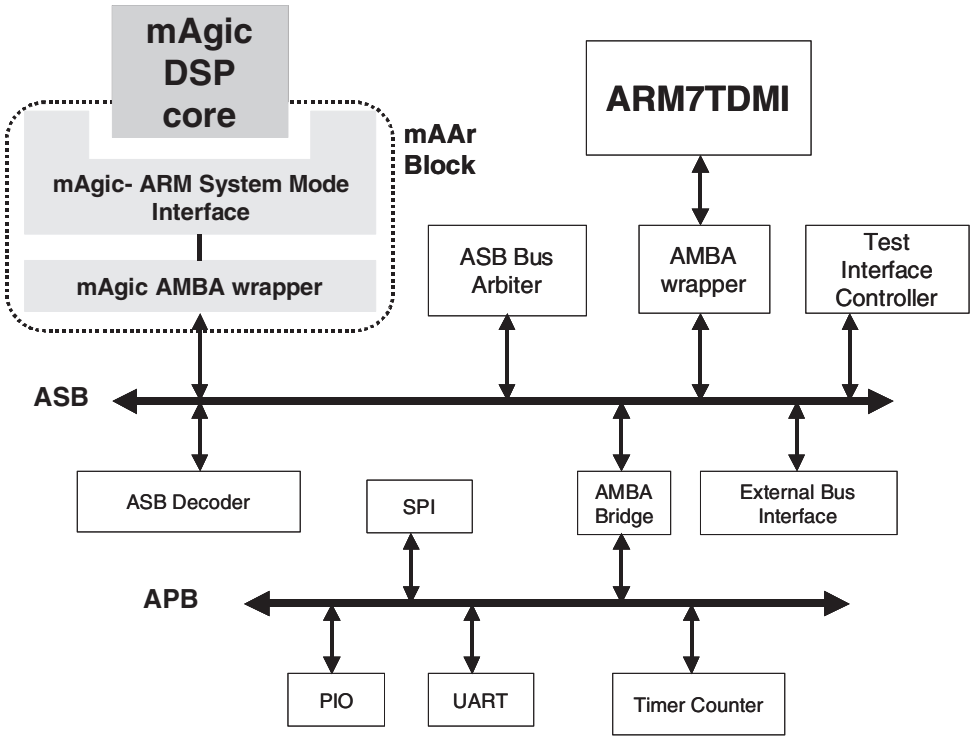
- fpu_halt, fpu_mode, fpu_exc → interrupt request from mAgic to ARM (through AIC) also available on external dedicated pads.

Source	Description	Destinations	
mAgic		**ARM (AIC)**	**PAD**
fpu_halt	mAgic halt	AIC_FIQ (IRQ(0))	FPU_HALT
fpu_mode	mAgic mode	AIC_IRQ(25)	FPU_MODE
fpu_exc	MAAR exception	AIC_IRQ(15)	FPU_EXC

Idle State

When mAgic is in Run Mode, it can be forced through software in an idle state. When the DSP executes a WAITEXT instruction the following actions occur: the mAgic core is stopped with a clock stretch, while all the DMAs, already launched or queued to be executed, are completed. The core waits for a signal, either external or generated by the ARM, which is able to reactivate it. When this kind of interrupt has been received, the mAgic core wakes-up and goes on with its program execution. This mechanism can be used every time the user wants to synchronize the DSP with the ARM or with an external device; moreover, it can be useful to reduce power dissipation.

Figure 19. Conceptual Model of the mAAr Block and the On-chip AMBA Bus



mAgic Interface Registers

The Interface mAgic registers, shortly named CSE registers (Command, Status & Exception), are a limited number of registers (four registers) used to interface the DSP with the ARM. They summarize the DSP status and the DSP exceptions, also determining the activation of some interrupts (halt, bus error) toward the ARM. Moreover, through the CSE Command register, the DSP can be forced to Run or to System Mode by the controller.

The Interface registers include the four following registers:

- Command register – used to control the core status mode (Run Mode / System Mode)
- Status register – used to reflect the current status of the core
- Exception register – used to collect the exceptions from the core and from the mAgic-ARM interface
- Exception Mask register – used to mask all the exceptions of the previous register

Table 28. mAgic Interface Registers

Address	Register	Name	Access	Reset State
0x460000	Command Register	MAAR_CMD	Read/Write	0x0
0x460004	Status and Configuration Reg.	MAAR_STATUS	Read Only	0x0
0x460008	Exception Register	MAAR_EXC	Read Only	0x0
0x46000C	Exception Mask Register	MAAR_MASK	Read/Write	0x0

The Command Register

Table 29 shows the format of the Command register. Only bit (0) is significant. When it is equal to 0 the mAgic DSP is in System Mode, when it is equal to 1 the mAgic DSP is in Run Mode.

During Run Mode (bit (0) =1), the bit (0) can be internally updated to 0 by mAgic when the HALT or the Exception (if not masked) signals are raised. This bit can also be set to 0 by the ARM, if the controller wants to stop the mAgic execution.

When reset signal (RESET) is active the register is cleared:

Table 29. Command Register

Spares	Mode
Bits (31:1)	Bit (0)

For example, possible uses of this register are:

1. System Mode to Run Mode transition: the user writes from the ARM program the value 0x00000001 to the Command register to run the program;
2. Run Mode to System Mode transition: the user writes from the ARM program the value 0x00000000 to the Command register, with the following effects:
   – The core (operators, Local Controller, address generator, register file) and all the mAgic Interface and Internal registers freeze.
   – All the internal buses are switched from Run Mode to System Mode. In this way the buses allow the access to all the mAgic memory mapped devices. Note that DMAs are not affected by this status change, so that if a DMA is running during this bit change, it is not stopped and it continues until the end of all the scheduled DMAs.

The Status Register

Table 30 shows the format of the Status Register:

Table 30. Status Register

Spares	mAgic-ARM Interface Exceptions	CORE Exceptions	Current Mode Status	Spares
Bits (31:19)	Bit (18)	Bit (17)	Bit (16)	Bits (15:0)

The Status register reflects the current status of the mAgic-ARM interface and of the mAgic DSP. The significant bits in this register are:

- Bit (16): current mode status bit. If mAgic is in Run Mode the bit is set to 1, if mAgic is in System Mode the bit is set to 0
- Bit (17): core exception. If an exception occurred in the mAgic core the bit is set to 1, otherwise it is equal to 0
- Bit (18): mAgic-ARM interface exception. If a decoder exception occurred in the mAgic-ARM interface the bit is set to 1, otherwise it is equal to 0

The Exception Register

The Exception Register monitors the exceptions of the mAgic DSP. Bits (3:0) display a decoder exception as detailed in Table 31. Bit (4) displays that an exception occurred in the mAgic core. If more details on the core exception are required, the user must read the GSR Exception register mapped at address 0x45000C of the ARM address space (See Figure 20 on page 81)

.

Table 31. Exception Register

Spares	Core Exception	Decoder Exceptions
Bits (31:5)	Bit (4)	Bits (3:0)

The Decoder Exceptions are:

Table 32. Decoder Exception Errors

Bits (3:0)	Decoder Exceptions
0000	No decoder error
0010	Write on read only registers
0100	Address error
1000	External access during run mode

Note that no exception is raised when a location, defined “unused” in the ARM memory space dedicated to the mAgic resource mapping (from address 0x400000 to address 0x4FFFFF), is accessed. For example, if the ARM wants to access a location between address 0x491000 and 0x49FFFF, no error is returned. On the contrary, if the ARM wants to write a read only register, as for instance the GSR_STAT register at address 0x450008, a decoder exception (0010) is produced.

When one bit of the CSE Exception register is set to 1, the interrupt IRQ_MAAR is sent to the AIC that receives it on its input line 15. Then, it is re-sent to the ARM according to its priority and the micro-controller starts the appropriate interrupt service routine.

The Exception Mask Register

The Exception Mask Register is used to mask the exceptions, but also to mask the HALT signal coming from the mAgic Core. When the user wants to mask the decoder exception, the corresponding bit (3:0) of the Exception Mask register equal 0x1111 must be set. If the user wants to mask the core exception the bit (4) equal 0x1 must be set.

In the same way, if the user wants to mask the HALT signal coming from mAgic, 0x1 must be written in the bit (6) of this register.

Table 33. Exception Mask Register

Spares	Halt from Core	Not used	Core Exception	Decoder Exceptions Mask
Bit (31:7)	Bit (6)	Bit (5) stuck at 0	Bit (4)	Bit (3:0)

mAgic Internal Registers

The Internal mAgic registers, also named GSR registers (Global Status Register), are sixteen registers used to describe as deeply as possible the internal status of the mAgic DSP. These registers collect detailed information on the machine status, the exceptions occurred in the DSP, the current value of several program-sequencing registers (Program Counter, Condition registers, Program Counter stack), the DMA registers, etc. They are visible in System Mode and they can be accessed by the ARM in read or write mode, according to the access type of each one of them. All these registers, summarized in the following table, are detailed in the current section.

Table 34. mAgic Internal Registers

Address	Register	Name	Access	Reset State
0x450000	Program Start Address	SAR	Read/Write	0x0
0x450004	Configuration	GSR CONF	Read/Write	0x1212
0x450008	Status	GSR STAT	Read Only	0x40000
0x45000C	Exception	GSR EXC	Read Only	0x0
0x450010	Exception Mask	GSR MASK	Read/Write	0x0
0x450014	PM Address	PC	Read Only	0x0
0x450018	Local Controller Right Stack	QF_CS	Read/Write	0x0
0x45001C	Local Controller Left Stack	IF_CS	Read/Write	0x0
0x450020	PM Address Stack	S&SP	Read Only	0x0
0x450024	DMA Type	DMA_TYPE	Read/Write	0x0
0x450028	DMA Length	DMA_LEN	Read/Write	0x0
0x45002C	DMA XM Modifier	DMA_MOD	Read/Write	0x0
0x450030	DMA Buffer/PM Start Address	DMA_CADD	Read/Write	0x0
0x450034	DMA XM Start Address	DMA_XMADD	Read/Write	0x0
0x450038	Start DMA	DMA_START	Read/Write	0x1
0x45003C	Step Mode	STEP_MODE	Read/Write	0x0

The Program Start Address Register (SAR)

The Program Start Address register contains the PC value used by the DSP when it is put in Run Mode. In other words, it represents the Program Memory address that is used as first in a program execution. The subsequent addresses are generated in the Local Controller after the program flow.

The SAR is a 32-bit register with only 13 significant bits (LSBs). It is located at address 0x450000 of the ARM memory map and it can be accessed by the ARM in read and in write mode during mAgic System Mode. Before running the DSP, the user writes the starting address of the mAgic program in this register. At reset, the value contained in this register is “0 × 0”.

The Configuration Register (GSR_CONF)

The Configuration register is a 32-bit read/write register containing all the information necessary to configure the DSP and its DMA with the External Memory. The register is configured in System Mode by the ARM. Only 15 bits are effectively used:

Table 35. Configuration Register

Bit	Description	Notes	Reset State
0	multi-DIOPSIS config	“1” multi-DIOPSIS config enabled “0” multi-DIOPSIS config disabled	0
1	Decompression enable/disable	“1” decompression enabled “0” decompression disabled	1
2	Bus grant enable/disable	“1” bus grant enabled “0” bus grant disabled	0
3	Not used	-	0

Table 35. Configuration Register (Continued)

Bit	Description	Notes	Reset State
7 to 4	read/write XM2DM address generation latency	the latency allowed is between 0x0 and 0×F	0001
11 to 8	read/write DM2XM address generation latency	the latency allowed is between 0x0 and 0×F	0010
15 to 12	read/write XM2PM address generation latency	the latency allowed is between 0x0 and 0×F	0001
16 to 32	Not used	-	0×0

- Bit 1: the decompression enable/disable allows the user to either run a compressed mAgic code (enabling the decompression device) or a non-compressed code (disabling the decompression device). The default state is enabled through decompression; the corresponding mAgic code is a compressed code.
- Bit 2: the bus grant is set. If the bit is equal to “1”, the external bus requests will be accepted. Otherwise not. This configuration bit can be changed both in System Mode by writing the register with the proper value, and with the use of GNTON and GNTOFF that set the bit respectively to 1 and to 0.
- Bits 7 to 4: four bits used to configure the External Memory read access latency for XM2DM instructions. The latency depends on the selected External Memory.
- Bits 11 to 8: four bits used to configure the External Memory write access latency for DM2XM instructions. The latency depends on the selected External Memory.
- Bits 15 to 12: four bits used to configure the External Memory read access latency for XM2PM instructions. The latency depends on the selected External Memory.

The Status Register (GSR_STAT)

The Status register is a 32-bit read only register containing all the information on the status of mAgic. The register can be read in System Mode by the ARM in order to know the status of the DSP. This register is updated at every cycle through the machine status. When a non-masked exception occurs, the register is frozen thus allowing the user to know the exact status of the DSP. Only 18 bits are effectively used:

Table 36. Status Register

Bit	Description	Notes	Reset State
0	Start	“1” DSP is running the 1st cycle “0” otherwise	0
1	Mode	“1” DSP in Run Mode “0” DSP in System Mode	0
2	Exception raised	“1” an exception occurred “0” no exception occurred	0
3	Clock enable	“1” DSP core running “0” DSP core stalled	0
4	Halt	“1” HALT instruction decoded “0” no HALT decoded	0
5	Runsys signal	“1” DSP in Run Mode “0” DSP in System Mode	0
6	Reserved	It is always set to “0”	0
9 to 7	Not used	-	000

Table 36. Status Register (Continued)

Bit	Description	Notes	Reset State
10	LRstacksel	“1” flag stack R selected “0” flag stack L selected	0
11	Rwstacksel	“1” write on flag stack “0” read on flag stack	0
15 to 12	DMA type	Define the DMA type: “1010” DM2XM “0101” DM2XMB “0111” XM2DM “0100” XM2DMB “1100” XM2PM	0000
16	Start transfer	“1” DMA started in that cycle “0” otherwise	0
17	End transfer	“1” no DMA running “0” DMA currently running	0
18	External bus request	“1” external bus request detected “0” otherwise	0
19	External bus grant	“1” external bus granted “0” otherwise	0
20	External interrupt	“1” external interrupt detected “0” otherwise	0
31 to 21	Not used	-	0×0

- Bit 0: the start bit is equal 1 only during the 1st run cycle. Then it returns to 0. It is used to start the decompression device.
- Bit 1: it displays the mAgic DSP status: 1 means Run Mode, 0 System Mode.
- Bit 2: when the bit is 1, at least one exception occurred. Fore more details, the user must read the exception register.
- Bit 3: this bit reports the status of an internal signal used to freeze the mAgic DSP when necessary (XM2PM, XM2DMB, WAITINT, etc.).
- Bit 4: this bit is set to 1 when an HALT instruction is decoded and it returns to 0 as soon as the DSP enters the System Mode state.
- Bit 5: this bit reports the status of an internal signal used to activate/deactivate all the mAgic internal devices when a transition Run/Sys or Sys/Run occurs.
- Bit 10: this bit reports the status of an internal signal used to read/write the mAgic Condition Flag Stacks. When the bit is 1 the right condition stack is selected, when the bit is 0 the left condition stack is selected.
- Bit 11: this bit reports the status of an internal signal used to access the L/R mAgic Condition Flag Stacks. When the bit is 1 the stack selected by the LRstacksel signal is written, when the bit is 0 the stack selected by the LRstacksel signal is read.
- Bits 15 to 12: these bits register the type of the running DMA, or of the last one. They can be:

  “1010” for a DM2XM

  “0101” for a DM2XMB

"0111" for a XM2DM

"0100" for a XM2DMB

"1100" for a XM2PM

- Bit 16: this bit is used to activate the DMA machine: it is set to 1 when the DMA is started, the following cycle after returns to 0.
- Bit 17: it displays the DMA status. When 1 a DMA is running. When 0 no DMA is running.
- Bit 18: when an external device requires the access to the external bus, this bit is set to 1. Otherwise it is 0.
- Bit 19: when the bit is 1 the external bus is granted, while when it is 0, it is not granted. Note that when an external device requires the bus to access to mAgic External Memory, the request is accepted only if this bit is 1. For instance, if a mAgic DMA is running, the bit is 0 and the bus is not granted.
- Bit 20: when an External Interrupt is detected, the bit is set. Otherwise it is equal 0.

The Exception Register (GSR_EXC)

The Exception Register is a 32-bit read only register containing the status of all the exceptions which occurred in the DSP. The register can be read in System Mode by the ARM. When the register is read, it is cleared. Only 8 bits are effectively used:

Table 37. Exception Register

Bit	Description	Notes	Reset State
0	Badinop	"1" exception detected "0" no exception detected	0
1	Badoutop	"1" exception detected "0" no exception detected	0
2	Divbyzero	"1" exception detected "0" no exception detected	0
3	Parameters FIFO full	"1" exception detected "0" no exception detected	0
4	DMA exception	"1" exception detected "0" no exception detected	0
5	Local Controller unknown code	"1" exception detected "0" no exception detected	0
9 to 6	Not used	-	0000
10	Data memory left write address equal to left read address	"1" exception detected "0" no exception detected	0
11	Data memory right write address equal to right read address	"1" exception detected "0" no exception detected	0
31 to 12	Not used	-	0×0

- Bit 0: badinop exception is composed of the logical *or* of all the "inop" operators exception (invalid operand).
- Bit 1: badoutop exception is composed of the logical *or* of all the "overflow" operators exception.

- Bit 2: divbyzero exception is the “division by zero” exception of the Div operator.
- Bit 3: this exception is raised when the FIFO collecting the parameters of the pending DMA is full. The depth of this FIFO is 4 when the parameters for the forth DMA are written and the exception is raised.
- Bit 4: this exception is raised when an exception occurs during a DMA burst.
- Bit 5: this exception is raised when an unknown code is received from the Local Controller.
- Bit 10: the exception is set when the left Internal Data Memory bank is accessed in read and in write mode at the same address.
- Bit 11: the exception is set when the right Internal Data Memory bank is accessed in read and in write mode at the same address.

The Exception Mask Register (GSR_MASK)

The Exception Mask Register is a 32-bit read/write register containing the masks for the Exception register. The register can be read and written in System Mode by the ARM. When the exception is masked, even if it is detected, it does not generate any interrupt. Only 8 bits are effectively used:

Table 38. Exception Mask Register

Bit	Description	Notes	Reset State
0	Mask for Badinop	“1” exception masked “0” exception not masked	0
1	Mask for Badoutop	“1” exception masked “0” exception not masked	0
2	Mask for Divbyzero	“1” exception masked “0” exception not masked	0
3	Mask for Parameters FIFO full	“1” exception masked “0” exception not masked	0
4	Mask for DMA exception	“1” exception masked “0” exception not masked	0
5	Mask for Local Controller unknown code	“1” exception masked “0” exception not masked	0
9 to 6	Not used	-	0000
10	Mask for Data Memory left write address equal to left read address	“1” exception masked “0” exception not masked	0
11	Mask for Data Memory right write address equal to right read address	“1” exception masked “0” exception not masked	0
31 to 12	Not used	-	0×0

The Program Memory Address Register (PC)

The Program Memory Address register is a 32-bit read only register with only 13 significant bits (LSBs). It contains the value of the current Program Counter. The register can be read by the ARM in System Mode. Its reset value is 0×0.

The Local Controller Right Condition Stack Register (QF_CS)

The Local Controller Right Condition Stack register is a 32-bit read/write register containing the value of the current Condition Right Stack. The register can be read and written by the ARM in System Mode. Its reset value is 0×0.

The Local Controller Left Condition Stack Register (IF_CS)

The Local Controller Left Condition Stack register is a 32-bit read/write register containing the value of the current Condition Left Stack. The register can be read and written by the ARM in System Mode. Its reset value is 0×0.

The Program Memory Address Stack Register (S&SP)

The Program Memory Address Stack register is a 32-bit read only register containing the value of the top of the PC Stack. The register can be read by the ARM in System Mode. Only the 13 LSBs are significant. Its reset value is 0×0.

The DMA Type Register (DMA_TYPE)

The DMA Type register is a 32-bit register. The first 8 bits (7 to 0) are read/write bits and contain the code for the DMA type. The three kinds of DMAs are identified by the following codes:

0 × C is used for XM2PM

0 × 4 or 0 × 7 are used for XM2DM

0 × 5 or 0 × A are used for DM2XM

Bits from 27 down to 8 are read only bits and are used to read, for debugging purpose, the value of the address generated by the MAGU (in this case from its left component, the LAGU) and it is used to access the Internal Buffer memory.

Bits 31 to 28 are unused.

In this way the 8 LSBs can be accessed by the ARM in read and write mode, while the other 20 bits are read only bits.

The DMA Type register reset value is 0 × 0.

The DMA Length Register (DMA_LEN)

The DMA Length register is a 32-bit read/write register with only 16 significant bits (LSBs). It contains the DMA length of the current burst transfer. The register can be read and written by the ARM in System Mode. In Run Mode mAgic writes the DMA length parameter of the current DMA burst in this register. Its reset value is 0 × 0.

The DMA External Memory Modifier Register (DMA_MOD)

The DMA External Memory Modifier register is a 32-bit read/write register with only 16 significant bits (LSBs). It contains the modifier used in the current DMA. The register can be read and written by the ARM in System Mode. In Run Mode mAgic writes the DMA modifier parameter of the current DMA burst in this register. Its reset value is 0 × 0.

The DMA Buffer/Program Memory Start Address Register (DMA_CADD)

The DMA Buffer/Program Memory Start Address register is a 32-bit read/write register with only 13 significant bits (LSBs). It contains the Buffer Data Memory address or the Program Memory address used in the current DMA (if the DMA is a XM2PM the address involved is a Program Memory address, otherwise, if it is either a XM2DM or a DM2XM the address involved is a Buffer Data Memory address). The register can be read and written by the ARM in System Mode. In Run Mode mAgic writes the address of the current DMA burst in this register. Its reset value is 0 × 0.

The DMA External Memory Start Address Register (DMA_XMADD)

The DMA External Memory Start Address register is a 32-bit read/write register with only 24 significant bits (LSBs). It contains the External Memory address used in the current DMA. The register can be read and written by the ARM in System Mode. In Run Mode mAgic writes the External Memory address of the current DMA burst in this register. Its reset value is 0 × 0.

The Start DMA Register (DMA_START)

The Start DMA register is a 32-bit register. The first 8 bits (7 to 0) are read/write bits and they contain the status of a DMA transfer. In more detail, if there is a running DMA the value of the bits is 0 × 0, if there is no running DMA the value is 0 × 1 (exactly the value of the bit 17 of the Status Register).

Bits from 27 down to 8 are read only bits and are used to read, for debugging purpose, the value of the address generated by the MAGU (in this case from its right component, the RAGU) and it is used to access to the External Memory.

Bits 31 to 28 are unused.

In this way the 8 LSBs can be accessed by the ARM in read and write mode, while the other 20 bits are read only bits.

The DMA Type register reset value is 0 × 1, because no DMA is supposed to be running at reset.

The Step Mode Register (STEP_MODE)

The Step Mode register is a 32-bit read/write register containing all the information necessary to perform the step mode of the mAgic DSP for debugging. In order to obtain a step execution the register must contain the value 0 × 3. The register can be accessed in System mode by the ARM. Only 2 bits are effectively used:

Table 39. Step Mode Register

Bit	Description	Notes	Reset State
0	Step mode activation	“1” step mode enabled “0” step mode disabled	0
1	Continue step mode	“1” next cycle step mode continue “0” next cycle step mode stop	0
31 to 2	Not used	-	0×0

– Bit 0: when the bit is 1 the DSP runs only one cycle then it returns to System Mode, when it is 0, it runs in the normal way.

– Bit 1: when the bit is 1 (and bit(0) is also 1) the DSP runs in Step Mode. This means that when the DSP is restarted from System Mode after the first cycle, it will continue the program execution previously stopped as if it had never been stopped. If the bit is 0, when the DSP is started again, it will run all the program left in Step Mode.

Memory Mapping of the mAgic DSP Internal Resources

Two different types of resources can be noticed in the following table:

- Memory resources: Program and Data Memories
- Resources for debugging and system initialization

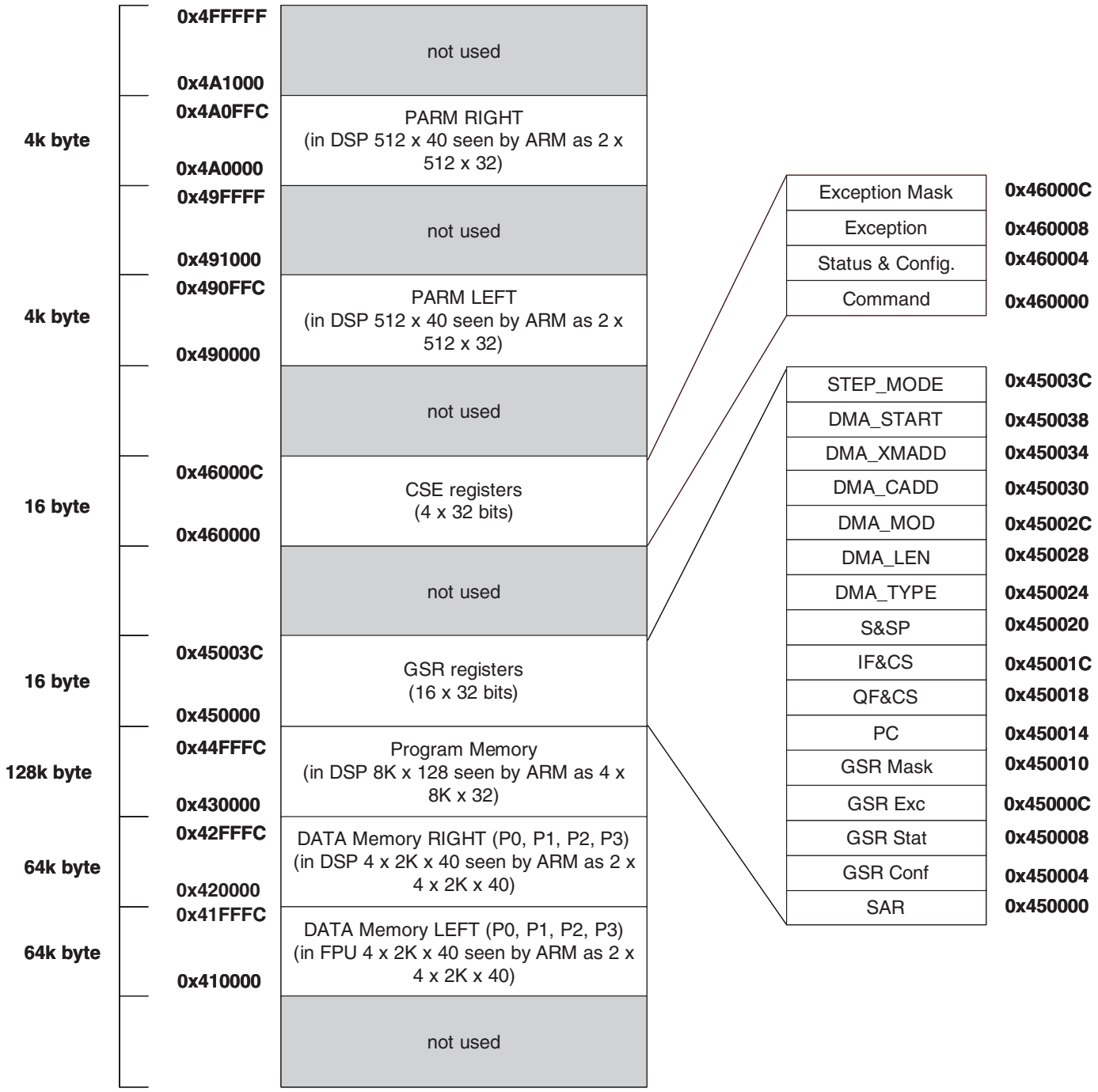
Accesses to the first resource type occur in a very rapid way.

For the second type of resources, there are no performance problems.

This implies that memory resources are mapped to allow burst transactions.

Both in AMBA and in PCI systems, the performance of a bus transaction, which concerns contiguous addresses, is better than non contiguous ones.

Figure 20. mAgic FPU Resource Map in the ARM Address Space



Instruction Set

The Very Long Instruction Word (VLIW)

The mAgic DSP is an architecture based on a Very Long Instruction Word (VLIW) structure. The use of this structure and of a Large Multiport Register File (LMRF) allows to obtain a large bandwidth and a high computational power.

The VLIW instructions allow the control of all the DSP devices cycle by cycle. The length of these instructions is 128 bits and these are collected in an 8k Program Memory. A compression mechanism allows expanding the capacity of this memory of an average 3 factor, depending on the application.

The instructions are organized in fields, with an almost direct correspondence between fields and a groups of devices driven by them.

Each decompressed VLIW instruction has 120 significant bits and it is organized as in the follow scheme:

DMO	FLOC	RFA								MAGU		DMO		
MUX	FLOC	7	6	5	4	3	2	1	0	LAGU	RAGU	ADD	MUL	RFIO
2	11	8	8	8	8	8	8	8	8	11	11	7	7	7

FLOC: it is an 11-bit field containing the codes for the Flow Control and the VLIW Decoder Unit;

- RFA: it is a 64-bit field divided in eight subfields of 8 bits each. They drive the input - output ports of the Register File, from port 0 to port 7;
- MAGU: it is a 22-bit field divided in two subfields of 11 bits each. They drive the Multiple Address Generation Unit, 11 bits for the LAGU and 11 bits for the RAGU;
- DMO: it is a 23-bit field driving the Data Memory Operators block and the multiplex in output from the operators block. The Data Memory Operators Block section is divided in three subfields: 7 bits for the adder opcode, 7 bits for the multiplier and the shifter opcode and 7 bits for the bank selection of port 0, 1, 2, 3, 4, 5 and 6. The MuxSel subfield drives the multiplex in output from the operators block for direct connection to the Internal Memories.

The FLOC Field

The FLOC VLIW field contains the nanocodes for the Flow Control and the VLIW Decoder Unit. It consists of 11 bits. The 7 LSBs (107-113 bits) are used for the program flow control: they are used by all the instructions changing the program flow, by the immediate loading instructions and by some move instructions.

The 4 MSBs (114-117 bits) are used for the program conditioning: the bit number 114 is connected to the addition instructions, the bit number 115 is connected to the multiply instructions, the bit number 116 is connected to the writing memory instructions and the bit number 117 is connected to the immediate loading instructions. When these bits are equal to "1" the corresponding set of instructions is conditioned with the condition contained on the top of the stack (if the condition is equal to "1" the instructions are executed, otherwise not), while if the bits are equal to "0" the instructions are not conditioned and they are always executed.

FLOC VLIW FIELD										
Conditioning				Flow Control						
117	116	115	114	113	112	111	110	109	108	107
imm	mem	mul	add	128 Flow Control Codes						

RFA VLIW Field

The RFA VLIW field drives the 4 input-4 output 512 registers MRF. This field is divided into eight subfields of 8 bits each. Each subfield drives one port of the Register File.

RFA VLIW FIELD																							
RFA7			RFA6			RFA5			RFA4			RFA3			RFA2			RFA1			RFA0		
106	...	99	98	...	91	90	...	83	82	...	75	74	...	67	66	...	59	58	...	51	50	...	43
8 bits			8 bits			8 bits			8 bits			8 bits			8 bits			8 bits			8 bits		

The MAGU VLIV Field

The Multiple Address Generation Unit (MAGU) is the device performing the address generation. The MAGU Block receives the MAGU VLIW field. This field is a 22-bit word having the following structure:

MAGU VLIW FIELD																						
LAGU											RAGU											
42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	
Address Mode		Register Group				Access Mode					Address Mode		Register Group				Access Mode					

The first 11 bits drive the LAGU, the other 11 bits the RAGU. For each LAGU/RAGU field, 2 bits specify the Addressing Mode (See Table 22 on page 46), 4 bits specify the number of the ARF registers, 5 bits specify the Accessing Mode (See Table 23 on page 46).

The DMO Field

The arithmetical and logical performances of the mAgic DSP are ensured by the Operators Block. This is driven by the DMO VLIW field (23 bits) that contains the Add and Mul/Shift op codes, the Register File Input Output field, and the MuxSel field in order to select the output mux from the Operators Block to the Memory Space. The RFIO is a 7-bit field, one bit for each port of the Register File, and it manages the parity of the input and of the output ports: bit set to 0 specifies that the port involved belongs to the left bank while bit set to 1 specifies that it belongs to the right bank. The structure of the DMO is as follows:

DMO VLIW FIELD																						
MuxSel		Add							Mul/Sh							Input/Output Selector						
119	118	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Add Codes							Mul/Sh Codes							pt0	pt1	pt4	pt2	pt3	pt5	pt6

The Instructions

The mAgic DSP instruction set collects all the instructions necessary for the DSP functionality. These instructions are divided into five main groups:

- Computation Instructions
  1. Add-Sub instructions
  2. Multiply instructions
  3. Logical and shift instructions
  4. Mathematical seed generation instructions
  5. Miscellaneous arithmetical instructions
  6. Conversion instructions

- Addressing instructions
  1. Move instructions
  2. ARF management instructions

- Program sequencing instructions
  1. Compare instructions
  2. Condition Code Stack instructions
  3. Pragmas instructions
  4. Branch instructions
  5. Loop instructions
  6. Repeat instructions
  7. Miscellaneous control instructions

- Immediate loading instructions
  1. Register File immediate loading
  2. Address Register File immediate loading

- DMA and interrupt instructions
  1. DMA instructions
  2. Interrupt management instructions

Only a general description of these main groups and the list of all the instructions is reported in this document. For a more detailed description of each instruction refer to [4] in Section “Related Documents” on page 268.

Group I: Computation Instructions

This group includes all the instructions using the operators block to perform arithmetical computations. They are divided according to the operator involved, and, for each operator, the data type (complex, vectorial or single) used by the instructions is detailed.

Add-Sub Instructions

The adder operators allow computing different types of addition, on integer and floating point data types: complex, vectorial and single additions. They also allow to explicitly execute an addition and a subtraction concurrently on the same data.

The adder floating-point instructions operate on 40-bit floating-point operands and output 40-bit floating-point results. The adder integer instructions operate on 32-bit integer operands and output 32-bit integer results.

The adder takes two input operands, which can be selected from the two Register File banks according to the input mixing mechanism. Usually, a result is returned to the Register File. Only in addsub operations two results are returned, while in compare operations no result is returned at all (only exception flags are updated).

Sometimes it is possible to specify a sign for the source operands as in the following example:

```
ADD     DestReg -SourceReg1 SourceReg2
```

that is equivalent to DestReg = -SourceReg1+ SourceReg2.

In order to improve the processor efficiency, the operators pipeline must be filled as much as possible.

Complex Instructions

Complex instructions include the instructions with floating point and integer complex input numbers.

The C[I]ADD takes two input values from the register file and stores the complex addition result in a destination register of the same register file. The C[I]JADD and the C[I]JJADD perform complex conjugated additions: in the C[I]JADD only the SourceReg2 is conjugated, while in the C[I]JJADD both SourceReg1 and SourceReg2 are conjugated. The C[I]RADD performs an addition between a complex operand (SourceReg1) and a real operand (SourceReg2). In every complex instruction the optional minus signs allow the sign change to the input operands.

Note that the prefix for these instructions is ‘C’. When the operands are integer numbers the operands type identifier ‘I’ follows it. The "J" and "JJ" letters identify conjugated or double conjugated operands, while the "R" means an addition with a real (not complex) number.

Add-Sub Complex Instructions		Description
CADD	DestC [-] SourceA [-] SourceB	Complex FP Addition
CJADD	DestC [-] SourceA [-] SourceB	Complex FP Conjugated Addition
CJJADD	DestC [-] SourceA [-] SourceB	Complex FP Double Conjugated Addition
CRADD	DestC [-] SourceA [-] SourceB	Complex FP with Real Addition
CIADD	DestC [-] SourceA [-] SourceB	Complex Int Addition
CIJADD	DestC [-] SourceA [-] SourceB	Complex Int Conjugated Addition
CIJJADD	DestC [-] SourceA [-] SourceB	Complex Int Double Conjugated Addition
CIRADD	DestC [-] SourceA [-] SourceB	Complex Int with Real Addition

Vectorial Instructions

The vectorial instructions manipulate integer or floating point numbers. Basically they perform simultaneously the same instruction on both the left and the right paths, taking 4 input data, two from the left bank and two from the right bank through the mixing input mechanism. The optional minus signs allow the sign change to the input operands.

In this group the VFABSADD and VIABSADD instructions allow the storage of the absolute value of the addition result on both the left and the right paths. If the minus sign before the second operand is present and the plus sign is before the first operand, the absolute value of the subtraction result is stored.

The VFADDSUB instruction allows the storage of both addition and subtraction results on two different pairs of destination registers.

Note that the prefix for these instructions is ‘V’, followed by the operands type identifier, ‘I’ or ‘F’, respectively for integers or floating point numbers.

Add-Sub Vectorial Instructions	Description
VFADDSUB DestC DestD SourceA SourceB	Vectorial FP Addition and Subtraction
VFABSADD DestC [-] SourceA [-] SourceB	Vectorial FP Absolute Addition
VIADDSUB DestC DestD SourceA SourceB	Vectorial Int Point Addition and Subtraction
VIABSADD DestC [-] SourceA [-] SourceB	Vectorial Int Absolute Addition

Single Instructions

The single instructions operate on one single path by taking input operands from the two banks through the input mixing mechanism.

The FADD and IADD allow the storage of the addition result, respectively float and integer, while if the minus sign before the second operand is present and the plus sign is before the first operand, the subtraction result is stored. The FADDSUB and the IADDSUB, instead, store both the addition and the subtraction results.

The single addition instruction can also be executed by performing the absolute value on the results before they are stored in the Register File (FABSADD and IABSADD).

Note that these instructions have no prefix and use ‘I’ or ‘F’, respectively for integers or floating point numbers.

Add-Sub Single Instructions	Description
FADD DestC [-] SourceA [-] SourceB	Floating Point Addition
FADDSUB DestC DestD SourceA SourceB	Floating Point Addition and Subtraction
FABSADD DestC [-] SourceA [-] SourceB	Floating Point Absolute Addition
IADD DestC [-] SourceA [-] SourceB	Integer Point Addition
IADDSUB DestC DestD SourceA SourceB	Integer Point Addition and Subtraction
IABSADD DestC [-] SourceA [-] SourceB	Integer Absolute Addition

Multiply Instructions

The multiplication instructions execute complex, vectorial and single products between integer and floating point numbers. In complex products it is feasible to conjugate the second or both the operands as well as multiply a complex number with a real one.

The multiplier floating-point instructions operate on 40-bit floating-point operands and output 40-bit floating-point results. The multiplier integer instructions operate on 32-bit integer operands and output 32-bit integer results.

The multiplier takes two input operands, which can be selected from the two Register File banks according to the input mixing mechanism. A result is returned to the Register File. It is always possible to specify a sign before the first source operand.

Complex Instructions

The complex instructions work on pairs of floating point or integer complex numbers through the mixing input mechanism.

The prefix for these instructions is ‘C’. When the operands are integer numbers the operands type identifier ‘I’ follows it. The "J" and "JJ" letters identify conjugated or double conjugated operands, while the "R" means that the second operand is a real (not complex) number.

Multiply Complex Instructions		Description
CMUL	DestC [-] SourceA SourceB	Complex FP Multiplication
CJMUL	DestC [-] SourceA SourceB	Complex FP Conjugated Multiplication
CJJMUL	DestC [-] SourceA SourceB	Complex FP Double Conj. Multiplication
CRMUL	DestC [-] SourceA SourceB	Complex FP with Real Multiplication
CIMUL	DestC [-] SourceA SourceB	Complex Int Multiplication
CIJMUL	DestC [-] SourceA SourceB	Complex Int Conjugated Multiplication
CIJJMUL	DestC [-] SourceA SourceB	Complex Int Double Conj. Multiplication
CIRMUL	DestC [-] SourceA SourceB	Complex Int with Real Multiplication

Vectorial Instructions

The vectorial instructions work on pairs of floating point or integer numbers, taking 4 input data, two from the left bank and two from the right bank through the mixing input mechanism.

The prefix is "V" followed by "F" for floating point instructions and "I" for integer instructions..

Multiply Vectorial Instructions		Description
VFMUL	DestC [-] SourceA SourceB	Vectorial Floating Point Multiplication
VIMUL	DestC [-] SourceA SourceB	Vectorial Integer Multiplication

Single Instructions

The single instructions work on single floating point or integer numbers through the mixing input mechanism.

Note that these instructions have no prefix and use either ‘I’ or ‘F’, respectively for integers or floating point numbers.

Multiply Single Instructions		Description
FMUL	DestC [-] SourceA SourceB	Floating Point Multiplication
IMUL	DestC [-] SourceA SourceB	Integer Multiplication

Logical and Shift Instructions

The instructions included in this group work with integer operands performing single and vectorial instructions.

They take two input operands from the Register File and return the result into the same register. The inputs are organized as follows:

- The input operand SourceRegE is pure data to be manipulated. It is 40 bits long, but only 32 LSB are used.

The input operand SourceRegN contains the possible mask in the 32 LSB and other information for shift or BIT instruction in the 8 MSB. The bits are arranged as follows::

MS Byte (S2high)								32 LSB (S2low)									
39	38	37	36	35	34	33	32	31	30	...	...	...	...	...	...	1	0

- Bits 31-0 contain the mask for the logic operations (SHAND, SHOR, SHXOR). They are reserved in BIT instructions.
- Bits 39-32 as in the following tables:

In shift instructions

39	**38**	**37**	**36**	**35**	**34**	**33**	**32**	
0	0	0	N	N	N	N	N	Left arithmetic shift and N. of shifts (0 to +31)
0	0	1	N	N	N	N	N	Right arithmetic shift and N. of shifts (0 to +31)
0	1	N	N	N	N	N	N	Left rotate mod. 40 and N. of shifts (0 to +39)
1	0	0	N	N	N	N	N	"1" Insert Left Shift and N. of shifts (0 to 31)
1	0	1	N	N	N	N	N	"1" Insert Right Shift and N. of shifts (0 to 31)
1	1	0	N	N	N	N	N	Left rotate mod. 32 and N. of shifts (0 to 31)
1	1	1	-	-	-	L | R	0 | 1	Encode 0|1 starting from Left(0)|Right(1)

In BIT instructions

39	38	37	36	35	34	33	32	
-	-	N	N	N	N	N	N	N of manipulated bit (0 to 39)

Where the symbol "-" means "Reserved bit".

The shift operation, SH, shifts SourceRegE by the amount and by the kind of shift (arithmetic, rotate, "0" insert, etc.) specified in the 8-MSB of SourceRegN and stores the result in DestRegF.

The instructions SH<log> DestRegF SourceRegE SourceRegN perform the following operation: F = (E <<>> Nhi) log Nlo, where <<>> is a kind of shift specified by 39-37 N bits of an amount specified by 36-32 N bits and log is AND, OR, XOR.

The arithmetic shift, left or right, is a shift that maintains the sign of the operand SourceRegE, while the “1” insert left shift inserts a “1” in the least significant bit of the shifted data. On the contrary the “0” insert right shift inserts a 0 in the most significant bit.

The rotation module 40 involves all the 40 bits, while the module 32 involves only the 32 LSBs.

The encode operation returns a code specifying the first position in SourceRegE in which 0 (or 1) appears, starting from the left (or the right).

The instructions BIT<log> DestRegF SourceRegE SourceRegN perform the following operation:

F = E log Nhi, where log is:

- SET: for bit set. It sets the Nhi-th bit of E.
- CLR: for bit clear. It clears the Nhi-th bit of E.
- TGL: for bit toggle. It toggles the Nhi-th bit of E.
- TST: for bit test. It tests the Nhi-th bit of E returning the test result (0 if "False", 1 if "True") in F.

Vectorial instructions.

Logical and Shift Vectorial Instructions		Description
VSH	DestF SourceE SourceN	Vectorial Shift
VSHAND	DestF SourceE SourceN	Vectorial Shift and Logical AND
VSHOR	DestF SourceE SourceN	Vectorial Shift and Logical OR
VSHXOR	DestF SourceE SourceN	Vectorial Shift and Logical XOR
VBITSET	DestF SourceE SourceN	Vectorial Bit Set
VBITCLR	DestF SourceE SourceN	Vectorial Bit Clear
VBITTGL	DestF SourceE SourceN	Vectorial Bit Toggle
VBITTST	DestF SourceE SourceN	Vectorial Bit Test

Single instructions.

Logical and Shift Single Instructions		Description
SH	DestF SourceE SourceN	Shift
SHAND	DestF SourceE SourceN	Shift and Logical AND
SHOR	DestF SourceE SourceN	Shift and Logical OR
SHXOR	DestF SourceE SourceN	Shift and Logical XOR
BITSET	DestF SourceE SourceN	Bit Set
BITCLR	DestF SourceE SourceN	Bit Clear
BITTGL	DestF SourceE SourceN	Bit Toggle
BITTST	DestF SourceE SourceN	Bit Test

Mathematical Seed Generation Instructions

This set includes vectorial and single instructions computing the seed for the inversion and for the inverse square root in floating point arithmetic.

Mathematical Seed Generation Instructions		Description
VINVSEED	DestReg SourceReg	Vectorial Inversion Seed Generation
VSQRTSEED	DestReg SourceReg	Vectorial Square Root Seed Generation
INVSEED	DestReg SourceReg	Inversion Seed Generation
SQRTSEED	DestReg SourceReg	Square Root Seed Generation

Miscellaneous Arithmetic Instructions

The instructions in this group allow performing different operations with the adder unit, such as min/max selection and some conditional instructions.

The FMIN and the IMIN allow the storage of a min result and FMAX and IMAX of a max result. The FMINMAX and the IMINMAX allow the simultaneous storage of a min result and of a max result.

The SEL, MASK and SWAP instructions are described in the following table. Note that these instructions are conditional operations.

Instruction	C-output	D-output
FSEL	A|-A	-B|B
ISEL	A|-A	-B|B
FCMASKA	A|0+B	A|0-B
ICMASKA	A|0+B	A|0-B
FCMASKB	A+0|B	A-0|B
ICMASKB	A+0|B	A-0|B
SWAP	A|B	B|A

The FSEL and ISEL allow a conditional selection between A and B inputs. The selection is driven by the condition generated by the Condition Generation Unit. The selection operations allow to select a value between A and –A or –B and B returning A on the C port and –B on the D port if condition code is true, otherwise –A on the C port and B on the D port.

The various MASK instructions (with floating point or integer numbers) perform additions and subtractions with an alternative selection between one input and zero. This selection is driven by the generated condition. Consider the FCMSKA operation, it allows to execute an addition or a subtraction (either changing or not the result signs) masking the operand A: if the condition code is "true", A+B will be returned on the C port and A-B on the D port. If the condition code is "false" the results are respectively B and –B, i.e. A is masked with 0.

The SWAP instruction allows the conditional swap between A and B inputs. The swap is driven by the condition generated by the Condition Generation Unit. If the condition is true the instruction returns the input B on the C port and the input A on the D port, performing a swap between the two input registers. If the condition code is false the results are respectively A and B (no swap is performed).

Vectorial Instructions

The vectorial instructions work on couples of floating point or integer numbers, through the mixing input mechanism.

The prefix is "V" followed by "F" for floating point instructions and "I" for integer instructions.

Miscellaneous Arithmetic Vectorial Instructions		Description
VFMIN	DestC SourceA SourceB	Vectorial Floating Point Minimum
VFMAX	DestC SourceA SourceB	Vectorial Floating Point Maximum
VFMINMAX	DestC DestD SourceA SourceB	Vectorial FP Minimum and Maximum
VIMIN	DestC SourceA SourceB	Vectorial Integer Minimum
VIMAX	DestC SourceA SourceB	Vectorial Integer Maximum
VIMINMAX	DestC DestD SourceA SourceB	Vectorial Integer Minimum and Maximum
VSWAP	RegisterA RegisterB	Vectorial Conditional Swap

Single Instructions

The single instructions work on single floating point or integer numbers, through the mixing input mechanism.

Note that these instructions have no prefix and use either ‘I’ or ‘F’, respectively for integers or floating point numbers.

Miscellaneous Arithmetic Single Instructions		Description
FMIN	DestC SourceA SourceB	Floating Point Minimum
FMAX	DestC SourceA SourceB	Floating Point Maximum
FMINMAX	DestC DestD SourceA SourceB	Floating Point Minimum and Maximum
IMIN	DestC SourceA SourceB	Integer Minimum
IMAX	DestC SourceA SourceB	Integer Maximum
IMINMAX	DestC DestD SourceA SourceB	Integer Minimum and Maximum
FSEL	DestC DestD SourceA SourceB	Floating Point Selection
ISEL	DestC DestD SourceA SourceB	Integer Selection
SWAP	RegisterA RegisterB	Conditional Swap
FCMASKA	DestC [-] SourceA [-] SourceB	FP Conditional Addition with A Masking
FCMASKB	DestC [-] SourceA [-] SourceB	FP Conditional Addition with B Masking
ICMASKA	DestC [-] SourceA [-] SourceB	Integer Conditional Addition with A Masking
ICMASKB	DestC [-] SourceA [-] SourceB	Integer Conditional Addition with B Masking

Conversion Instructions

These operations allow to automatically convert a number from one representation to another: from integer to floating point (ITOF) and from floating point to integer (FTOI). Vectorial and single conversions are allowed.

Conversion Instructions		Description
VFTOI	DestReg SourceReg	Vectorial Floating Point to Integer Conversion
VITOF	DestReg SourceReg	Vectorial Integer to Floating Point Conversion
FTOI	DestReg SourceReg	Floating Point to Integer Conversion
ITOF	DestReg SourceReg	Integer to Floating Point Conversion

Group 2: Addressing Instructions

Instructions performing addressing and data movements between the Register File and the Internal Memory, but also all the instructions useful for the Address Register File management, are in this group.

Move Instructions

The Move instructions are used to transfer data from Register File to Register File, from the local memory to the Register File and vice versa. The memory access can be executed in several ways allowing to read/write up to two data in the same instruction.

The addressing and the accessing modes must be specified in order to control the address generation. The accessing mode allows:

- to select the reading or the writing mode
- to read or write using the bit reversal mode
- to read and write simultaneous

The addressing mode defines the way in which the address will be computed. The addressing modes give four increment modes:

- no increment
- increment mode without update
- post-increment mode
- pre-increment mode

According to the values of the S and L registers, the address generation is linear (S=L=0), with offset (S≠0 and L=0), circular or modular (L≠0).

Move Instructions	Description
VMS2RF[_<M|U|MU>] DestReg Arf [BTR]	Vectorial move from Memory to RF
VRF2MS[_<M|U|MU>] Arf SourceReg [BTW]	Vectorial move from RF to Memory
VRF2RF DestReg SourceReg	Vectorial move from RF to RF
MEM2RF[_<M|U|MU>] DestReg Arf [BTR]	Move from Memory to RF
RF2L[_<M|U|MU>] Arf SourceReg [BTW]	Move from RF to Left Memory
RF2R[_<M|U|MU>] Arf SourceReg [BTW]	Move from RF to Right Memory
RF2RF DestReg SourceReg	Move from RF to RF
VRW2MS[_<M|U|MU>] Reg1 Arf2 Reg2 Arf1	Vectorial RF to Memory and viceversa
RW2L[_<M|U|MU>] Reg1 Arf2 Reg2 Arf1	Move from RF to Left Memory and viceversa
RW2R[_<M|U|MU>] Reg1 Arf2 Reg2 Arf1	Move from RF to Right Memory and viceversa

ARF Management Instructions

This group of instructions includes all the instructions useful for the ARF management. Whenever the user wants to do some manipulations on the ARF registers, the selected ARF register must first be moved in the Register File, and then all the arithmetical instructions can be executed on it. At the end, the register (or its subfield) can be moved back into the Address Register File.The instructions for these kinds of movements are summarized in the following table.

ARF Management Instructions	Description
RF2A_[S][L][A][M] .....Arf SourceReg	Move from RF to ARF Selectively
RF2A_P .....Arf SourceReg	Move from RF to the P field
DRF2A_P .....Arf Arf2 SourceReg	Double Move from RF to the P field
A2RF .....DestReg Arf	Move SLAM from ARF to RF
A2RF_P .....DestReg Arf	Move P from ARF to RF
DA2RF_P DestReg1 DestReg2 Arf1 Arf2	Double Move P from ARF to RF
DA2RF DestReg1 DestReg2 Arf1 Arf2	Double Move SLAM from ARF to RF
RF2A_SLAMP Arf SourceReg P=*val*	Move from RF to ARF and load P with *val*
CLEAR_L .....Arf	Clear L field in an ARF register
CLEAR_SL .....Arf	Clear S and L fields in an ARF register

Group 3: Program Sequencing Instructions

The assembly language supports all the common instructions used in order to control the program flow. These instructions are divided into the following basic groups: Compare instructions, Condition Code Stack instructions, Pragma instructions, Branch instructions, Loop instructions and Control instructions. All these groups are detailed in the next sections.

Compare Instructions

The compare instructions perform logical comparison between two values. The result is pushed onto the top of the Condition Code Stack.

With complex numbers the comparison is limited to equality and not equality conditions. The inputs for complex comparison are two complex numbers, with their real part on the RF left bank and the imaginary part on the RF right bank. The condition is computed as the logical AND of the two conditions computed on the two (left and right) paths and then it is pushed onto both Condition Code Stacks.

For vectorial and single instructions the possible logical comparisons are: equate (EQ), not equate (NE), less than (LT), greater than (GT), less equal (LE) and greater equal (GE). The inputs for vectorial instructions are two vectorial numbers (two pairs from both the left and the right banks). Two conditions are computed, one for each path, and they are pushed independently onto the two Condition Code Stacks. The single instructions take as input two single numbers from whichever bank of the Register File. The condition is computed on a single path and then pushed onto both Condition Code Stacks.

Compare Instructions	Description
CEQ SourceReg1 SourceReg2	Complex Equal
CNE SourceReg1 SourceReg2	Complex Not Equal
VFEQ SourceReg1 SourceReg2	Vectorial Floating Point Equal
VFLT SourceReg1 SourceReg2	Vectorial Floating Point Less Than
VFGT SourceReg1 SourceReg2	Vectorial Floating Point Greater Than
VFLE SourceReg1 SourceReg2	Vectorial Floating Point Less or Equal

Compare Instructions		Description
VFGE	SourceReg1 SourceReg2	Vectorial Floating Point Greater or Equal
VFNE	SourceReg1 SourceReg2	Vectorial Floating Point Not Equal
VIEQ	SourceReg1 SourceReg2	Vectorial Integer Equal
VILT	SourceReg1 SourceReg2	Vectorial Integer Less Than
VIGT	SourceReg1 SourceReg2	Vectorial Integer Greater Than
VILE	SourceReg1 SourceReg2	Vectorial Integer Less or Equal
VIGE	SourceReg1 SourceReg2	Vectorial Integer Greater or Equal
VINE	SourceReg1 SourceReg2	Vectorial Integer Not Equal
FEQ	SourceReg1 SourceReg2	Floating Point Equal
FLT	SourceReg1 SourceReg2	Floating Point Less Than
FGT	SourceReg1 SourceReg2	Floating Point Greater Than
FLE	SourceReg1 SourceReg2	Floating Point Less or Equal
FGE	SourceReg1 SourceReg2	Floating Point Greater or Equal
FNE	SourceReg1 SourceReg2	Floating Point Not Equal
IEQ	SourceReg1 SourceReg2	Integer Equal
ILT	SourceReg1 SourceReg2	Integer Less Than
IGT	SourceReg1 SourceReg2	Integer Greater Than
ILE	SourceReg1 SourceReg2	Integer Less or Equal
IGE	SourceReg1 SourceReg2	Integer Greater or Equal
INE	SourceReg1 SourceReg2	Integer Not Equal

Condition Code Stack Instructions

These instructions operate on the Condition Code Stack. Logical operations between pairs of values on the top of the stack are possible through them.

FSTRUE and FSFALSE push onto the top of the stack respectively a true boolean value (“1”) and a false boolean value (“0”).

FSPOP pops a condition from the top of the stack. The FSNOT substitutes the first element with its logical negation.

FSAND, FSOR and FSXOR pop the first two elements of the stack, perform the logical operations AND, OR or XOR on them and push the result back. The FSREPUSH duplicates the first element of the stack and the FSSWAP swaps the first two elements.

FSAND2 performs the logical operations AND on the two elements on the top of the stack and pushes back the result by popping only the element on the top.

FSI2IQ pushes the top of the LStack on the RStack, so that the condition present on the top of the LStack is duplicated on the top of the RStack. In the same way, the FSQ2IQ pushes the top of the RStack on the LStack.

Condition Code Stack Instructions	Description
FSTRUE	Flag Stacks Push True
FSFALSE	Flag Stacks Push False
FSPOP	Condition Code Stack Pop

Condition Code Stack Instructions	Description
FSNOT	Condition Code Stack NOT
FSAND	Condition Code Stack AND
FSOR	Condition Code Stack OR
FSXOR	Condition Code Stack XOR
FSREPUSH	Condition Code Stack Re-push
FSSWAP	Condition Code Stack Swap
FSAND2	Condition Code Stack AND with only 1 pop
FSI2IQ	LStack Top Condition Pushed on RStack
FSQ2IQ	RStack Top Condition Pushed on LStack

Pragmas Instructions

The Pragmas instructions are directives to the assembler-optimizer. They act by identifying a block of instructions with a particular behavior. For instance, the PRAGMA_WAIT_ARF_ON/OFF fixes a relationship between all the instructions using the identified ARF and all the DMA instructions inside the block; through the PRAGMA_FREE_ARF_ON/OFF, instead, the optimizer obtains better code optimization results by acting on the specified ARF registers.

The PRAGMA CONDxxx_ON and CONDxxx_OFF are used to identify blocks of conditioned operations: the instructions after the CONDxxx_ON and before the CONDxxx_OFF are executed only if the condition COND, generated by the Condition Generation Unit, is true. It is possible to selectively condition different groups of instructions (arithmetical instructions, memory writing instructions and immediate loading instructions) or all the instructions together.

Pragmas Instructions	Description
PRAGMA_WAIT_ARF_ON ..... CondArf	The identified CondArf in this block of instructions is related to DMA instructions
PRAGMA_WAIT_ARF_OFF..... CondArf	
PRAGMA_FREE_ARF_ON..... CondArf	The optimizer can freely move the identified CondArf inside this block of instructions
PRAGMA_FREE_ARF_OFF .... CondArf	
PRAGMA CONDARIT_ON	All the arithmetical instructions in this block are conditioned by the top of the stack
PRAGMA CONDARIT_OFF	
PRAGMA CONDMEM_ON	All the memory access instructions in this block are conditioned by the top of the stack
PRAGMA CONDMEM_OFF	
PRAGMA CONDIMM_ON	All the immediate loading instructions in this block are conditioned by the top of the stack
PRAGMA CONDIMM_OFF	
PRAGMA CONDALL_ON	All the instructions in this block are conditioned by the top of the stack
PRAGMA CONDALL_OFF	

Branch Instructions

This group includes immediate and conditioned jump, call and return instructions. They are of two main types: always executed or executed only if the current condition is false (the latter case is identified by the presence of the suffix: IF_NOT).

The jump address for a branch or a call instruction can either be defined with a label or be preloaded into a register of the Register File. Two different instructions must be used:

BR or CALL if the jump address is given by a label, BR_REG or CALL_REG if it is given by a register.

Branch Instructions		Description
BR	label	Branch
BRIF_NOT	label	Branch if Condition is False
BR_REG		Branch to a Preloaded Jump Address
CALLlabel		Call a Subroutine
CALLIF_NOT	label	Call a Subroutine if Condition is False
CALL_REG		Call a Subroutine to a Preloaded Jump Address
RETURN		Return from a Subroutine
RETURNIF_NOT		Return from a Subroutine if Condition is False

Loops Instructions

The mAgic DSP Assembly instructions allow loops management by the use of the INCCOMP instruction. This instruction simultaneously performs both the loop counter increment by the specified step amount and the comparison between the new calculated index and the loop end value. The comparison sets the flags in the Flow Control Unit in order to generate the condition for the execution (or not) of the subsequent conditioned branch instruction.

In more detail, before a loop start, all the loop information (initial and final values for the loop counter, length of the step on the loop counter) is stored in an ARF register. The ARF register is loaded in the following way: the A field with the starting value of the loop counter, the L field with the loop end value and the M field with the step value. Whenever an INCCOMP instruction is encountered, the LOOPGU is activated. This unit simultaneously performs the addition A' = A + M and the check A'<L. The result of this check sets the condition flags used by the subsequent branch instruction.

Loop Instructions		Description
INCCOMP	AddressReg	Increment and Compare

Repeat Instructions

These kinds of instructions allow very close loops, and in the meantime they allow the update of the register file addresses. The repeat instruction it is ideal for loop unrolling and fast loops.

microcode	enableloop	loopcounter	start block	end block	increment
repeati	Vliw[120:118]	Vliw[41:26]	Vliw[12:0]	Vliw[25:13]	Vliw[106:43]
repeatr	Vliw[120:118]	Rfl[31:0]	Vliw[12:0]	Vliw[25:13]	Vliw[106:43]

The difference between the two instructions is that repeatr uses a 32 bits wide loopcounter from the RF, this instruction is mainly used to do loops; repeati uses an immediate 16-bit loopcounter, this instruction is mainly used to do loop unrolling.

addr7incr	addr6incr	add5incr	add4incr	add3incr	add2incr	add1incr	add0incr
vliw[104:99]	vliw[96:91]	vliw[88:83]	vliw[80:75]	vliw[72:67]	vliw[64:59]	vliw[56:51]	vliw[48:43]

At each loop cycle the RF addresses will be incremented by the corresponding incr. The enable loop parameter is used to disable for enable loopcycles, write in RF, and/or MAGU address update.

Notes: 1. It is not possible to do nested repeat instructions.
2. There must be at least two cycles between the repeat and the end block instructions

Repeat Instructions	Description
REPEATI	Repeat with Immediate Parameters
REPEATR	Repeat with Parameters from the Register File

Miscellaneous Control Instructions

The remaining Program Sequencing instructions that cannot be included in the previous groups are included in this section.

The HALT instruction stops the program execution and switches the mAgic DSP in System Mode.

The NOP instruction performs the "nop operation" by increasing the Program Counter by one.

The PC2RF, STAT2RF and STACK2RF instructions move to the specified registers of the Register File respectively the Program Counter, the Status Register and the Condition Stack Register. The destination registers are two, one in the left bank and the other in the right bank. The Program Counter and the Status Register are duplicated in both banks, while for the Condition Stack Register the LStack is moved into the left bank and the RStack is moved into the right bank.

The ONES instruction counts the number of “1” that are in a Register of the Register File.

Miscellaneous Control Instructions Instructions		Description
HALT		Halt Execution
NOP		Nop Instruction
PC2RF	DestReg1 DestReg2	Move from PC Register to RF
STAT2RF	DestReg1 DestReg2	Move from Status Register to RF
STACK2RF	DestReg1 DestReg2	Move from Condition Stack Register to RF
ONES	DestReg SourceReg	“1” counter on RF Register

Group 4: Immediate Loading Instructions

The immediate loading instructions can be distinguished in those loading the Address Register File and the those loading the Data Register File. They are summarized in the next two sections.

Register File Immediate Loading

Only one instruction allows loading immediate values in the Register File. Two registers are simultaneously written through this instruction; the first one must be allocated in the left Register File bank, the second one in the right Register File bank. The addresses for the EvenReg and for the OddReg can be either of the same pair (address n and n+1) or completely different. The only requisite is that the first must be even and the second odd.

If the user is interested in loading only one register, a trash register must be defined for the other address (for instance the mAgic C compiler uses, as trash registers, the register 0 for the even bank and 1 for the odd bank) in order to avoid data overwrite.

Register File Immediate Loading Instructions	Description
DIMM2RF EvenReg val1OddReg val2	Double Immediate Loading into RF

Address Register File Immediate Loading

Three different instructions are available for loading the Address Register File. The IMM2A loads the S, L, A and M fields of one ARF register selectively. The P field cannot be loaded through this instruction. Because of this, the instruction is useful when the P field of the selected ARF is already loaded with the right value (or for instance, after it is loaded with an appropriate ARF management instruction) and the user only wants to load some or all of the other fields.

The DIMM2ASEQ has almost the same behavior as the previous one, but it loads two subsequent ARF registers (if ARF is 7 it loads register 7 and register 8) with the specified fields. In this case the fields can also be loaded selectively; moreover, the P field of both registers can be laoded.

The third instruction, DIMM2A, loads two ARF as specified by the user (not necessarily one subsequent the other). All the fields must be loaded. This instruction is useful at initialization time.

Address Register File Immediate Loading Instructions	Description
IMM2A ARF [S=*val*] [L=*val*] [A=*val*] [M=*val*]	Immediate Loading into ARF
DIMM2ASEQ ARF [S1=*val*] [L1=*val*] [A1=*val*] [M1=*val*] [S2=*val*] [L2=*val*] [A2=*val*] [M2=*val*]	Double Immediate Loading into sequential ARF
DIMM2A ARF1 S1=*val* L1=*val* A1=*val* M1=*val* P1=*val* ARF2 S2=*val* L2=*val* A2=*val* M2=*val* P2=*val*	Double Immediate Loading into two ARF with P Loading

Group 5: DMA and Interrupt Instructions

DMA Instructions

As explained in Section “Direct Memory Access” on page 53, DMAs are allowed between the mAgic DSP Internal Data Buffer Memory and the External Memory. DMAs are possible in both directions (from the Internal Data Buffer to the External Memory and from the External to the Internal Data Buffer Memory) and they can be of two types: a blocking DMA or a non-blocking one.

When an instruction starting a DMA is decoded, the DMA machine can be in two states: either idle or executing a previously started DMA. If it is idle, the DMA (no matter which kind it is) is immediately launched. If it is already executing a DMA, the behavior is different according to the type of instruction. A non-blocking DMA (XM2DM or DM2XM) queues the request of the DMA by writing the whole set of parameters (DMA code, starting address in the Internal Memory, starting address in the External Memory, burst length and burst step) in a FIFO able to collect up to 4 requests. When the DMA controller becomes idle, the first request is popped from the FIFO and executed. On the contrary, if the instruction is a blocking one (XM2DMB or DM2XMB), the clock of the

mAgic DSP core is stalled and the whole machine waits for the right moment to start the DMA. Only then the mAgic clock is re-activated and the core goes on with the program.

The parameters used by the DMA must be previously loaded in reserved registers through an IMM2BS or a LDRB instruction. The IMM2BS loads these registers with immediate values. The LDRB, instead, also uses RF and ARF registers.

The XM2PM instruction is also included among the DMA instructions. It is used for moving a section of the mAgic program stored in the External Memory to the mAgic Program Memory at the address specified by label. During this transfer, the mAgic core clock is stalled. As soon as the program transfer is over, the core is resumed with the Program Counter pointing to label.

The last instruction is the WAITTRX. It stretches the core waiting for all the scheduled DMAs (the current ones and all those in the FIFO) to be finished. In this way, the user can synchronize the core execution with the data transfer.

All the instructions used for instructing and activating a DMA are summarized in the following table. For more details on these instructions refer to [4] in Section “Related Documents” on page 268.

DMA Instructions		Description
IMM2BS	<parameters>	Immediate Loading DMA Parameters
LDRB	<parameters>	Loading DMA Parameters
XM2PM	label	External Memory to Program Memory Transfer
XM2DM		ExtMem to Buffer Memory Transfer
XM2DMB		ExtMem to Buffer Memory Blocking Transfer
DM2XM		Buffer Memory to ExtMem Transfer
DM2XMB		Buffer Memory to ExtMem Blocking Transfer
WAITTRX		Wait for End of Transfer

Interrupt Management Instructions

The instructions included in this group allow mAgic to exchange interrupts with the ARM system and with the outside. With the six SIRQ instructions mAgic can send interrupts to the AIC. The SIRQ(0:5) are also connected to the PIO outputs.

All the SIRQ instructions generate an high pulse (3 mAgic clock cycle wide) on the related interrupt internal lines (to the AIC) and the PIO outputs; except for the SIRQ5 instruction that when executed make the related interrupt internal line and the PIO output level toggle (default level low). SIRQ5 provide the appropriate functionality for dual D740 shared XM management.

The LDAI and LDAR instructions are used to load, with immediate values (LDAI) or with values contained in the registers (LDAR), all the registers used for the interrupt management: the ISR (Interrupt Services Routine Registers), the IMR (Interrupt Mask Register), the IPR (Interrupt Priority Register) and also the JMPA (Jump Addresses Register) used in BR_REG and CALL_REG instructions.

The WAITEXT is the instruction used to synchronize mAgic with an external device: when this instruction is decoded, the core is stalled waiting for the arrival of the external interrupt corresponding to the PIO signal EXTIRQ2

The WATCHINT also looks for pending interrupts requests. If one or more interrupts have been received, they are served by starting an interrupt service routine, according

to the priority assigned in the IPR register. If no pending interrupt is found, the program flow goes on without any change. This instruction never stretches the core.

Interrupt Management Instructions		Description
SIRQ0		Interrupt on SIRQ0 line (positive edge)
SIRQ1		Interrupt on SIRQ1 line (positive edge)
SIRQ2		Interrupt on SIRQ2 line (positive edge)
SIRQ3		Interrupt on SIRQ3 line (positive edge)
SIRQ4		Interrupt on SIRQ4 line (positive edge)
SIRQ5		Interrupt on SIRQ5 line (positive level)
LDAI	ImmVal BitField	Immediate Loading of Interrupt Services Routine Addresses, Masks, Priorities and Jump Addresses
LDAR	SourceReg BitField	Loading of Interrupt Services Routine Addresses, Masks, Priorities and Jump Addresses from a Register
WAITEXT		Wait for External Interrupt
WATCHINT		Check for interrupt pending requests and eventually jump to the ISVR

External Memory Synchronization Mechanisms

The mAgic DSP can share its External Memory with another processor. A safe mechanism of bus request/bus grant allows a secure bus sharing between mAgic and the external processor.

GNTON enables the mAgic DSP to accept access requests to the External Memory coming from an external device. This access enable can be removed with the instruction GNTOFF. When the enable is “on” and an external device asks to access the External Memory, the bus is grant as soon as the current mAgic transfer (if there is one) is finished. If the enable is “off” no grant is given to any request from an external device..

External Memory Bus Grant Synchronization Instructions	Description
GNTON	Grant On: mAgic DSP is enabled to accept access requests to its External Memory coming from an external device
GNTOFF	Grant Off: mAgic DSP is disabled to accept access requests to its External Memory coming from an external device

Instruction Set Syntax

- The mAgic DSP assembly is case sensitive.
- The instruction mnemonics and their suffixes, prefixes and option fields are reserved words.
- The preprocessor directives have a slash in front of them or are enclosed in angle brackets. They are lower case.

- The integer numbers can be represented both in decimal and in hexadecimal format. The representation is in C style, for example the decimal number 450, as described in the following tables:

Table 40. Prefix for Integer Numbers Representation

Numeric Format	Integer Representation
Decimal	450
Hexadecimal	0x1C2

- Representation of floating point numbers is given by the following examples:

  4.65. 6.5 0.65 4. 4.0 4.6e10 4.65e-10

- The labels are defined by LABEL followed by an identifier:

```
...
BRIF_NOT        start
...
LABEL start
...
...
```

- Every assembly program must include the file “assembly.exp” in the first line.
- The comment tag is “!”. All the text placed between this character and a new line is discarded.

Instruction Set Summary

Table 41. Instruction Set Summary

Mnemonic	Description
Add-Sub Instructions	
CADD	Complex Addition
CJADD	Complex Conjugated Addition
CJJADD	Complex Double Conjugated Addition
CRADD	Complex with Real Addition
CIADD	Complex Integer Addition
CIJADD	Complex Integer Conjugated Addition
CIJJADD	Complex Integer Double Conjugated Addition
CIRADD	Complex Integer with Real Addition
VFABSADD	Vectorial Floating Point Absolute Addition
VIABSADD	Vectorial Integer Absolute Addition
VFADDSUB	Vectorial Floating Point Addition and Subtraction
VIADDSUB	Vectorial Integer Addition and Subtraction
FADD	Floating point Addition
FADDSUB	Floating Point Addition and Subtraction

Table 41. Instruction Set Summary (Continued)

Mnemonic	Description
IADD	Integer Addition
IADDSUB	Integer Addition and Subtraction
FABSADD	Floating Point Absolute Addition
IABSADD	Integer Absolute Addition
Miscellaneous Arithmetic Instructions	
VFMIN	Vectorial Floating Point Minimum
VIMIN	Vectorial Integer Minimum
VFMAX	Vectorial Floating Point Maximum
VIMAX	Vectorial Integer Maximum
VFMINMAX	Vectorial Floating Point Minimum and Maximum
VIMINMAX	Vectorial Integer Minimum and Maximum
VSWAP	Vectorial Conditional Swap
FMIN	Floating Point Minimum
IMIN	Integer Minimum
FMAX	Floating Point Maximum
IMAX	Integer Maximum
FMINMAX	Floating Point Minimum and Maximum
IMINMAX	Integer Minimum and Maximum
FSEL	Floating Point Selection
ISEL	Integer Selection
FCMSKA	Floating Point Addition with A Masking
ICMSKA	Integer Point Addition with A Masking
FCMSKB	Floating Point Addition with B Masking
ICMSKB	Integer Addition with B Masking
SWAP	Conditional Swap
Multiply Instructions	
CMUL	Complex Multiplication
CJMUL	Complex Conjugated Multiplication
CJJMUL	Complex Double Conjugated Multiplication
CRMUL	Complex with Real Multiplication
CIMUL	Complex Integer Multiplication
CIJMUL	Complex Integer Conjugated Multiplication
CIJJMUL	Complex Integer Double Conjugated Multiplication
CIRMUL	Complex Integer with Real Multiplication
VFMUL	Vectorial Floating Point Multiplication
VIMUL	Vectorial Integer Multiplication

Table 41. Instruction Set Summary (Continued)

Mnemonic	Description
FMUL	Floating Point Multiplication
IMUL	Integer Multiplication
Logical and Shift Instructions	
VSH	Vectorial Logical and Arithmetical Shift
VSHAND	Vectorial Shift and Logical AND
VSHOR	Vectorial Shift and Logical OR
VSHXOR	Vectorial Shift and Logical XOR
VBITSET	Vectorial Bit Set
VBITCLR	Vectorial Bit Clear
VBITTGL	Vectorial Bit Toggle
VBITTST	Vectorial Bit Test
SH	Logical and Arithmetical Shift
SHAND	Shift and Logical AND
SHOR	Shift and Logical OR
SHXOR	Shift and Logical XOR
BITSET	Bit Set
BITCLR	Bit Clear
BITTGL	Bit Toggle
BITTST	Bit Test
Mathematical Seed Generation Instructions	
VINVSEED	Vectorial Inversion Seed Generation
VINVSQRTSEED	Vectorial Inversion Square Root Seed Generation
INVSEED	Inversion Seed Generation
INVSQRTSEED	Inversion Square Root Seed Generation
Conversion Instructions	
VFTOI	Vectorial Floating Point to Integer Conversion
VITOF	Vectorial Integer to Floating Point Conversion
FTOI	Floating Point To Integer Conversion
ITOF	Integer To Floating Point Conversion
Move Instructions	
VMS2RF	Vectorial Memory Space to Register File
VRF2MS	Vectorial Register File to Memory Space
VRF2RF	Vectorial Register File to Register File
MEM2RF	Move from Memory Space To Register File
RF2L	Move from Register File To Left Memory Space
RF2R	Move from Register File To Right Memory Space

Table 41. Instruction Set Summary (Continued)

Mnemonic	Description
RF2RF	Move from Register File To Register File
VRW2MS	Vectorial Register File to Memory Space and vice versa
RW2L	Move from Register File To Left Memory Space and vice versa
RW2R	Move from Register File To Right Memory Space and vice versa
ARF Management Instructions	
RF2A_[S][L][A][M]	Move from RF to ARF Selectively
RF2A_P	Move from RF to the P field in ARF
DRF2A_P	Double Move from RF to the P field in ARF
A2RF	Move SLAM from ARF to RF
A2RF_P	Move P from ARF to RF
DA2RF_P	Double Move P from ARF to RF
DA2RF	Double Move SLAM from ARF to RF
RF2A_SLAMP	Move from RF to ARF and load P with an Immediate
CLEAR_L	Clear L field in an ARF register
CLEAR_SL	Clear S and L fields in an ARF register
Compare Instructions	
CEQ	Complex Floating Point Equal
CNE	Complex Floating Point Not Equal
CIEQ	Complex Integer Equal
CINE	Complex Integer Not Equal
VFEQ	Vectorial Floating Point Equal
VFLT	Vectorial Floating Point Less Than
VFGT	Vectorial Floating Point Greater Than
VFLE	Vectorial Floating Point Less or Equal
VFGE	Vectorial Floating Point Greater or Equal
VFNE	Vectorial Floating Point Not Equal
VIEQ	Vectorial Integer Equal
VILT	Vectorial Integer Less Than
VIGT	Vectorial Integer Greater Than
VILE	Vectorial Integer Less or Equal
VIGE	Vectorial Integer Greater or Equal
VINE	Vectorial Integer Not Equal
FEQ	Floating Point Equal
FLT	Floating Point Less Than
FGT	Floating Point Greater Than
FLE	Floating Point Less or Equal

Table 41. Instruction Set Summary (Continued)

Mnemonic	Description
FGE	Floating Point Greater or Equal
FNE	Floating Point Not Equal
IEQ	Integer Equal
ILT	Integer Less Than
IGT	Integer Greater Than
ILE	Integer Less or Equal
IGE	Integer Greater or Equal
INE	Integer Not Equal
Condition Code and Loop Instructions	
FSPOP	Flag Stacks Pop
FSNOT	Flag Stacks NOT
FSAND	Flag Stacks AND
FSOR	Flag Stacks OR
FSXOR	Flag Stacks Xor
FSREPUSH	Flag Stacks Repush
FSSWAP	Flag Stacks Swap
FSTRUE	Flag Stacks Push True
FSFALSE	Flag Stacks Push False
FSAND2	Flag Stacks AND2
FSI2IQ	Flag IStack Push
FSQ2IQ	Flag QStack Push
INCCOMP	Increment and Compare
Branch Instructions	
BR	Branch
BRIF_NOT	Branch if Condition is False
BR_REG	Branch to a Preloaded Jump Address
CALL	Call a Subroutine
CALLIF_NOT	Call a Subroutine if Condition is False
CALL_REG	Call a Subroutine to a Preloaded Jump Address
RETURN	Return from a Subroutine
RETURNIF_NOT	Return from a Subroutine if Condition is False
Repeat Instructions	
REPEATI	Repeat of a Block of Instructions with All Immediate Parameters
REPEATR	Repeat of a Block of Instructions with Loop Counter from Register File
Control and Miscellaneous Instructions	

Table 41. Instruction Set Summary (Continued)

Mnemonic	Description
HALT	Halt
NOP	No Operation
PC2RF	Move from PC Register To Register File
STAT2RF	Move from Status Register To Register File
STACK2RF	Move from Condition Stack Registers To Register File
ONES	“1” counter
Immediate Loading Instructions	
DIMM2RF	Double Immediate Loading into RF
IMM2A	Immediate Loading into ARF
DIMM2ASEQ	Double Immediate Loading into two Sequential ARF
DIMM2A	Double Immediate Loading into two ARF with Page loading
Interrupt Management Instructions	
GNTON	Grant On
GNTOFF	Grant Off
LDAI	Immediate Loading of Interrupt Services Routine Addresses, Masks, Priorities and Jump Addresses
LDAR	Loading of Interrupt Services Routine Addresses, Masks, Priorities and Jump Addresses from a Register
WATCHINT	Watch for Interrupt
Burst Transfer Instructions	
IMM2BS	Immediate Loading Burst Parameters
LDRB	Loading Burst Parameters
XM2PM	External Memory to Program Memory Transfer
XM2DM	External Memory to Buffer Data Memory Transfer
XM2DMB	External Memory to Buffer Data Memory Blocking Transfer
DM2XM	Buffer Data Memory to External Memory Transfer
DM2XMB	Buffer Data Memory to External Memory Blocking Transfer
WAITEXT	Wait for External Interrupt
WAITTRX	Wait for End of Transfer

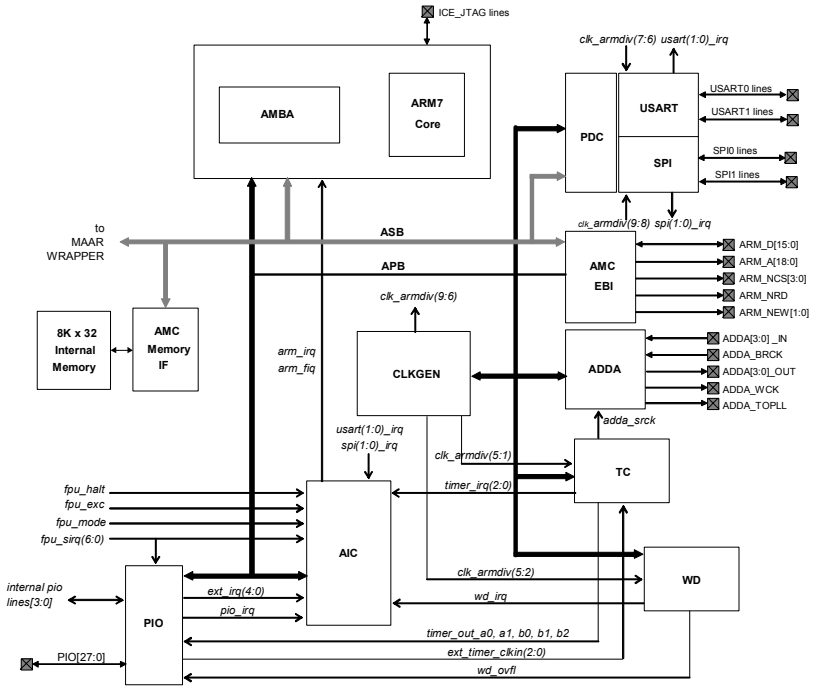
ARM System

The ARM7TDMI micro controller and its peripherals are referred to as ARM System. The ARM System architecture consists of two main buses based on the AMBA specifications (refer to [2] and [3] in Section “Related Documents” on page 268), the Advanced System Bus (ASB) and the Advanced Peripheral Bus (APB). The ASB is the high-performance system ‘backbone’ bus, able to sustain the external memory bandwidth, on which the CPU and Memory Interface devices reside, plus a ‘bridge’ to a narrower APB bus on which the lower bandwidth peripheral devices are located.

The two master devices on the ASB are the ARM7TDMI and the Peripheral Data Controller (PDC).

The slave devices in the ARM System are the bridge to the Peripheral Bus (APB BRIDGE), the Advanced Memory Controller (AMC) and the mAgic-ARM Interface (mAAr).

Figure 21. Armsystem Architecture



The AMC controls the accesses to the on-chip 8Kx32 bit memory and to the external memory. The AMC does not decode transfer protection information because ASB BPROT[1:0] signals are not used. Each bus master (ARM and PDC) can initiate read and write operations by providing address and control information. Only one bus master is allowed to actively use the bus at any one time. A bus slave responds to a read or write operations within a given address-space range. The bus slave signals back to the active master the success, failure or waiting of the data transfer.

The APB is the bus connecting the on-chip peripherals. The APB is connected to the ASB through a Slave ASB interface, the AMBA Bridge. The peripherals on the APB cannot be accessed in burst mode, they do not handle any response signal such as “wait”, “error” or “last”, they can only return data. The available peripherals are described with more details in the next sections. The ARM System block diagram is shown in Table 21.

Each peripheral in the ARM System has a 16k-byte address space allocated in the upper 3M bytes of the 4G-byte address space. Except for the interrupt controller, the peripheral base address is the lowest address of its memory space. The peripheral register set is composed of control, mode, data, status and interrupt registers.

To maximize the efficiency of bit manipulation, frequently written registers are mapped into three memory locations.

The first address is used to set the individual register bits, the second address resets the bits and the third address reads the value stored in the register. A bit can be set or reset by writing a one at the corresponding position of the appropriate address. Writing a zero has no effect. Individual bits can thus be modified without having to use costly read-modify-write and complex bit manipulation instructions.

ARM7TDMI Burst Transfers are not supported in D740. The ASB locked transfer is instead supported: BLOK signal is automatically asserted when the ARM executes a SWP instruction (register-memory data exchange).

The peripherals available in the ARM System are:

- **ADDA**. The ADDA block provides 4 channel serial interfaces (ADDA0, 1, 2, 3) toward stereo audio 24-bit ADCs and DACs. From the ADCs the ADDA receives a serial stream at programmable bit-rate frequency that is 64 x the audio sampling frequency. The ADDA transmits to the DACs serial streams with the same characteristics of that coming from the ADCs. In addition it provides an associated interrupt toward the AIC.

- **USART.** Two full-duplex, universal synchronous/asynchronous receiver/transmitters (USART0 and USART1). The main features are:
  - Programmable baud rate generator
  - Parity, framing and overrun error detection
  - Line break generation and detection
  - Automatic echo, local loopback and remote loopback channel modes
  - Multidrop mode: address detection and generation
  - Interrupt generation
  - Two dedicated Peripheral Data Controller channels
  - 5-, 6-, 7-, 8- and 9-bit character length

  USART1 also provides the following serial i/f:
  - Modem
  - Hardware handshaking
  - RS485
  - ISO7816 (T = 0 and T = 1).

- **SPI**: the two four wire Serial Peripheral Interface (SPI0 and SPI1) provide serial interfaces in compliant to industry-standard. The SPIs provide full duplex communication with external devices in master and slave mode as well as in multi-master mode. The SPIs have four external chip selects that can be connected to up to 15 devices. The data length is programmable, from 8- to 16-bit. The baud rates, clock polarities, and phases can also be programmed.

- **PDC**: the 8-channel Peripheral Data Controller is dedicated to the two USARTs and to the two SPIs. One PDC channel is connected to the receiving channel and one to the transmitting channel of each peripheral. The user interface of a PDC channel is integrated in the memory space of each USART and SPI. It contains a 32-bit address pointer register and a 16-bit count register. When the programmed data is transferred, an end-of-transfer interrupt is generated by the corresponding peripheral.

- **AIC**: the Advanced Interrupt Controller reduces the CPU overhead in handling internal and external interrupts. It provides 8 levels of priority, with 31 interrupt sources individually maskable and vectored.

- **PIO**: the programmable Parallel I/O Controller multiplexes a set of general input/output signals with the peripherals I/Os. This reduces the external pin count, and provides additional possibilities for external communication. Moreover, the I/O controller is used to control and set internal functional bits from and to the FPU and others.

- **TC**: the Timer Counter 1 block includes three identical 16-bit counter channels. Each channel can be independently programmed to perform a wide range of functions, including pulse generation, delay timing and pulse width modulation. The timer/counter channel has three external clock inputs, five internal clock inputs, and two multi-purpose output signals, which can be configured by the user. Each channel drives an internal interrupt signal, which can be programmed to generate processor interrupts via the Advanced Interrupt Controller (AIC). The timer/counter block features two global registers that act upon all three TC channels. The Block Control Register allows the three channels to be started simultaneously with the same instruction. The Block Mode Register defines the external clock inputs for each timer/counter channel, allowing them to be chained.

- **WD:** the WatchDog timer can prevent system lockup if the main processor is trapped in an endless loop. If an overflow does occur, the watchdog timer generates processor interrupts via the Advanced Interrupt Controller (AIC) and an external low pulse through the PIO. The time-out period is programmable up to a maximum of 32 seconds. The user should reload the watchdog at regular intervals before the timer overflow occurs.

- **CLKGEN**: the CLocK GENerator is under control of ARM. It can be programmed to enable and disable the clocks for some ARM System peripherals. The system clock is free-running. The clock generator provides divided clocks for: Timer Counter, Watchdog, USART and SPI devices. The system clock is divided by integers, which can be programmed through software. An 11-bit counter is used to generate the divided clocks.

The configuration and the status registers of these blocks are written and read through the APB.

The ARM bootstrap sequence is described in Section “D740 System” on page 240.

Memory Map

The first four Megabytes are reserved to the Internal Memory. The addressing area of the EBI (External Bus Interface, inside the AMC) starts at address 0 × 00500000 (after the re-mapping following the boot sequence). The peripheral base addresses are located in the three upper Megabytes of the ARM memory space. The ARM programs reside in the NCS0 address region (See Table 42).

Table 42. ARM Memory Map

Fixed Internal Register Mapping (512 K × 32 = 2 MB)	AIC	0xFFFF FFFF 0xFFFF C000		16KB
	WD	0xFFFF BFFF 0xFFFF 8000		16KB
	Not Used	0xFFFF 7FFF 0xFFFF 4000		16KB
	PIO	0xFFFF 3FFF 0xFFFF 0000		16KB
	TC	0xFFFE FFFF 0xFFFE C000		16KB
	Not Used	0xFFFE BFFF 0xFFFE 8000		16KB
	AMC-EBI	0xFFFE 7FFF 0xFFFE 4000		16KB
	Not Used	0xFFFE 3FFF 0xFFFD 0000		5x16KB
	SPI1	0xFFFC FFFF 0xFFFC C000		16KB
	SPI0	0xFFFC BFFF 0xFFFC 8000		16KB
	USART1	0xFFFC 7FFF 0xFFFC 4000		16KB
	USART0	0xFFFC 3FFF 0xFFFC 0000		16KB
	Not Used	0xFFFB FFFF 0xFFF0 C000		45×16KB
	ADDA	0xFFF0 BFFF 0xFFF0 8000		16KB
	Not Used	0xFFF0 7FFF 0xFFF0 4000		16KB
	CLKGEN	0xFFF0 3FFF 0xFFF0 0000		16KB

Table 42. ARM Memory Map (Continued)

		Not Used	0xFFEF FFFF 0xFFE0 0000	1MB
External 16-bit Memories (4 × 512 K × 16 = 4 MB)	RAM	NCS3	0x008F FFFF 0x0080 0000	1MB
	RAM	Data Memory NCS2	0x007F FFFF 0x0070 0000	1MB
	RAM	Data Memory NCS1	0x006F FFFF 0x0060 0000	1MB
	ROM	ARM Program Memory NCS0	0x005F FFFF 0x0050 0000	1MB
Fixed Internal Register Mapping (256 K × 32 = 1 MB)		MAAR	0x004F FFFF 0x0040 0000	1 MB
Internal 32-bit RAM (8 K × 32 = 32 KBytes)		Internal Memory	0x0000 7FFF 0x0000 0020	32KB
		FIQ	0x0000 001C	
		IRQ	0x0000 0018	
		Reset	0x0000 0000	

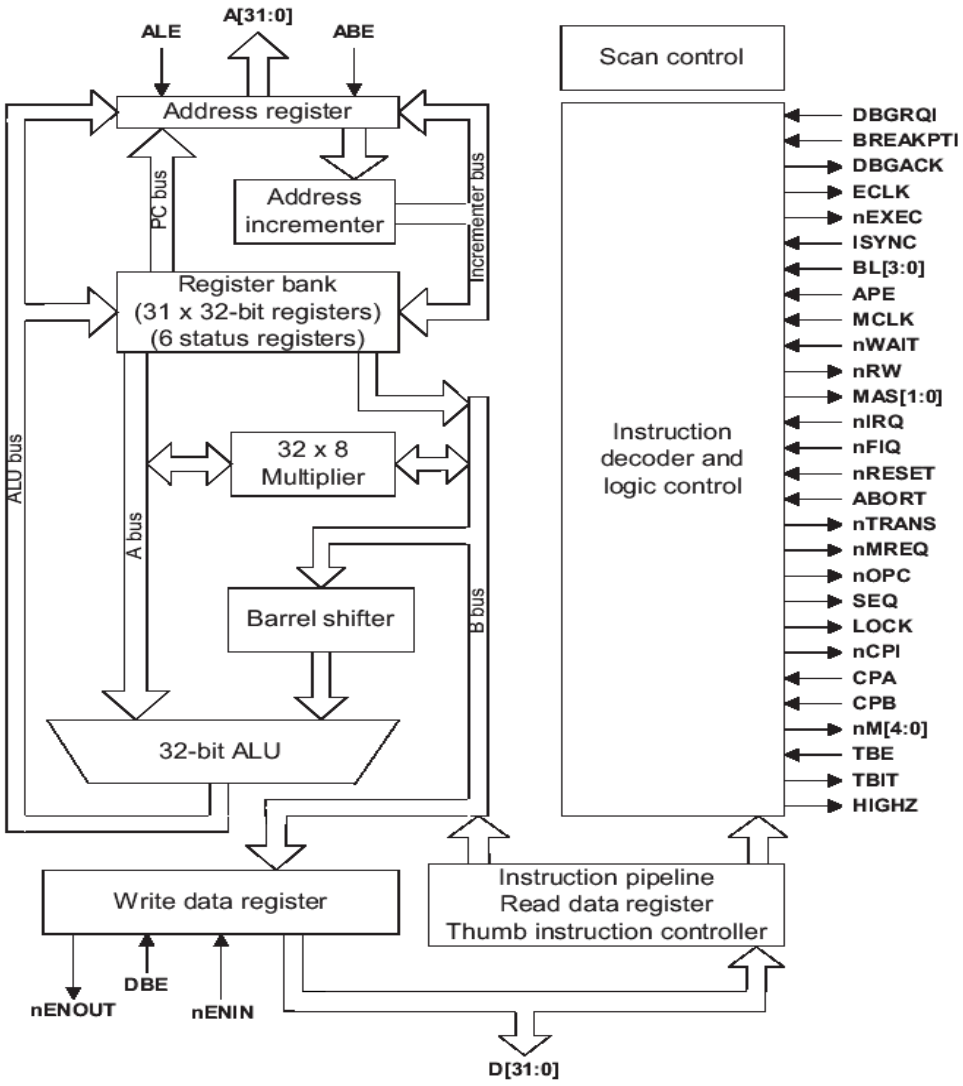
ARM7TDMI

The ARM7TDMI is a member of the Advanced RISC Machines (ARM) family of general-purpose high performance 32-bit microprocessors.

The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles. The instruction set and the related decode mechanism are much simpler than those of microprogrammed Complex Instruction Set Computers. This simplicity results in a high instruction throughput and impressive real-time interrupt response from a small and cost-effective chip. Pipelining is employed so that all parts of the processing and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory.

The ARM memory interface has been designed to allow the performance potential to be realized without incurring high costs in the memory system. Speed-critical control signals are pipelined to allow system control functions to be implemented in standard low-power logic, and these control signals facilitate the exploitation of the fast local access modes offered by industry standard dynamic RAMs.

Figure 22. ARM7TDMI Core Diagram



The ARM7TDMI is a 3-stage pipeline, 32-bit RISC processor. The processor architecture is Von Neumann load/store architecture, which is characterized by a single data and address bus for instructions and data.

The CPU has seven operating modes. Each operating mode has dedicated banked registers for fast exception handling. The processor has a total of 37 32-bit registers, including 6 status registers.

The ARM7TDMI processor employs a unique architectural strategy known as THUMB, which allows optimizing the code density. The key idea behind THUMB is that of a super-reduced instruction set. Essentially, the ARM7TDMI processor has two instruction sets:

- the standard 32-bit ARM set
- a 16-bit THUMB set

The THUMB set's 16-bit instruction length allows to approach twice the density of a standard ARM code while retaining most of the ARM's performance advantage over a

traditional 16-bit processor using 16-bit registers. This is possible because the THUMB code operates on the same 32-bit register set as the ARM code. The THUMB code is able to provide up to 65% of the code size of the ARM, and 160% of the performance of an equivalent ARM processor connected to a 16-bit memory system. THUMB instructions operate with the standard ARM register configuration, allowing excellent interoperability between the ARM and the THUMB states. Each 16-bit THUMB instruction has a corresponding 32-bit ARM instruction with the same effect on the processor model.

The major advantage of a 32-bit (ARM) architecture over a 16-bit architecture is its ability to manipulate 32-bit integers with single instructions, and to efficiently address a large address space. When processing 32-bit data, a 16-bit architecture will take at least two instructions to perform the same task as a single ARM instruction. However, not all the code in a program will process 32-bit data (for example, code that performs character string handling), and some instructions, like Branches, do not process any data at all. If a 16-bit architecture only has 16-bit instructions, and a 32-bit architecture only has 32-bit instructions, then overall the 16-bit architecture will have better code density, and better than one half the performance of the 32-bit architecture. Clearly, the 32-bit performance comes at the cost of code density.

THUMB breaks this constraint by implementing a 16-bit instruction length on a 32-bit architecture, making the processing of 32-bit data efficient with a compact instruction coding. This provides far better performance than a 16-bit architecture, with better code density than a 32-bit architecture. THUMB has also a major advantage over other 32-bit architectures with 16-bit instructions. This is the ability to switch back to full ARM code and execute at full speed. Thus critical loops for applications such as fast interrupts can be coded using the full ARM instruction set, and linked with the THUMB code.

The overhead of switching from the THUMB code to the ARM code is folded into subroutine entry time. Various portions of a system can be optimized for speed or for code density by switching between THUMB and ARM execution as appropriate.

ARM7TDMI has a total of 37 registers – 31 general-purpose32-bit registers and six status registers – but these cannot be seen at once. The processor state and the operating mode dictate which registers are available to the programmer. In ARM state, 16 general registers and one or two status registers are visible at any one time. Mode-specific banked registers are switched on in privileged (non-User) mode.Figure 23 on page 114 shows which registers are available in each mode: the banked registers are marked with a shaded triangle. The ARM state register set contains 16 directly accessible registers: R0 to R15. All of these except R15 are general-purpose, and may be used to hold either data or address values. In addition to these, there is a seventeenth register used to store status information.

The THUMB state register subset of the ARM state set is shown in Figure 24 on page 115. The programmer has direct access to eight general registers, R0-R7, as well as the Program Counter (PC), a stack pointer register (SP), a link register (LR), and the Current Program Status Register (CPSR). There are banked Stack Pointers, Link Registers and Saved Process Status Registers (SPSRs) for each privileged mode.

In D740 the ARM7 is the system master. Whatever mAgic functional mode, the ARM processor and its peripherals (referred to as ARM System) will always be running.

ARM7TDMI views memory as a linear collection of bytes numbered upwards from zero. Bytes 0 to 3 hold the first stored word, bytes 4 to 7 the second and so on. In D740 the endian option chosen for ARM7TDMI is Little Endian. ARM7TDMI supports byte (8-bit), half-word (16-bit) and word (32-bit) data types. The word size is selected by means of BSIZE[1:0] and BA[1:0] signals. Words must be aligned to four-byte boundaries and half

words to two-byte boundaries. In ARM state, all instructions are conditionally executed according to the state of the CPSR condition codes and the instruction's condition field. This field (bits 31:28) determines the circumstances under which an instruction is to be executed. If the state of the C, N, Z and V flags fulfil the conditions encoded by the field, the instruction is executed otherwise it is ignored. There are sixteen possible conditions, each represented by a two-character suffix that can be appended to the instruction's mnemonic. For example, a Branch (B in assembly language) becomes BEQ for "Branch if Equal", which means the Branch will only be taken if the Z flag is set. In the absence of a suffix, the condition field of most instructions is set to "Always" (sufix AL). This means the instruction will always be executed regardless of the CPSR condition codes.

For further reading and documentation on the ARM7 architecture, refer to [1] in Section "Related Documents" on page 268.

Figure 23. Register Organization in ARM State

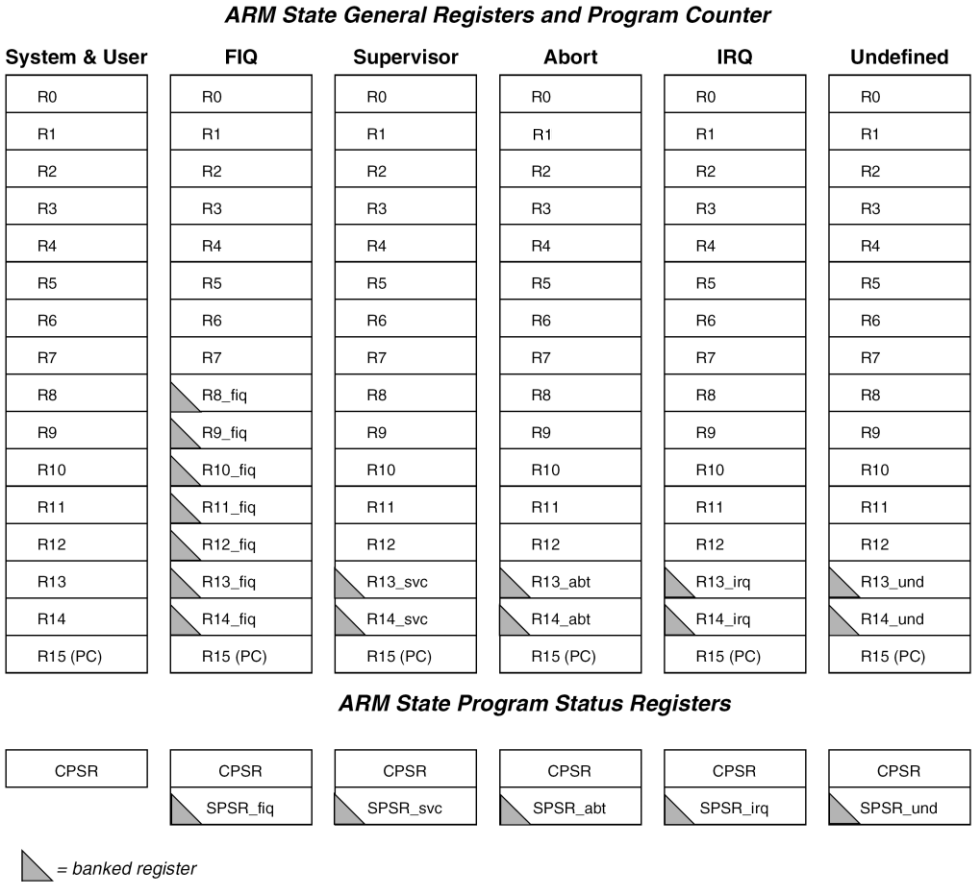
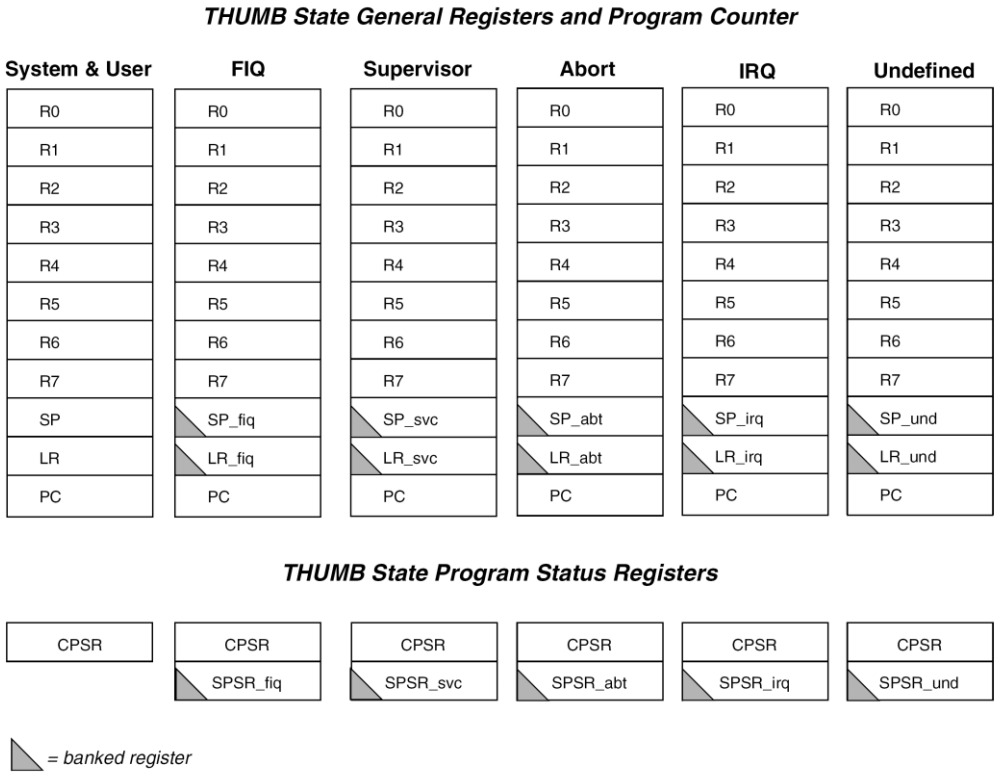


Figure 24. Register Organization in THUMB State



PDC: Peripheral Data Controller

The PDC transfers data between on-chip peripherals (two USARTs and two SPIs) and the on- and off-chip memories. This transfer is achieved via the AMBA Bridge using a simple arbitration mechanism between the AMBA System Bus (ASB) and the PDC to control Bridge access. This avoids processor intervention and removes the processor interrupt handling overhead.

The PDC acts as a master of ASB.

This significantly reduces the number of clock cycles required for a data transfer and, as a result, improves the performance of the ARM and makes it more power efficient.

The PDC channels are implemented in pairs, each pair being dedicated to a particular peripheral. One PDC channel in the pair is dedicated to the receiving channel and one to the transmitting channel of each USART, and SPI.

The user interface of a PDC channel is integrated in the memory space of each peripheral.

It contains a 32-bit memory pointer register and a 16-bit transfer count register plus a 32-bit register for next memory pointer and a 16-bit register for next transfer count. The peripheral triggers PDC transfers using transmit and receive signals.

When the programmed data is transferred, an end of transfer interrupt is generated by the corresponding peripheral.

PDC Configuration

The user interface of a PDC channel is integrated into the user interface of the peripheral which it is related to. Per peripheral, it contains four 32-bit Pointer Registers (RPR, RNPR, TPR, TNPR) and four 16-bit Counter Registers (RCR, RNCR, TCR, TNCR).

The size of the transfer (number of transfers) is configured in an internal 16-bit transfer counter register, and it is possible, at any moment, to read the number of transfers left for each channel.

The base memory address is configured in a 32-bit memory pointer by defining the location of the first access point in the memory. It is possible, at any moment, to read the location of the next transfer in memory.

The PDC has dedicated status registers that indicate if the transfer is enabled or disabled for each channel — the remaining status for each channel is located in the peripheral. Transfers can be enabled and/or disabled by setting TXTEN/TXTDIS and RXTEN/RXTDIS in PDC Transfer Control Registers. The PDC sends status flags (periph_rx_end, periph_tx_end, rx_buffer_full, tx_buffer_empty) to the peripheral, which can latch the flags in its status register.

PDC Memory Pointers

Each peripheral is connected to the PDC by a receive data channel and a transmit data channel. Each channel has an internal 32-bit memory pointer. Each memory pointer points to a location in the system bus memory space (on-chip memory or external bus interface memory).

Depending on the type of transfer (byte, half-word or word), the memory pointer is incremented by 1, 2 or 4, respectively for peripheral transfers.

If a memory pointer is reprogrammed while the PDC is in operation, the transfer addresses are changed, and the PDC performs transfers using the new address.

PDC Transfer Counters

There is one internal 16-bit transfer counter for each channel. Each counter is used to count the size of the block already transferred by its associated channel. These counters are decremented after each data transfer. When the counter reaches zero, the

transfer is complete and the PDC stops transferring data and disables the trigger while activating the related periph_end flag if the Next Counter Register is equal to zero.

If the counter is reprogrammed while the PDC is operating then the number of transfers is changed and the PDC counts transfers from the new value.

When the Next Counter Register is not equal to zero, for example, the values have been programmed into Next Pointer/Counter Registers, the behavior is the same, except that, after activating the flag periph_end when the transfer counter reaches zero, the values of the Next Pointer/Counter Registers are loaded into the Pointer/Counter Registers in order to re-enable triggers. The flag periph_end is automatically cleared when one of the counter registers (Counter or Next Counter Register) is written.

Note: Note: When the Next Counter Register is loaded into the Counter Register, it is set to zero.

PDC Data Transfers

The peripheral triggers PDC transfers using transmit (periph_tx_rdy) and receive (periph_rx_rdy) signals.

When the peripheral receives an external character, it sends a Receive Ready signal to the PDC, which then requests access to the system bus (ASB) from the Bus Arbiter.

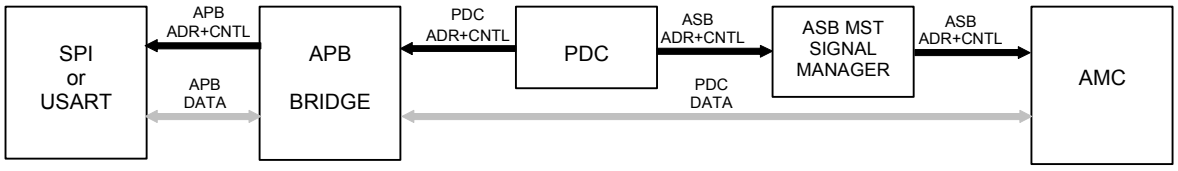
When access is granted, the PDC starts a read of the peripheral Receive Holding Register, via the dedicated pdc_add, pdc_sel, pdc_write and pdc_size signals to the Bridge.

Next, the PDC triggers a write in the memory by setting the ASB control signals and, at the same time, the Bridge provides the data that is to be written to the memory.

After each transfer, the relevant PDC memory pointer is incremented and the number of transfers left is decremented. When the memory block size is reached, a signal is sent to the peripheral and the transfer stops.

The same procedure is followed, in reverse, for transmit transfers. These data exchanges are shown in Figure 25.

Figure 25. PDC Connections



PDC Registers

Base Address: 0xFFFC 0000 for USART0
0xFFFC 4000 for USART1
0xFFFC 8000 for SPI0
0xFFFC C000 for SPI1

Table 43. PDC Register Map

Offset	Register	Name	Access	Reset State
0x100	Receive Pointer Register	PERIPH_RPR	Read/Write	0
0x104	Receive Counter Register	PERIPH_RCR	Read/Write	0
0x108	Transmit Pointer Register	PERIPH_TPR	Read/Write	0
0x10C	Transmit Counter Register	PERIPH_TCR	Read/Write	0
0x110	Receive Next Pointer Register	PERIPH_RNPR	Read/Write	0
0x114	Receive Next Counter Register	PERIPH_RNCR	Read/Write	0
0x118	Transmit Next Pointer Register	PERIPH_TNPR	Read/Write	0
0x11C	Transmit Next Counter Register	PERIPH_TNCR	Read/Write	0
0x120	PDC Transfer Control Register	PERIPH_PTCR	Only Write	-
0x124	PDC Transfer Status Register	PERIPH_PTSR	Only Read	0

PDC Receive Pointer Register

Register Name PERIPH_RPR
Access Type: Read/Write
Reset Value See Table 43
Absolute Address 0×FFFC 0100 for USART0
0×FFFC 4100 for USART1
0×FFFC 8100 for SPI0
0×FFFC C100 for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 0	RXPTR: Peripheral Receive Pointer Register	32	RXPTR must be loaded with the address of the receive buffer.

PDC Receive Counter Register

Register Name	PERIPH_RCR
Access Type	Read/Write
Reset Value	See Table 43 on page 118
Absolute Address	0xFFFC 0104 for USART0
	0xFFFC 4104 for USART1
	0xFFFC 8104 for SPI0
	0xFFFC C104 for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 16	UNUSED	16	
15 downto 0	RXCTR: Receive Counter Register	16	RXCTR must be loaded with the size of the receive buffer. 0 = Stop peripheral data transfer to the receiver 1 - 65535 = Start peripheral data transfer if corresponding periph_rx_rdy is active

PDC Transmit Pointer Register

Register Name	PERIPH_TPR
Access Type	Read/Write
Reset Value	See Table 43 on page 118
Absolute Address	0xFFFC 0108 for USART0
	0xFFFC 4108 for USART1
	0xFFFC 8108 for SPI0
	0xFFFC C108 for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 0	TXPTR: Peripheral Receive Pointer Register	32	TXPTR must be loaded with the address of the transmit buffer.

PDC Transmit Counter Register

Register Name	PERIPH_TCR
Access Type	Read/Write
Reset Value	See Table 43 on page 118
Absolute Address	0xFFFC 010C for USART0
	0xFFFC 410C for USART1
	0xFFFC 810C for SPI0
	0xFFFC C10C for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 0	UNUSED	16	
15 downto 0	TXCTR: Transmit Counter Register	16	TXCTR must be loaded with the size of the receive buffer. 0 = Stop peripheral data transfer to the transmitter 1 - 65535 = Start peripheral data transfer if corresponding periph_tx_rdy is active

PDC Receive Next Pointer Register

Register Name	PERIPH_RNPR
Access Type	Read/Write
Reset Value	See Table 43 on page 118
Absolute Address	0xFFFC 0110 for USART0
	0FFFC 4110 for USART1
	0xFFFC 8110 for SPI0
	0xFFFC C110 for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 0	RXNPTR: Peripheral Receive Next Pointer Register	32	RXNPTR contains the address of the next buffer to fill with received data when the current one is completed. Must be loaded with the address of the receive buffer.

PDC Receive Next Counter Register

Register Name	PERIPH_RNCR
Access Type	Read/Write
Reset Value	See Table 43 on page 118
Absolute Address	0xFFFC 0114 for USART0
	0xFFFC 4114 for USART1
	0xFFFC 8114 for SPI0
	0xFFFC C114 for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 16	UNUSED	16	
15 downto 0	RXNCTR: Receive Next Counter Register	16	RXNCTR contains the next buffer maximum size.

PDC Transmit Next Pointer Register

Register Name	PERIPH_TNPR
Access Type	Read/Write
Reset Value	See Table 43 on page 118
Absolute Address	0xFFFC 0118 for USART0
	0xFFFC 4118 for USART1
	0xFFFC 8118 for SPI0
	0xFFFC C118 for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 0	TXNPTR: Peripheral Transmit Next Pointer Register	32	TXNPTR contains the address of the next buffer where to read data from when the current one is complete.

PDC Next Transmit Counter Register

Register Name	PERIPH_TNCR
Access Type	Read/Write
Reset Value	See Table 43 on page 118
Absolute Address	0xFFFC 011C for USART0
	0xFFFC 411C for USART1
	0xFFFC 811C for SPI0
	0xFFFC C11C for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 16	UNUSED	16	
15 downto 0	TXNCTR: Peripheral Transmit Next Counter Register	16	TXNCTR contains the next buffer maximum size.

PDC Transfer Control Register

Register Name	PERIPH_ PTCR
Access Type	Write
Absolute Address	0xFFFC 0120 for USART0
	0xFFFC 4120 for USART1
	0xFFFC 8120 for SPI0
	0xFFFC C120 for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 10	UNUSED	22	
9	TXTDIS: Transmit Transfer Disable	1	0 = No effect 1 = Disables the transmitter PDC requests.
8	TXTEN: Transmit Transfer Enable	1	0 = No effect 1 = Enables the transmitter PDC requests.

Bit(s) Number (Continued)	Name	Length	Comments
7 downto 2	UNUSED	6	
1	RXTDIS: Receive Transfer Disable	1	0 = No effect 1 = Disables the receiver PDC requests.
0	RXTEN: Receive Transfer Enable	1	0 = No effect 1 = Enables the receiver PDC requests.

PDC Transfer Status Register

Register Name	PERIPH_ PTSR
Access Type	Read
Reset Value	See Table 43 on page 118
Absolute Address	0xFFFC 0124 for USART0
	0xFFFC 4124 for USART1
	0xFFFC 8124 for SPI0
	0xFFFC C124 for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 2	UNUSED	22	
1	RXTDIS: Transmit Transfer Disable	1	0 = Receiver PDC transfer requests are disabled. 1 = Receiver PDC transfer requests are enabled.
0	RXTEN: Transmit Transfer Enable	1	0 = Transmitter PDC transfer requests are disabled. 1 = Transmitter PDC transfer requests are enabled.

AMC-EBI: Advanced Memory Controller - External Bus Interface

The External Bus Interface (EBI) is designed to ensure the successful data transfer between several external devices and the embedded memory controller of ARM7TDMI based device. The EBI include a Static Memory Controller capable of handling several types of external memory and peripheral devices, such as

SRAM, PROM, EPROM, EEPROM, Flash.

The EBI is fully programmable and can address up to 4M bytes. It has four chip selects and a 19-bit address bus.

The 16-bit data bus can be configured to interface with 8-or 16-bit external devices. Separate write control signals allow direct memory and peripheral interfacing.

The EBI supports different access protocols, allowing single-clock-cycle memory accesses.

The main features are:

- Programmable External memory mapping
- Up to four chip select lines
- 8- or 16-bit data bus
- Byte-write lines
- Remap of boot memory
- Two different read protocols
- Programmable wait state generation
- Programmable data float time

The memory map associates the internal 32-bit address space with the external 19-bit address bus ARM_A[0:18]. The memory map is defined by programming the base address and page size of the external memories.

If the physical memory device is smaller than the programmed page size, it wraps around and appears to be repeated within the page. The EBI correctly handles any valid access to the memory device within the page. In the event of an access request to an address outside any programmed page, an abort signal is generated. Two types of aborts are possible: instruction prefetch abort and data abort. The corresponding exception vector addresses are, respectively, 0x0000000C and 0x00000010. It is up to the system programmer to program the error handling routine to use in case of an abort.

After reset ARM_NCS0 is mapped at address 0 × 0 (it normally correspond to a non volatile program memory), while the internal SRAM at address 0 × 300000.

This is because ARM, after reset, fetches automatically its first instruction from address 0 × 0.

However, after initialization the internal SRAM (it stores the hard exception vectors) has to be mapped at address 0 × 0.

This is achieved by a remap procedure (through EBI configuration registers) that let the internal SRAM to be mapped at address 0 × 0 and the CS0 NVM at 0 × 500000.

ARM external data bus is 16 bits wide, so organized:

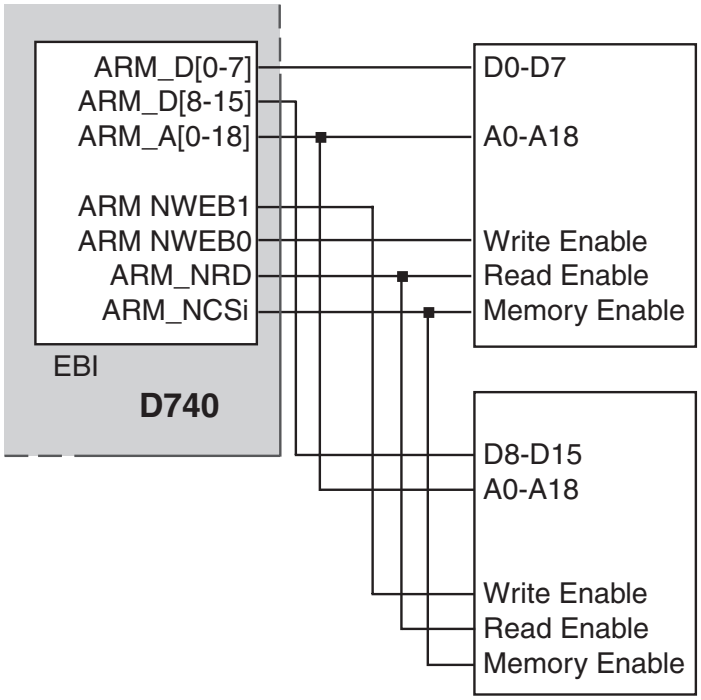
	Ext Mem	
bit	15 8	7 0
addr n	byte 1	byte 0
addr n +1	byte 3	byte 2

related to instruction and double word data as follows:

ARM Data / Instruction			
31 24	23 16	15 8	7 0
byte 3	byte 2	byte 1	byte 0

A typical external memory configuration (two 512K × 8) is shown in Figure 26.

Figure 26. ARM External Memory Architecture Example



In order to interface slower memory devices up to 8 wait states (1 WS = 1 arm_clock period) can be inserted to stretch the access timing.

The EBI supports both standard read protocol and an early mode protocol that let the read command start half arm_clock cycle before its standard behavior. The early read protocol, allowing the assertion of the read signal at the beginning of the access facilitates running at 0 wait state with external devices.

Furthermore the EBI supports up to 7 data float times (1 FT = 1 arm_clock period) that consist in wait time inserted after a read access is finished to prevent any bus contention

in case the external device is too slow in releasing the bus or latches the data too long after the chip select rising edge.

The EBI is programmed by using the registers listed in Table 44 on page 126. Four Chip Select Registers (EBI_CSR0 to EBI_CSR3) are used to program the parameters for the individual external memories.

EBI Register

Base Address: 0×FFFE400.

Table 44. EBI Register Map

Offset	Register	Name	Access	Reset State
0×00	Chip Select Register 0	EBI_CSR0	Read/Write	0×0000 203D
0×04	Chip Select Register 1	EBI_CSR1	Read/Write	0×1000 0000
0×08	Chip Select Register 2	EBI_CSR2	Read/Write	0×2000 0000
0×0C	Chip Select Register 3	EBI_CSR3	Read/Write	0×3000 0000
0×10	Chip Select Register 4 (UNUSED)	EBI_CSR4	Read/Write	0×4000 0000
0×14	Chip Select Register 5 (UNUSED)	EBI_CSR5	Read/Write	0×5000 0000
0×18	Chip Select Register 6 (UNUSED)	EBI_CSR6	Read/Write	0×6000 0000
0×1C	Chip Select Register 7 (UNUSED)	EBI_CSR7	Read/Write	0×7000 0000
0×20	Remap Control Register	EBI_RCR	Write Only	-
0×24	Memory Control Register	EBI_MCR	Read/Write	0×0000 0000

EBI Chip Select Register

Register Name	EBI_CSR0 – EBI_CSR
Access Type	Read/Write
Reset Value	See Table 44
Absolute Address	0xFFFE4000 – 0xFFFE401C

Bit(s) Number	Name	Length	Comments
31 downto 20	BA: Base Address	12	
19 downto 14	UNUSED	4	
13	CSEN: Chip Select Enable	1	0 = Chip select is disabled 1 = Chip select is enabled
12	BAT: Byte Access Type	1	0 = Byte write access type 1 = Byte select access type
11 downto 9	TDF: Data Float Output Time	3	Number of cycles added after the transfer.

Bit(s) Number (Continued)	Name	Length	Comments		
8 downto 7	PAGES: Page Size	2		**Page Size**	**Active bits in base address**
			0 0	1M byte	12 bits (31-20)
			0 1	4M byte	10 bits (31-22)
			1 0	16M byte	8 bits (31-24)
			1 1	64M byte	6 bits (31-26)
6	UNUSED				
5	WSE: Wait State Enable	1	0 = Wait state generation is disabled 1 = Wait state generation is enabled		
4 downto 2	NWS: Number of Wait States	3	This field is valid only if WSE is set		
1 downto 0	DBW: Data Bus Width	2	0 1 = 16-bit data bus width 1 0 = 8-bit data bus width		

EBI_CSRx [5:2]				Number of Added Wait States
WSE	**NWS[2]**	**NWS[1]**	**NWS[0]**	
0	X	X	X	0
1	0	0	0	1
1	0	0	1	2
1	0	1	0	3
1	0	1	1	4
1	1	0	0	5
1	1	0	1	6
1	1	1	0	7
1	1	1	1	8

EBI Remap Control Register

Register Name	EBI_RCR
Access Type	Write Only
Absolute Address	0xFFFE4020

Bit(s) Number	Name	Length	Comments
31 downto 1	UNUSED	31	
0	RCB: Remap Command Bit	1	0 = No Effect 1 = Cancels the remapping (performed at reset) of the two devices.

EBI Memory Control Register

Register Name	EBI_MCR
Access Type	Read/Write
Reset ValueV	See Table 44 on page 126
Absolute Address	0xFFFE4024

Bit(s) Number	Name	Length	Comments
31 downto 5	UNUSED	27	
4	DRP: Data Read Protocol	1	0 = Standard read protocol. 1 = Early read protocol.
3	UNUSED	1	
2 downto 0	ALE: Address Line Enable	3	This field determines the number of valid address lines and the number of valid chip select lines. See Table 45

Table 45. Address Line Enable

ALE			Valid Address Bits	Maximum Addressable Space	Valid Additional Chip Select
0	X	X	A20, A21, A22, A23	16M bytes	None
1	0	0	A20, A21, A22	8M bytes	CS4 (1)
1	0	1	A20, A21	4M bytes	CS4, CS5 (1)
1	1	0	A20	2M bytes	CS4, CS5, CS6 (1)
1	1	1	None	1M bytes	CS4, CS5, CS6, CS7 (1)

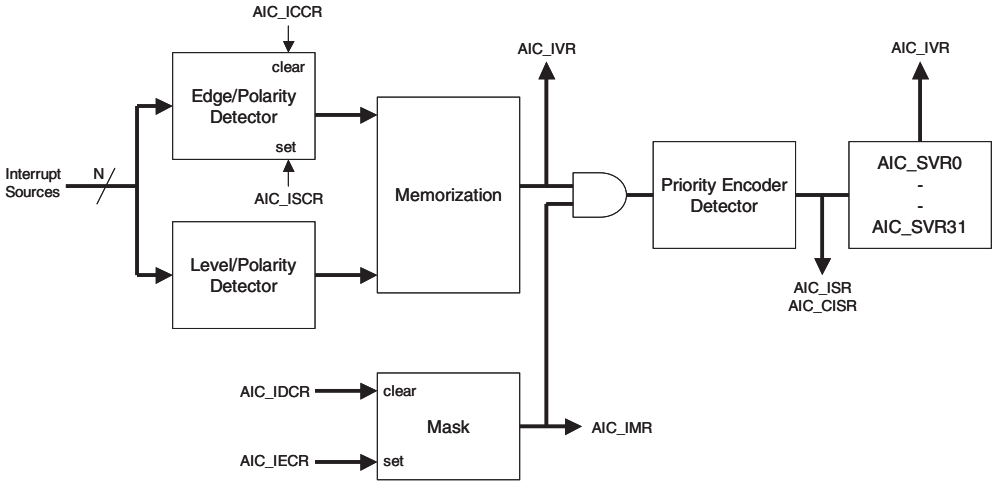
1. This configuration is not applicable in D740, because chip selects from CS4 to CS7 are not available as output pin.

AIC: Advanced Interrupt Controller

The AIC is an 8-level priority, individually maskable, vectored interrupt controller. The ARM programs the AIC configuration registers through the APB. The AIC provides two interrupts outputs directly connected to the NFIQ (Fast Interrupt reQuest) and the NIRQ (standard interrupt request) inputs of the ARM7TDMI processor. The processor's NFIQ line can only be asserted by the external fast interrupt request input: FIQ while to the ARM nIRQ are associated 24 interrupt sources. The interrupt source mapping is defined in Table 46 on page 133.

The fast forcing feature available in the AIC allows redirection of any internal or external interrupt source to provide a fast interrupt rather than a normal interrupt. The processor's FIQ line can be asserted either by the external fast interrupt request input FIQ, or by the other interrupt sources via the fast forcing feature. The NIRQ line is asserted by all other interrupt sources except those that assert FIQ. An 8-level priority encoder allows the programmer to define the priority between the different nIRQ interrupt sources. Internal interrupt sources can be programmed to be level sensitive or edge triggered. External interrupt sources can be programmed to be positive-edge or negative-edge triggered or high-level or low-level sensitive.

Figure 27. AIC Internal Architecture



AIC Interrupt Vectoring

The hardware interrupt vectoring reduces the number of instructions required to manage the interrupt handlers to only one. By storing the following instruction at address 0×00000018,

```
LDR PC,[PC,# -&F20]
```

ARM loads the program counter with the interrupt handler address stored in the AIC_IVR register. Execution is then vectored to the interrupt handler corresponding to the current interrupt.

The current interrupt is the interrupt with the highest priority when the Interrupt Vector Register (AIC_IVR) is read. The value read in the AIC_IVR corresponds to the address stored in the Source Vector Register (AIC_SVR) of the current interrupt. Each interrupt source has its corresponding AIC_SVR. In order to take advantage of the hardware interrupt vectoring, it is necessary to store the address of each interrupt handler in the corresponding AIC_SVR, at system initialization.

AIC Interrupt Priority

An 8-level priority encoder allows the customer to define the priority between the different nIRQ interrupt sources.

Each source has a programmable priority level of 7 to 0. Level 7 is the highest priority and level 0 the lowest.

When the AIC receives more than one unmasked interrupt at a time, the interrupt with the highest priority is serviced first. If both interrupts have equal priority, the interrupt with the lowest interrupt source number is serviced first.

The current priority level is defined as the priority level of the current interrupt at the time the register AIC_IVR is read (the interrupt which will be serviced).

In the case a higher priority unmasked interrupt occurs while an interrupt already exists, there are two possible outcomes depending on whether the AIC_IVR has been read.

1. If the nIRQ line has been asserted but the AIC_IVR has not been read, then the processor will read the new higher priority interrupt handler address in the AIC_IVR register and the current interrupt level is updated
2. If the processor has already read the AIC_IVR then the nIRQ line is reasserted

When the processor has authorized nested interrupts to occur and reads the AIC_IVR again, it reads the new, higher priority interrupt handler address. At the same time the current priority value is pushed onto a first-in last-out stack and the current priority is updated to the higher priority.

When the end of interrupt command register (AIC_EOICR) is written the current interrupt level is updated with the last stored interrupt level from the stack (if any). Hence at the end of a higher priority interrupt, the AIC returns to the previous state corresponding to the preceding lower priority interrupt which had been interrupted.

The interrupt handler must read the AIC_IVR as soon as possible. This de-asserts the nIRQ request to the processor and clears the interrupt in case it is programmed to be edge triggered.

This permits the AIC to assert the nIRQ line again when a higher priority unmasked interrupt occurs.

At the end of the interrupt service routine, the end of interrupt command register (AIC_EOICR) must be written. This allows pending interrupts to be serviced.

AIC Interrupt Masking

Each interrupt source, including FIQ, can be enabled or disabled using the command registers AIC_IECR and AIC_IDCR. The interrupt mask can be read in the read only register AIC_IMR. A disabled interrupt does not affect the servicing of other interrupts.

AIC Interrupt Set & Clear

All interrupt sources which are programmed to be edge triggered (including FIQ) can be individually set or cleared by respectively writing to the registers AIC_ISCR and AIC_ICCR. This function of the interrupt controller is available for auto-test or software debug purposes.

AIC Fast Interrupt

The external FIQ line is the only source which can raise a fast interrupt request to the processor.

Therefore, it has no priority controller. It can be programmed to be positive or negative edge triggered or high- or low-level sensitive. The fast interrupt handler address can be stored in the related AIC_SVR register (SVR[0]). The value written into this register is available by reading the AIC_FVR register when an FIQ interrupt is raised. By storing the following instruction at address 0×0000001C:

```
LDR PC,[PC,# -&F20]
```

ARM will load the program counter with the interrupt handler address stored in the AIC_FVR register.

Alternatively, the interrupt handler can be stored starting from address 0×0000001C.

AIC Fast Forcing

The Fast Forcing feature is enabled or disabled by using the Fast Forcing Enable Register (AIC_FFER) or the Fast Forcing Disable Register (AIC_FFDR). Writing to these registers results in an update of the Fast Forcing Status Register (AIC_FFSR) that controls the feature for each internal or external interrupt source. When the feature is disabled, the interrupt source is handled as is typical for the current operating mode.

When Fast Forcing is enabled, the edge/level programming and, in certain cases, edge detection of the interrupt source are both still active, but the source cannot trigger a normal interrupt to the core and is not seen by the priority handler.

If the interrupt source is programmed in level mode and an active level is sampled, the Fast Forcing results in the assertion of the nFIQ line to the core.

If the interrupt source is programmed in edge mode and an active edge is detected, the Fast Forcing results in the assertion of the nFIQ line to the core.

The Fast Forcing feature does not affect the Source 0 pending bit in the Interrupt Pending Register.

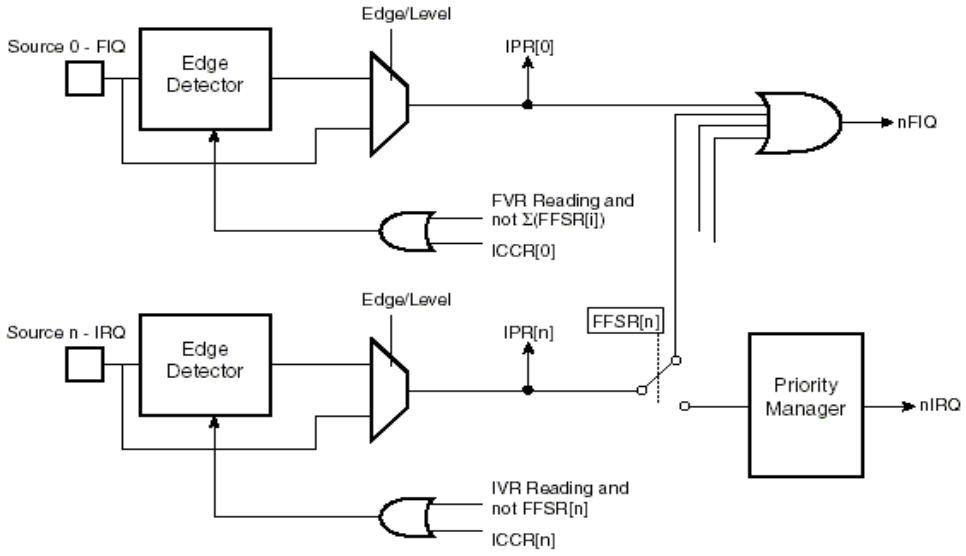
The Fast Interrupt Vector Register (AIC_FVR) reads the contents of the Source Vector Register 0, whatever the source of the Fast Interrupt may be. The read of the FVR does not clear the Source 0 when the Fast Forcing feature is used; the interrupt is cleared by writing to the Interrupt Clear Command Register.

All enabled and pending interrupt sources that have the Fast Forcing feature enabled and that are programmed in edge mode must be cleared by writing to the Interrupt Clear Command Register. In doing so, they are cleared independently and thus lost interrupts are prevented.

The read of the IVR does not clear the source that has the Fast Forcing feature enabled.

The FIQ pin continues to operate normally and becomes one of the Fast Interrupt sources.

Figure 28. AIC Fast Forcing Interaction



AIC Interrupt Source Configuration

IRQ1 to IRQ17 interrupt sources can be programmed to be high level sensitive or positive edge triggered.

FIQ and IRQ24 to IRQ31 can be programmed to be positive-edge or negative-edge triggered or high-level or low-level sensitive.

Table 46. ARMSYSTEM AIC Interrupt Source Mapping

Source	Name	Description	Type	Assignment
0	FIQ	Fast Interrupt	Edge/Level Negative/Positive	FPU_HALT from mAgic
1	IRQ1	Interrupt 1	Edge/Level Positive only	TIMER_INT(0) from TIMER
2	IRQ2	Interrupt 2		TIMER_INT(1) from TIMER
3	IRQ3	Interrupt 3		TIMER_INT(2) from TIMER
4	IRQ4	Interrupt 4		USART0_INT from USART-0
5	IRQ5	Interrupt 5		USART1_INT from USART-1
6	IRQ6	Interrupt 6		SPI0_INT from SPI0
7	IRQ7	Interrupt 7		SPI1_INT from SPI1
8	IRQ8	Interrupt 8		FPU_SIRQ(0) from mAgic
9	IRQ9	Interrupt 9		FPU_SIRQ(1) from mAgic
10	IRQ10	Interrupt 10		FPU_SIRQ(2) from mAgic
11	IRQ11	Interrupt 11		FPU_SIRQ(3) from mAgic
12	IRQ12	Interrupt 12		FPU_SIRQ(4) from mAgic
13	IRQ13	Interrupt 13		FPU_SIRQ(5) from mAgic
14	IRQ14	Interrupt 14		not used
15	IRQ15	Interrupt 15		FPU_EXC from mAgic
16	IRQ16	Interrupt 16		WD_IRQ from WD
17	IRQ17	Interrupt 17		PIO_INT from PIO
18	IRQ18	Interrupt 18		not used
19	IRQ19	Interrupt 19		not used
20	IRQ20	Interrupt 20		not used
21	IRQ21	Interrupt 21		not used
22	IRQ22	Interrupt 22		not used
23	IRQ23	Interrupt 23		not used
24	IRQ24	Interrupt 24	Edge/Level Negative/Positive	EXT_IRQ[4] from PIO[27] pad
25	IRQ25	Interrupt 25		FPU_MODE from mAgic
26	IRQ26	Interrupt 26		PIO_LOOP_IN from PIO[5] pad
27	IRQ27	Interrupt 27		FPU_SIRQ(6) from mAgic
28	IRQ28	Interrupt 28		EXT_IRQ(0) from PIO[12] pad
29	IRQ29	Interrupt 29		EXT_IRQ(1) from PIO[13] pad
30	IRQ30	Interrupt 30		EXT_IRQ(2) from PIO[14] pad
31	IRQ31	Interrupt 31		EXT_IRQ(3) from PIO[15] pad

Base Address 0xFFFFC000

Table 47. AIC Register Map

Offset	Register	Name	Access	Reset State
0x3000	Source Mode Register 0	AIC_SMR0	Read/Write	0
0x3004	Source Mode Register 1	AIC_SMR1	Read/Write	0
-	-	-	-	-
0x307C	Source Mode Register 31	AIC_SMR31	Read/Write	0
0x3080	Source Vector Register 0	AIC_SVR0	Read/Write	0
0x3084	Source Vector Register 1	AIC_SVR1	Read/Write	0
-	-	-	-	-
0x30FC	Source Vector Register 31	AIC_SVR31	Read/Write	0
0x3100	IRQ Vector Register	AIC_IVR	Read Only	0
0x3104	FIQ Vector Register	AIC_FVR	Read Only	0
0x3108	Interrupt Status Register	AIC_ISR	Read Only	0
0x310C	Interrupt Pending Register	AIC_IPR	Read Only	(1)
0x3110	Interrupt Mask Register	AIC_IMR	Read Only	0
0x3114	Core Interrupt Status Register	AIC_CISR	Read Only	0
0x3120	Interrupt Enable Command Register	AIC_IECR	Write Only	-
0x3124	Interrupt Disable Command Register	AIC_IDCR	Write Only	-
0x3128	Interrupt Clear Command Register	AIC_ICCR	Write Only	-
0x312C	Interrupt Set Command Register	AIC_ISCR	Write Only	-
0x3130	End of Interrupt Command Register	AIC_EOICR	Write Only	-
0x3134	Spurious Interrupt Vector Register	AIC_SPU	Read/Write	0
0x3138	Debug Control Register (Protect)	AIC_DEBUG	Read/Write	0
0x3140	Fast Forcing Enable Register	AIC_FFER	Write Only	-
0x3144	Fast Forcing Disable Register	AIC_ FFDR	Write Only	-
0x3148	Fast Forcing Status Register	AIC_ FFSR	Read Only	0

1. The reset value of this register depends on the level of the External IRQ lines. All other sources are cleared at reset.

AIC Source Mode Register

Register Name	AIC_SMR0 – AIC_SMR31
Access Type	Read/Write
Reset Value	See Table 47 on page 134
Absolute Address	0×FFFF F000 – 0×FFFF F07C

Bit(s) Number	Name	Length	Comments		
31 downto 7	UNUSED	25			
6 downto 5	SCRTYPE: Interrupt Source Type	2		**Internal Sources**	**External Sources**
			0 0	Level Sensitive	Low Level Sensitive
			0 1	Edge Triggered	Negative Edge Triggered
			1 0	Level Sensitive	High Level Sensitive
			1 1	Edge Triggered	Positive Edge Triggered
4 downto 3	UNUSED	2			
2 downto 0	PRIOR: Priority Level	3	Program the priority level for all sources except 0 (FIQ). The priority level can be between 0 (lowest) and 7. The priority level is not used for the FIQ, in the SMR0.		

Notes:
1. when two or more interrupts (IRQ1:IRQ31) with the same programmed priority level are active, the first to be served is the one corresponding to the lowest interrupt source number.
2. Note: since the internal sourced interrupts are always “positive only”, bit(6) of AIC_SMRx cannot be written and it is fixed low when reading.

AIC Source Vector Register

Register Name	AIC_SVR0 – AIC_SVR31
Access Type	Read/Write
Reset Value	See Table 47 on page 134
Absolute Address	0xFFFF F080 – 0xFFFF F0FC

Bit(s) Number	Name	Length	Comments
31 downto 0	VECTOR: Interrupt Handler Address	32	For each interrupt source, the user may store the addresses of the corresponding handler in these registers.

AIC IRQ Vector Register

Register Name	AIC_IVR
Access Type	Read Only
Reset Value	See Table 47 on page 134
Absolute Address	0xFFFF F100

Bit(s) Number	Name	Length	Comments
31 downto 0	IRQV: Interrupt Vector Register	32	The IRQ Vector Interrupt contains the vector programmed by the user in the Source Vector Register corresponding to the current interrupt. The Source Vector Register (31 down to 1) is indexed using the current interrupt number when the Interrupt Vector Register is read. When there is no current interrupt, the IRQ Vector Register reads 0.

AIC FIQ Vector Register

Register Name	AIC_FVR
Access Type	Read Only
Reset Value	See Table 47 on page 134
Absolute Address	0xFFFF F104

Bit(s) Number	Name	Length	Comments
31 downto 0	FIQV:FIQ Vector Register	32	The IRQ Vector Register contains the vector programmed by the user in the Source Vector Register 0 which corresponds to FIQ.

AIC Interrupt Status Register

Register Name	AIC_ISR
Access Type	Read Only
Reset Value	See Table 47 on page 134
Absolute Address	0xFFFF F108

Bit(s) Number	Name	Length	Comments
31 downto 5	UNUSED	27	
4 downto 0	IRQID: Current IRQ Identifier	5	The interrupt Status Register returns the current interrupt source number.

AIC Interrupt Pending Register

Register Name	AIC_IPR
Access Type	Read Only
Reset Value	See Table 47 on page 134
Absolute Address	0xFFFF F10C

Bit(s) Number	Name	Length	Comments
31 downto 1	IRQ Interrupt Pending	31	0 = Corresponding interrupt is inactive. 1 = Corresponding interrupt is pending
0	FIQ Interrupt Pending	1	0 = Corresponding interrupt is inactive. 1 = Corresponding interrupt is pending

AIC Interrupt Mask Register

Register Name	AIC_IMR
Access Type	Read Only
Reset Value	See Table 47 on page 134
Absolute Address	0xFFFF F110

Bit(s) Number	Name	Length	Comments
31 downto 1	IRQ Interrupt Mask	31	0 = Corresponding interrupt is disabled. 1 = Corresponding interrupt is enabled
0	FIQ Interrupt Mask	1	0 = Corresponding interrupt is disabled. 1 = Corresponding interrupt is enabled

AIC Core Interrupt Status Register

Register Name	AIC_CISR
Access Type	Read Only
Reset Value	See Table 47 on page 134
Absolute Address	0xFFFF F114

Bit(s) Number	Name	Length	Comments
31 downto 2	UNUSED		
1	NIRQ Status	1	0 = NIRQ line inactive. 1 = NIRQ line active.
0	NFIQ Status	1	0 = NFIQ line inactive. 1 = NFIQ line active.

AIC Interrupt Enable Command Register

Register Name	AIC_IECR
Access Type	Write Only
Absolute Address	0xFFFF F120

Bit(s) Number	Name	Length	Comments
31 downto 1	IRQ Interrupt Enable	31	0 = No effect. 1 = Enables corresponding interrupt.
0	FIQ Interrupt Enable	1	0 = No effect. 1 = Enables corresponding interrupt.

AIC Interrupt Disable Command Register

Register Name	AIC_IDCR
Access Type	Write Only
Absolute Address	0xFFFF F124

Bit(s) Number	Name	Length	Comments
31 downto 1	IRQ Interrupt Disable	31	0 = No effect. 1 = Disables corresponding interrupt.
0	FIQ Interrupt Disable	1	0 = No effect. 1 = Disables corresponding interrupt.

Each interrupt source, including FIQ, can be enabled or disabled using the command register AIC_IECR and AIC_IDCR. The interrupt mask can be read in the read only register AIC_IMR. A disabled interrupt does not affect the servicing of other interrupts.

AIC Interrupt Clear Command Register

Register Name	AIC_ICCR
Access Type	Write Only
Absolute Address	0xFFFF F128

Bit(s) Number	Name	Length	Comments
31 downto 1	IRQ Interrupt Clear	31	0 = No effect. 1 = Clears corresponding interrupt.
0	FIQ Interrupt Clear	1	0 = No effect. 1 = Clears corresponding interrupt.

AIC Interrupt Set Command Register

Register Name	AIC_ISCR
Access Type	Write Only
Absolute Address	0xFFFF F12C

Bit(s) Number	Name	Length	Comments
31 downto 1	IRQ Interrupt Set	31	0 = No effect. 1 = Sets corresponding interrupt.
0	FIQ Interrupt Set	1	0 = No effect. 1 = Sets corresponding interrupt.

All interrupt sources, which are programmed to be edge triggered (including FIQ) can be individually set or cleared by respectively writing to the register AIC_ISCR, and AIC_ICCR. This interrupt controller function is available for auto-test or software debug purposes.

This is totally independent from the SWI instruction of the ARM7 processor.

AIC End of Interrupt Command Register

Register Name	AIC_EOICR
Access Type	Write Only
Absolute Address	0xFFFF F130

The End of the Interrupt Command Register is used by the interrupt routine to indicate that the interrupt treatment is complete. Any value can be written because it is only necessary to write to this register location to signal the end of interrupt treatment.

When the End of Interrupt Command Register is written, the current interrupt level is updated with the last stored interrupt level from the stack (if any).

AIC SPU Vector Register

Register Name	AIC_SPU
Access Type	Read/Write
Reset Value	See Table 47 on page 134
Absolute Address	0xFFFF F134

Bit(s) Number	Name	Length	Comments
31 downto 0	SPU: Spurious Interrupt Vector Register	32	For a spurious interrupt source, the user may store the addresses of the corresponding handler in these registers.

The programmed address value of AIC_SPU is read in the AIC_IVR if it is read when the NIRQ line is not asserted.

The programmed address value of AIC_SPU is read in the AIC_FVR if it is read when the NFIQ line is not asserted.

Debug Control Register (Protect)

Register Name	AIC_DEBUG
Access Type	Read/Write
Reset Value	See Table 47 on page 134
Absolute Address	0xFFFF F138

Bit(s) Number	Name	Length	Comments
31 downto 2	UNUSED	30	
1	DBG_MASK: Interrupt output mask	1	When high, nfiq and nirq outputs are disabled (set to high). Note that nint output is not affected.
0	DBG_MODE: Protect mode enable	0	Activates Protect Mode when set to high. Protect Mode allows reading the Interrupt Vector Register without performing the associated automatic operations.

AIC Fast Forcing Enable Register

Register Name	AIC_FFER
Access Type	Write Only
Absolute Address	0xFFFF F140

Bit(s) Number	Name	Length	Comments
31 downto 1	IRQ Interrupt Fast Forcing Enable	31	0 = No effect. 1 = Enables Forcing on corresponding interrupt to FIQ.
0	FIQ Interrupt Fast Forcing Enable	1	0 = No effect. 1 = Enables Forcing on corresponding interrupt to FIQ.

The fast forcing feature allows redirection of any internal or external interrupt source in order to provide a fast interrupt rather than a normal interrupt. The processor's nFIQ line can be asserted either by the external fast interrupt request input FIQ, or by the other interrupt sources via the fast forcing feature. The nIRQ line is asserted by all other interrupt sources except by those that assert FIQ.

AIC Fast Forcing Disable Register

Register Name	AIC_FFDR
Access Type	Write Only
Absolute Address	0xFFFF F144

Bit(s) Number	Name	Length	Comments
31 downto 1	IRQ Interrupt Fast Forcing Disable	31	0 = No effect. 1 = Disables Forcing on corresponding interrupt to FIQ.
0	FIQ Interrupt Fast Forcing Disable	1	0 = No effect. 1 = Disables Forcing on corresponding interrupt to FIQ.

AIC Fast Forcing Status Register

Register Name	AIC_FFDR
Access Type	Read Only
Reset Value	See Table 47 on page 134
Absolute Address	0xFFFF F148

Bit(s) Number	Name	Length	Comments
31 downto 1	IRQ Interrupt Fast Forcing Status	31	0 = Status Fast Forcing feature disabled on corresponding interrupt. 1= Status Fast Forcing feature enabled on corresponding interrupt.
0	FIQ Interrupt Fast Forcing Status	1	0 = Status Fast Forcing feature disabled on corresponding interrupt. 1= Status Fast Forcing feature enabled on corresponding interrupt.

PIO: Parallel I/O Controller

The Parallel Input/Output (PIO) peripheral features 32 programmable input/output lines, each of which may be dedicated as general purpose I/O (SW controlled) or be multiplexed with a signal generated by another embedded peripheral (peripheral linked), in order to optimize the use of available package pins in the overall system-on-chip design. After reset, all channels are connected to the PIO controller as I/O and are in input mode. I/O lines may be multiplexed with the input and output signals of another on-chip peripheral. The state of each bit of the PIO Status Register (PIO_PSR) determines whether the related channel is sw controlled or peripheral linked.

After reset, all channels are SW controlled and are in input mode.

The PIO Status Register (PIO_PSR) is controlled by writing logic 1 to the relevant bit of the PIO Enable (PIO_PER) or to the PIO Disable (PIO_PDR) Control Registers. Writing a 0 to a Control Register has no effect. When the sw controlled mode is selected, the peripheral input line is connected to zero. If a pin is a general-purpose parallel I/O pin (not multiplexed with a peripheral), PIO_PER and PIO_PDR has no effect and reading PIO_PSR returns a logic 1 for the bits corresponding to these pins.

PIO Output Enabling

Each channel is effectively bi-directional. A channel output may be disabled by suitably programming the Output Status Register (PIO_OSR) via the PIO_ODR (Output Disable) write-only Control Register. Conversely, the user can enable a channel as an output by writing logic 1 to the relevant bit of the PIO_OER (Output Enable) Control Register. Reading the Output Status Register (PIO_OSR) monitors the status of the various channels. Output enabling/disabling may only be controlled by user software when the channel has been configured as sw controlled (via PIO Status Register, PIO_PSR); when the channel is configured as peripheral linked, the pad output is enabled by a logic 0 state on the Peripheral Output Enable terminal.

PIO Pin Signal Levels

Each pin may be actively driven high or low, or its level may be determined by off-chip circuitry. The actual logic level on the pin is governed by the following conditions:

- If a channel is sw controlled and is defined as an output (see Section "PIO Output Enabling"), its pin level is governed by the state of the Output Data Status Register (PIO_ODSR) which may be controlled by writing a logic 1 to the Set Output Data Control Register (PIO_SODR) or to the Clear Output Data Control Register (PIO_CODR). The programmed value may be read from the Output Data Status Register (PIO_ODSR).
- If a channel is sw controlled, but is not enabled as an output, the external off-chip circuit, irrespective of the Output Data Status Register contents, will determine the pin level.
- If a channel is peripheral linked and the PIO Status Register (PIO_PSR) is programmed to enable the peripheral connection, its level will be defined by the peripheral.

In all cases, reading the corresponding bit of the Pin Data Status Register (PIO_PDSR) allows monitoring the actual level on a pin.

PIO Interrupt

Each I/O channel can be programmed to generate an interrupt when a voltage level change occurs on the related external pin. Each individual channel may have its interrupt generation enabled or disabled according to the logic condition of the related bit in the Interrupt Mask Register (PIO_IMR), which is controlled by writing a logic 1 to the relevant bit of the Interrupt Enable (PIO_IER) or to the Interrupt Disable (PIO_IDR) Control Register, in the same fashion as for the other Control Registers described above. When a change in level occurs, the corresponding bit in the Interrupt Status Register (PIO_ISR) register is set, whether the pin is used for a peripheral function or for processor addressed PIO, and whether the channel is working in input or in output mode. When the Interrupt Status Register (PIO_ISR) is read by user software, its contents are automatically cleared. If the corresponding interrupt is enabled in the Interrupt Mask Register (PIO_IMR), the relevant interrupt signal is generated. The interrupt signals from all 32 channels are gated together to generate a single PIO Interrupt signal (PIO_INT) that is connected to the AIC on AIC_IQR(17).

PIO Glitch Filter

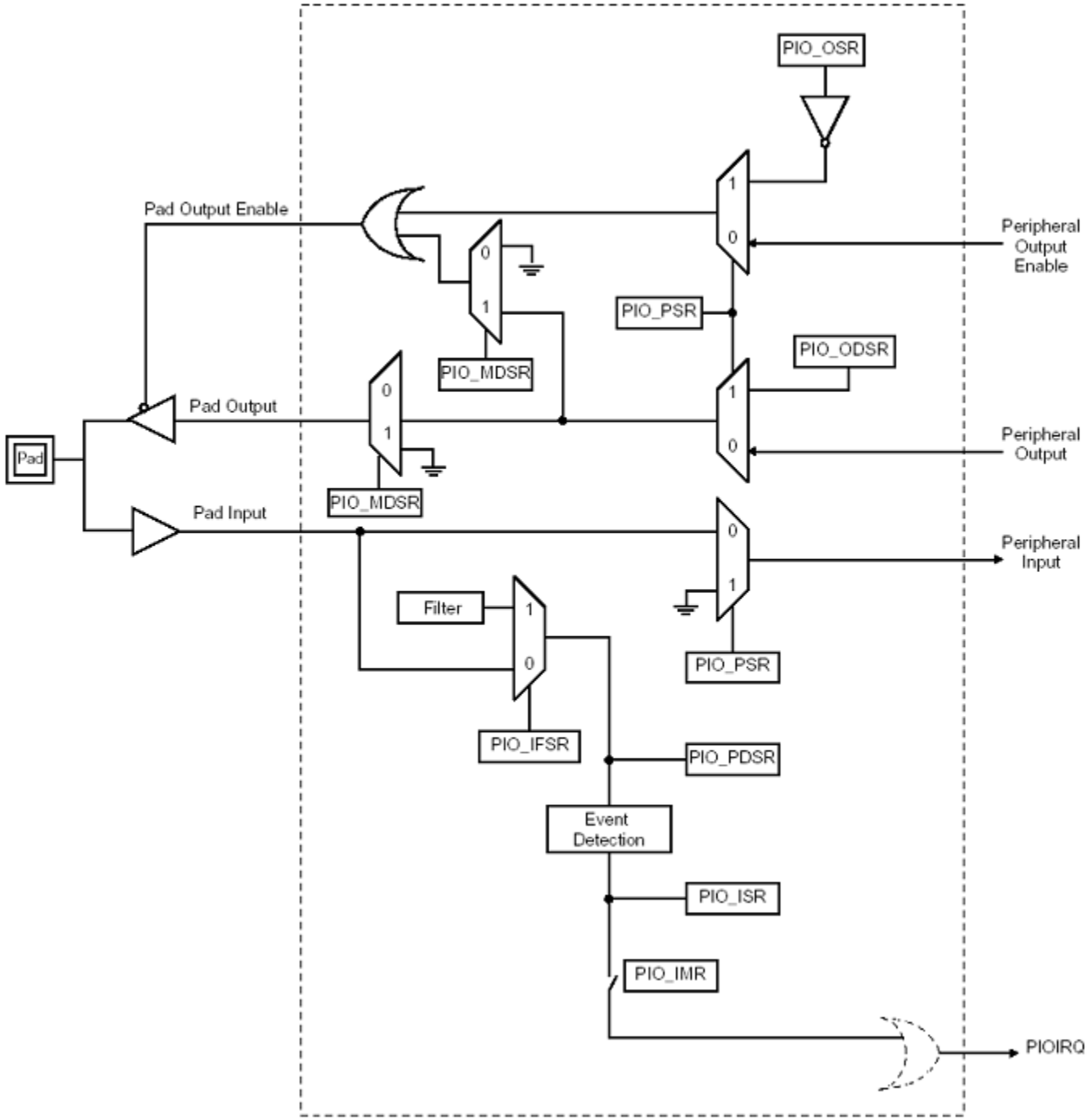
Optional glitch filtering is available for each channel. It is governed by the state of the Input Filter Status Register (PIO_IFSR) which is controlled by writing a logic 1 to the relevant bit of the Input Filter Enable (PIO_IFER) or to the Input Filter Disable (PIO_IFDR) Control Registers, in the same fashion as for the other control registers described above. Input glitch filtering may be selected, whether the pin is used for a peripheral function or for processor addressed parallel input. When the filter is activated, a glitch with duration of less than 1/2 clock cycle is automatically rejected, while a pulse with duration of 1 clock cycle or more is accepted. For pulse durations between 1/2 clock cycle and 1 clock cycle the pulse may or may not be taken into account, depending on the precise timing of its occurrence. Thus for a pulse to be visible it must exceed 1 clock cycle, whereas for a glitch to be reliably filtered out, its duration must not exceed 1/2 clock cycle.

PIO Multi Driver Option

Each I/O can be programmed for multi-driver option. This means that the I/O is configured as open drain (can only drive a low level) in order to support external drivers on the same pin. An external pull-up is necessary to guarantee a logic level of one when the pin is not being driven. Registers PIO_MDER (Multi-driver Enable) and PIO_MDDR (Multi-driver Disable) control this option. Multi-driver can be selected whether the PIO controller or the peripheral controls the I/O pin. PIO_MDSR (Multi-driver Status) indicates which pins are configured to support external drivers.

The PIO peripheral is fully-controllable via five sets of three 32-bit registers; pin data and interrupt source conditions are available to user software via two 32-bit registers. The following figure illustrates PIO functionality and the effect of register programming.

Figure 29. PIO Internal Architecture



Four of the 32 PIO lines work only internally; only 28 are available on D740 pads.

So there is a mismatch between the PIO PAD line number and the related PIO configuration register bit index number.

The following table summarizes the correspondence and the linked peripheral for each PIO line:

Table 48. PIO Mapping

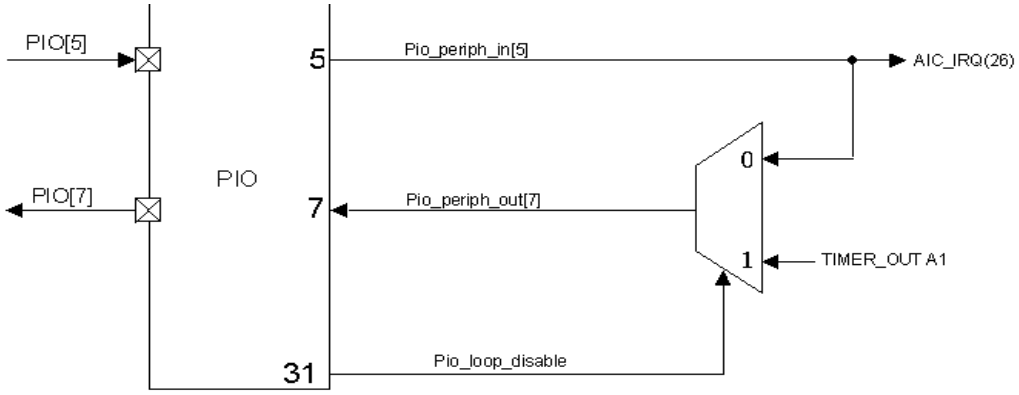
PIO Internal Line Number (Registers bit numbers)	PIO External Line Number (PAD line number)	Source/Destination when configured as Peripheral linked
bit 0	PIO[0]	TIMER input: TIMER_IN 0
bit 1	PIO[1]	TIMER input: TIMER_IN 1
bit 2	PIO[2]	TIMER input: TIMER_IN 2
bit 3	PIO[3]	WD output: Watch Dog Overflow
bit 4	PIO[4]	mAgic output: FPU_SIRQ(5)
bit 5	PIO[5]	AIC input: AIC_IRQ(26)
bit 6	PIO[6]	TIMER output: TIMER_OUT A0
bit 7	PIO[7]	TIMER output: TIMER_OUT A1
bit 8	PIO[8]	mAgic output: FPU_SIRQ(0)
bit 9	PIO[9]	mAgic output: FPU_SIRQ(1)
bit 10	PIO[10]	mAgic output: FPU_SIRQ(2)
bit 11	PIO[11]	mAgic output: FPU_SIRQ(3)
bit 12	PIO[12]	AIC input: AIC_IRQ(28)
bit 13	PIO[13]	AIC input: AIC_IRQ(29)
bit 14	PIO[14]	AIC input: AIC_IRQ(30) mAgic input: INTM(0)
bit 15	PIO[15]	AIC input: AIC_IRQ (31) mAgic input: INTM(1)
bit 16	externally not available: internally connected to mAgic input INTM(2)	TIMER output: TIMER_OUT B0
bit 17	externally not available: internally connected to mAgic input INTM(3)	TIMER output: TIMER_OUT B1
bit 18	externally not available: internally connected to mAgic input INTM(4)	TIMER output: TIMER_OUT B2
bit 19	PIO[16]	mAgic output: FPU_SIRQ(4)
bit 20	PIO[17]	mAgic output: FPU_SIRQ(6)
bit 21	PIO[18]	no peripheral linked
bit 22	PIO[19]	no peripheral linked
bit 23	PIO[20]	no peripheral linked
bit 24	PIO[21]	no peripheral linked
bit 25	PIO[22]	no peripheral linked
bit 26	PIO[23]	no peripheral linked

Table 48. PIO Mapping (Continued)

PIO Internal Line Number (Registers bit numbers)	PIO External Line Number (PAD line number)	Source/Destination when configured as Peripheral linked
bit 27	PIO[24]	no peripheral linked
bit 28	PIO[25]	no peripheral linked
bit 29	PIO[26]	no peripheral linked
bit 30	PIO[27]	AIC input: AIC_IRQ(24)
bit 31	externally not available: internally connected to Pio_loop_disable	no peripheral linked

For test purpose PIO[5] peripheral input can be routed in loop toward PIO[7] peripheral output through the control of PIO bit 31 internal line as shown in Figure 30.

Figure 30. PIO Test Loop Architecture



PIO Registers

Base Address 0xFFFF0000

Table 49. PIO Register Map

Offset	Register	Name	Access	Reset State
0x00	PIO Enable Register	PIO_PER	Write Only	-
0x04	PIO Disable Register	PIO_PDR	Write Only	-
0x08	PIO Status Register	PIO_PSR	Read Only	0xFFFF FFFF
0x0C	Reserved	-	-	-
0x10	Output Enable Register	PIO_OER	Write Only	-
0x14	Output Disable Register	PIO_ODR	Write Only	-
0x18	Output Status Register	PIO_OSR	Read Only	0
0x1C	Reserved	-	-	-
0x20	Input Filter Enable Register	PIO_IFER	Write Only	-
0x24	Input Filter Disable Register	PIO_IFDR	Write Only	-
0x28	Input Filter Status Register	PIO_IFSR	Read Only	0
0x2C	Reserved	-	-	-
0x30	Set Output Data Register	PIO_SODR	Write Only	-
0x34	Clear Output Data Register	PIO_CODR	Write Only	-
0x38	Output Data Status Register	PIO_ODSR	Read Only	0
0x3C	Pin Data Status Register	PIO_PDSR	Read Only	(1)
0x40	Interrupt Enable Register	PIO_IER	Write Only	-
0x44	Interrupt Disable Register	PIO_IDR	Write Only	-
0x48	Interrupt Mask Register	PIO_IMR	Read Only	0
0x4C	Interrupt Status Register	PIO_ISR	Read Only	(2)
0x50	Multi Driver Enable Register	PIO_MDER	Write Only	-
0x54	Multi Driver Disable Register	PIO_MDDR	Write Only	-
0x58	Multi Driver Status Register	PIO_MDSR	Read Only	0

1. The reset value of this register depends on the level of the external pins at reset.
2. This register is cleared at reset. However, the first read of the register can give a value not equal to zero if any changes have occurred on any pins between the reset and the read.

PIO Enable Register

Register Name	PIO_PER
Access Type	Write Only
Absolute Address	0xFFFF0000

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = No effect. 1 = Enables PIO sw control for the corresponding pin (disables peripheral link to the pin).

This register is used to enable individual pins to be controlled by the PIO Controller instead of the associated peripheral. When the PIO is enabled, the associated peripheral input (if any) is held at logic zero.

PIO Disable Register

Register Name	PIO_PDR
Access Type	Write Only
Absolute Address	0xFFFF0004

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = No effect. 1 = Disables PIO sw control for the corresponding pin (enables peripheral link to the pin).

This register is used to disable PIO control of individual pins. When the PIO control is disabled, the normal peripheral function is enabled on the corresponding pin.

PIO Status Register

Register Name	PIO_PSR
Access Type	Read Only
Reset Value	See Table 49 on page 148
Absolute Address	0xFFFF0008

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = sw control is disabled on the corresponding line (peripheral link is enabled). 1 = sw control is enabled on the corresponding line (peripheral link is disabled).

This register indicates which pins are enabled for PIO control. This register is updated when PIO lines are enabled or disabled.

PIO Output Enable Register

Register Name	PIO_OER
Access Type	Write Only
Absolute Address	0xFFFF0010

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = No Effect. 1 = Enables the PIO output on the corresponding pin.

This register is used to enable PIO output drivers. If the pin is driven by a peripheral, this has no effect on the pin, but the information is stored.

PIO Output Disable Register

Register Name	PIO_ODR
Access Type	Write Only
Absolute Address	0xFFFF00014

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = No effect. 1 = Disables the PIO output on the corresponding pin.

This register is used to disable PIO output drivers. If the pin is driven by the peripheral, this has no effect on the pin, but the information is stored.

PIO Output Status Register

Register Name	PIO_OSR
Access Type	Read Only
Reset Value	See Table 49 on page 148
Absolute Address	0xFFFF00018

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = The corresponding PIO is input on this line. 1 = The corresponding PIO is output on this line.

This register shows the PIO pin control (output enable) status that is programmed through PIO_OER and PIO_ODR. The defined value is effective only if the pin is controlled by the PIO.

PIO Input Filter Enable Register

Register Name	PIO_IFER
Access Type	Write Only
Absolute Address	0xFFFF00020

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = No effect. 1 = Enables the glitch filter on the corresponding pin.

This register is used to enable input glitch filters. It affects the pin whether or not the PIO is enabled.

PIO Input Filter Disable Register

Register Name	PIO_IFDR
Access Type	Write Only
Absolute Address	0xFFFF00024

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = No effect. 1 = Disables the glitch filter on the corresponding pin.

This register is used to disable input glitch filters. It affects the pin whether or not the PIO is enabled.

PIO Input Filter Status Register

Register Name	PIO_IFSR
Access Type	Read Only
Reset Value	See Table 49 on page 148
Absolute Address	0xFFFF00028

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = Filter is not selected on the corresponding input. 1 = Filter is selected on the corresponding input (peripheral and PIO).

This register indicates which pins have glitch filters selected. It is updated when PIO outputs are enabled or disabled by writing to PIO_IFER or PIO_IFDR.

PIO Set Output Data Register

Register Name	PIO_SODR
Access Type	Write Only
Absolute Address	0xFFFF00030

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = No effect. 1 = PIO output data on the corresponding pin is set.

This register is used to set PIO output data. It affects the pin only if the corresponding PIO output is enabled and if the pin is controlled by the PIO. Otherwise, the information is stored.

PIO Clear Output Data Register

Register Name	PIO_CODR
Access Type	Write Only
Absolute Address	0xFFFF00034

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = No effect. 1 = PIO output data on the corresponding pin is cleared.

This register is used to clear PIO output data. It affects the pin only if the corresponding PIO output is enabled and if the pin is controlled by the PIO. Otherwise, the information is stored.

PIO Output Data Status Register

Register Name	PIO_ODSR
Access Type	Read Only
Reset Value	See Table 49 on page 148
Absolute Address	0xFFFF00038

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = The output data for the corresponding line is programmed to 0. 1 = The output data for the corresponding line is programmed to 1.

This register shows the output data status, which is programmed in PIO_SODR or PIO_CODR. The defined value is effective only if the pin is controlled by the PIO Controller and only if the pin is defined as an output.

PIO Pin Data Status Register

Register Name	PIO_PDSR
Access Type	Read Only
Reset Value	See Table 49 on page 148
Absolute Address	0xFFFF0003C

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = The corresponding pin is at logic 0. 1 = The corresponding pin is at logic 1.

This register shows the state of the physical pin of the chip. The pin values are always valid regardless of whether the pins are enabled as PIO, peripheral, input or output.

PIO Interrupt Enable Register

Register Name	PIO_IER
Access Type	Write Only
Absolute Address	0xFFFF00040

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = No effect. 1 = Enables an interrupt when a change of logic level is detected on the corresponding pin.

This register is used to enable PIO interrupts on the corresponding pin. It has effect whether PIO is enabled or not.

PIO Interrupt Disable Register

Register Name	PIO_IDR
Access Type	Write Only
Absolute Address	0xFFFF00044

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = No effect. 1 = Disables the interrupt on the corresponding pin. Logic level changes are still detected.

This register is used to disable PIO interrupts on the corresponding pin. It has effect whether PIO is enabled or not.

PIO Interrupt Mask Register

Register Name	PIO_IMR
Access Type	Read Only
Reset Value	See Table 49 on page 148
Absolute Address	0xFFFF00048

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = Interrupt is not enabled on the corresponding input pin. 1 = Interrupt is enabled on the corresponding input pin.

This register shows which pins have interrupts enabled. It is updated when interrupts are enabled or disabled by writing to PIO_IER or PIO_IDR.

PIO Interrupt Status Register

Register Name	PIO_ISR
Access Type	Read Only
Reset Value	See Table 49 on page 148
Absolute Address	0xFFFF0004C

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = No input change has been detected on the corresponding pin since the register was last read. 1 = At least one input change has been detected on the corresponding pin since the register was last read.

This register indicates for each pin when an input logic value change has been detected (rising or falling edge). This is valid whether the PIO is selected for the pin or not.

PIO Multidrive Enable Register

Register Name	PIO_MDER
Access Type	Write Only
Absolute Address	0xFFFF00050

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = No effect. 1 = Enables multi-drive option on the corresponding pin.

This register is used to enable PIO output drivers to be configured as open drain to support external drivers on the same pin.

PIO Multidrive Disable Register

Register Name	PIO_MDDR
Access Type	Write Only
Absolute Address	0xFFFF00054

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = No effect. 1 = Disables multidrive option on the corresponding pin.

PIO Multidrive Status Register

Register Name	PIO_MDSR
Access Type	Read Only
Reset Value	See Table 49 on page 148
Absolute Address	0xFFFF00058

Bit(s) Number	Name	Length	Comments
31 downto 0	P31 downto P0	32	0 = PIO is not configured as an open drain. 1 = PIO is configured as an open drain.

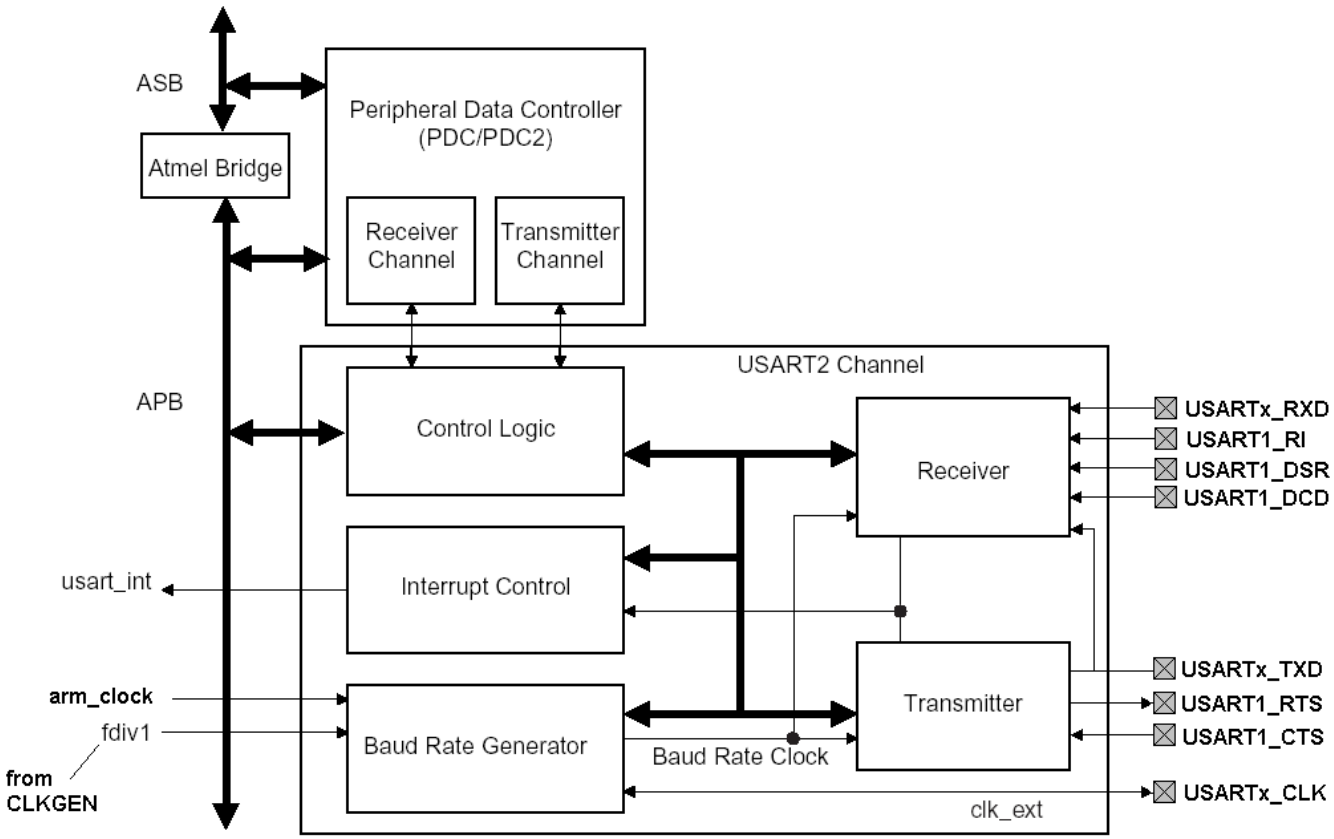
USART: Universal Sync/Asynchronous Rx/Tx

ARM System provides two (USART0 and USART1), full-duplex, universal synchronous/asynchronous receiver/transmitters that interface with the APB for configuration and are connected to the PDC for data transfers.

Two dedicated Peripheral Data Controller (PDC) channels are connected to each USART. They allow a DMA transfer of data either sent or received by the USARTs.

One of the channels is dedicated to the receiver. The other one is dedicated to the transmitter. The generation of interrupts is controlled in the status register by asserting the corresponding interrupt line. The four PDC configuration registers for the two USART are mapped in the USART memory space. These registers are physically located in the PDC and are accessed when selecting the USART. For more details concerning the PDC usage, see Section “PDC: Peripheral Data Controller” on page 116.

Figure 31. USART Block Diagram



The USART1 can be programmed to operate in five different modes:

- normal
- modem
- handshaking
- RS485
- ISO7816 (T = 0 and T = 1).

The USART0 can be programmed to operate in only two different modes:

- normal
- ISO7816 (T = 0 and T = 1)

Each USART can generate an interrupt toward the AIC:

- UASRT0 ➔ AIC_IRQ(4)
- USART1 ➔ AIC_IRQ(5)

USART Baud Rate Generator

For each USARTs an embedded Baud Rate Generator provides the bit period clock to both the receiver and the transmitter.

The Baud Rate Generator can select between external and internal clock sources.

The external clock sources (for synchronous mode only) are:

- UASRT0 ➔ USART0_SCK
- USART1 ➔ USART1_SCK

Note: In all cases, if an external clock is used, the duration of each of its levels must be longer than the arm_clock period. The external clock frequency must be at least 4.5 times lower than the arm_clock.

The internal clock sources are:

- UASRT0 ➔ CLK_ARM_DIV6 (from CLKGEN) or arm_clock
- USART1 ➔ CLK_ARM_DIV7 (from CLKGEN) or arm_clock

For the programmable Baud Rate is:

- Asynchronous Mode ➔ BR = Source Clock / (N × CD)
- Synchronous Mode ➔ BR = Source Clock / CD

with CD = 2 to 65535 (if CD = 0 Clock is disables; if CD = 1 the division is bypassed)

and N = 16 if oversampling is enabled or N = 8 if oversampling is disabled.

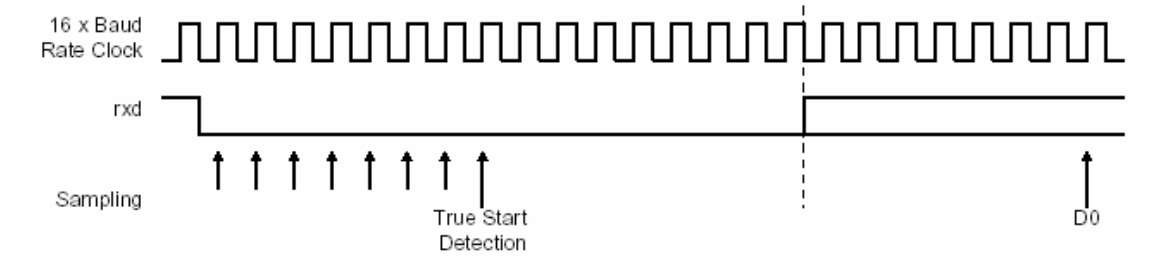
In synchronous mode with external clock selected, the clock is provided directly by the signal on the USARTx_SCK. No division is active.

When the USART is programmed to operate in IS07816 mode, the source clock is divided by the value (FI_DI_RATIO) written in US_FIDI. If US_FIDI is set to 0, the Baud Rate Clock is disabled.

USART Asynchronous Receiver

In asynchronous mode, the USART detects the start of a received character by sampling the USARTx_RXD signal until it detects a valid start bit. A low level (space) is interpreted as a valid start bit if it is detected for more than 7 cycles of the sampling clock, which is 16 times the baud rate. Hence, a space that is longer than 7/16 of the bit period is detected as a valid start bit. A space which is 7/16 of a bit period or shorter is ignored and the receiver continues to wait for a valid start bit.

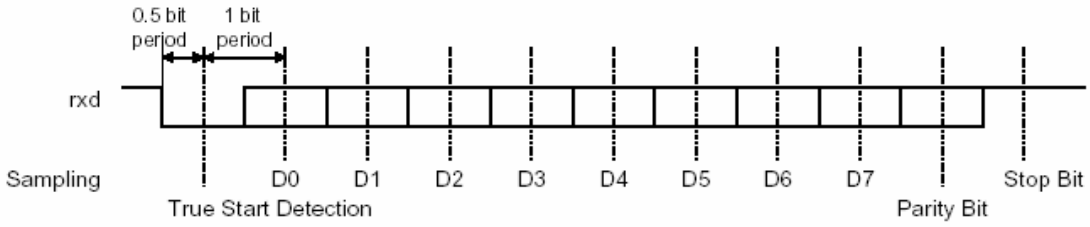
Figure 32. USART Asynchronous Mode: Start Bit Detection



When a valid start bit has been detected, the receiver samples the USARTx_RXD at the theoretical mid-point of each bit. It is assumed that each bit lasts 16 cycles of the sampling clock (1-bit period) so the sampling point is eight cycles (0.5 bit period) after the start of the bit.

The first sampling point is therefore 24 cycles (1.5 bit periods) after the falling edge of the start bit was detected. Each subsequent bit is sampled 16 cycles (1-bit period) after the previous one.

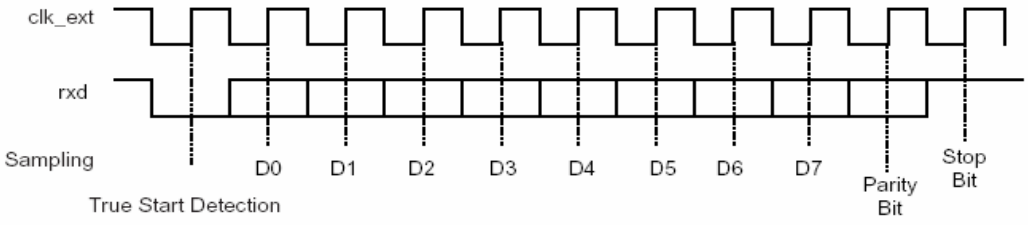
Figure 33. USART Asynchronous Mode: Character Reception



USART Synchronous Receiver

When configured for synchronous operation, the receiver samples the USARTx_RXD signal on each rising edge of the Baud Rate Clock. If a low level is detected, it is considered a start. The data bits, parity bit and stop bit are sampled and the receiver waits for the next start bit.

Figure 34. USART Synchronous Mode: Character Reception



When a complete character is received, it is transferred to the US_RHR and the RXRDY status bit in US_CSR is set. If US_RHR has not been read since the last transfer, the OVRE status bit in US_CSR is set.

USART Parity Check

Each time a character is received, the receiver calculates the parity of the received data bits, in accordance with the field PAR in US_MR. It then compares the result with the received parity bit. If different, the parity error bit PARE in US_CSR is set.

USART Frame Check

If a character is received with a stop bit at low level and with at least one data bit at high level, a framing error is generated. This sets FRAME in US_CSR.

USART Time-Out Check

The Time-Out function allows an idle condition on the rxd line to be detected. The maximum delay for which the USART should wait for a new character to arrive while the rxd line is inactive (high level) is programmed in US_RTOR (Receiver Time-out). When this register is set to 0, no time-out is detected. Otherwise, the receiver waits for a first character and then initializes a counter, which is decremented at each bit period and reloaded at each byte reception. When the counter reaches 0, the TIMEOUT bit in US_CSR is set. The user can restart the wait for a first character with the STTTO (Start Time-out) bit in US_CR.

The time-out duration is: D = US_RTOR Value x Bit Period.

USART Generating CLK_TXD

In synchronous mode, clk_txd is the data clock. In asynchronous mode the clock is defined below:

clk_txd = 16 x Baud Rate Clock

or

clk_txd = 8 x Baud Rate Clock

depending on the OVER bit in US_MR register,

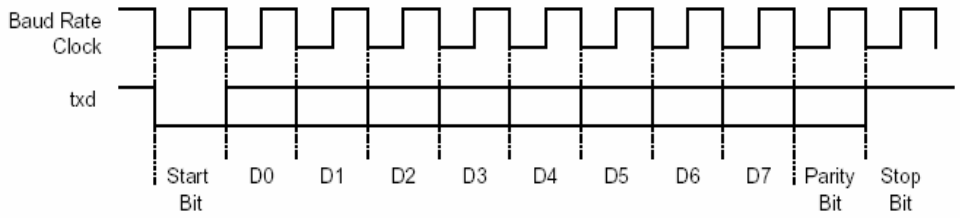
or

clk_txd = FI_DI_RATIO x Baud Rate Clock in ISO7816 mode.

USART Transmitter

The transmitter has the same behavior in both synchronous and asynchronous operating modes. The start bit, data bits, parity bit and stop bits are serially shifted, lowest significant bit first, on the falling edge of the serial clock.

Figure 35. USART A/Synchronous Mode: Character Transmission



The number of data bits, the parity bit, the number of stop bits are all configurable.

When a character is written to US_THR (Transmit Holding), it is transferred to the Shift Register as soon as it is empty. When the transfer occurs, the TXRDY bit in US_CSR is set until a new character is written to US_THR. If Transmit Shift Register and US_THR are both empty, the TXEMPTY bit in US_CSR is set.

USART Time-guard

The time-guard function allows the transmitter to insert an idle state on the txd line between two characters. The duration of the idle state is programmed in US_TTGR (Transmitter Time-guard). When this register is set to zero, no time-guard is generated. Otherwise, the transmitter holds a high level on txd after each transmitted byte during the number of bit periods programmed in US_TTGR.

USART Multi Drop Mode

When the field PAR in US_MR equals 11X (binary value), the USART is configured to run in multi-drop mode. In this case, the parity error bit PARE in US_CSR is set when data is detected with a parity bit set to identify an address byte. PARE is cleared with the Reset Status Bits command (RSTSTA) in US_CR. If the parity bit is detected low, identifying a data byte, PARE is not set.

The transmitter sends an address byte (parity bit set) when a Send Address command (SENDA) is written to US_CR. In this case, the next byte written to US_THR will be transmitted as an address. After this, any byte transmitted will have the parity bit cleared.

USART Transmit Break

The transmitter generates a break condition on the txd line when STTBRK is set in US_CR (Control Register). In this case, the character present in the Transmit Shift Register is completed before the line is held low. To cancel a break condition on the txd line, the STPBRK command in US_CR must be set. The USART completes minimum break duration of one character length. The txd line then returns to high level (idle state) for at least 12-bit periods, or the value of the Time-guard register if it is greater than 12, to ensure that the end of the break is correctly detected. Then the transmitter resumes normal operation. The BREAK is managed like a character:

- The STTBRK and the STPBRK commands are performed only if the transmitter is ready (bit TXRDY = 1 in US_CSR).
- The STTBRK command blocks the transmitter holding register (bit TXRDY is cleared in US_CSR) until the break has started.
- A break is started when the Shift Register is empty (any previous character is fully transmitted). US_CSR.TXEMPTY is cleared. The break blocks the transmitter shift register until it is completed (high level for at least 12-bit periods after the STPBRK command is requested).
- In order to avoid unpredictable states:
- STTBRK and STPBRK commands must not be requested at the same time.
- Once an STTBRK command is requested, further STTBRK commands are ignored until the BREAK is ended (high level for at least 12-bit periods).
- All STPBRK commands requested without a previous STTBRK command are ignored.
- A byte written into the Transmit Holding Register while a break is pending but not started (bit TXRDY = 0 in US_CSR) is ignored.
- It i s not permitted to write new data in the Transmit Holding Register while a break is in progress (STPBRK has not been requested), even though TXRDY = 1 in US_CSR.
- A new STTBRK command must not be issued until an existing break has ended (TXEMPTY=1 in US_CSR).

The standard break transmission sequence is:

1. Wait for the transmitter ready (US_CSR.TXRDY = 1)
2. Send the STTBRK command (write 0x0200 to US_CR)
3. Wait for the transmitter ready (bit TXRDY = 1 in US_CSR)
4. Send the STPBRK command (write 0x0400 to US_CR)
5. The next byte can then be sent.
6. Wait for the transmitter ready (bit TXRDY = 1 in US_CSR)
7. Send the next byte (write byte to US_THR)

Each of these steps can be scheduled by using the interrupt if the bit TXRDY in US_IMR is set. For character transmission, the USART channel must be enabled before sending a break.

USART Receive Break

The receiver detects a break condition when all data, parity and stop bits are low. When the low stop bit is detected, the receiver asserts the RXBRK bit in US_CSR. An end of receive break is detected by a high level for at least 2/16 of a bit period in asynchronous operating mode or at least one sample in synchronous operating mode. RXBRK is also asserted when an end-of-break is detected. Both the beginning and the end of a break can be detected by an interrupt if the bit US_IMR.RXBRK is set.

USART Peripheral Data Controller Channels

Each USART channel is closely connected to a corresponding Peripheral Data Controller channel. One is dedicated to the receiver. The other is dedicated to the transmitter.

The PDC channel is programmed using US_TPR (Transmit Pointer) and US_TCR (Transmit Counter) for the transmitter and US_RPR (Receive Pointer) and US_RCR (Receive Counter) for the receiver. The status of the PDC is given in US_CSR by the ENDTX bit for the transmitter and by the ENDRX bit for the receiver.

The PDC has two parameter buffers. When the first buffer is full, the PDC buffers may fill, thereby permitting continuous data transfer. The pointer registers (US_TPR and US_RPR) are used to store the address of transmit or receive buffers. The counter registers (US_TCR and US_RCR) are used to store the size of these buffers. The RXRDY bit triggers the receiver data transfer and the transmitter data transfer is triggered by the TXRDY.

When a transfer is performed, the counter is decremented and the pointer is incremented. When the counter reaches 0, the status bit is set (ENDRX for the receiver, ENDTX for the transmitter in US_CSR) and can be programmed to generate an interrupt. While the counter is at 0, the status bit is asserted and transfers are disabled.

USART Interrupts

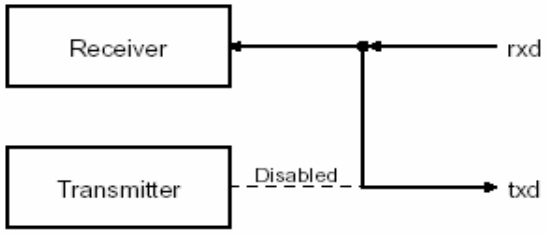
Each USART US_CSR status bits control the generation of the related interrupt by asserting the two USART interrupt lines toward the AIC. US_IMR (Interrupt Mask Register) indicates the status of the corresponding bits. When a bit is set in US_CSR and the same bit is set in US_IMR, the interrupt line is asserted.

USART Test Configurations

The USARTs can be programmed to operate in three different test modes.

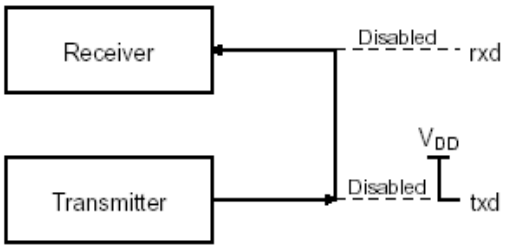
Automatic echo mode allows bit-by-bit retransmission. When a bit is received on the rxd line, it is sent to the txd line. Programming the transmitter has no effect.

Figure 36. USART Automatic Echo Test Mode



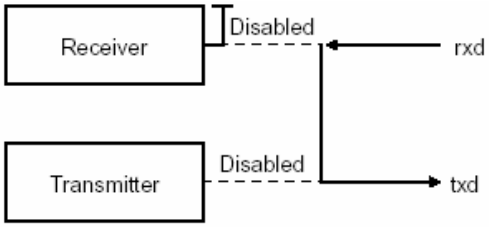
Local loopback mode allows the transmitted characters to be received. txd and rxd pins are not used and the output of the transmitter is internally connected to the input of the receiver. The rxd pin level has no effect and the txd pin is held high, as in idle state.

Figure 37. USART Automatic Local Loopback Mode



Remote loopback mode directly connects the rxd pin to the txd pin. The Transmitter and the Receiver are disabled and have no effect. This mode allows bit-by-bit retransmission.

Figure 38. USART Automatic Remote Loopback Mode



USART ISO7816 Mode

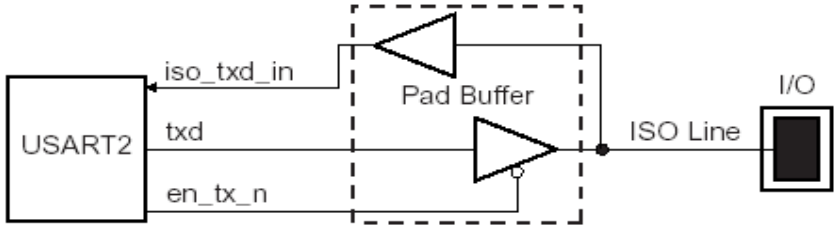
The USARTs handle all specific requirements defined in ISO7816 T = 0 and T = 1 protocol types. To select ISO7816 mode, the protocol and master or slave field USART_MODE in the Mode Register (US_MR) has to be set correctly depending on the desired configuration.

Note: The channel test configuration is not available in ISO mode.

The IS07816 USART is configured and controlled via several registers:

- USART Mode Register (US_MR)
- USART Control Register (US_CR)
- USART Interrupt Enable Register (US_IER)
- USART Interrupt Disable Register (US_IDR)
- USART Interrupt Mask Register (US_IMR)
- USART Channel Status Register (US_CSR)
- USART Fi_Di_Ratio Register (US_FIDI)
- USART Nb Errors Register (US_NER)
- USART Receiver Time Out Register (US_RTOR)

Figure 39. USART ISO Mode Connection



The bit rate is determined by the following formula:

$$B = D / F \times f$$

where:

B = bit rate

D = bit-rate adjustment factor

F = clock frequency division factor

f = iso clock frequency (Hz) as defined with iso clock

D is a 4-bit encoded binary value (DI) with the corresponding decimal index (Di) as shown in Table 50. F is a 4-bit encoded binary value (FI) with the corresponding decimal index (Fi) as shown in Table 51.

Table 50. USART Binary and Decimal Values for D

DI_{bits}	0001	0010	0011	0100	0101	0110	1000	1001
$DI_{decimal}$	1	2	4	8	16	32	12	20

Table 51. USART Binary and Decimal Values for F

FI_{bits}	0000	0001	0010	0011	0100	0101	0110	1001	1010	1011	1100	1101
$FI_{decimal}$	372	372	558	744	1116	1488	1860	512	768	1024	1536	2048

The Baud Rate Clock used in mode ISO7816 is configured via the field FI_DI_RATIO in US_FIDI. This is a 11-bit divider, hence the divider must be programmed between 0 and 2047. The FI_DI_RATIO value loaded in US_FIDI will define the bit rate. The FI/DI value shown in Table 52 is loaded in the US_FIDI register. For example, if an FI/DI of 372 is desired, 174 H is programmed into the US_FIDI register. Reset value for FI_DI_RATIO is 174.

Table 52. USART Fi/Di Ratio Possible Values

FI/DI	372	558	774	1116	1488	1806	512	768	1024	1536	2048
1	372	558	744	1116	1488	1860	512	768	1024	1536	2048
2	186	279	372	558	744	930	256	384	512	768	1024

Table 52. USART Fi/Di Ratio Possible Values

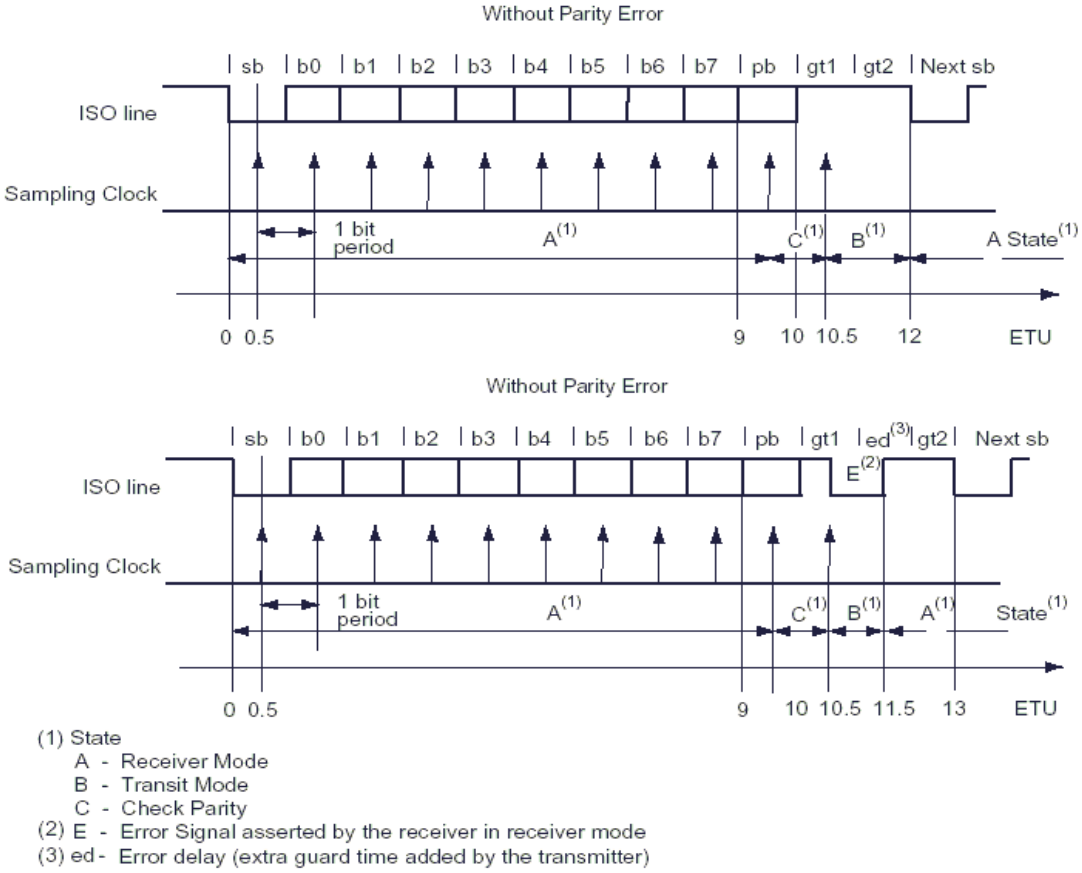
4	93	139.5	186	279	372	465	128	192	256	384	512
8	46.5	69.75	93	139.5	186	232.5	64	96	128	192	256
16	23.25	34.87	46.5	69.75	93	116.2	32	48	64	96	128
32	11.62	17.43	23.25	34.87	46.5	58.13	16	24	32	48	64
12	31	46.5	62	93	124	155	42.66	64	85.33	128	170.6
20	18.6	27.9	37.2	55.8	74.4	93	25.6	38.4	51.2	76.8	102.4

Bit Time ETU. The relation between the ISO clock and the baud rate clock divisor factor FI_DI_RATIO determines the bit duration. As a result, in ISO7816 mode, there are FI_DI_RATIO clock cycles in one elementary time unit (ETU).

The following table summarizes the ISO protocol (T=1 and T=0):

Protocol T=1	
Receiver Side	Upon detection of a start bit, the following data byte and the parity bit are shifted in the Receiver Holding Register (US_RHR) when the shift is completed and the parity is checked. If a parity error is detected, the PARE bit is set in US_CSR. In this protocol there is only one stop bit.
Transmitter Side	The data is loaded in the Transmit Holding Register and parity is calculated. The data and the parity bit are shifted out following the start bit.
Protocol T=0	
Receiver Side	Upon detection of a start bit, the following data byte and the parity bit are shifted in the Receiver Holding Register (US_RHR) when the shift is completed and the parity is checked. If a parity error is detected, a low error signal is sent for one Elementary Time Unit (ETU), 10.5 ETUs after the start bit. This error signal is sent from the receiver to the transmitter. The shift register value is not loaded in the Receiver Holding Register.
Transmitter Side	The data is loaded in the Transmit Holding Register and parity is calculated. The data and the parity bit are shifted out following the start bit. The TxD line returns to high impedance 10 ETUs after the start bit. In this protocol there are two stop bits.

Figure 40. USART T = 0 Protocol without and with Parity Error



(1) State
A - Receiver Mode
B - Transit Mode
C - Check Parity
(2) E - Error Signal asserted by the receiver in receiver mode
(3) ed - Error delay (extra guard time added by the transmitter)

Additional Features in Protocol T = 0	
Error Number	If errors occurred during a transfer, it is possible to know the total amount of errors by reading the field NB_ERRORS in US_NER. This is a read only register reset by a read action. Up to 255 errors can be recorded.
Non Acknowledge Inhibited	In some cases it can be important to inhibit an error. This can be achieved by setting the INACK bit in US_MR. This means that even if an error is detected by the receiver, it will not send an error signal (NACK). If INACK is reset, on each error, the NACK bit in US_SR will be set. It can be reset by using the RSTNACK command in US_CR. Moreover, when INACK is reset and the USART2 receives a byte with a parity error, this value is available in the US_RHR register but the US_RXRDY bit will not be set.

Additional Features in Protocol T = 0 (Continued)	
Character Repetition	A character repetition can be executed if the MAX_ITERATION field in US_MR is different from 0. MAX_ITERATION is a 3-bit field configurable with a value between 0 and 7. This implies that a character can be repeated up to seven times. (Initial send + seven repetitions). If MAX_ITERATION is equal to zero, the protocol T = 0 transfer format explained previously is still valid. If MAX_ITERATION is different from zero and no parity error has been detected, the protocol T = 0 transfer format explained previously is still valid. If MAX_ITERATION is different from zero and a parity error has been detected, the transmitter will again send the value. If a parity error is still detected, the value is sent as many times as the value loaded in the MAX_ITERATION field. If the number of repetitions of the value reaches the value loaded in the MAX_ITERATION field, the ITERATION (US_CSR) flag is set. If at some stage during the repetition schematic, no error parity is detected by the far-end receiver, the repetition is stopped. To reset the ITERATION (US_CSR) flag, the RSIT bit must be set in the USART Control Register (US_CR).
Disable Successive NACK	In some cases it may be useful to limit the number of successive NACKs sent on the line. This can be achieved by setting the DSNACK bit in US_MR. If this bit is set, successive parity errors are counted up to the value specified in the MAX_ITERATION field. As soon as this value is reached, no additional NACK is sent on the ISO line and the flag ITERATION is asserted. If it is not set, NACK is sent on the ISO line as soon as a parity error occurs in the received character (unless INACK is set). Bit Order If set, MSB is sent first, otherwise LSB.

USART HW Handshake mode This mode is applicable only for USART1.

In this mode, an additional output USART1_RTS (Request to Send) is generated. If an active RX_DMA_END is received, the receiver will force the RTS pin to a high level on the next start bit reception.

An additional input USART1_CTS (Clear to Send) is activated in this mode. If a high level signal is received on this pin, the transmitter will be disabled after finishing the current character transmission.

Figure 41. USART HW Handshake Connection

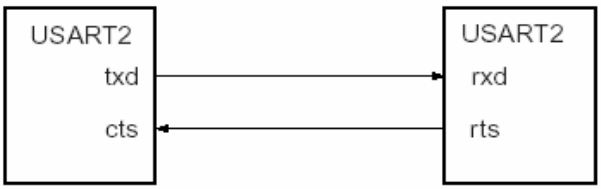
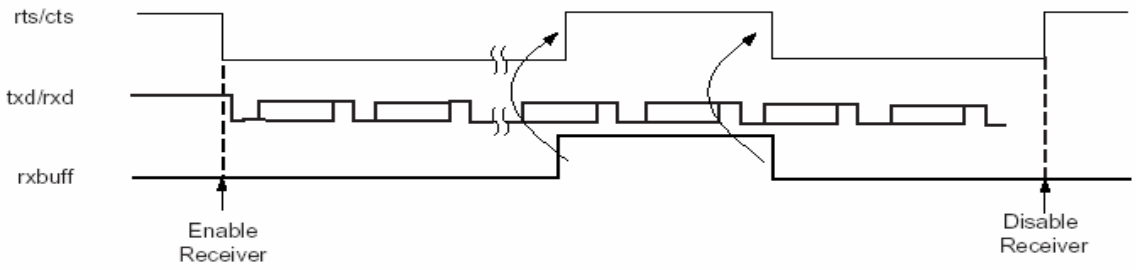


Figure 42. USART RTS/CTS Behavior in Hardware Handshaking Mode

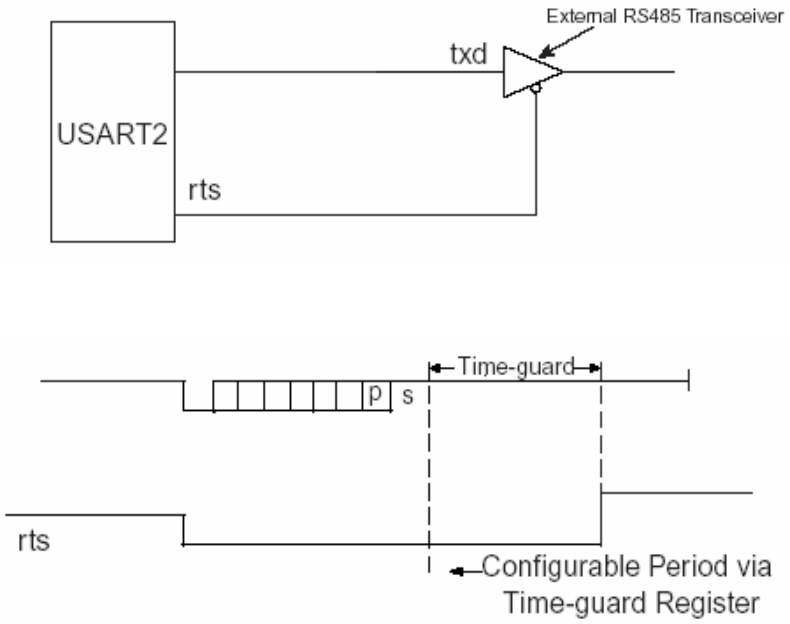


USART RS485 Mode

This mode is applicable only for USART1.

In this mode, an additional output USART1_RTS (Request to Send) is generated. The behavior of USART1_RTS will follow the behavior of TXEMPTY.

Figure 43. USART RS485 Connection



USART Modem Mode

This mode is applicable only for USART1.

In this mode, two additional outputs are generated:

1. Request to Send: USART1_RTS
2. Data Terminal Ready: USART1_DTR

If the RTS transmitter is enabled, the USART1_RTS pin will be forced to its active level (low). The behavior of the USART1_DTR pin is controlled via 2 bits (DTRDIS and DTREN) located in the Control Register (US_CR). If DTREN is set, the DTR pin will be forced to its active level (low). If DTRDIS is set, the DTR pin will be forced to its inactive level (high).

The Modem mode also activates four additional inputs:

1. Ring Indicator: USART1_RI
2. Data Set Ready USART1_DSR
3. Data Carrier Detect USART1_DCD
4. Clear to Send USART1_CTS

If an input change is detected on the USART1_RI pin, the RIIC flag in the Channel Status Register (US_CSR) is raised. To reset this flag, the US_CSR register must be read.

Furthermore, if an input change is received on the USART1_DSR pin, the flag DSRIC in the Channel Status Register (US_CSR) is raised. To reset this flag, the US_CSR register must be read.

In addition, if an input change is received on the USART1_DCD pin, the DCDIC flag in the Channel Status Register (US_CSR) is raised. To reset this flag, the US_CSR register must be read.

Moreover, if an active signal is detected on the USART1_CTS pin, the transmitter will be disabled after finishing the current character transmission. If an input change is received on this pin, the CTSIC flag in the Channel Status Register (US_CSR) is raised. To reset this flag, the US_CSR register must be read.

USART Registers

Base Address 0xFFFC 0000 for USART0 and
0xFFFC 4000 for USART1

Table 53. USART Register Map

Offset	Register	Name	Access	Reset State
0x00	Control Register	US_CR	Write Only	-
0x04	Mode Register	US_MR	Read/Write	0
0x08	Interrupt Enable Register	US_IER	Write Only	-
0x0C	Interrupt Disable Register	US_IDR	Write Only	-
0x10	Interrupt Mask Register	US_IMR	Read Only	0
0x14	Channel Status Register	US_CSR	Read Only	-
0x18	Receiver Holding Register	US_RHR	Read Only	0
0x1C	Transmitter Holding Register	US_THR	Write Only	-
0x20	Baud Rate Generator Register	US_BRGR	Read/Write	0
0x24	Receiver Time-out Register	US_RTOR	Read/Write	0
0x28	Transmitter Time-guard Register	US_TTGR	Read/Write	0
0x2C	Reserved	-	-	-

Table 53. USART Register Map (Continued)

Offset	Register	Name	Access	Reset State
0x30	Reserved	-	-	-
0x34	Reserved	-	-	-
0x38	Reserved	-	-	-
0x3C	Reserved	-	-	-
0x40	FI_DI_Ratio Register	US_FIDI	Read/Write	0
0x44	Nb Errors Register	US_NER	Read Only	0

USART Control Register

Register Name	US_CR
Access Type	Write Only
Absolute Address	0xFFFC 0000 for USART0 and 0xFFFC 4000 for USART1

Bit(s) Number	Name	Length	Comments
31 downto 20	UNUSED	12	
19	RTSDIS: Request to Send Disable	1	0 = No effect. 1 = The RTS pin is forced to 1.
18	RTSEN:Request to Send Enable	1	0 = No effect. 1 = The RTS pin is forced to 0.
17	DTRDIS: Data Terminal Ready Disable	1	0 = No effect. 1 = The DTR pin is forced to 1.
16	DTREN: Data Terminal Ready Enable	1	0 = No effect. 1 = The DTR pin is forced to 0.
15	RETTO: Rearm Time-out	1	0 = No effect. 1 = Restarts a time-out.
14	RST_NACK: Reset Non-acknowledge	1	0 = No effect. 1 = Resets the Nack status bit.
13	RSTIT: Reset Iterations	1	Note: This bit only takes effect in ISO7816 Mode. 0 = No effect. 1 = Resets the status bit Iteration.
12	SENDA: Send Address	1	0 = No effect. 1 = In multidrop Mode only, the next character written to the US_THR is sent with the address bit set.

Bit(s) Number (Continued)	Name	Length	Comments
11	STTO: Start Time-out	1	0 = No effect. 1 = Start waiting for a character before clocking the time-out counter.
10	STPBRK: Stop Break	1	0 = No effect. 1 = If a break is being transmitted, stop transmission of the break after a minimum of one character length and transmit a high level during 12 bit periods.
9	STTBRK: Start Break	1	0 = No effect. 1 = If break is not being transmitted, start transmission of a break after the characters present in US_THR and in the Transmit Shift Register have been transmitted.
8	RSTSTA: Reset Status Bits	1	0 = No effect. 1 = Resets the status bits PARE, FRAME, OVRE and RXBRK in the US_CSR.
7	TXDIS: Transmitter Disable	1	0 = No effect. 1 = The transmitter is disabled.
6	TXEN: Transmitter Enable	1	0 = No effect. 1 = The transmitter is enabled if TXDIS is 0. (1)
5	RXDIS: Receiver Disable	1	0 = No effect. 1 = The receiver is disabled.
4	RXEN: Receiver Enable	1	0 = No effect. 1 = The receiver is enabled if RXDIS is 0.
3	RSTTX:Reset Transmitter	1	0 = No effect. 1 = The transmitter logic is reset. (2)
2	RSTRX: Reset Receiver	1	0 = No effect. 1 = The receiver logic is reset.
1 downto 0	UNUSED		

1. When the Mode is not ISO7816 and the transmitter is disabled, the USART TX out is in high-importance.
2. The transmitter is disabled after reset.

USART Mode Register

Register Name	US_MR
Access Type	Read/Write
Reset Value	See Table 53 on page 169
Absolute Address	0xFFFC 0004 for USART0 and 0xFFFC 4004 for USART1

Bit(s) Number	Name	Length	Comments
31 downto 29	UNUSED	3	
28	FILTER: Receive Line Filter	1	0 = The USART does not filter receive line. 1 = The USART filters receive line using a three sample filter.
26 downto 24	MAX_ITERATION: Maximum Number of Iterations	3	0-7 This will operate in mode ISO7816, Protocol T = 0 only
23 downto 22	UNUSED	2	
21	DSNACK: Disable Successive NACK	1	0 = NACK is sent on the ISO line as soon as a parity error occurs in the received character (unless INACK is set). 1 = Successive parity errors are counted up to the value specified in the MAX_ITERATION field. These parity errors generate a NACK on the ISO line. As soon as this value is reached, no additional NACK is sent on the ISO line. The flag ITERATION is asserted.
20	INACK:Inhibit Non-acknowledge	1	Note: This bit will be used only in ISO7816 mode, protocol T = 0 receiver. 0 = The NACK is generated. 1 = The NACK is not generated.
19	OVER: Oversampling Mode	1	0 = 16x Oversampling. 1 = 8x Oversampling.
18	CKLO: Clock Output Select	1	0 = The USART does not drive the SCK pin. 1 = The USART drives the SCK pin if USCLK[1] is 0.
17	MODE9: 9-bit Character Length	1	0 = CHRL defines character length. 1 = 9-bit character length. MODE9 has priority on character length.

Bit(s) Number (Continued)	Name	Length	Comments
16	MSBF: Bit Order	1	0 = LSB First. 1 = MSB First.
15 downto 14	CHMODE: Channel Mode	2	**Mode Description** 0 0 — Normal Mode The Channel operates as an Rx/Tx USART 0 1 — Automatic Echo RXD Input is connected to the TXD pin. 1 0 — Local Loopback Transmitter Output Signal is connected to Receiver Input Signal. 1 1 — Remote Loopback RXD is internally connected to TXD pin.
13 downto 12	NBSTOP: Number of Stop Bits (1)	2	0 0 — 1 stop bit / 1 stop bit 0 1 — 1.5 stop bits / Reserved 1 0 — 2 stop bits / 2 stop bits 1 1 — Reserved / Reserved (Asynchronous (SYNC=0) / Synchronous (SYNC=1))
11 downto 9	PAR: Parity Type	3	**Parity type** 0 0 0 — Even Parity 0 0 1 — Odd Parity 0 1 0 — Parity forced to 0 (Space) 0 1 1 — Parity forced to 1 (Mark) 1 0 x — No parity 1 1 x — Multidrop mode
8	SYNC: Synchronous Mode Select	1	0 = USART operates in Asynchronous Mode. 1 = USART operates in Synchronous Mode.

Bit(s) Number (Continued)	Name	Length	Comments	
7 downto 6	CHRL: Character Length [2]	2		**Character Length**
			0 0	Five bits
			0 1	Six bits
			1 0	Seven bits
			1 1	Eight bits
5 downto 4	USCLKS: Clock Selection	2		**Selected Clock**
			0 0	MCKI (ARM clock)
			0 1	CLK_ARM_DIV (from CLKGEN)
			1 x	External (SCK)
3	UNUSED	1		
2 downto 0	USART_MODE	3		**Mode**
			0 0 0	Normal
			0 0 1	RS485
			0 1 0	HW Handshaking
			0 1 1	Modem
			1 0 0	ISO07816: T = 0
			1 1 0	ISO07816: T = 1

1. The interpretation of the number of stop bits depends on SYNC.
2. Start, stop and parity bits are added to the character length.

USART Interrupt Enable Register

Register Name	US_IER
Access Type	Write Only
Base Address	0xFFFC 0008 for USART0 and 0xFFFC 4008 for USART1

Bit(s) Number	Name	Length	Comments
31	COMM_RX: Enable COMM_RX (from ARM) Interrupt	1	0 = No Effect. 1 = Enables COMM_RX Interrupt.
30	COMM_TX: Enable COMM_TX (from ARM) Interrupt	1	0 = No Effect. 1 = Enables COMM_TX Interrupt.
29 downto 20	UNUSED	10	
19	CTSIC: Enable Clear to Send Input Change Interrupt	1	0 = No Effect. 1 = Enables CTS Interrupt.
18	DCDIC: Enable Data Carrier Detect Input Change Interrupt	1	0 = No Effect. 1 = Enables DCD Interrupt.
17	DSRIC: Enable Data Set Ready Input Change	1	0 = No Effect. 1 = Enables DCR Interrupt.
16	RIIC: Enable Ring Indicator Input Change	1	0 = No Effect. 1 = Enables RI Interrupt.
15 downto 14	UNUSED	2	
13	NACK: Enable Non-acknowledge Interrupt	1	0 = No effect. 1 = Enable Non-acknowledge Interrupt.
12	RXBUFF: Enable Buffer Full Interrupt	1	0 = No effect. 1 = Enable Buffer Full Interrupt.
11	TXBUFE: Enable Buffer Empty Interrupt	1	0 = No effect. 1 = Enables Buffer Empty Interrupt.
10	ITERATION: Enable Iteration Interrupt	1	Note: This will operate only in IS07816 mode, Protocol T = 0. 0 = No effect. 1 = Enables ITERATION interrupt.
9	TXEMPTY: Enable TXEMPTY Int.	1	0 = No effect. 1 = Enables TXEMPTY Interrupt.

Bit(s) Number (Continued)	Name	Length	Comments
8	TIMEOUT: Enable Time-out Int.	1	0 = No effect. 1 = Enables Reception Time-out Interrupt.
7	PARE: Enable Parity Error Int.	1	0 = No effect. 1 = Enables Parity Error Interrupt.
6	FRAME: Enable Framing Error Int.	1	0 = No effect. 1 = Enables Framing Error Interrupt.
5	OVRE: Enable Overrun Error Int.	1	0 = No effect. 1 = Enables Overrun Error Interrupt.
4	ENDTX: Enable End of Transmit Int.	1	0 = No effect. 1 = Enables End of Transmit Interrupt.
3	ENDRX: Enable End of Receive Int.	1	0 = No effect. 1 = Enables End of Receive Transfer Interrupt.
2	RXBRK: Enable Receiver Break Int.	1	0 = No effect. 1 = Enables Receiver Break Interrupt.
1	TXRDY: Enable TXRDY Int.	1	0 = No effect. 1 = Enables TXRDY Interrupt.
0	RXRDY: Enable RXRDY Int.	1	0 = No effect. 1 = Enables TXRDY Interrupt.

USART Interrupt Disable Register

Register Name	US_IDR
Access Type	Write Only
Base Address	0xFFFC000C for USART0 and 0xFFFC 400C for USART

Bit(s) Number	Name	Length	Comments
31	COMM_RX: Enable COMM_RX (from ARM) Interrupt	1	0 = No Effect. 1 = Disables COMM_RX Interrupt.
30	COMM_TX: Disable COMM_TX (from ARM) Interrupt	1	0 = No Effect. 1 = Disables COMM_TX Interrupt.
29 downto 20	UNUSED	10	
19	CTSIC: Disable Clear to Send Input Change Interrupt	1	0 = No Effect. 1 = Disables CTS Interrupt.
18	DCDIC: Disable Data Carrier Detect Input Change Interrupt	1	0 = No Effect. 1 = Disables DCD Interrupt.
17	DSRIC: Disable Data Set Ready Input Change	1	0 = No Effect. 1 = Disables DCR Interrupt.
16	RIIC: Disable Ring Indicator Input Change	1	0 = No Effect. 1 = Disables RI Interrupt.
15 downto 14	UNUSED	2	
13	NACK: Disable Non-acknowledge Interrupt	1	0 = No effect. 1 = Disables Non-acknowledge Interrupt.
12	RXBUFF: Disable Buffer Full Interrupt	1	0 = No effect. 1 = Disables Buffer Full Interrupt.
11	TXBUFE: Disable Buffer Empty Interrupt	1	0 = No effect. 1 = Disables Buffer Empty Interrupt.
10	ITERATION: Disable Iteration Interrupt	1	Note: This will operate only in IS07816 mode, Protocol T = 0. 0 = No effect. 1 = Disables ITERATION interrupt.
9	TXEMPTY: Disable TXEMPTY Int.	1	0 = No effect. 1 = Disables TXEMPTY Interrupt.

Bit(s) Number (Continued)	Name	Length	Comments
8	TIMEOUT: Disable Time-out Int.	1	0 = No effect. 1 = Disables Reception Time-out Interrupt.
7	PARE: Disable Parity Error Int.	1	0 = No effect. 1 = Disables Parity Error Interrupt.
6	FRAME: Disable Framing Error Int.	1	0 = No effect. 1 = Disables Framing Error Interrupt.
5	OVRE: Disable Overrun Error Int.	1	0 = No effect. 1 = Disables Overrun Error Interrupt.
4	ENDTX: Disable End of Transmit Int.	1	0 = No effect. 1 = Disables End of Transmit Interrupt.
3	ENDRX: Disable End of Receive Int.	1	0 = No effect. 1 = Disables End of Receive Transfer Interrupt.
2	RXBRK: Disable Receiver Break Int.	1	0 = No effect. 1 = Disables Receiver Break Interrupt.
1	TXRDY: Disable TXRDY Int.	1	0 = No effect. 1 = Disables TXRDY Interrupt.
0	RXRDY: Disable RXRDY Int.	1	0 = No effect. 1 = Disables TXRDY Interrupt.

USART Interrupt Mask Register

Register Name	US_IMR
Access Type	Read Only
Reset Value	See Table 53 on page 169
Base Address	0xFFFC 0010 for USART0 and 0xFFFC 4010 for USART1

Bit(s) Number	Name	Length	Comments
31	COMM_RX: Enable COMM_RX (from ARM) Interrupt	1	0 = COMM_RX Interrupt is disabled. 1 = COMM_RX Interrupt is enabled.
30	COMM_TX: Disable COMM_TX (from ARM) Interrupt	1	0 = COMM_TX Interrupt is disabled. 1 = COMM_TX Interrupt is enabled.
29 downto 20	UNUSED	10	
19	CTSIC: Disable Clear to Send Input Change Interrupt	1	0 = CTS Interrupt is disabled. 1 = CTS Interrupt is enabled.
18	DCDIC: Disable Data Carrier Detect Input Change Interrupt	1	0 = DCD Interrupt is disabled. 1 = DCD Interrupt is enabled.
17	DSRIC: Disable Data Set Ready Input Change	1	0 = DSR Interrupt is disabled. 1 = DSR Interrupt is enabled.
16	RIIC: Disable Ring Indicator Input Change	1	0 = RI Interrupt is disabled. 1 = RI Interrupt is enabled.
15 downto 14	UNUSED	2	
13	NACK: Disable Non-acknowledge Interrupt	1	0 = NACK Interrupt is disabled. 1 = NACK Interrupt is enabled.
12	RXBUFF: Disable Buffer Full Interrupt	1	0 = RXBUFF Interrupt is disabled. 1 = RXBUFF Interrupt is enabled.
11	TXBUFE: Disable Buffer Empty Interrupt	1	0 = TXBUFF Interrupt is disabled. 1 = TXBUFF Interrupt is enabled.
10	ITERATION: Disable Iteration Interrupt	1	Note: This will operate only in IS07816 mode, Protocol T = 0. 0 = ITERATION interrupt is disabled. 1 = ITERATION Interrupt is enabled.
9	TXEMPTY: Mask TXEMPTY Int.	1	0 = TXEMPTY Interrupt is disabled. 1 = TXEMPTY Interrupt is enabled.

Bit(s) Number (Continued)	Name	Length	Comments
8	TIMEOUT: Mask Time-out Int.	1	0 = TIMEOUT Interrupt is disabled. 1 = TIMEOUT Interrupt is enabled.
7	PARE: Mask Parity Error Int.	1	0 = PARE Interrupt is disabled. 1 = PARE Interrupt is enabled.
6	FRAME: Mask Framing Error Int.	1	0 = FRAME Interrupt is disabled. 1 = FRAME Interrupt is enabled.
5	OVRE: Mask Overrun Error Int.	1	0 = OVRE Interrupt is disabled. 1 = OVRE Interrupt is enabled.
4	ENDTX: Mask End of Transmit Int.	1	0 = ENDTX Interrupt is disabled. 1 = ENDTX Interrupt is enabled.
3	ENDRX: Mask End of Receive Int.	1	0 = ENDRX Interrupt is disabled. 1 = ENDRX Interrupt is enabled.
2	RXBRK: Mask Receiver Break Int.	1	0 = RXBRK Interrupt is disabled. 1 = RXBRK Interrupt is enabled.
1	TXRDY: Mask TXRDY Int.	1	0 = TXRDY Interrupt is disabled. 1 = TXRDY Interrupt is enabled.
0	RXRDY: Mask RXRDY Int.	1	0 = RXRDY Interrupt is disabled. 1 = RXRDY Interrupt is enabled.

USART Channel Status Register

Register Name	US_CSR
Access Type	US_CSR
Reset Value	See Table 53 on page 169
Base Address	0xFFFC 0014 for USART0 and 0xFFFC 4014 for USART1

Bit(s) Number	Name	Length	Comments
31	COMM_RX: (from ARM)	1	0 = COMM_RX is at 0 1 = ARM debug communication channel receive buffer is empty.
30	COMM_TX: (from ARM)	1	0 = COMM_RX is at 0 1 = ARM debug communication channel transmit buffer is empty.
29 downto 24	UNUSED	10	
23	CTS: Image of CTS input	1	Fixed to 1 (for USART0) 0 = CTS is at 0. 1 = CTS is at 1.
22	DCD: Image of DCD input	1	Fixed to 1 (for USART0) 0 = DCD is at 0. 1 = DCD is at 1.
21	DSR: Image of DSR input	1	Fixed to 1 (for USART0) 0 = DSR is at 0. 1 = DSR is at 1.
20	RI: Image of RI input	1	Fixed to 1 (for USART0) 0 = RI is at 0. 1 = RI is at 1.
19	CTSIC: Clear to Send Input Change Flag	1	Fixed to 0 (for USART0) 0 = No input change has been detected on CTS pin since last read of US_CSR. 1 = An input change has been detected on CTS pin.
18	DCDIC: Disable Data Carrier Detect Input Change Flag	1	Fixed to 0 (for USART0) 0 = No input change has been detected on DCD pin since last read of US_CSR. 1 = An input change has been detected on DCD pin.
17	DSRIC: Disable Data Set Ready Input Change Flag	1	Fixed to 0 (for USART0) 0 = No input change has been detected on DSR pin since last read of US_CSR. 1 = An input change has been detected on DSR pin.

Bit(s) Number (Continued)	Name	Length	Comments
16	RIIC: Disable Ring Indicator Input Change Flag	1	Fixed to 0 (for USART0) 0 = No input change has been detected on RI pin since last read of US_CSR. 1 = An input change has been detected on RI pin.
15 downto 14	UNUSED	2	
13	NACK: Non-acknowledge	1	0 = A Non-acknowledgement has not been detected. 1 = A Non-acknowledgement has been detected.
12	RXBUFF: Reception Buffer Full	1	0 = PDC Reception Buffer is not full. 1 = PDC Reception Buffer is full.
11	TXBUFE: Transmission Buffer Empty	1	0 = PDC Transmission Buffer is not empty. 1 = PDC Transmission Buffer is empty.
10	ITERATION: Max Number of Repetitions Reached	1	Fixed to 0 (for USART0) Note: This bit will operate only in IS07816 mode, Protocol T = 0. 0 = Max number of repetitions has not been reached. 1 = Max number of repetitions has been reached.Not Applicable in our configuration.
9	TXEMPTY: Transmitter Empty. (1)	1	0 = There are characters either in US_THR or in the Transmit Shift Register. 1 = There are no characters neither in US_THR nor in the Transmit Shift Register.
8	TIMEOUT: Receiver Time-out.	1	0 = There has not been a time-out since the last “Start Time-out” command or the Time-out Register is 0. 1 = There has been a time-out since the last “Start Time-out” command.
7	PARE: Parity Error.	1	1 = At least one parity bit has been detected false (or a parity bit high in multi-drop mode) since the last “Reset Status Bits” command. 0 =No parity bit has been detected false (or a parity bit high in multi-drop mode) since the last “Reset Status Bits” command.
6	FRAME: Framing Error.	1	0 = No stop bit has been detected low since the last “Reset Status Bits” command. 1 = At least one stop bit has been detected low since the last “Reset Status Bits” command.

Bit(s) Number (Continued)	Name	Length	Comments
5	OVRE: Overrun Error.	1	0 = No byte has been transferred from the Receive Shift Register to the US_RHR when RxRDY was asserted since the last “Reset Status Bits” command. 1 = At least one byte has been transferred from the Receive Shift Register to the US_RHR when RxRDY was asserted since the last “Reset Status Bits” command.
4	ENDTX: End of Transmit.	1	0 = The End of Transfer signal from the PDC channel dedicated to the transmitter is inactive. 1 = The End of Transfer signal from the PDC channel dedicated to the transmitter is active.
3	ENDRX: End of Receive.	1	0 = The End of Transfer signal from the PDC channel dedicated to the receiver is inactive. 1 = The End of Transfer signal from the PDC channel dedicated to the receiver is active.
2	RXBRK: Break Receiver/ End of Break.	1	0 = No Break Receiver or End of Break has been detected since the last “Reset Status Bits” command in the Control Register. 1 = Break Receiver or End of Break has been detected since the last “Reset Status Bits” command in the Control Register.
1	TXRDY: Transmitter Ready. (2)	1	0 = A character is in US_THR waiting to be transferred to the Transmit Shift Register or the transmitter is disabled. 1 = There is no character in the US_THR.
0	RXRDY: Receiver Ready.	1	0 = No complete character has been received since the last read of the US_RHR or the receiver is disabled. 1 = At least one complete character has been received and the US_RHR has not been read yet.

1. Equal to zero when USART is disabled or at reset. Transmitter Enable command (in US_CR) sets this bit to one.
2. Equal to zero when USART is disabled or at reset. Transmitter Enable command (in US_CR) sets this bit to one.

USART Receiver Holding Register

Register Name	US_RHR
Access Type	Read Only
Reset Value	See Table 53 on page 169
Base Address	0xFFFC 0018 for USART0 and 0xFFFC 4018 for USART1

Bit(s) Number	Name	Length	Comments
31 downto 8	UNUSED	24	
7 downto 0	RXCHR: Received Character	8	Last character received if RXRDY is set. When the number of data bits is less than 9 bits, the bits are right-aligned. All non-significant bits read zero

USART Transmitter Holding Register

Register Name	US_THR
Access Type	Write Only
Base Address	0xFFFC 001C for USART0 and 0xFFFC 401C for USART1

Bit(s) Number	Name	Length	Comments
31 downto 8	UNUSED	24	
7 downto 0	TXCHR: Character to be Transmitted.	8	Next character to be transmitted after the current character if TXRDY is not set. When the number of data bits is less than 9 bits, the bits are right-aligned.

USART Baud Rate Generator Register

Register Name	US_BRGR
Access Type	Read/Write
Reset Value	See Table 53 on page 169
Base Address	0xFFFC 0020 for USART0 and 0xFFFC 4020 for USART1

Bit(s) Number	Name	Length	Comments	
31 downto 16	UNUSED	16		
15 downto 0	CD: Clock Divisor	16	This register has no effect if Synchronous Mode is selected with an external clock.	
			CD	
			0	Disables Clock
			1	Clock Divisor bypass
			2 to 65535	Baud Rate (Asynchronous Mode) = Selected Clock / (N×CD) Baud Rate (Synchronous Mode) = Selected Clock / CD

Notes:
1. In CD field N = 16 if OVER = 0 in US_MR; N = 8 if OVER = 1 in US_MR
2. In Synchronous Mode, the value programmed must be even to ensure a 50:50 ratio.
3. Clock divisor bypass (CD=1) must not be used when internal clock MCKI is selected (USCLKS=0)

USART Receiver Time-out Register

Register Name	US_RTOR
Access Type	Read/Write
Reset Value	See Table 53 on page 169
Base Address	0xFFFC 0024 for USART0 and 0xFFFC 4024 for USART1

Bit(s) Number	Name	Length	Comments	
31 downto 16	UNUSED	24		
15 downto 0	TO: Time-out Value	16	When a value is written to this register, a Start Time-out Command is automatically performed.	
			TO	
			0	Disables the Rx Time-out function.
			1 to 65535	The Time-out counter is loaded with TO when the Start Time-out Command is given or when each new data character is received (after reception has started).

Note: Time-out duration = TO x 4 x Bit period

USART Transmitter Time-guard Register

Register Name	US_TTGR
Access Type	Read/Write
Reset Value	See Table 53 on page 169
Base Address	0xFFFC 0028 for USART0 and 0xFFFC 4028 for USART1

Bit(s) Number	Name	Length	Comments	
31 downto 16	UNUSED	24		
7 downto 0	TG:Time-guard Value	8	**TG**	
			0	Disables the Tx Time-guard function.
			1 to 255	TXD is inactive high after the transmission of each character for the time-guard duration.

Note: Time-guard duration = TG x Bit period

USART FI_DI RATIO Register

Register Name	US_FIDI
Access Type	Read/Write
Reset Value	See Table 53 on page 169
Base Address	0xFFFC 0040 for USART0 and 0xFFFC 4040 for USART1

Bit(s) Number	Name	Length	Comments	
31 downto 11	UNUSED	21		
10 downto 0	FIDI: FI Over DI Ratio Value	11	**FIDI**	
			0	Baud Rate = 0.
			1 to 2047	Baud Rate = Selected clock / FI_DI_RATIO / 16.

USART Nb Errors Register

Register Name	US_NER
Access Type	Read
Reset Value	See Table 53 on page 169
Base Address	0xFFFC 0044 for USART0 and 0xFFFC 4044 for USART1

Bit(s) Number	Name	Length	Comments
31 downto 8	UNUSED	24	
7 downto 0	NB_ERRORS: Number of Errors During ISO7816 Transfers	8	This 8-bit register informs the user of the total amount of errors that have occurred during an ISO7816 transfer. It is a read-only register and it is reset by reading the register.

SPI: Serial Peripheral Interface

ARM System features two SPIs (SPI0 and SPI1) that provide communication with external devices in master or slave mode.

The SPIs interface to the APB and are connected to the Peripheral Data Controller PDC.

A divided clock is available for each SPI from CLKGEN:

- clk_armdiv8 ➔ SPI0
- clk_armdiv9 ➔ SPI1

Each SPI can generate an interrupt toward the AIC:

- SPI0 ➔ INT[6]
- SPI1 ➔ INT[7]

As for the USART, a two-channel PDC is used to move data directly between memory and each SPI without CPU intervention for maximum real-time processing throughput.

Each SPI can work in master and slave mode.

SPI Master Mode

In Master Mode, the SPI controls data transfers to and from the slave(s) connected to the SPI bus. The SPI drives the chip select(s) (SPIx_NSS; SPIx_NSS[1:3]) to the slave(s) and the serial clock (SPIx_SCK).

After enabling the SPI, a data transfer begins when a data is written to the SP_TDR (Transmit Data Register).

Transmit and Receive buffers maintain the data flow at a constant rate with a reduced requirement for high priority interrupt servicing. When new data is available in the SP_TDR the SPI continues to transfer data. If the SP_RDR (Receive Data Register) has not been read before new data is received, the Overrun Error (OVRES) flag is set.

The delay between the activation of the chip select and the start of the data transfer (DLYBS) as well as the delay between each data transfer (DLYBCT) can be programmed for each of the four external chip selects.

In master mode the peripheral selection can be defined in two different ways:

- Fixed Peripheral Select: SPI exchanges data with only one peripheral Fixed Peripheral Select mode is ideal for transferring memory blocks without the extra overhead in the transmit data register to determine the peripheral. This option is only available when the SPI is programmed in master mode.
- Variable Peripheral Select: Data can be exchanged with more than one peripheral. The PCS field in SP_TDR (Transmit Data Register) is used to select the destination peripheral. The data transfer characteristics are changed when the selected peripheral changes, according to the associated chip select register. This option is only available when the SPI is programmed in master mode.

The Chip Select lines are driven by the SPI only if it is programmed in Master Mode. These lines are used to select the destination peripheral. The PCSDEC field in SP_MR (Mode Register) selects 1 to 4 peripherals (PCSDEC = 0) or up to 15 peripherals (PCSDEC = 1).

If Variable Peripheral Select is active, the chip select signals are defined for each transfer in the PCS field in SP_TDR. Chip select signals can thus be defined independently for each transfer.

If Fixed Peripheral Select is active, Chip Select signals are defined for all transfers by the field PCS in SP_MR. If a transfer with a new peripheral is necessary, the software must wait until the current transfer is completed, then change the value of PCS in SP_MR before writing new data in SP_TDR.

The value on the NPCS pins at the end of each transfer can be read in the SP_RDR (Receive Data Register).

By default, all NPCS signals are high (equal to one) before and after each transfer.

A mode fault is detected when the SPI is programmed in Master Mode and a low level is driven by an external master on the SPIx_NSS signal.

When a mode fault is detected, the MODF bit in the SP_SR is set until the SP_SR is read and the SPI is disabled until re-enabled by bit SPIEN in the SP_CR (Control Register).

By default, Mode Fault Detection is enabled. It is disabled by setting the MODFDIS bit in the SPI Mode Register.

Figure 44. SPI Master Mode

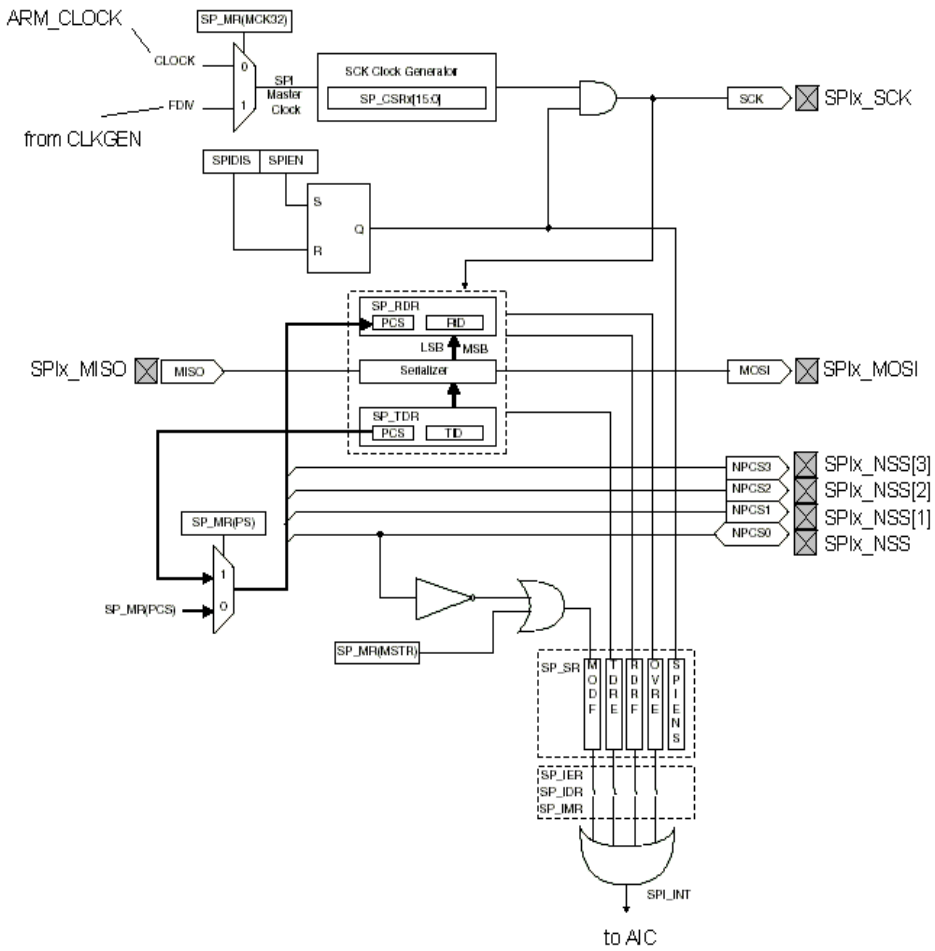
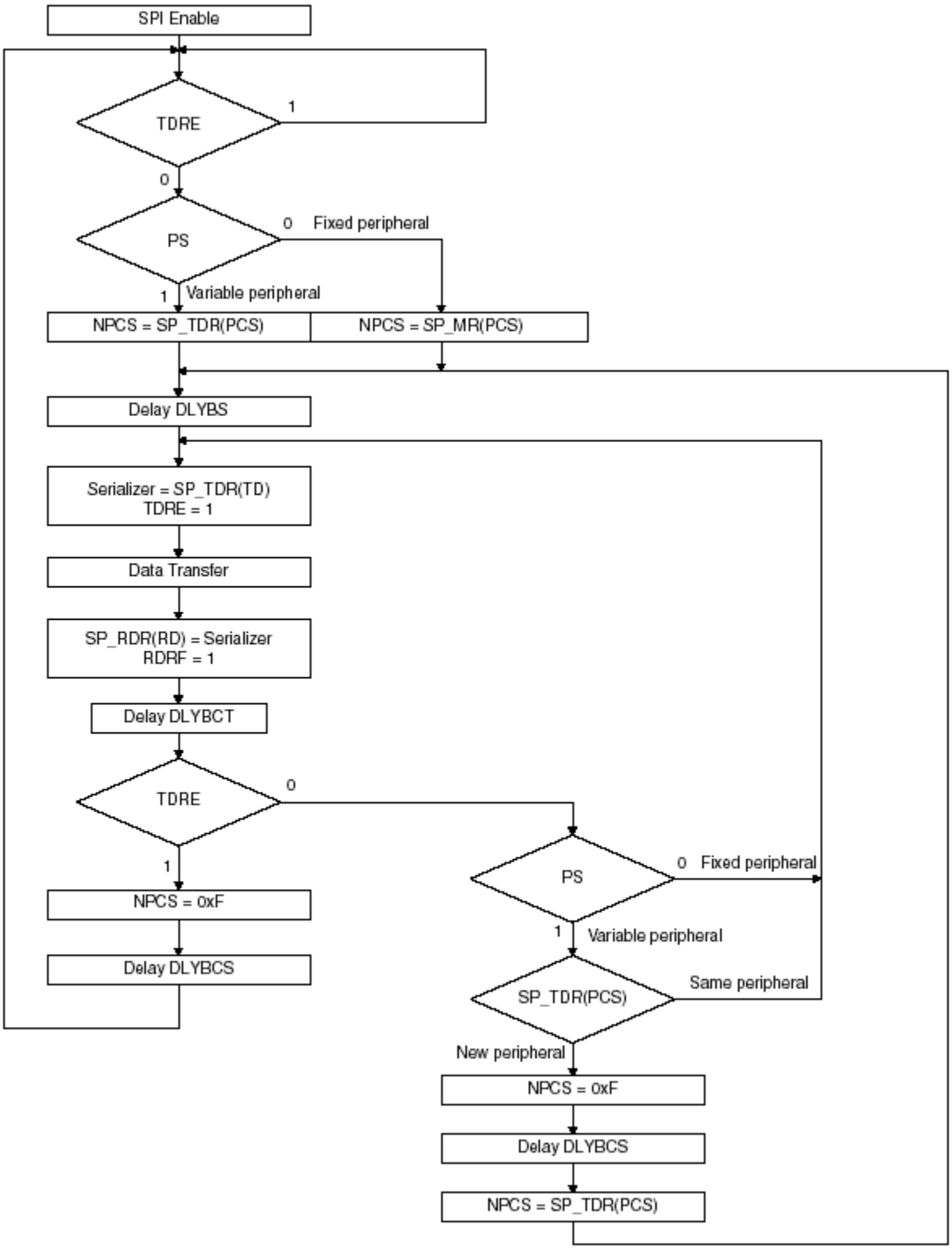


Figure 45. SPI Functional Flow Diagram in Master Mode



SPI Slave Mode

In Slave Mode, the SPI waits for SPIx_NSS to go active low before receiving the serial clock from an external master.

In slave mode CPOL, NCPHA and BITS fields of SP_CSR0 are used to define the transfer characteristics.

The other Chip Select Registers are not used in slave mode.

Figure 46. SPI Slave Mode

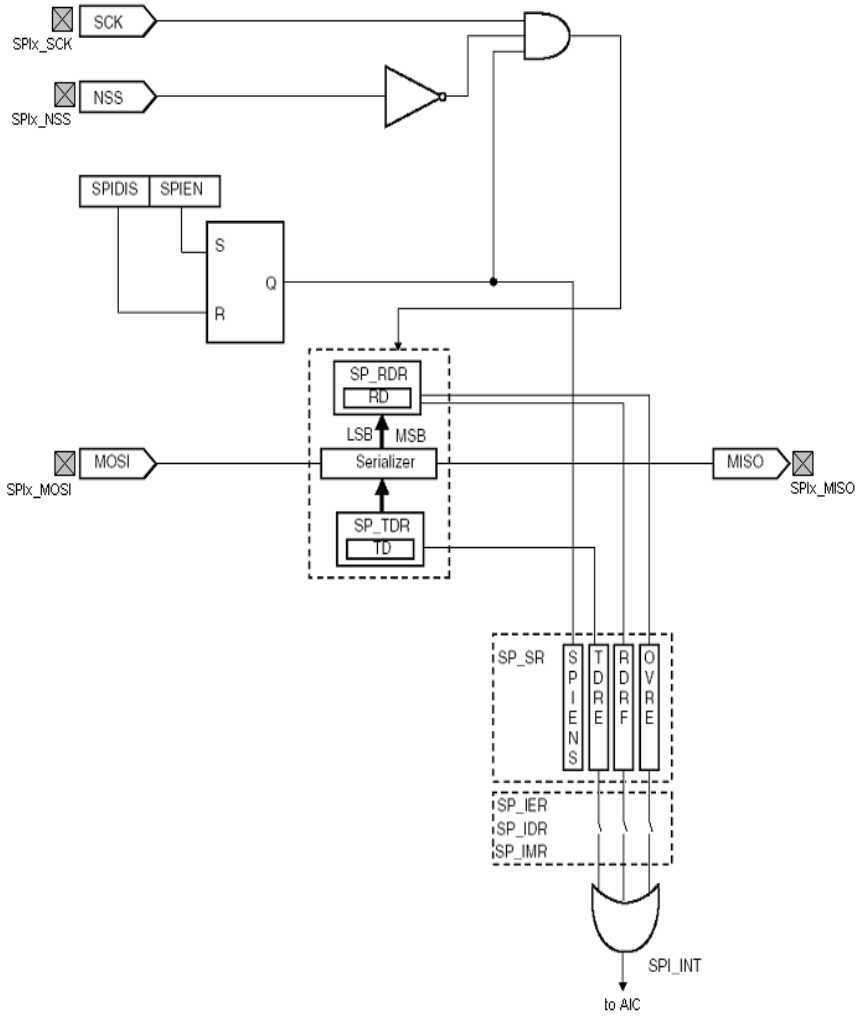


Figure 47. SPI Transfer Format (NCPHA Equals One, 8 Bits per Transfer)

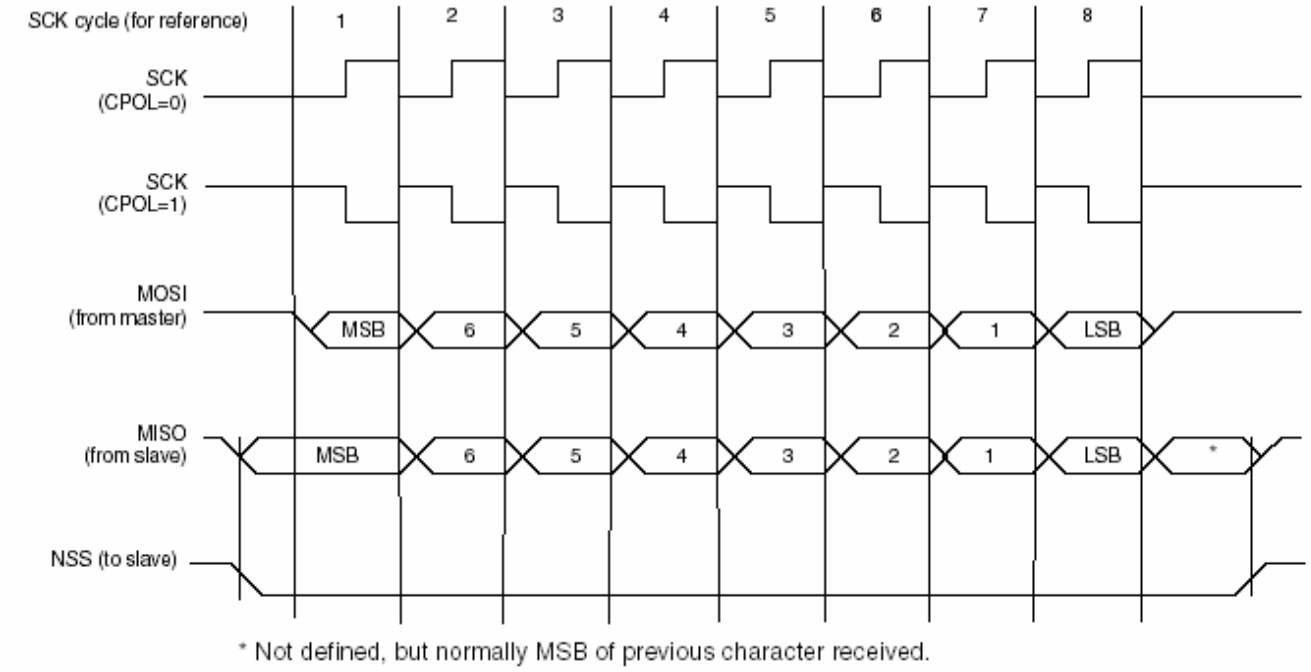


Figure 48. SPI Transfer Format (NCPHA Equals Zero, 8 Bits per Transfer)

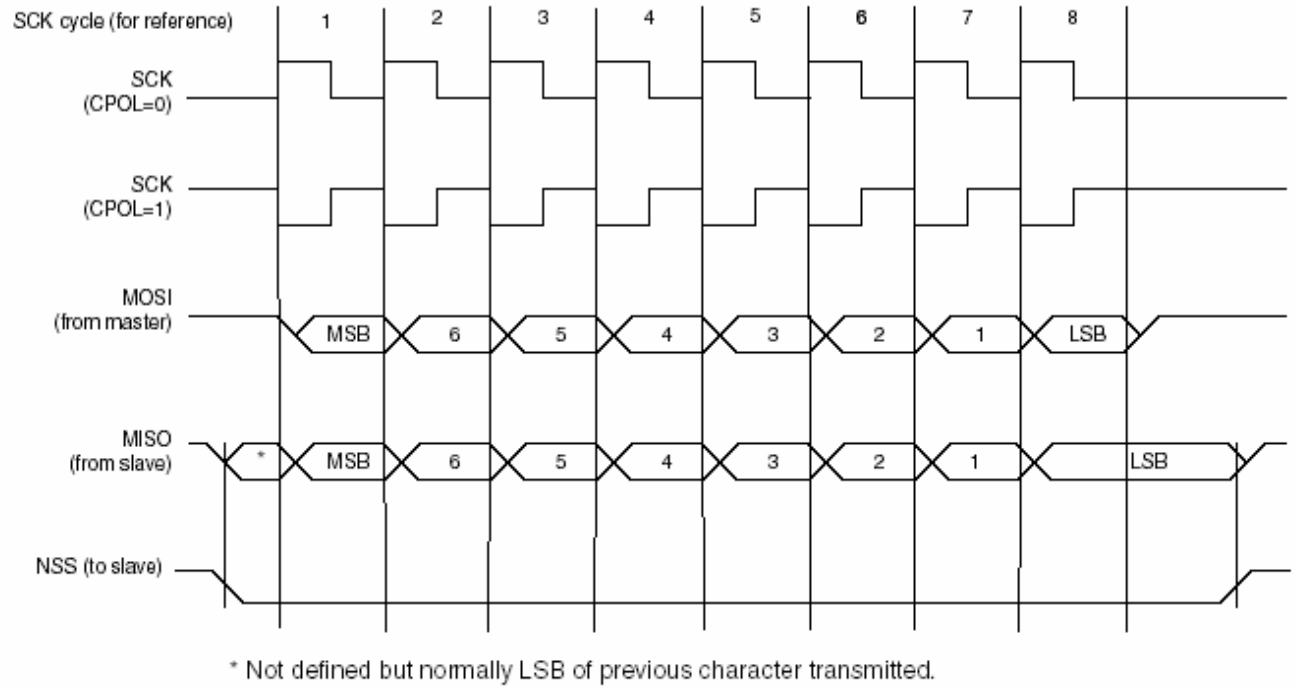
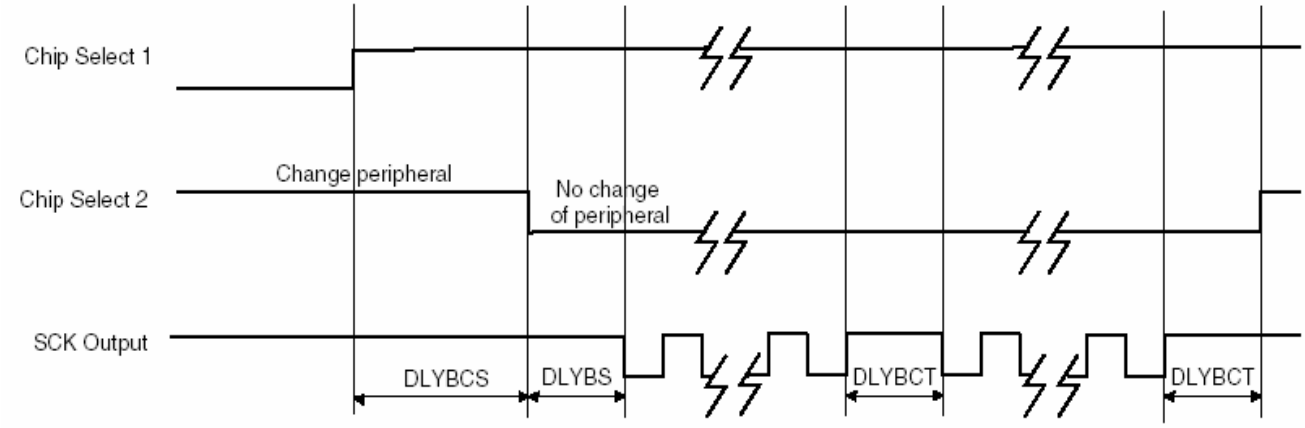


Figure 49. SPI Programmable Delays (DLYBCS, DLYBS and DLYBCT)



SPI Clock Generation

In master mode the SPI Master Clock can be (as defined by the MCK32 field of SP_MR):

- SPI0 ➔ clk_armdiv8 (from CLKGEN) or arm_clock
- SPI1 ➔ clk_armdiv9 (from CLKGEN) or arm_clock

The SPI baud rate clock is generated by dividing the SPI Master Clock by a value between 4 and 510. The divisor is defined in the SCBR field in each Chip Select Register. The transfer speed can thus be defined independently for each chip select signal.

CPOL and NCPHA in the Chip Select Registers define the clock/data relationship between master and slave devices. CPOL defines the inactive value of the SCK. NCPHA defines which edge causes data to change and which edge causes data to be captured.

In Slave Mode, the input clock low and high pulse duration must strictly be longer than two arm_clock periods.

SPI Registers

Base Address 0xFFFC 8000 for SPI0 and
0xFFFC C000 for SPI1

Table 54. SPI Register Map

Offset	Register	Name	Access	Reset State
0x00	Control Register	SP_CR	Write Only	-
0x04	Mode Register	SP_MR	Read/Write	0
0x08	Receive Data Register	SP_RDR	Read Only	0
0x0C	Transmit Data Register	SP_TDR	Write Only	-
0x10	Status Register	SP_SR	Read Only	0
0x14	Interrupt Enable Register	SP_IER	Write Only	-
0x18	Interrupt Disable Register	SP_IDR	Write Only	-
0x1C	Interrupt Mask Register	SP_IMR	Read Only	0
0x20	Reserved	-	-	-
0x24	Reserved	-	-	-
0x28	Reserved	-	-	-
0x2C	Reserved	-	-	-
0x30	Chip Select Register 0	SP_CSR0	Read/Write	0
0x34	Chip Select Register 1	SP_CSR1	Read/Write	0
0x38	Chip Select Register 2	SP_CSR2	Read/Write	0
0x3C	Chip Select Register 3	SP_CSR3	Read/Write	0

SPI Control Register

Register Name	SP_CR
Access Type	Write Only
Base Address	0xFFFC 8000 for SPI0 and 0xFFFC C000 for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 8	UNUSED	24	
7	SWRST: SPI Software reset	1	0 = No effect. 1 = Resets the SPI.
6 downto 2	UNUSED	5	
1	SPIDIS: SPI Disable	1	0 = No effect. 1 = Disables the SPI.
0	SPIEN: SPI Enable	1	0 = No effect. 1 = Enables the SPI.

SPI Mode Register

Register Name	SP_MR
Access Type	Read/Write
Reset Value	See Table 54 on page 195
Base Address	0xFFFC 8004 for SPI0 and 0xFFFC C004 for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 24	DLYBCS: Delay Between Chip Selects	8	Number of Delay cycles between two CS activation. If DLYBCS is less than or equal to six, six SPI Master Clock periods will be inserted by default.
23 downto 20	UNUSED	4	

Bit(s) Number (Continued)	Name	Length	Comments
19 downto 16	PCS: Peripheral Chip Select	4	This field is only used if Fixed Peripheral Select is active (PS=0). If PCSDEC=0: PCS = xxx0 NPCS[3:0] = 1110 PCS = xx01 NPCS[3:0] = 1101 PCS = x011 NPCS[3:0] = 1011 PCS = 0111 NPCS[3:0] = 0111 PCS = 1111 forbidden (no peripheral is selected) (x = don't care) If PCSDEC=1: NPCS[3:0] output signals = PCS
7	LLB: Local Loopback Enable	1	0 = Local loop-back path disabled 1 = Local loop-back path enabled
6 downto 5	UNUSED	2	
4	MODFDIS: Mode Fault Detection	1	0 = Mode fault detection is enabled. 1 = Mode fault detection is disabled.
3	MCK32: Clock Selection	1	0 = SPI Master Clock equals CLOCK 1 = SPI Master Clock equals FDIV
2	PCSDEC: Chip Select Decode	1	0 = The chip selects are directly connected to a peripheral device. 1 = The four chip select lines are connected to a 4- to 16-bit decoder. When PCSDEC equals one, up to 16 Chip Select signals can be generated with the four lines using an external 4- to 16-bit decoder. The Chip Select Registers define the characteristics of the 16 chip selects according to the following rules: SP_CSR0 defines peripheral chip select signals 0 to 3. SP_CSR1 defines peripheral chip select signals 4 to 7. SP_CSR2 defines peripheral chip select signals 8 to 11. SP_CSR3 defines peripheral chip select signals 12 to 15 (1).
1	PS: Peripheral Select	1	0 = Fixed Peripheral Select. 1 = Variable Peripheral Select.
0	MSTR: Master/Slave Mode	1	0 = SPI is in Slave mode. 1 = SPI is in Master mode.

1. The 16th state corresponds to a state in which all chip selects are inactive. This allows a different clock configuration.

SPI Receive Data Register

Register Name	SP_RDR
Access Type	Read
Reset Value	See Table 54 on page 195
Base Address	0xFFFC 8008 for SPI0 and 0xFFFC C008 for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 20	UNUSED	12	
19 downto 16	PCS: Peripheral Chip Select	4	In Master Mode only, these bits indicate the value on the NPCS pins at the end of a transfer. Otherwise, these bits read zero.
15 downto 0	RD: Receive Data	16	Data received by the SPI Interface is stored in this register right-justified. Unused bits read zero.

SPI Transmit Data Register

Register Name	SP_TDR
Access Type	Write
Base Address	0xFFFC 800C for SPI0 and 0xFFFC C00C for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 20	UNUSED	12	
19 downto 16	PCS: Peripheral Chip Select	4	This field is only used if Variable Peripheral Select is active (PS = 1). If PCSDEC = 0: PCS = xxx0 NPCS[3:0] = 1110 PCS = xx01 NPCS[3:0] = 1101 PCS = x011 NPCS[3:0] = 1011 PCS = 0111 NPCS[3:0] = 0111 PCS = 1111 forbidden (no peripheral is selected) (x = don't care) If PCSDEC = 1: NPCS[3:0] output signals = PCS

Bit(s) Number (Continued)	Name	Length	Comments
15 downto 0	TD: Transmit Data	16	Data to be transmitted by the SPI Interface is stored in this register. Information to be transmitted must be written to the transmit data register in a right-justified format.

SPI Status Register

Register Name	SP_SR
Access Type	Read
Reset Value	See Table 54 on page 195
Base Address	0xFFFC 8010 for SPI0 and 0xFFFC C010 for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 17	UNUSED	15	
16	SPIENS: SPI Enable Status	1	0 = SPI is disabled 1 = SPI is enabled
15 downto 8	UNUSED	12	
7	TXBUFFE: SPI Transmit Buffer Empty	1	0 = SPI transmit buffer (in PDC) not empty. 1 = SPI transmit buffer (in PDC) empty. Input signal from PDC block.
6	RXBUFFF: SPI Receive Buffer Full	1	0 = SPI transmit buffer (in PDC) not full. 1 = SPI transmit buffer (in PDC) full. Input signal from PDC block.
5	ENDTX: SPI Transmit End	1	0 = SPI transmit not ended. 1 = SPI transmit ended Input signal from PDC block.
4	ENDRX: SPI Receive End	1	0 = SPI receive not ended. 1 = SPI receive ended. Input signal from PDC block.
3	OVRES: Overrun Error Status	1	0 = No overrun has been detected since the last read of SP_SR. 1 = An overrun has occurred since the last read of SP_SR. An overrun occurs when SP_RDR is loaded at least twice from the serializer since the last read of the SP_RDR.

Bit(s) Number (Continued)	Name	Length	Comments
2	MODF: Mode Fault Error	1	0 = No Mode Fault has been detected since the last read of SP_SR. 1 = A Mode Fault occurred since the last read of the SP_SR.
1	TDRE: Transmit Data Register Empty	1	0 = Data has been written to SP_TDR and not yet transferred to the serializer. 1 = The last data written in the Transmit Data Register has been transferred into the serializer. TDRE equals zero when the SPI is disabled or at reset. The SPI enable command sets this bit to one.
0	RDRF: Receive Data Register Full	1	0 = No data has been received since the last read of SP_RDR 1 = Data has been received and the received data has been transferred from the serializer to SP_RDR since the last read of SP_RDR.

SPI Interrupt Enable Register

Register Name	SP_IER
Access Type	Write
Base Address	0xFFFC 8014 for SPI0 and 0xFFFC C014 for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 8	UNUSED	28	
7	TXBUFFE: SPI Transmit Buffer Empty interrupt enable	1	0 = No effect. 1 = Enables TXBUFFE interrupt.
6	RXBUFFF: SPI Receive Buffer Full interrupt enable	1	0 = No effect. 1 = Enable RXBUFFF interrupt.
5	ENDTX: SPI Transmit End interrupt enable	1	0 = No effect. 1 = Enables ENDTX interrupt.
4	ENDRX: SPI Receive End interrupt enable	1	0 = No effect. 1 = Enables ENDRX interrupt.
3	OVRES: Overrun Error Status interrupt enable	1	0 = No effect. 1 = Enables the Overrun Error interrupt.

Bit(s) Number (Continued)	Name	Length	Comments
2	MODF: Mode Fault Error interrupt enable	1	0 = No effect. 1 = Enables the Mode Fault Error Interrupt.
1	TDRE: Transmit Data Register Empty interrupt enable	1	0 = No effect. 1 = Enables the Transmit Data Register Empty Interrupt.
0	RDRF: Receive Data Register Full interrupt enable	1	0 = No effect. 1 = Enables the Receive Data Register Full interrupt.

SPI Interrupt Disable Register

Register Name	SP_IDR
Access Type	Write
Base Address	0xFFFC 8018 for SPI0 and 0xFFFC C018 for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 8	UNUSED	28	
7	TXBUFFE: SPI Transmit Buffer Empty interrupt disable	1	0 = No effect. 1 = Disables TXBUFFE interrupt.
6	RXBUFFF: SPI Receive Buffer Full interrupt disable	1	0 = No effect. 1 = Disables RXBUFFF interrupt.
5	ENDTX: SPI Transmit End interrupt disable	1	0 = No effect. 1 = Disables ENDTX interrupt.
4	ENDRX: SPI Receive End interrupt disable	1	0 = No effect. 1 = Disables ENDRX interrupt.
3	OVRES: Overrun Error Status interrupt disable	1	0 = No effect. 1 = Disables the Overrun Error Interrupt.
2	MODF: Mode Fault Error interrupt disable	1	0 = No effect. 1 = Disables the Mode Fault Error Interrupt.
1	TDRE: Transmit Data Register Empty interrupt disable	1	0 = No effect. 1 = Disables the Transmit Data Register Empty Interrupt.
0	RDRF: Receive Data Register Full interrupt disable	1	0 = No effect. 1 = Disables the Receive Data Register Full Interrupt.

SPI Interrupt Mask Register

Register Name	SP_IMR
Access Type	Read
Reset Value	See Table 54 on page 195
Base Address	0xFFFC 801C for SPI0 and 0xFFFC C01C for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 8	UNUSED	28	
7	TXBUFFE: SPI Transmit Buffer Empty interrupt mask	1	0 = TXBUFFE interrupt disabled. 1 = TXBUFFE interrupt enabled.
6	RXBUFFF: SPI Receive Buffer Full interrupt mask	1	0 = RXBUFFF interrupt disabled. 1 = RXBUFFF interrupt enabled.
5	ENDTX: SPI Transmit End interrupt mask	1	0 = ENDTX interrupt disabled. 1 = ENDTX interrupt enabled.
4	ENDRX: SPI Receive End interrupt mask	1	0 = ENDRX interrupt disabled. 1 = ENDRX interrupt enabled.
3	OVRES: Overrun Error Status interrupt mask	1	0 = Overrun Error Interrupt is disabled. 1 = Overrun Error Interrupt is enabled.
2	MODF: Mode Fault Error interrupt mask	1	0 = Mode Fault Error Interrupt is disabled. 1 = Mode Fault Error Interrupt is enabled.
1	TDRE: Transmit Data Register Empty interrupt mask	1	0 Transmit Data Register Empty Interrupt is disabled. 1 = Transmit Data Register Empty Interrupt is enabled.
0	RDRF: Receive Data Register Full interrupt mask	1	0 = Receive Data Register Full Interrupt is disabled. 1 = Receive Data Register Full Interrupt is enabled.

SPI Chip Select Register

Register Name:	SP_CSR0 … SP_CSR3
Access Type:	Read/Write
Reset Value	See Table 54 on page 195
Base Addresses:	SP_CSR0 → 0xFFFC 8030 for SPI0 and 0xFFFC C030 for SPI1

SP_CSR1 → 0xFFFC 8034 for SPI0 and 0xFFFC C034 for SPI1

SP_CSR2 → 0xFFFC 8038 for SPI0 and 0xFFFC C038 for SPI1

SP_CSR3 → 0xFFFC 803C for SPI0 and 0xFFFC C03C for SPI1

Bit(s) Number	Name	Length	Comments
31 downto 24	DLYRCT: Delay Between Consecutive Transfers	8	This field defines the delay between two consecutive transfers with the same peripheral without removing the chip select. The delay is always inserted after each transfer and before removing the chip select if needed. When DLYBCT equals zero, a delay of four SPI Master Clock periods are inserted. Otherwise, the following equation determines the delay: Delay_After_Transfer = 32 × DLYBCT × SPI_Master_Clock_period
23 downto 16	DLYBS: Delay Before SCK	8	This field defines the delay from NPCS valid to the first valid SCK transition. When DLYBS equals zero, the NPCS valid to SCK transition is 1/2 the SCK clock period. Otherwise, the following equation determines the delay: NPCS_to_SCK_Delay = DLYBS × SPI_Master_Clock_period
15 downto 8	SCRB: Serial Clock Baud Rate	8	In Master Mode, the SPI Interface uses a modulus counter to derive the SCK baud rate from the SPI Master Clock (selected between CLOCK and FDIV). The Baud rate is selected by writing a value from 2 to 255 in the field SCBR. The following equation determines the SCK baud rate: SCK_Baud_Rate = SPI_Master_Clock_frequency / 2 x SCBR Giving SCBR a value of zero or one it disables the baud rate generator. SCK is disabled and assumes its inactive state value. No serial transfers may occur. At reset, baud rate is disabled.

<table>
<tr><th>Bit(s) Number (Continued)</th><th>Name</th><th>Length</th><th colspan="2">Comments</th></tr>
<tr><td rowspan="12">7 downto 4</td><td rowspan="12">BITS: Bits per transfer</td><td rowspan="12">4</td><td colspan="2">The BITS field determines the number of data bits transferred. Reserved values should not be used.</td></tr>
<tr><td>BITS [3:0]</td><td>Bits per transfer</td></tr>
<tr><td>0000</td><td>8</td></tr>
<tr><td>0001</td><td>9</td></tr>
<tr><td>0010</td><td>10</td></tr>
<tr><td>0011</td><td>11</td></tr>
<tr><td>0100</td><td>12</td></tr>
<tr><td>0101</td><td>13</td></tr>
<tr><td>0110</td><td>14</td></tr>
<tr><td>0111</td><td>15</td></tr>
<tr><td>1000</td><td>16</td></tr>
<tr><td>Other values</td><td>Reserved</td></tr>
<tr><td>3 downto 2</td><td>UNUSED</td><td>2</td><td colspan="2"></td></tr>
<tr><td>1</td><td>NCPHA: Clock Phase</td><td>1</td><td colspan="2">0 = Data is changed on the leading edge of SCK and captured on the following edge of SCK.
1 = Data is captured on the leading edge of SCK and changed on the following edge of SCK.
NCPHA determines which edge of SCK causes data to change and which edge causes data to be captured. NCPHA is used with CPOL to produce a desired clock/data relationship between master and slave devices.</td></tr>
<tr><td>0</td><td>CPOL: Clock Polarity</td><td>1</td><td colspan="2">0 = The inactive state value of SCK is logic level zero.
1 = The inactive state value of SCK is logic level one.
CPOL is used to determine the inactive state value of the serial clock (SCK). It is used with NCPHA to produce a desired clock/data relationship between master and slave devices.</td></tr>
</table>

TC: Timer Counter

ARM System features a timer/counter block, containing three identical 16-bit timer/counter channels (TC0, TC1, TC2). Each channel can be independently programmed to perform a wide range of functions, including pulse generation, delay timing and pulse width modulation.

The timer/counter channel has three external clock inputs, five internal clock inputs, and two multi-purpose output signals, which can be configured by the user. Each channel drives an internal interrupt signal, which can be programmed to generate processor interrupts via the Advanced Interrupt Controller (AIC).

The timer/counter block features two global registers that act upon all three TC channels. The Block Control Register allows the three channels to be started simultaneously with the same instruction. The Block Mode Register defines the external clock inputs for each timer/counter channel, allowing them to be chained.

The value of the counter is incremented at each positive edge of the selected clock. When the counter has reached the value 0xFFFF and passes to 0x0000, an overflow occurs and the bit COVFS in the TC_SR (Status Register) is set. The current value of the counter is accessible in real time by reading the Counter Value Register, TC_CV. A trigger can reset the counter. In this case, the counter value passes to 0x0000 on the next valid edge of the selected clock. Each TC channel has also three programmable Compare registers (A, B and C) which stored values are compared with the counter value and whenever equal they can have different effects on the TC channel outputs (set, reset, toggle or none) depending on TC_CMR register configuration.

Each TC channel can operate in two different modes:

- Capture Mode allows measurement on signals (not used: all inputs to be measured fixed to 0).
- Waveform Mode allows wave generation.

The Waveform mode must be selected programming the WAVE field in the TC_CMR register.

Each TC channel has an interrupt output (TIMER_INT(0), (1), (2)) which activation is conditioned to user configuration (through TC_IER and TC_IDR). The interrupts are connected to AIC inputs respectively: INT1, 2, 3.

TC Clock Selection

At block level, input clock signals of each channel can either be connected to the external inputs TCLK0, TCLK1 or TCLK2, or be connected to the configurable I/O signals TIOA0, TIOA1 or TIOA2 for chaining by programming the TC_BMR (Block Mode).

Each channel can independently select an internal or external clock source for its counter:

- Internal clock signals: timer_div1_clock, timer_div2_clock, timer_div3_clock, timer_div4_clock, timer_div5_clock
- External clock signals: XC0, XC1 or XC2

The TCCLKS bits in the TC Channel Mode Register (Capture Mode) makes this selection.

The selected clock can be inverted with the CLKI bit in TC_CMR (Capture Mode). This allows counting on the opposite edges of the clock. The burst function allows the clock to be validated when an external signal is high. The BURST parameter in the Mode Register defines this signal (none, XC0, XC1, XC2).

Each TC channel has 8 clock sources:

- Timer_div1_clock (connected to clk_armdiv5 from CLKGEN)
- Timer_div2_clock (connected to clk_armdiv1 from CLKGEN)
- Timer_div3_clock (connected to clk_armdiv2 from CLKGEN)
- Timer_div4_clock (connected to clk_armdiv3 from CLKGEN)
- Timer_div5_clock (connected to clk_armdiv4 from CLKGEN)
- XC0 (for TC0 can be TC1 output, TC2 output or PIO[0] pad; for TC1 and TC2 is PIO[0] pad)
- XC1 (for TC1 can be TC0 output, TC2 output or PIO[1] pad; for TC0 and TC2 is PIO[1] pad)
- XC2 (for TC2 can be TC0 output, TC1 output or PIO[2] pad; for TC0 and TC1 is PIO[2] pad)

The source can be selected programming the TCCLKS field in the TC_CMR register, and in addition, in the TC_BMR when an external source is selected.

Note: When an external source is selected synchronization with the arm_clk is provided in order to avoid metastability problems. This causes a limited jitter of an arm_clk period on the generated clock. The maximum frequency for the clock sources is half the arm-clock.In all cases, if an external clock is used, the duration of each of its levels must be longer than the master clock (CLOCK_CHx) period. The external clock frequency must be at least 2.5 times lower than the master clock (CLOCK_CHx)

Each TC channel has two outputs:

- Timer_ext_out_A0 (connected to PIO[6] pad)
- Timer_ext_out_B0 (connected to pio(16) internal line toward magic INTM(2))
- Timer_ext_out_A1 (connected to PIO[7 pad])
- Timer_ext_out_B1 (connected to pio(17) internal line toward magic INTM(2))
- Timer_ext_out_A2 (connected to ADDA_SRCK)
- Timer_ext_out_B2 (connected to pio(18) internal line toward magic INTM(2))

The registers for channel programming are listed in the next section.

TC Clock Control

The clock of each counter can be controlled in two different ways: it can be enabled/disabled and started/stopped.

- The clock can be enabled or disabled by the user with the CLKEN and the CLKDIS commands in the Control Register. In Capture Mode it can be disabled by an RB load event if LDBDIS is set to 1 in TC_CMR. In Waveform Mode, it can be disabled by an RC Compare event if CPCDIS is set to 1 in TC_CMR. When disabled, the start or the stop actions have no effect: only a CLKEN command in the Control Register can re-enable the clock. When the clock is enabled, the CLKSTA bit is set in the Status Register.
- The clock can also be started or stopped: a trigger (software, synchro, external or compare) always starts the clock. The clock can be stopped by an RB load event in Capture Mode (LDBSTOP = 1 in TC_CMR) or a RC compare event in Waveform Mode (CPCSTOP = 1 in TC_CMR). The start and the stop commands have effect only if the clock is enabled.

TC Trigger

A trigger resets the counter and starts the counter clock. Three types of triggers are common to both modes, and a fourth external trigger is available to each mode.

The following triggers are common to both modes:

- Software Trigger: Each channel has a software trigger, available by setting SWTRG in TC_CCR.
- SYNC: Each channel has a synchronization signal SYNC. When asserted, this signal has the same effect as a software trigger. The SYNC signals of all channels are asserted simultaneously by writing TC_BCR (Block Control) with SYNC set.
- Compare RC Trigger: RC is implemented in each channel and can provide a trigger when the counter value matches the RC value if CPCTRG is set in TC_CMR.

The channel can also be configured to have an external trigger. In Capture Mode, the external trigger signal can be selected between TIOA and TIOB. In Waveform Mode, an external event can be programmed on one of the following signals: TIOB, XC0, XC1 or XC2. This external event can then be programmed to perform a trigger by setting ENETRG in TC_CMR.

If an external trigger is used, the duration of the pulses must be longer than the master clock (CLOCK_CHx) period in order to be detected.

Whatever the trigger used, it will be taken into account at the following active edge of the selected clock. This means that the counter value can be read different from zero just after a trigger, especially when a low frequency signal is selected as the clock.

Waveform Operating Mode allows the TC1 channel to generate 1 or 2 PWM signals with the same frequency and independently programmable duty cycles, or to generate different types of one-shot or repetitive pulses.

In this mode, TIOA is configured as output and TIOB is defined as output if it is not used as an external event (EEVT parameter in TC_CMR).

TC Waveform Selection

Depending on the parameter WAVSEL in TC_CMR (Channel Mode Register), the behavior of TC_CV varies.

With any selection, RA, RB and RC can all be used as compare registers.

RA Compare may be used to control the TIOA output, RB Compare may be used to control the TIOB output (if correctly configured) and RC Compare may be used to control TIOA and/or TIOB outputs.

When **WAVSEL = 00**, the value of TC_CV is incremented from 0 to 0xFFFF. Once 0xFFFF has been reached, the value of TC_CV is reset. Incrementation of TC_CV starts again and the cycle continues.

A trigger such as an external event or a software trigger can reset the value of TC_CV. It is important to note that the trigger may occur at any time. RC Compare cannot be programmed to generate a trigger in this configuration. At the same time, RC Compare can stop the counter clock (CPCSTOP = 1 in TC_CMR) and/or disable the counter clock (CPCDIS = 1 in TC_CMR).

Figure 50. TC WAVESEL = 00 without Trigger

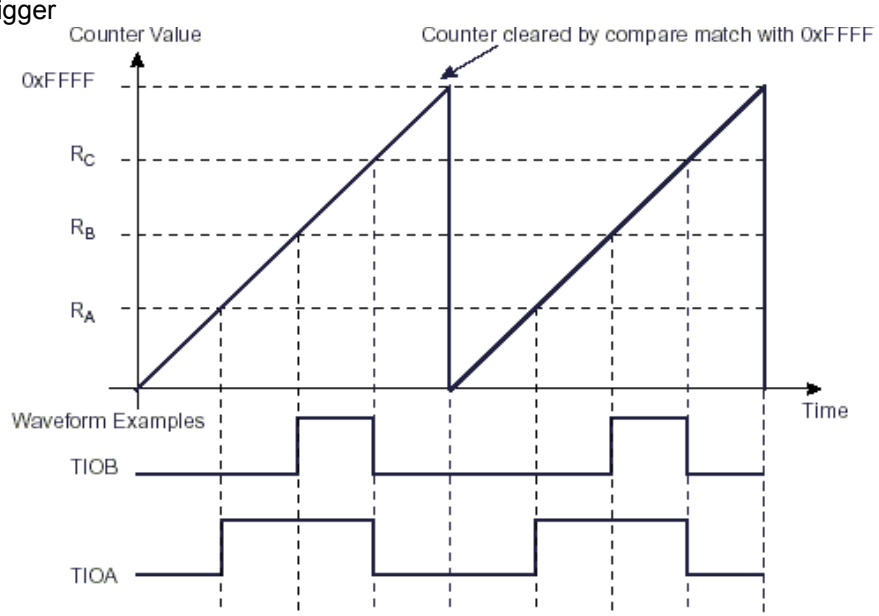
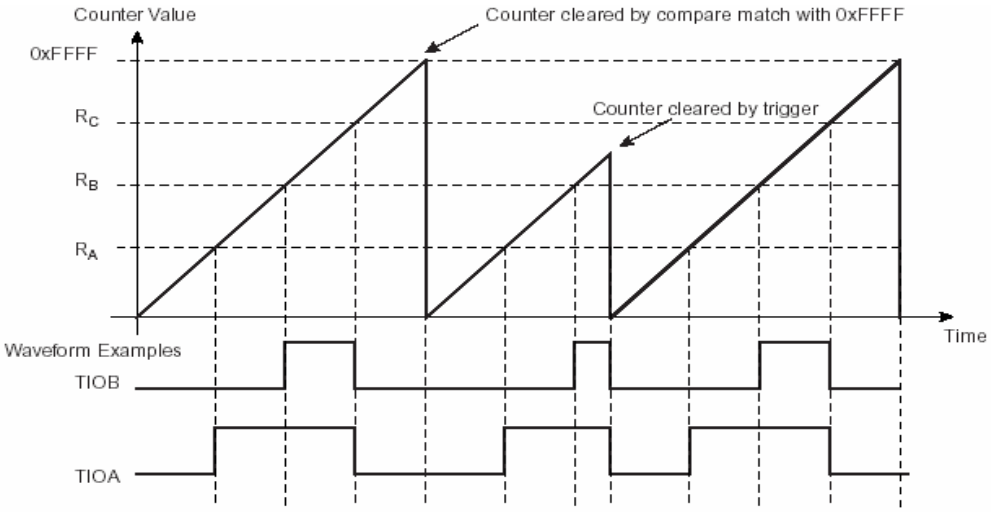


Figure 51. TC WAVESEL = 00 with Trigger



When **WAVSEL = 10**, the value of TC_CV is incremented from 0 to the value of RC, then automatically reset on an RC Compare. Once the value of TC_CV has been reset, it is then incremented and so on. It is important to note that at any time TC_CV can be reset by an external event or a software trigger if both are programmed correctly. In addition, RC Compare can stop the counter clock (CPCSTOP = 1 in TC_CMR) and/or disable the counter clock (CPCDIS = 1 in TC_CMR).

Figure 52. TC WAVESEL = 10 without Trigger

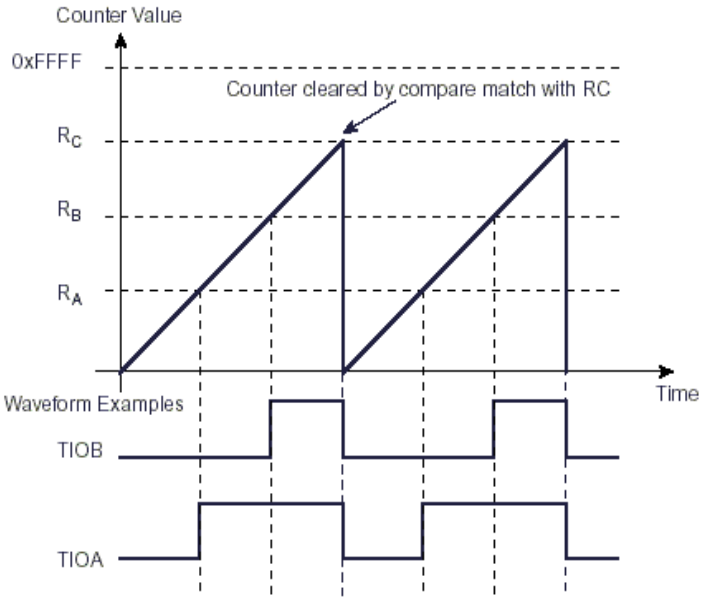
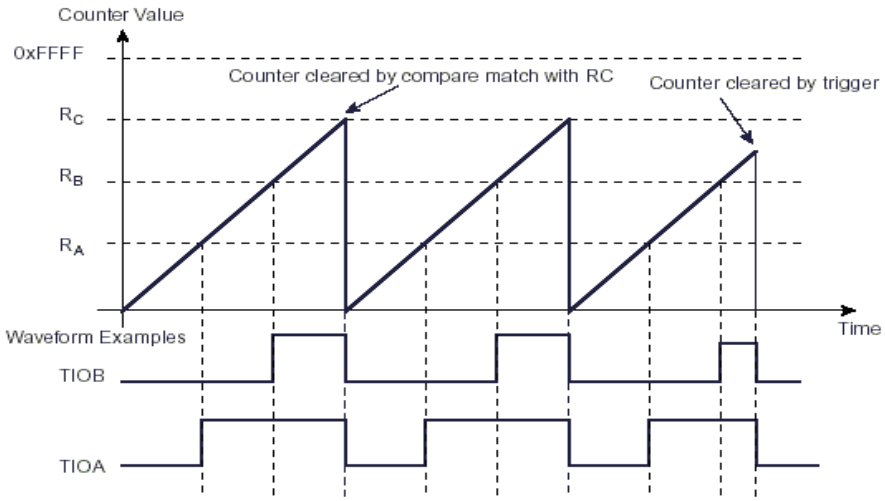


Figure 53. TC WAVESEL = 10 with Trigger



When **WAVSEL = 01**, the value of TC_CV is incremented from 0 to 0xFFFF. Once 0xFFFF is reached, the value of TC_CV is decremented to 0, then re-incremented to 0xFFFF and so on. A trigger such as an external event or a software trigger can modify TC_CV at any time. If a trigger occurs while TC_CV is incrementing, TC_CV then decrements.

If a trigger is received while TC_CV is decrementing, TC_CV then increments. RC Compare cannot be programmed to generate a trigger in this configuration. At the same time, RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

Figure 54. TC WAVESEL = 01 without Trigger

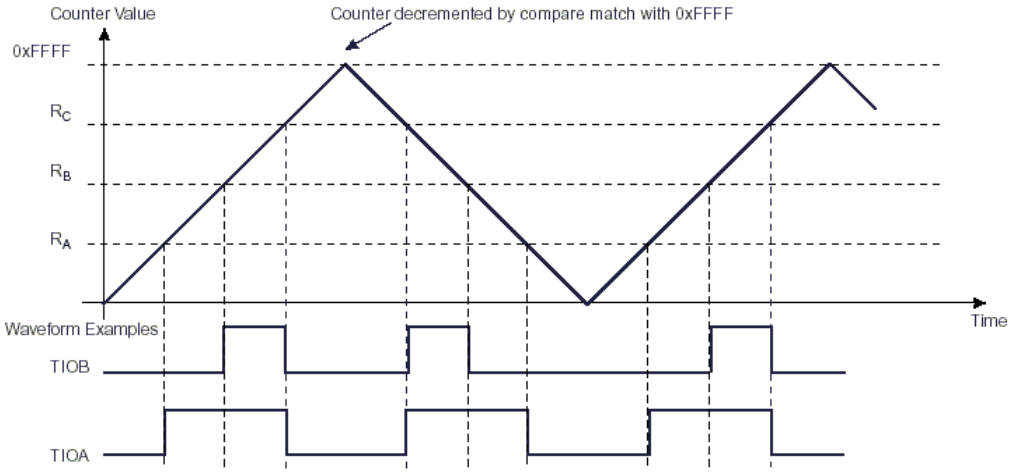
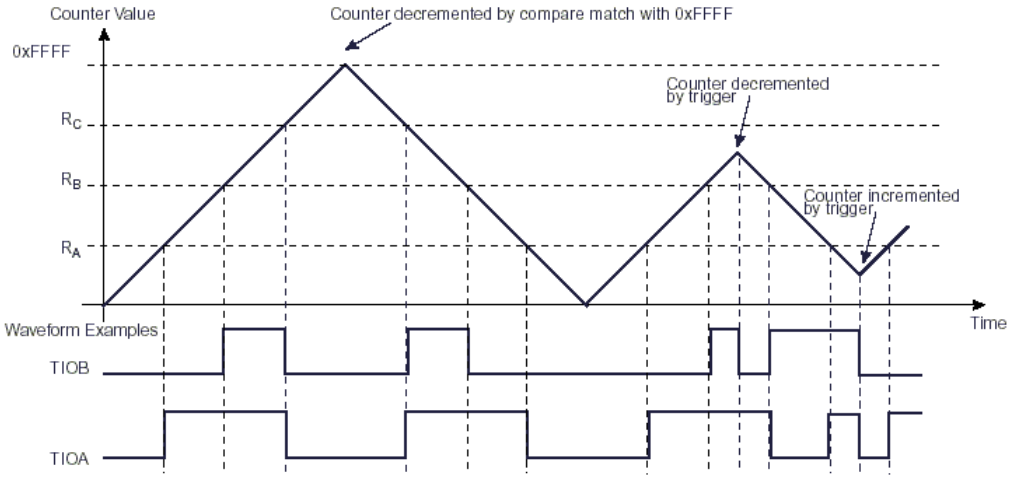


Figure 55. TC WAVESEL = 01 with Trigger



When **WAVSEL = 11**, the value of TC_CV is incremented from 0 to RC. Once RC is reached, the value of TC_CV is decremented to 0, then re-incremented to RC and so on. A trigger such as an external event or a software trigger can modify TC_CV at any time. If a trigger occurs while TC_CV is incrementing, TC_CV then decrements. If a trigger is received while TC_CV is decrementing, TC_CV then increments.

RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

Figure 56. TC WAVESEL = 11 without Trigger

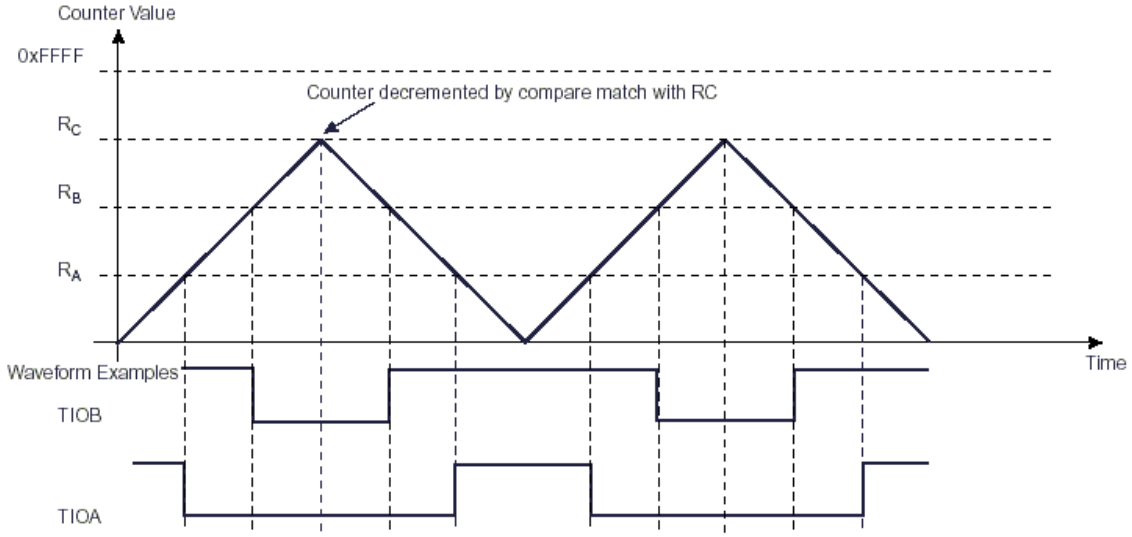


Figure 57. TC WAVESEL = 11 with Trigger

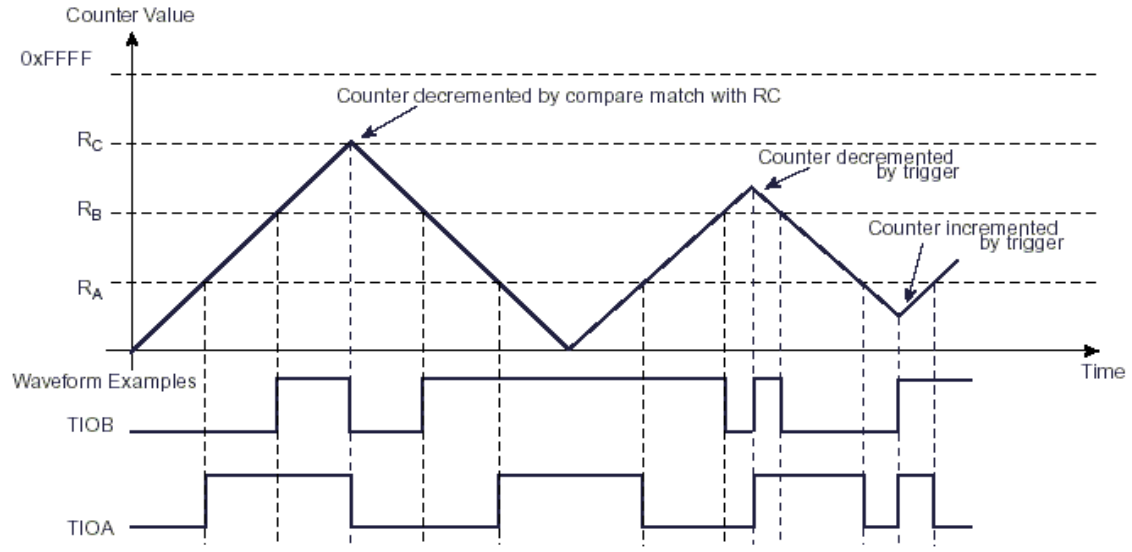
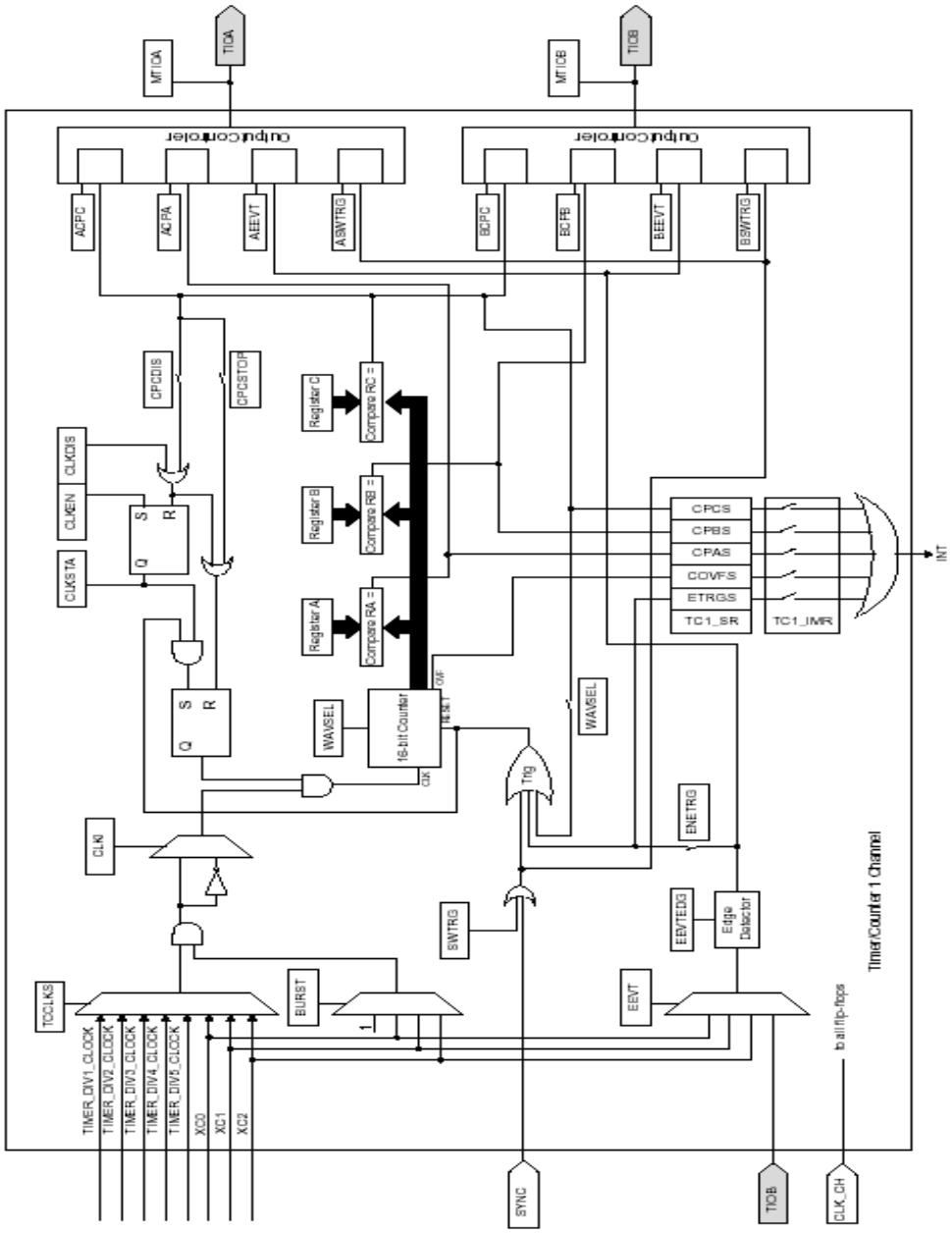


Figure 58. TC Single Channel in WAVE Mode



TC External Event/Trigger Conditions

An external event can be programmed to be detected on one of the clock sources (XC0, XC1, XC2) or TIOB. The external event selected can then be used as a trigger.

The parameter EEVT in TC_CMR selects the external trigger. The parameter EEVT-EDG defines the trigger edge for each of the possible external triggers (rising, falling or both). If EEVTEDG is cleared (none), no external event is defined.

If TIOB is defined as an external event signal (EEVT = 0), TIOB is no longer used as output and the TC1 channel can only generate a waveform on TIOA.

When an external event is defined, it can be used as a trigger by setting bit ENETRG in TC_CMR.

As in Capture Mode, the SYNC signal and the software trigger are also available as triggers.

RC Compare can also be used as a trigger depending on the parameter WAVSEL.

TC Output Controller

The output controller defines the output level changes on TIOA and TIOB following an event.

TIOB control is used only if TIOB is defined as output (not as an external event).

The following events control TIOA and TIOB: software trigger, external event and RC compare.

RA compare controls TIOA and RB compare controls TIOB. Each of these events can be programmed to set, clear or toggle the output as defined in the corresponding parameter in TC_CMR.

The tables below show which parameter in TC_CMR is used to define the effect of each event.

Parameter	TIOA Event
ASWTRG	Software trigger
AEEVT	External event
ACPC	RC compare
ACPA	RA compare

Parameter	TIOB Event
BSWTRG	Software trigger
BEEVT	External event
BCPC	RC compare
BCPB	RB compare

If two or more events occur at the same time, the priority level is defined as follows:

1. Software trigger
2. External event
3. RC compare
4. RA or RB compare

The following bits in the Status Register TC_SR are significant in Waveform Mode:

- CPAS: RA Compare Status

There has been a RA Compare match at least once since the last read of the status

- CPBS: RB Compare Status

There has been a RB Compare match at least once since the last read of the status

- CPCS: RC Compare Status

There has been a RC Compare match at least once since the last read of the status

- COVFS: Counter Overflow

Counter has attempted to count past 0xFFFF since the last read of the status

- ETRGS: External Trigger

External trigger has been detected since the last read of the status.

TC Registers

Base Address 0xFFFE C000

Table 55. TC Global Register Map

Offset	Register	Name	Access	Reset State
0x00	TC Channel 0	See Table 56		
0x40	TC Channel 1	See Table 56		
0x80	TC Channel 2	See Table 56		
0xC0	TC Block Control Register	TC_BCR	Write Only	-
0xC4	TC Block Mode Register	TC_BMR	Read/Write	0

Table 56. TC Channel 0-1-2 Register Map

Offset	Register	Name	Access	Reset State
0x00	Channel Control Register	TC_CCR	Write Only	-
0x04	Channel Mode Register	TC_CMR	Read/Write	0
0x10	Counter Value	TC_CV	Read/Write	0
0x14	Register A	TC_RA	Read/Write	0
0x18	Register B	TC_RB	Read/Write	0
0x1C	Register C	TC_RC	Read/Write	0
0x20	Status Register	TC_SR	Read Only	0
0x24	Interrupt Enable Register	TC_IER	Write Only	-
0x28	Interrupt Disable Register	TC_IDR	Write Only	-
0x2C	Interrupt Mask Register	TC_IMR	Read Only	0

TC Block Control Register (Global)

Register Name	TC_BCR
Access Type	Write Only
Absolute Address	0xFFFE C0C0

Bit(s) Number	Name	Length	Comments
31 downto 1	UNUSED	31	
0	SYNC: Synchro Command	1	0 = No effect. 1 = Asserts the SYNC signal which generates a software trigger simultaneously for each of the channels.

TC Block Mode Register (Global)

Register Name	TC_BMR
Access Type	Read/Write
Reset Value	See Table 56 on page 214
Absolute Address	0xFFFE C0C4

Bit(s) Number	Name	Length	Comments	
31 downto 6	UNUSED	26		
5 downto 4	TC0XC0S: Timer Channel 0 External Clock Signal 0 Selection	2		**Signal Connected to XC0**
			0 0	TCLK0 (from PIO[0])
			0 1	none
			1 0	TIOA1
			1 1	TIOA2
3 downto 2	TC1XC1S: Timer Channel 1 External Clock Signal 1 Selection	2		**Signal Connected to XC1**
			0 0	TCLK1 (from PIO[1])
			0 1	none
			1 0	TIOA0
			1 1	TIOA2

Bit(s) Number (Continued)	Name	Length	Comments	
1 downto 0	TC2XC2S: Timer Channel 2 External Clock Signal 2 Selection	2		**Signal Connected to XC2**
			0 0	TCLK2 (from PIO[2])
			0 1	none
			1 0	TIOA0
			1 1	TIOA1

TC Channel Control Register

Register Name	TC_CCR
Access Type	Write Only
Absolute Address	0xFFFE C000 – 0xFFFE C040 – 0xFFFE C080

Bit(s) Number	Name	Length	Comments
31 downto 3	UNUSED	31	
2	SWTRG: Software Trigger Command	1	0 = No effect. 1 = A software trigger is performed: the counter is reset and the clock is started.
1	CLKDIS: Counter Clock Disable Command	1	0 = No effect. 1 = Disables the clock.
0	CLKEN: Counter Clock Enable Command	1	0 = No effect. 1 = Enables the clock if CLKDIS is not 1

TC Channel Mode Register: Waveform Mode

Register Name	TC_CMR
Access Type	Read/Write
Reset Value	See Table 56 on page 214
Absolute Address	0xFFFE C004 – 0xFFFE C044 – 0xFFFE C084

Bit(s) Number	Name	Length	Comments
31 downto 30	BSWTRG: Software Trigger Effect on TIOB.	2	
29 downto 28	BEEVT:External Event Effect on TIOB	2	
27 downto 26	BCPC: RC Compare Effect on TIOB	2	0 0 none
25 downto 24	BCPB: RB Compare Effect on TIOB	2	0 1 set
23 downto 22	ASWTRG: Software Trigger Effect on TIOA	2	1 0 clear
21 downto 20	AEEVT: External Event Effect on TIOA	2	1 1 toggle
19 downto 18	ACPC: RC Compare Effect on TIOA	2	
17 downto 16	ACPA: RA Compare Effect on TIOA	2	
15	WAVE	1	0 = Waveform Mode is disabled (Capture Mode is enabled). 1 = Waveform Mode is enabled.
14 downto 13	WAVSEL: Waveform selection	2	**Edge** 0 0 UP mode without automatic trigger on RC compare 0 1 UP mode with automatic trigger on RC compare 1 0 UPDOWN mode without automatic trigger on RC compare 1 1 UPDOWN mode with automatic trigger on RC compare
12	ENETRG:External Event Trigger Enable.	1	0 = The external event has no effect on the counter and its clock. In this case, the selected external event only controls the TIOA output. 1 = The external event resets the counter and starts the counter clock.

Bit(s) Number (Continued)	Name	Length	Comments		
11 downto 10	EEVT: External Event Selection.	2		**Signal selected as external event**	**TIOB Direction**
			0 0	TIOB	input (1)
			0 1	XC0	output
			1 0	XC1	output
			1 1	XC2	output
9 downto 8	EEVTEDG: External Event Edge Selection.	2		**Edge**	
			0 0	none	
			0 1	rising edge	
			1 0	falling edge	
			1 1	each edge	
7	CPCDIS: Counter Clock Disable with RC Compare.	1	0 = Counter clock is not disabled when counter reaches RC. 1 = Counter clock is disabled when counter reaches RC.		
6	CPCSTOP: Counter Clock Stopped with RC Compare.	1	0 = Counter clock is not stopped when counter reaches RC. 1 = Counter clock is stopped when counter reaches RC.		
5 downto 4	BUSRT: Burst Signal Selection	2	0 0	The clock is not gated by an external signal.	
			0 1	XC0 is ANDed by the selected clock.	
			1 0	XC1 is ANDed by the selected clock.	
			1 1	XC2 is ANDed by the selected clock.	
3	CLKI:Clock Invert	1	0 = Counter is incremented on rising edge of the clock. 1 = Counter is incremented on falling edge of the clock.		

Bit(s) Number (Continued)	Name	Length	Comments	
2 downto 0	TCCLKS:Clock Selection	3		Clock Selected
			0 0 0	Timer_div1_clock
			0 0 1	Timer_div2_clock
			0 1 0	Timer_div3_clock
			0 1 1	Timer_div4_clock
			1 0 0	Timer_div5_clock
			1 0 1	XC0
			1 1 0	XC1
			1 1 1	XC2

1. If TIOB is chosen as the external event signal, it is configured as an input and it no longer generates waveforms.

TC Counter Value Register

Register Name	TC_CVR
Access Type	Read Only
Reset Value	See Table 56 on page 214
Absolute Address	0xFFFE C010 – 0xFFFE C050 – 0xFFFE C090

Bit(s) Number	Name	Length	Comments
31 downto 16	UNUSED	16	
15 downto 0	CV: Counter Value	16	CV contains the counter value in real time

TC Register A

Register Name	TC_RA
Access Type	Read Only if Wave=0, Read/Write if Wave=1
Reset Value	See Table 56 on page 214
Absolute Address	0xFFFE C014 – 0xFFFE C054 – 0xFFFE C094

Bit(s) Number	Name	Length	Comments
31 downto 16	UNUSED	16	
15 downto 0	RA: Register A	16	RA contains the Register A value in real time.

TC Register B

Register Name	TC_RB
Access Type	Read Only if Wave=0, Read/Write if Wave=1
Reset Value	See Table 56 on page 214
Absolute Address	0xFFFE C018 – 0xFFFE C058 – 0xFFFE C098

Bit(s) Number	Name	Length	Comments
31 downto 16	UNUSED	16	
15 downto 0	RB: Register B	16	RB contains the Register B value in real time.

TC Register C

Register Name	TC_RC
Access Type	Read/Write
Reset Value	See Table 56 on page 214
Absolute Address	0xFFFE C01C – 0xFFFE C05C – 0xFFFE C09C

Bit(s) Number	Name	Length	Comments
31 downto 16	UNUSED	16	
15 downto 0	RC: Register C	16	RC contains the Register C value in real time.

TC Status Register

Register Name	TC_SR
Access Type	Read
Reset Value	See Table 56 on page 214
Absolute Address	0xFFFE C020 – 0xFFFE C060 – 0xFFFE C0A0

Bit(s) Number	Name	Length	Comments
31 downto 19	UNUSED	13	
18	MTIOB:TIOB Mirror	1	0 = TIOB is low. If WAVE=0, it means that TIOB pin is low. If WAVE=1, it means that TIOB is driven low. 1 = TIOB is high. If WAVE=0, this means that TIOB pin is high. If WAVE=1, this means that TIOB is driven high.
17	MTIOA:TIOA Mirror	1	0 = TIOA is low. If WAVE=0, it means that TIOA pin is low. If WAVE=1, it means that TIOA is driven low. 1 = TIOA is high. If WAVE=0, it means that TIOA pin is high. If WAVE=1, it means that TIOA is driven high.
16	CLKSTA: Clock Enabling Status	1	0 = Clock is disabled. 1 = Clock is enabled.
15 downto 8	UNUSED	8	
7	ETRGS:External Trigger Status	1	0 = External trigger has not occurred since the last read of the Status Register. 1 = External trigger has occurred since the last read of the Status Register.

Bit(s) Number (Continued)	Name	Length	Comments
6	LDRBS:RB Loading Status	1	0 = RB load has not occurred since the last read of the Status Register or WAVE=1. 1 = RB load has occurred since the last read of the Status Register, if WAVE=0.
5	LDRAS: RA Loading Status	1	0 = RA load has not occurred since the last read of the Status Register or WAVE=1. 1 = RA load has occurred since the last read of the Status Register, if WAVE=0.
4	CPCS: RC Compare Status	1	0 = RC Compare has not occurred since the last read of the Status Register. 1 = RC Compare has occurred since the last read of the Status Register.
3	CPBS: RB Compare Status	1	0 = RB Compare has not occurred since the last read of the Status Register or WAVE=0. 1 = RB Compare has occurred since the last read of the Status Register, if WAVE=1.
2	CPAS: RA Compare Status	1	0 = RA Compare has not occurred since the last read of the Status Register or WAVE=0. 1 = RA Compare has occurred since the last read of the Status Register, if WAVE=1.
1	LOVRS: Load Overrun Status	1	0 = Load overrun has not occurred since the last read of the Status Register or WAVE=1. 1 = RA or RB have been loaded at least twice without any read of the corresponding register since the last read of the Status Register, if WAVE=0.
0	COVFS:Counter Overflow Status	1	0 = No counter overflow has occurred since the last read of the Status Register. 1 = A counter overflow has occurred since the last read of the Status Register.

Note: When TC_SR is read the interrupt sources (if any) are de-asserted.

TC Interrupt Enable Register

Register Name	TC_IER
Access Type	Write Only
Absolute Address	0xFFFE C024 – 0xFFFE C064 – 0xFFFE C0A4

Bit(s) Number	Name	Length	Comments
31 downto 8	UNUSED	24	
7	ETRGS:External Trigger	1	0 = No effect. 1 = Enables the External Trigger Interrupt.
6	LDRBS: RB Loading	1	0 = No effect. 1 = Enables the RB Load Interrupt.
5	LDRAS:RA Loading	1	0 = No effect. 1 = Enables the RA Load Interrupt.
4	CPCS: RC Compare	1	0 = No effect. 1 = Enables the RC Compare Interrupt.
3	CPBS: RB Compare	1	0 = No effect. 1 = Enables the RB Compare Interrupt.
2	CPAS: RA Compare	1	0 = No effect. 1 = Enables the RA Compare Interrupt.
1	LOVRS:Load Overrun	1	0 = No effect. 1 = Enables the Load Overrun Interrupt.
0	COVFS:Counter Overflow	1	0 = No effect. 1 = Enables the Counter Overflow Interrupt.

TC Interrupt Disable Register

Register Name	TC_IDR
Access Type	Write Only
Absolute Address	0xFFFE C028 – 0xFFFE C068 – 0xFFFE C0A8

Bit(s) Number	Name	Length	Comments
31 downto 8	UNUSED	24	
7	ETRGS:External Trigger	1	0 = No effect. 1 = Disables the External Trigger Interrupt.
6	LDRBS: RB Loading	1	0 = No effect. 1 = Disables the RB Load Interrupt (if WAVE=0).
5	LDRAS: RA Loading	1	0 = No effect. 1 = Disables the RA Load Interrupt (if WAVE=0).
4	CPCS: RC Compare	1	0 = No effect. 1 = Disables the RC Compare Interrupt.
3	CPBS: RB Compare	1	0 = No effect. 1 = Disables the RB Compare Interrupt (if WAVE=1).
2	CPAS: RA Compare	1	0 = No effect. 1 = Disables the RA Compare Interrupt (if WAVE=1).
1	LOVRS: Load Overrun	1	0 = No effect. 1 = Disables the Load Overrun Interrupt (if WAVE=0).
0	COVFS: Counter Overflow	1	0 = No effect. 1 = Disables the Counter Overflow Interrupt.

TC Interrupt Mask Register

Register Name	TC_IMR
Access Type	Read Only
Reset Value	See Table 56 on page 214
Absolute Address	0xFFFE C02C – 0xFFFE C06C – 0xFFFE C0AC

Bit(s) Number	Name	Length	Comments
31 downto 8	UNUSED	24	
7	ETRGS: External Trigger	1	0 = The External Trigger Interrupt is disabled. 1 = The External Trigger Interrupt is enabled.
6	LDRBS:RB Loading	1	0 = The Load RB Interrupt is disabled. 1 = The Load RB Interrupt is enabled.
5	LDRAS: RA Loading	1	0 = The Load RA Interrupt is disabled. 1 = The Load RA Interrupt is enabled.
4	CPCS: RC Compare	1	0 = The RC Compare Interrupt is disabled. 1 = The RC Compare Interrupt is enabled.
3	CPBS: RB Compare	1	0 = The RB Compare Interrupt is disabled. 1 = The RB Compare Interrupt is enabled.
2	CPAS: RA Compare	1	0 = The RA Compare Interrupt is disabled. 1 = The RA Compare Interrupt is enabled.
1	LOVRS: Load Overrun	1	0 = The Load Overrun Interrupt is disabled. 1 = The Load Overrun Interrupt is enabled.
0	COVFS: Counter Overflow	1	0 = The Counter Overflow Interrupt is disabled. 1 = The Counter Overflow Interrupt is enabled.

WD: Watchdog Timer

ARM System features an internal Watchdog Timer that can be used to guard against system lock-up if the software becomes trapped in a deadlock. The Wd is controlled through the APB.

During normal operation the user reloads the watchdog at regular intervals before the timer overflow occurs. If an overflow does occur, the watchdog timer generates one pulse on the signal WDIRQ that is connected to the AIC_IRQ(16) and a low level is driven on the NWDOVF signal for a duration of 8 x arm_clk cycles toward PIO[3] pad.

The watchdog timer has a 16-bit down counter. Bits 12-15 of the value loaded when the watchdog is restarted are programmable using the HPVC parameter in WD_CMR (Clock Mode).

Four clock sources are available to the watchdog counter from CLKGEN:

- clk_armdiv2
- clk_armdiv3
- clk_armdiv4
- clk_armdiv5

The selection between the clock sources is made using the WDCLKS parameter in WD_CMR.

The dividing factor can be from 4 to any required value. The high level period of any of these inputs must be greater than 1 CLOCK period.

The time-out period is calculated as follows:

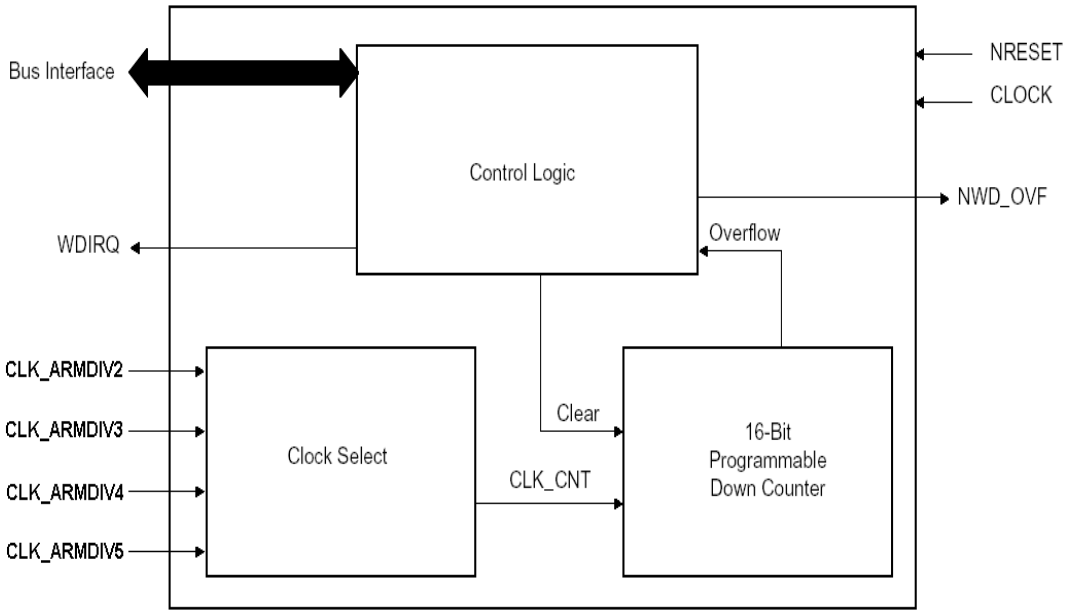
$$Period = \frac{1}{CLOCK(Hz)} \times DIVFAC \times 2^{12}(HPCV + 1)$$

Note: With HPCV in range 0 to 15.

The selection between the clock sources is made using the WDCLKS parameter in WD_CMR.

All write accesses are protected by control access keys to help prevent corruption of the watchdog should an error condition occur. To update the contents of the mode and control registers it is necessary to write the correct bit pattern to the control access key bits at the same time as the control bits are written (the same write access).

Figure 59. WD Block Diagram



WD Registers

Base Address 0xFFFF 8000

Table 57. WD Register Map

Offset	Register	Name	Access	Reset
0x00	Overflow Mode Register	WD_OMR	Read/Write	0
0x04	Clock Mode Register	WD_CMR	Read/Write	0
0x08	Control Register	WD_CR	Write Only	-
0x0C	Status Register	WD_SR	Read Only	0

WD Overflow Mode Register

Register Name	WD_OMR
Access Type	Read/Write
Reset Value	See Table 57 on page 227
Absolute Address	0xFFFF 8000

Bit(s) Number	Name	Length	Comments
31 downto 16	UNUSED	16	
15 downto 4	OKEY: Overflow Access Key (1)	11	0x234 = Write access in WD_OMR is allowed. Other value = Write access in WD_OMR is prohibited.
3	EXTEN: External Signal Enable	1	0 = Generation of a pulse on the pin NWDOVF by the watchdog is disabled. 1 = When an overflow occurs, a pulse on the pin NWDOVF is generated.
2	IRQEN: Interrupt Enable	1	0 = Generation of an interrupt by the watchdog is disabled. 1 = When an overflow occurs, the watchdog generates an interrupt.
1	RSTEN: Internal Reset Enable	1	This feature is not implemented
0	WDEN: Watch Dog Enable	1	0 = Watchdog is disabled and does not generate any signals. 1 = Watchdog is enabled and generates enabled signals.

1. Used only when writing WD_OMR. OKEY is read as 0.

WD Clock Mode Register

Register Name	WD_CMR
Access Type	Read/Writ
Reset Value	See Table 57 on page 227
Absolute Address	0xFFFF 8004

Bit(s) Number	Name	Length	Comments	
31 downto 16	UNUSED	16		
15 downto 7	CKEY: Clock Access Key (1)	11	0x06E = Write access in WD_CMR is allowed. Other value = Write access in WD_CMR is prohibited.	
6	UNUSED	1		
5 downto 2	HPCV: High Preload Counter Value	4	Counter is preloaded when watchdog counter is restarted with bits 0 to 11 set (FFF) and bits 12 to 15 equaling HPCV.	
1 downto 0	WDCLKS: Clock Selection	2		**Clock Selected**
			0 0	clk_armdiv2
			0 1	clk_armdiv3
			1 0	clk_armdiv4
			1 1	clk_armdiv5

1. Used only when writing WD_CMR. CKEY is read as 0.

WD Control Register

Register Name	WD_CR
Access Type	Write Only
Absolute Address	0xFFFF 8008

Bit(s) Number	Name	Length	Comments
31 downto 16	UNUSED	16	
15 downto 0	RSTKEY: Restart Key	16	0xC071 = Watchdog counter is restarted. Other value = No effect.

WD Status Register

Register Name	WD_S
Access Type	Read Only
Reset Value	See Table 57 on page 227
Absolute Address	0xFFFF 800

Bit(s) Number	Name	Length	Comments
31 downto 1	UNUSED	31	
0	WDOVF: Watchdog Overflow	1	0 = No watchdog overflow. 1 = A watchdog overflow has occurred since the last restart of the watchdog counter.

Note: if watchdog overflow determines an external reset through external logic this data is not significant

WD Enabling Sequence

To enable the Watchdog Timer the sequence is as follows:

- Disable the watchdog by clearing the bit WDEN by writing 0x234 to WD_OMR; this step is unnecessary if the WD is already disabled (reset state)
- Initialize the WD Clock Mode Register by writing 0x373C to WD_CMR (HPCV = 15 and WDCLKS = CLK_ARMDIV2: these values depend on user application)
- Restart the timer by writing 0xC071 to WD_CR
- Enable the watchdog by writing 0x2345 to WD_OMR (interrupt enabled: this value depends on user's application)

ADDA: Analog-to-digital / Digital-to-analog IF

ADDA block provides the interface toward 4 output + 4 input serial stereo digital audio channels (24-bit DAC and ADC).

From the ADC the ADDA receives a serial stream at programmable bit-rate frequency that is 64 x the audio sampling frequency.

ADDA transmits to the DAC a serial stream with the same characteristics of that coming from the ADC.

In addition it provides an associated interrupt toward the AIC.

The ADDA is configurable through the APB.

The ADDA peripheral features four programmable input and four programmable output lines, each of which may be directly connected to ADC and DAC converters.

The eight I/O channels have the same sampling frequency.

The ADDA can also provide a configurable edge on input bit rate clock.

The functionality of the ADDA is governed by one read/write configuration register. The configuration register provides enabling bits for DAC and ADC lines, clocks polarity control, a “busy” bit (read only) and a soft reset bit.

The soft reset has to be used to correctly initialize the interface. The ADDA is not reset by HW.

Each individual I/O channel is associated with a bit in the configuration register interface.

ADDA SYNCHRO & Interrupt

The TC is used for generating audio synchronization signals toward the ADDA through the ADDA_SRCK signal (Sample Rate ClocK). The ADDA_SRCK is the source for the D740 output line ADDA_TOPLL.

By programming the relevant bit of the configuration register is possible to configure the polarity of ADDA_TOPLL.

D740 receives ADDA_BRCK (Bit Rate Clock) and generates ADDA_WCK (Word Clock) through a counter whose reset is synchronised with ADDA_SRCK.

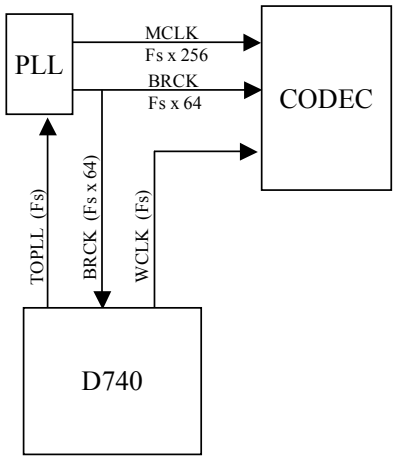
ADDA_WCK is used toward the converters both as left-right word identifier and as word synchronism. Its polarity can be programmed trough the configuration register: writing a logic 1 the output signal send a value “1” for the left output word and a value “0” for the right output word, otherwise writing a logic “0” to the relevant bit of the configuration register the output signal send a value “0” for the left output word and a value “1” for the right output word.

ADDA_BRCK frequency is 64 times the ADDA_SRCK (and ADDA_WCK) sampling frequency.

Two possible configurations for synchronism generation are possible:

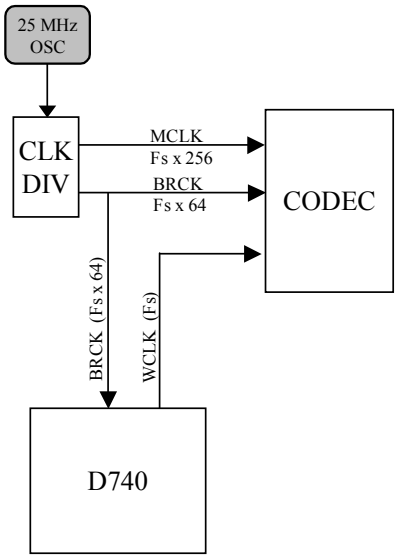
– By multiplication of the WCK (e.g. with an external PLL). The first solution using and external PLL allows the selection of a sampling frequency (Fs) independently from the system clock frequency.

Figure 60. ADDA Synchronisms from External PLL



– By division of D740 main clock to produce the BRCK and the WCK. This can be accomplished by using either an external device or the clock dividing capabilities of the Timer Counter. This solution uses a clock divider in place of an external PLL, but the system clock frequency must be a multiple of Fs.

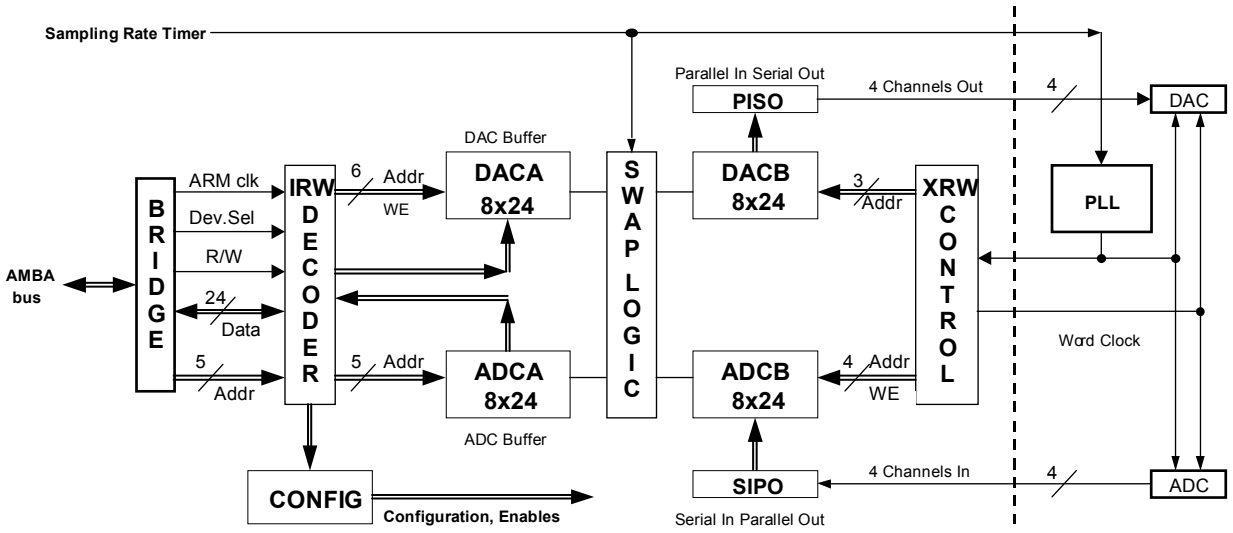
Figure 61. ADDA Synchronisms from System Clock



The ADDA interrupt (AIC_IRQ(3)) toward the AIC is provided by the TC through the signal TIMER_INT(2), based on ADDA_SRCK transitions.

ADDA Data In & Out

Figure 62. ADDA Block Diagram



Each data input and output channel is monodirectional (ADDA[3:0]_IN, ADDA[3:0]_OUT). Programming the configuration register with logic 1 to the relevant bit enables a channel input or output. Writing logic 0 to the same bit disables the channel. The configuration of the various channels can be read from the configuration register.

In order to decouple the ARM7 and the converters accesses to the registers, the double buffer technique has been used: the ADDA unit contains two sets of registers for each I/O channel. In coincidence of every interrupt, the two sets of registers are swapped and ARM can read the acquired data. The Busy Bit in the Configuration Register is active during the data swap time. Thus there are two ways of acquiring data from the ADDA interface:

1. Interrupt Based:
   - Reset ADDA
   - Set the ISR by writing the ISR function and doing the correct configuration of the AIC register
   - Enable the ISR
   - Enable ADDA

2. Polling:
   - Reset ADDA
   - Enable ADDA
   - Poll the Busy Bit: data is valid if Busy Bit = 0. The polling frequency has to be high enough to sample the level change of the Busy Bit. The Busy Bit active time is 4 times the BRCK period

The ADDA sends and receives the serial stream from the ADC and to the DAC at BRCK bit-rate frequency. The 64-bit acquired data corresponds to a pair of 32-bit samples associated to the left and right channels. The 24 MSBs of each word correspond to the bit coming from the ADC while the 8 LSB are unused (don't care).

NB: the SRCK high-level time must remain active more than a BRCK period.

The ADDA transmits a serial stream to the DACs with the same characteristics of that coming from the ADC.

Figure 63. ADDA Timing Diagram (OWCKP=0; SARK=0; PISOCKP=1; OWCKS=1; SIPCK=1)

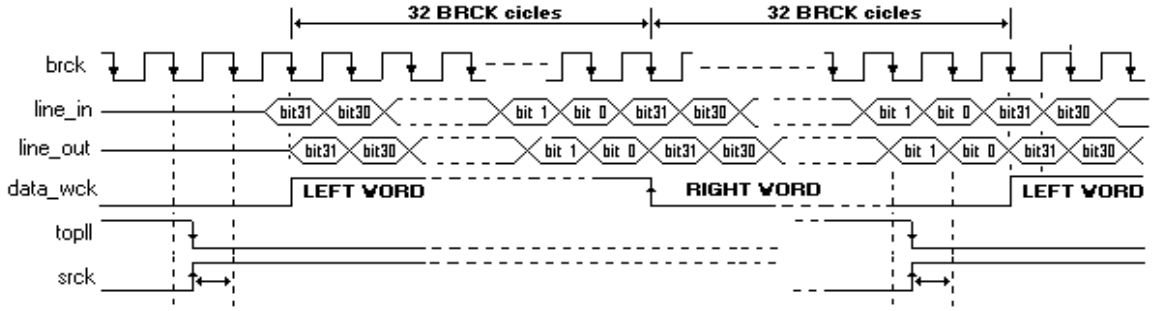
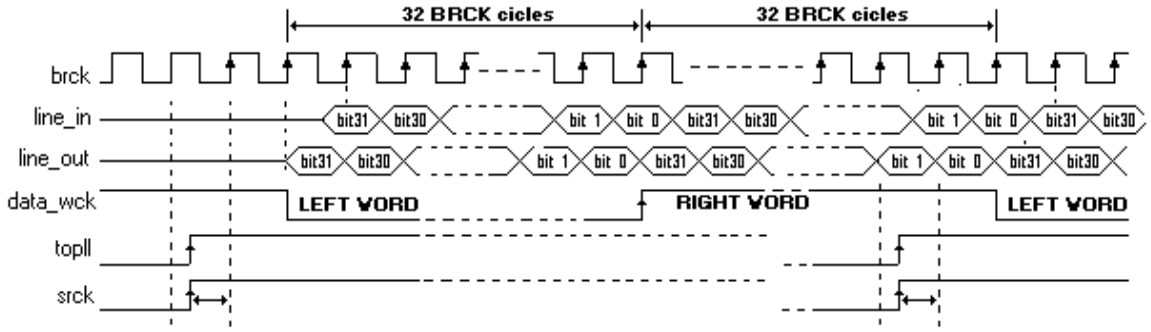


Figure 64. ADDA Timing Diagram (OWCKP=1; SARK=1; PISOCKP=0; OWCKS=0; SIPCK=0)



ADDA Registers

Base Address 0xFFF0 8000

Table 58. ADDA Register Map

Offset	Register	Name	Access	Reset State
0x00	ADDA Configuration Register	ADDA_CR	Read/Write	(No HW reset) 0x0000 0000 after SW reset
0x20	ADC Input Channel 0 LEFT	ADC_0	Read Only	No HW reset
0x24	ADC Input Channel 0 RIGHT	ADC_1		
0x28	ADC Input Channel 1 LEFT	ADC_2		
0x2C	ADC Input Channel 1 RIGHT	ADC_3		
0x30	ADC Input Channel 2 LEFT	ADC_4		
0x34	ADC Input Channel 2 RIGHT	ADC_5		
0x38	ADC Input Channel 3 LEFT	ADC_6		
0x3C	ADC Input Channel 3 RIGHT	ADC_7		
0x40	DAC Output Channel 0 LEFT	DAC_0	Write Only	No HW reset
0x44	DAC Output Channel 0 RIGHT	DAC_1		
0x48	DAC Output Channel 1 LEFT	DAC_2		
0x4C	DAC Output Channel 1 RIGHT	DAC_3		
0x50	DAC Output Channel 2 LEFT	DAC_4		
0x54	DAC Output Channel 2 RIGHT	DAC_5		
0x58	DAC Output Channel 3 LEFT	DAC_6		
0x5C	DAC Output Channel 3 RIGHT	DAC_7		

ADDA Configuration Register

Register Name	ADDA_CR
Access Type	Read/Write
Absolute Address	0xFFF0 8000

Bit(s) Number	Name	Length	Comments
31 downto 23	UNUSED	9	
22	B: Busy Bit	1	Read Only (active high). Signals data overrun.
21	SIPOCK: Input Lines Bit Rate Clock Polarity	1	0: serial data sampled on BRCK rising edge 1: serial data sampled on BRCK falling edge
20	OWCKS: Output word clock edge synchronization	1	0: WCK generated on BRCK rising edge 1: WCK generated on BRCK falling edge
19	PISOCKP: Output Lines Bit Rate Clock Polarity	1	0: serial data generated on BRCK rising edge 1: serial data generated on BRCK falling edge
18	R: Soft Reset	1	Low Active
17	SARK: Sampling Rate Clock Polarity (Topll pulse polarity)	1	0: TOPLL negate polarity of SRCK 1: TOPLL same polarity of SRCK
16	OWCKP: Output Word Clock Polarity	1	0: WCK High for left word 1: WCK Low for left word
15 downto 12	ILE: Input Stereo Line Enables	4	4 inputs 1: input enabled 0: input disabled
11 downto 8	OLE: Output Stereo Line Enables	4	4 outputs 1: output enabled 0: output disabled
7 downto 0	UNUSED	8	

ADC Channel 0 - ADC Channel 7

Register Name	ADC_0 – ADC_7
Access Type	Read Only
Absolute Address	0xFFF0 8020 - 0xFFF0 803C

Bit(s) Number	Name	Length	Comments
31 downto 8	Audio data	24	
7 downto 0	UNUSED	8	

DAC Channel 0 - DAC Channel 7

Register Name	DAC_0 – DAC_7
Access Type	Write Only
Absolute Address	0xFFF0 8040 - 0xFFF0 805C

Bit(s) Number	Name	Length	Comments
31 downto 8	Audio data	24	
7 downto 0	UNUSED	8	

CLKGEN: Clock Generator

The Clock peripheral Generator block is an APB slave.

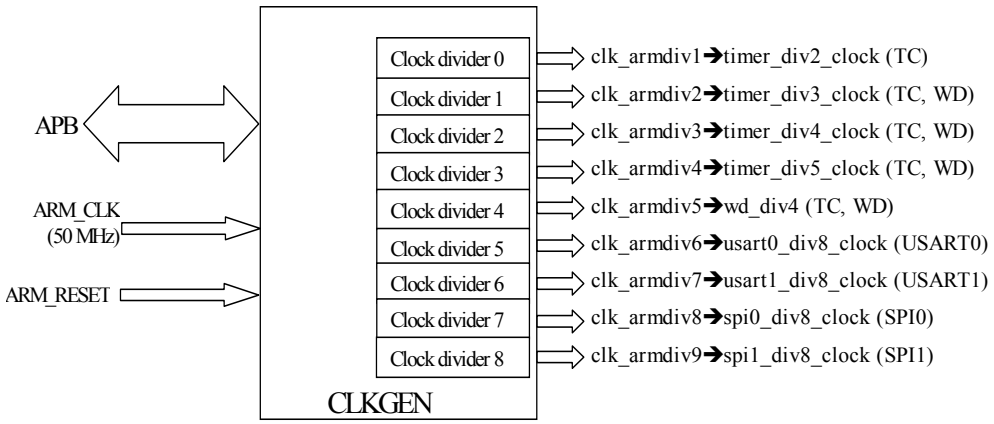
It generates divided clocks for several peripherals: the Timer Counter, the Watchdog, the USARTs and the SPIs.

The source clock is the arm_clk.

This clock is divided by integers, which can be set by software: the dividers are stored in memory-mapped registers, written through the APB. They are coded on 11 bits (unsigned).

A 11 bit counter is used to generate the divided clocks. The divider must be 2 or an integer multiple of 2 (0 and 1 as divider values do not produce any output clock). The smallest divider is 2 (25 MHz), the greatest is 2046 (24.426 KHz). There is an enable and a disable register. When any data is written in the enable register, the output clocks are enabled (on reset the output clock are fixed at 0). When any data is written in the disable register, the output clocks are set to 0.

Figure 65. CLKGEN Block Diagram



CLKGEN Registers

Base Address 0xFFF0 0000

Table 59. CLKGEN Register Map

Offset	Register	Name	Access	Reset State
0x00	Clock Divider 0	CPT_MAX0	Read/Write	0
0x04	Clock Divider 1	CPT_MAX1	Read/Write	0
0x08	Clock Divider 2	CPT_MAX2	Read/Write	0
0x0C	Clock Divider 3	CPT_MAX3	Read/Write	0
0x10	Clock Divider 4	CPT_MAX4	Read/Write	0
0x14	Clock Divider 5	CPT_MAX5	Read/Write	0
0x18	Clock Divider 6	CPT_MAX6	Read/Write	0
0x1C	Clock Divider 7	CPT_MAX7	Read/Write	0
0x20	Clock Divider 8	CPT_MAX8	Read/Write	0
0x24	Clock Global Enable	ENABLE	Write Only	-
0x28	Clock Global Disable	DISABLE	Write Only	-

Note: For ENABLE and DISABLE register the written data is “don’t care”

CLKGEN Clock Divider n Register

Register Name CPT_MAXn (n = 0 ÷ 8)

Access Type Read/Write

Reset Value See Table 59

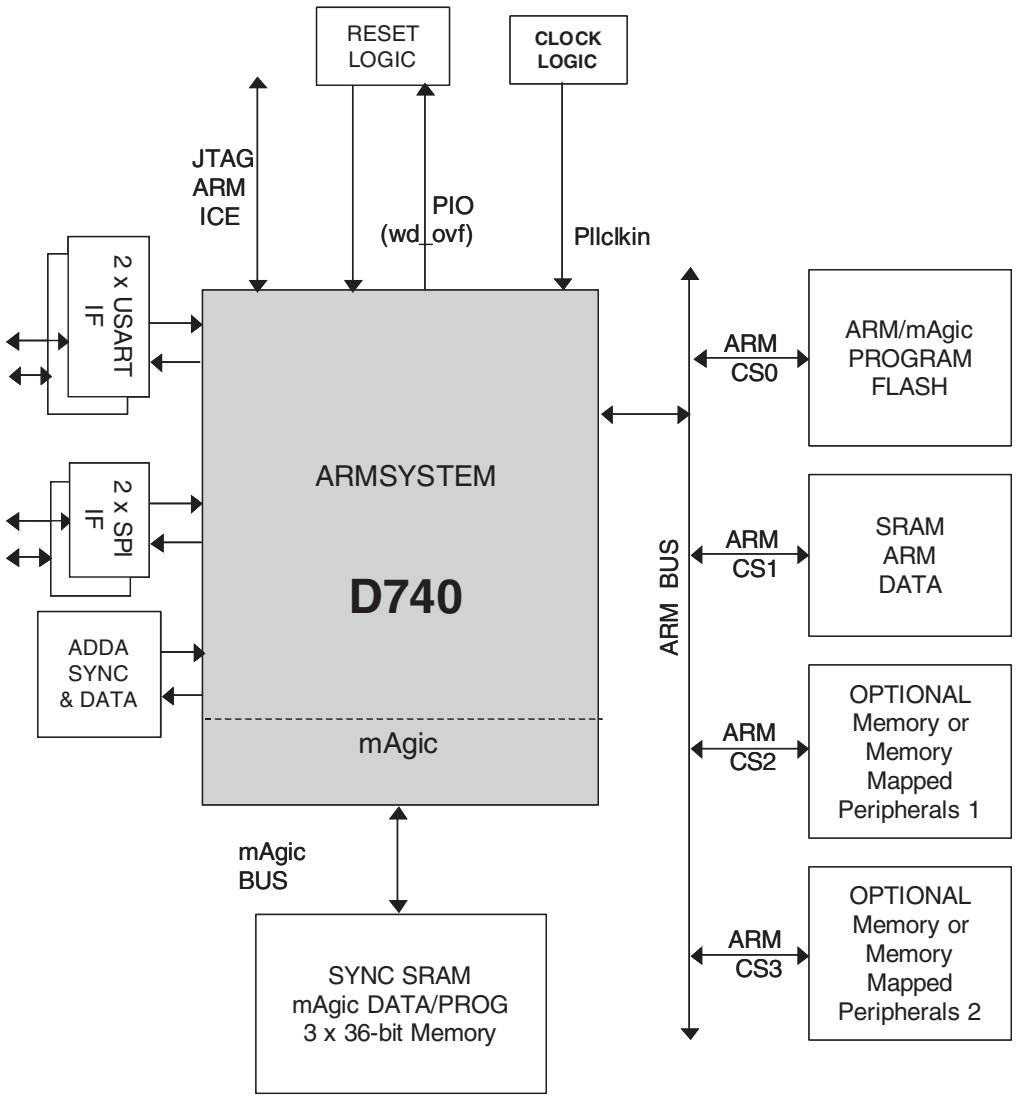
Absolute Address 0xFFF0 0000 - 0xFFF0 0020

Bit(s) Number	Name	Length	Comments
31 downto 11	UNUSED	21	
10 downto 0	CPT_MAXn: counter max value = divider	11	value of CPT_MAXn drives the clock divider Clk_armdiv(n+1): 0, 1 and all odd values = not significant

D740 System

A typical system configuration is shown in Figure 66.

Figure 66. Typical D740 System Configuration



The on-chip decoding logic allows an easy connection of external memories or memory mapped devices to the ARM processor. The optional mAgic external memory is a synchronous static memory. The operating clock can be directly supplied to D740 if an operating frequency lower than 80 MHz is required: in this case the internal PLL has to be disabled.

Bootstrap

Conventional program operation requires RAM memory at page zero to support dynamic exception vector. However it is necessary to boot from non-volatile memory at page zero. When D740 is reset, the memory map is modified to place non-volatile memory at page zero.

The on-chip RAM is remapped to address 0x00300000 and the external non-volatile memory is remapped to address 0x00000000 (See Section “EBI Remap Control Register” on page 127). The off-chip non-volatile memory is selected by the CS0 of the EBI.

The external boot memory is a 16-bit memory on NCS0). The memory map is returned to its conventional configuration by writing 1 to the RCB bit of the EBI_RCR (Remap Control Register).

This cancels the re-mapping and enables normal operation of the EBI, as programmed (See “EBI Chip Select Register” on page 126). It is not possible to re-map the memory by writing 0 to the RCB bit in EBI_RCR. The chip select addresses that are programmed, take effect when memory re-mapping is cancelled. Only NCS0 is active while the memory is remapped. Wait states take effect immediately when they are programmed to allow Boot program execution to be optimized.

Table 60. ARM External Program Memory (CSO - 16bit)

Tasks	Memory Map on Reset		Memory Map after Memory Swap	
			Branch on C main and enter THUMB state	
			Initialize memory needed by C code	
			Initialize the stacks	
			Copy the hard exception vectors in internal memory	
post remap			PtInitRemap	0x0050 004C
Jump to post remap	mov pc, r12	0x0000 003C		
	Remap instruction			
	EBI settings			
Jump address after remap	ldr r12, PtInitRemap	0x0000 0024		
	InitReset	0x0000 0020		
	FIQ Vector	0x0000 001C		
	IRQ Vector	0x0000 0018		
Boot ⟶	Undef vector	0x0000 0004		
	Reset vector: B InitReset	0x0000 0000		0x0050 0000

During boot, the number of external devices (number of active chip selects) and their configurations must be programmed as described in Section “EBI Chip Select Register” on page 126.

Reset initializes the user interface registers to their default states as defined in the peripheral sections of this datasheet and forces the ARM7TDMI to perform the next instruction fetch from address zero. Except for the program counter, the ARM core reg-

isters do not have defined reset states. When reset is active, the inputs of D740 must be held at valid logic levels. The EBI address lines drive low during reset.

Table 61. ARM Internal Memory (32-bit)

Tasks	Memory Map on Reset		Memory Map after Memory Swap	
			Internal memory available for Critical routines, Stacks or generic usage (depending on software configuration)	0x0000 07FF
				0x0000 0020
			FIQ vector	0x0000 001C
			IRQ vector	0x0000 0018
Hard exception vectors			Reserved vector	0x0000 0014
			Data ABorT Vector (DABT)	0x0000 0010
			Program AborT Vector (PABT)	0x0000 000C
			SoftWare Interrupt Vector (SWI)	0x0000 0008
			Undef vector	0x0000 0004
		0x0030 0000	Reset vector	0x0000 0000

The bootstrap sequence of the ARM is:

1. The ARM is put in ARM32 run mode
2. The program fetch starts from the boot memory at address 0x00000000; the external arm_csb[0] signal is activated
3. At this moment the Interrupt vectors are defined as follows:
   – Reset: branch to InitReset section (EBI settings, …)
   – Undef, Software Interrupt, Program Abort, Data Abort, and Reserved interrupt are not handled
   – Set the FIQ handler: ldr pc, [pc,#-0xF20] (jump to AIC_IVR/AIC_SVR)

   Note: At reset, all interrupts are disabled (AIC_IMR = 0x00000000).

4. Memorize by software the jump address for post re-mapping initialization
5. Set the EBI with the correct configuration values (See “EBI Chip Select Register” on page 126). A configuration example is reported in the following table (the

selection: Byte write access 1Meg pages, data 16-bit, 1 wait state is common to all the pages, page base addresses are different):

EBI_CSR0	0x0050 2001 (page base address 0x0050 0000)
EBI_CSR1	0x0060 2001 (page base address 0x0060 0000)
EBI_CSR2	0x0070 2001 (page base address 0x0070 0000)
EBI_CSR3	0x0080 2001 (page base address 0x0080 0000)
EBI_CSR4	0x0090 0001 (To be programmed even if not used)
EBI_CSR5	0x00A0 0001 (To be programmed even if not used)
EBI_CSR6	0x00B0 0001 (To be programmed even if not used)
EBI_CSR7	0x00C0 0001 (To be programmed even if not used)
EBI_MCR	0x0000 0000

Note: even if CSR[4:7] are not used in D740 it is necessary to write them with the proper 1Meg spacing values in the page base addresses field in order to avoid incorrect operations.

6. Execute the Re-map by writing EBI_RVR = 0x0000 0001
7. Jump to post re-mapping address

Post Re-mapping Initialization

After re-mapping, the non-volatile memory that was mapped at address 0x0 on CS0 is re-mapped to address 0x00500000, always plugged on CS0. Internal RAM is remapped on address 0x00000000. The runtime environment initialization can then be started. Typical operations are:

1. Copy of the hard exception vectors from ROM to RAM at address 0x00000000
2. Clear all the pending interrupt to avoid spurious IRQ activation
3. Initialize C runtime environment
4. Initialize the peripheral configuration registers
5. Branch on C code main function

After the ARM system initialization steps, the ARM can initialize mAgic.

Development Tools

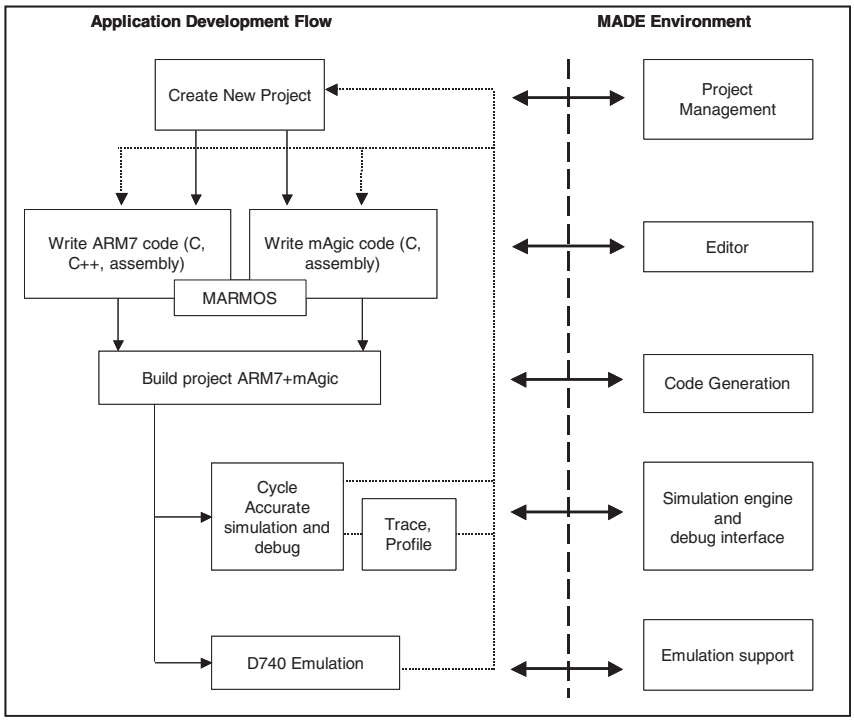
D740 is supported with a complete set of software and hardware development tools.

MADE

The D740 is supported by a set of development tools and an integrated development environment called MADE (Multicore Application Development Environment).

Figure 67 shows a typical application development flow using MADE.

Figure 67. Application Development Flow Using MADE



MADE provides the user with an integrated environment for creating, building and debugging applications for both the D740 cores, the ARM7TDMI and the mAgic DSP, by means of a common project management and support for the ECOS operating system.

Code generation tools for the ARM include the GNU Code Development Chain for ARM7 (C-C++ compiler, assembler, linker and utilities).

Code generation tools for mAgic include C compiler (GNU gcc based, ANSI compliant), VLIW assembler-optimizer, code compressor, linker and utilities.

MADE supports the ECOS operaring system, that provides the user with device drivers for DIOPSIS peripherals.

MADE provides the user with a software Cycle-Accurate simulator and supports the JTST board through serial connection.

The Cycle-Accurate simulator is a cycle-level, bit-true simulator built to replicate as accurately as possible the real behavior of the D740 system, both in cycle count and in data representation, by using an accurate description of the internal blocks and interconnections corresponding one to one with the real D740 hardware. The simulator exposes all the internal resources of the D740 system giving the user many debug facilities. It also supports very powerful application profiling. The Cycle-Accurate simulator can run at an approximate speed of 10000 instructions per second on average PC (1

GigaHz Pentium III™), thus is best suited for deeply optimizing the core kernels of the application.

MADE provides the user with a GNU GDB based interface to the JTST D740 development Board. The usual debug capabilities for the ARM System are supported, while the support for the mAgic core is limited to memory and status registers inspection.

JTAG-ICE

The ARM Standard In-Circuit-Emulation debug interface is supported via the JTAG-ICE port of the D740.

When the ARM ICE configuration is selected, the usual debug capabilities for the ARM System are supported, while the support for the mAgic core is limited to memory and status registers inspection.

The 5 jtag pins are shared between ARM7TDMI ICE functionality and the D740 chip Boundary Scan Logic. The “JCFG” pin acts as ARM jtag / D740 BSL selector. When “JCFG” pin is high the ARM ICE is selected, while D740 BSL is selected when “JCFG” is low.

JTST

JTST is a low cost general-purpose module that provides the appropriate resources in order to test D740.

JTST provides the following resources to D740:

- mAgic SSRAM, ARM FLASH and SRAM
- 4 Stereo Audio 20 bit CODECs
- 1 USB 2.0 Full (12 Mbps)
- 2 RS232/LVTTL a/synchronous serial I/O lines
- 2 SPI serial I/O lines
- Reset Logic (Power ON, Push Button, WDG)
- IO connectors (USART, SPI, USB, PIO, AUDIO)
- PLL-Clock Logic (25 MHz oscillator + CLK connector)
- DIP SWITCH & Status 7-segment Display
- Voltage Regulators 5V/3.3V & 5V/1.8V
- M-ICE JTAG

JTAG Boundary Scan

The JTAG interface is in compliant to the IEEE 1149.1-1990 standard with the following exceptions:

- only the three “required” instructions are supported (BYPASS, EXTEST, SAMPLE/PRELOAD). The six “optional” instructions (INTEST, RUNBIST, CLAMP, HIGHZ, IDCODE, USERCODE) are not supported.
- It is not possible to 3-state the TDO pin

The Boundary Scan configuration depends on the level of “JCFG” pin. The 5 jtag pins are shared between ARM7TDMI ICE functionality and the D740 chip Boundary Scan Logic. The “JCFG” pin acts as ARM jtag / D740 BSL selector. When “JCFG” pin is high the ARM ICE is selected, while D740 BSL is selected when “JCFG” is low.

The Boundary-scan Register (BSR) contains 395 bits which correspond to active pins and associated control signals.

Each D740 input pin has a corresponding bit in the Boundary-scan Register for observability.

Each D740 output pin with tristate capability has a corresponding 2-bit register in the BSR. The OUTPUT bit contains data that can be forced on the pad. The CTRL bit can put the pad into high impedance.

Each D740 output pin has a corresponding bit in the BSR that contains data that can be forced on the pad.

Each D740 in/out pin corresponds to a 2-bit register in the BSR. The OUTPUT/INPUT bit contains data that can be forced on the pad when the pad acts as output and is for the observability of data applied to the pad when the pad acts as input. The CTRL bit selects the direction of the pad. A further flip-flop, at pad level, provides the insulation between data in input and in output.

The Table 62 on page 247 all the bits of the BSR are listed.

Table 62. Boundary-scan Register

Index	Port	Function
394	RESET	input
393	BIST_RUN	input
392	BIST_RES	output
391	SCAN_EN	input
390	SCAN_TEST	input
389	XM_REQ	input
388	XM_GNT	output
387	XM_NCS	control
386		output-tristate
385	XM_NWE	control
384		output-tristate
383	XM_A[0]	control
382		output-tristate
381	XM_A[10]	control
380		output-tristate
379	XM_A[11]	control
378		output-tristate
377	XM_A[12]	control
376		output-tristate
375	XM_A[13]	control
374		output-tristate
373	XM_A[14]	control
372		output-tristate
371	XM_A[15]	control
370		output-tristate
369	XM_A[16]	control
368		output-tristate
367	XM_A[17]	control
366		output-tristate
365	XM_A[18]	control
364		output-tristate
363	XM_A[19]	control
362		output-tristate
361	XM_A[1]	control
360		output-tristate
359	XM_A[20]	control
358		output-tristate
357	XM_A[21]	control
356		output-tristate
355	XM_A[22]	control
354		output-tristate
353	XM_A[23]	control
352		output-tristate
351	XM_A[2]	control
350		output-tristate
349	XM_A[3]	control
348		output-tristate
347	XM_A[4]	control
346		output-tristate
345	XM_A[5]	control
344		output-tristate
343	XM_A[6]	control
342		output-tristate
341	XM_A[7]	control
340		output-tristate
339	XM_A[8]	control
338		output-tristate
337	XM_A[9]	control
336		output-tristate
335	XM_D[0]	control
334		bidir
333	XM_D[10]	control
332		bidir
331	XM_D[11]	control
330		bidir
329	XM_D[12]	control
328		bidir
327	XM_D[13]	control
326		bidir

Table 62. Boundary-scan Register (Continued)

Index	Port	Function
325	XM_D[14]	control
324		bidir
323	XM_D[15]	control
322		bidir
321	XM_D[16]	control
320		bidir
319	XM_D[17]	control
318		bidir
317	XM_D[18]	control
316		bidir
315	XM_D[19]	control
314		bidir
313	XM_D[1]	control
312		bidir
311	XM_D[20]	control
310		bidir
309	XM_D[21]	control
308		bidir
307	XM_D[22]	control
306		bidir
305	XM_D[23]	control
304		bidir
303	XM_D[24	control
302		bidir
301	XM_D[25]	control
300		bidir
299	XM_D[26]	control
298		bidir
297	XM_D[27]	control
296		bidir
295	XM_D[28]	control
294		bidir
293	XM_D[29]	control
292		bidir
291	XM_D[2]	control
290		bidir
289	XM_D[30]	control
288		bidir
287	XM_D[31]	control
286		bidir
285	XM_D[32]	control
284		bidir
283	XM_D[33]	control
282		bidir
281	XM_D[34]	control
280		bidir
279	XM_D[35]	control
278		bidir
277	XM_D[36]	control
276		bidir
275	XM_D[37]	control
274		bidir
273	XM_D[38]	control
272		bidir
271	XM_D[39]	control
270		bidir
269	XM_D[3]	control
268		bidir
267	XM_D[40]	control
266		bidir
265	XM_D[41]	control
264		bidir
263	XM_D[42]	control
262		bidir
261	XM_D[43]	control
260		bidir
259	XM_D[44]	control
258		bidir
257	XM_D[45]	control
256		bidir
255	XM_D[46]	control
254		bidir

Table 62. Boundary-scan Register (Continued)

Index	Port	Function
253	XM_D[47]	control
252		bidir
251	XM_D[48]	control
250		bidir
249	XM_D[49]	control
248		bidir
247	XM_D[4]	control
246		bidir
245	XM_D[50]	control
244		bidir
243	XM_D[51]	control
242		bidir
241	XM_D[52]	control
240		bidir
239	XM_D[53]	control
238		bidir
237	XM_D[54]	control
236		bidir
235	XM_D[55]	control
234		bidir
233	XM_D[56]	control
232		bidir
231	XM_D[57]	control
230		bidir
229	XM_D[58]	control
228		bidir
227	XM_D[59]	control
226		bidir
225	XM_D[5]	control
224		bidir
223	XM_D[60]	control
222		bidir
221	XM_D[61]	control
220		bidir
219	XM_D[62]	control
218		bidir

Index	Port	Function
217	XM_D[63]	control
216		bidir
215	XM_D[64]	control
214		bidir
213	XM_D[65]	control
212		bidir
211	XM_D[66]	control
210		bidir
209	XM_D[67]	control
208		bidir
207	XM_D[68]	control
206		bidir
205	XM_D[69]	control
204		bidir
203	XM_D[6]	control
202		bidir
201	XM_D[70]	control
200		bidir
199	XM_D[71]	control
198		bidir
197	XM_D[72]	control
196		bidir
195	XM_D[73]	control
194		bidir
193	XM_D[74]	control
192		bidir
191	XM_D[75]	control
190		bidir
189	XM_D[76]	control
188		bidir
187	XM_D[77]	control
186		bidir
185	XM_D[78]	control
184		bidir
183	XM_D[79]	control
182		bidir

Table 62. Boundary-scan Register (Continued)

Index	Port	Function
181	XM_D[7]	control
180		bidir
179	XM_D[8]	control
178		bidir
177	XM_D[9]	control
176		bidir
175	FPU_HALT	output
174	FPU_EXC	output
173	FPU_MODE	output
172	PIO[0]	control
171		bidir
170	PIO[10]	control
169		bidir
168	PIO[11]	control
167		bidir
166	PIO[12]	control
165		bidir
164	PIO[13]	control
163		bidir
162	PIO[14]	control
161		bidir
160	PIO[15]	control
159		bidir
158	PIO[16]	control
157		bidir
156	PIO[17]	control
155		bidir
154	PIO[18]	control
153		bidir
152	PIO[19]	control
151		bidir
150	PIO[1]	control
149		bidir
148	PIO[20]	control
147		bidir
146	PIO[21]	control
145		bidir
144	PIO[22]	control
143		bidir
142	PIO[23]	control
141		bidir
140	PIO[24]	control
139		bidir
138	PIO[25]	control
137		bidir
136	PIO[26]	control
135		bidir
134	PIO[27]	control
133		bidir
132	PIO[2]	control
131		bidir
130	PIO[3]	control
129		bidir
128	PIO[4]	control
127		bidir
126	PIO[5]	control
125		bidir
124	PIO[6]	control
123		bidir
122	PIO[7]	control
121		bidir
120	PIO[8]	control
119		bidir
118	PIO[9]	control
117		bidir
116	ARM_A[0]	output
115	ARM_A[10]	output
114	ARM_A[11]	output
113	ARM_A[12]	output
112	ARM_A[13]	output
111	ARM_A[14]	output

Table 62. Boundary-scan Register (Continued)

Index	Port	Function
110	ARM_A[15]	output
109	ARM_A[16]	output
108	ARM_A[17]	output
107	ARM_A[18]	output
106	ARM_A[1]	output
105	ARM_A[2]	output
104	ARM_A[3]	output
103	ARM_A[4]	output
102	ARM_A[5]	output
101	ARM_A[6]	output
100	ARM_A[7]	output
99	ARM_A[8]	output
98	ARM_A[9]	output
97	ARM_D[0]	control
96		bidir
95	ARM_D[10]	control
94		bidir
93	ARM_D[11]	control
92		bidir
91	ARM_D[12]	control
90		bidir
89	ARM_D[13]	control
88		bidir
87	ARM_D[14]	control
86		bidir
85	ARM_D[15]	control
84		bidir
83	ARM_D[1]	control
82		bidir
81	ARM_D[2]	control
80		bidir
79	ARM_D[3]	control
78		bidir
77	ARM_D[4]	control
76		bidir
75	ARM_D[5]	control
74		bidir
73	ARM_D[6]	control
72		bidir
71	ARM_D[7]	control
70		bidir
69	ARM_D[8]	control
68		bidir
67	ARM_D[9]	control
66		bidir
65	ARM_NRD	control
64		bidir
63	ARM_NWEB0	control
62		bidir
61	ARM_NWEB1	control
60		bidir
59	ARM_NCS0	output
58	ARM_NCS1	output
57	ARM_NCS2	output
56	ARM_NCS3	output
55	ADDA0_IN	input
54	ADDA1_IN	input
53	ADDA2_IN	input
52	ADDA3_IN	input
51	ADDA_BRCK	input
50	ADDA0_OUT	output
49	ADDA1_OUT	output
48	ADDA2_OUT	output
47	ADDA3_OUT	output
46	ADDA_WCK	output
45	ADDA_TOPLL	output
44	USART0_RXD	input
43	USART1_RXD	input
42	USART0_TXD	control
41		bidir

Table 62. Boundary-scan Register (Continued)

Index	Port	Function
40	USART1_TXD	control
39		bidir
38	USART0_SCK	control
37		bidir
36	USART1_SCK	control
35		bidir
34	USART1_CTS	input
33	USART1_DCD	input
32	USART1_RI	input
31	USART1_DSR	input
30	USART1_RTS	output
29	USART1_DTR	output
28	SPI0_MOSI	control
27		bidir
26	SPI1_MOSI	control
25		bidir
24	SPI0_MISO	control
23		bidir
22	SPI1_MISO	control
21		bidir
20	SPI0_SCK	control
19		bidir
18	SPI1_SCK	control
17		bidir
16	SPI0_NSS	control
15		bidir
14	SPI1_NSS	control
13		bidir
12	SPI0_NSS[1]	control
11		output-tristate
10	SPI0_NSS[2]	control
9		output-tristate
8	SPI0_NSS[3]	control
7		output-tristate
6	SPI1_NSS [1]	control
5		output-tristate
4	SPI1_NSS [2]	control
3		output-tristate
2	SPI1_NSS [3]	control
1		output-tristate
0	SINGLE	input

Absolute Maximum Ratings*

Operating Temperature (Commercial) …..... -40°C to +70°C
Storage Temperature.....……..... -60°C to +150°C
Voltage on Input Pin with respect to GND ……. -0.3V to +3.6V
Maximum Operating Voltage (VDDIO)…….... +3.6V
Maximum Operating Voltage (VDD & VDDPLL)…..... +1.95V

*NOTICE: Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC Characteristics

The following characteristics are applicable over the Operating Temperature range: TA = 0°C to +70°C, unless otherwise specified and are certified for a Junction Temperature up to TJ = 100°C.

Table 63. DC Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{DD}	DC Supply Core		1.65	1.8	1.95	V
V_{DDPLL}	DC Supply PLL		1.65	1.8	1.95	V
V_{DDIO}	DC Supply IO		3.0	3.3	3.6	V
V_{IL}	Input Low-level Voltage		-0.3		+0.8	V
V_{IH}	Input High-level Voltage		2.0		V_{DDIO} + 0.3	V
V_{OL}	Output Low-level Voltage					
	all drives	I_{OL} = 0 mA			0.2	V
	01 drives	I_{OL} = 2 mA			0.4	V
	02 drives	I_{OL} = 4 mA			0.4	V
	03 drives	I_{OL} = 8 mA			0.4	V
	04 drives	I_{OL} = 16 mA			0.4	V
V_{OH}	Output High-level Voltage					
	all drives	I_{OH} = 0 mA	V_{DDIO} - 0.2			V
	01 drives	I_{OH} = 2 mA	V_{DDIO} - 0.4			V
	02 drives	I_{OH} = 4 mA	V_{DDIO} - 0.4			V
	03 drives	I_{OH} = 8 mA	V_{DDIO} - 0.4			V
	04 drives	I_{OH} = 16 mA	V_{DDIO} - 0.4			V
I_{LEAK}	Input Leakage Current				0.4	A
I_{PULLUP}	Input Pull-Up Current	V_{DDIO} = 3.6V, V_{IN} = 0			322	A
$I_{PULLDOWN}$	Input Pull-Down Current	V_{DDIO} = 3.6V, V_{IN} = 3.6V			356	A
C_{IN}	Input Capacitance				8	pF

Note: The number codes associated to the drivers represent the strength of the output lines and the bi-directional lines (when acting as outputs), see Table 6 on page 6.

Timing Specifications

All timing values refer to the following conditions:

$V_{DD} = V_{DDPLL} = 1.8V$

$V_{DDIO} = 3.3V$

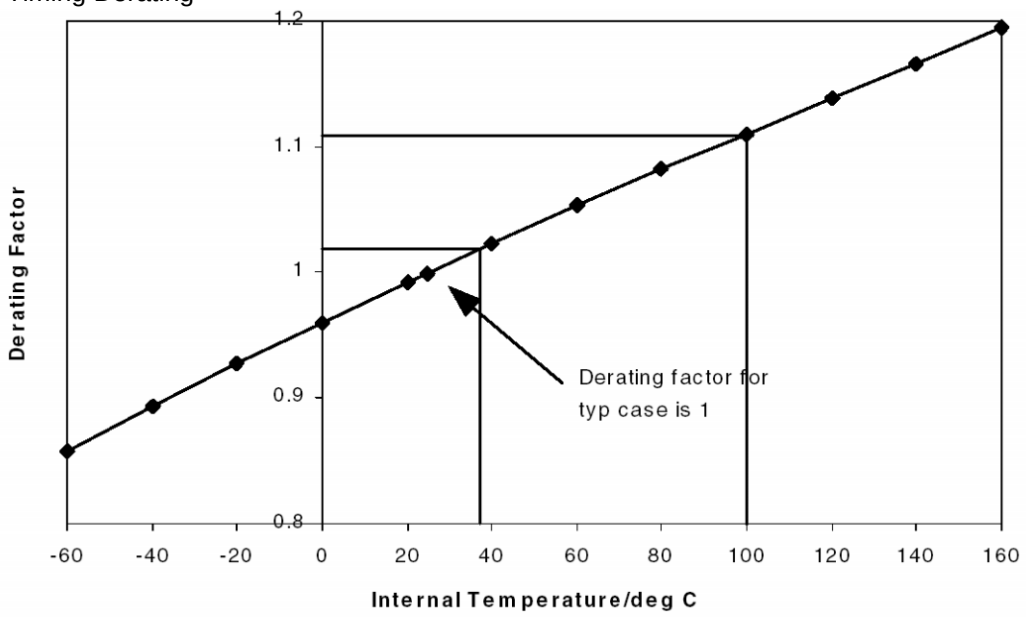
$T_A = 25°C$

$P_D = 0.5$ W

With $P_D = 0.5$ W and $T_A = 25°C$, internal junction temperature is $T_J = 37.5°C$.

In order to obtain the timing for other temperature conditions, the following figures should be used:

Figure 68. Junction Temperature Timing Derating



The derating factor is normalized to $T_J = 25°C$, so for example with $T_J = 100°C$ the timing values should be multiplied by the factor: 1.12 / 1.02 = 1.1.

Referring to the load capacitance:

$C_L = 50$ pF for all outputs and bidir, but XM_D = 22 pF, XM_Clk = 24 pF, XM_A = 52 pF.

In order to obtain the derating corresponding to loads different from these, consider the following factors (delay for each load pF) to be added (if the load is greater) or subtracted (if the load is smaller)

Table 64. External Load Timing Derating

Pad type	Derating
out-02	0.090 ns/pF
out-03	0.046 ns/pF
bi-02	0.092 ns/pF
bi-03	0.046 ns/pF

CLOCK

Symbol	Parameter	Conditions	Min	Max	Units
T_{PLLCLK}	PLL clock input period	PLL_EN = 1 (enabled)	40	50 (1)	ns
T_{PLLCLK}	PLL clock input period	PLL_EN = 0 (disabled)	10	10000 (2)	ns
T_{MCLK}	Magic Internal clock period		10	10000 (2)	ns
T_{ACLK}	ARM Internal clock period		20	20000 (2)	ns
	PLL clock input duty cycle		40	60	%

1. PLL lock minimum output frequency is 80 MHz, that corresponds to a period of 12.5 ns
2. Measured value

RESET

Symbol	Parameter	Conditions	Min	Max	Units
T_{RST}	RESET low active pulse		10 x T_{PLLCLK}		ns

ARM External Memory Read

Table 65. ARM External Memory Standard Mode Read Times (n= 0 to 8 Wait States)

Symbol	Parameter	Min	Max	Units
T_{PRD}	ARM_NRD low pulse	(n+0.5) x T_{ACLK}		ns
T_{SCS}	ARM_NCS (1) active to ARM_NRD fall	(0.5 x T_{ACLK}) – 2.9	(0.5 x T_{ACLK}) – 4.3	ns
T_{HCS}	ARM_NRD rise to ARM_NCS inactive	4.6	3.1	ns

Table 65. ARM External Memory Standard Mode Read Times (n= 0 to 8 Wait States) (Continued)

Symbol	Parameter	Min	Max	Units
T_{SADR}	ARM_A valid to ARM_NRD fall	(0.5 x T_{ACLK}) – 4.6	(0.5 x T_{ACLK}) – 6.9	ns
T_{HADR}	ARM_NRD rise to ARM_A invalid	4.5	6.6	ns
T_{SDAT}	ARM_D required setup to ARM_NRD fall	9.1		ns
T_{HDAT}	ARM_D required hold from ARM_NRD fall	0		ns

1. ARM_NCS represents one of the four ARM chip select commands ARM_NCS[0:3].

Figure 69. ARM External Memory Standard Mode Read Waveform

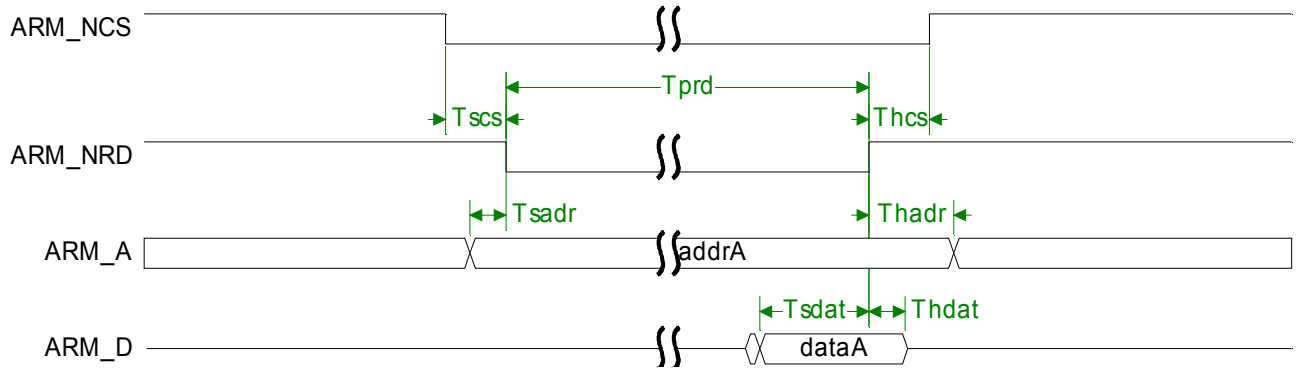
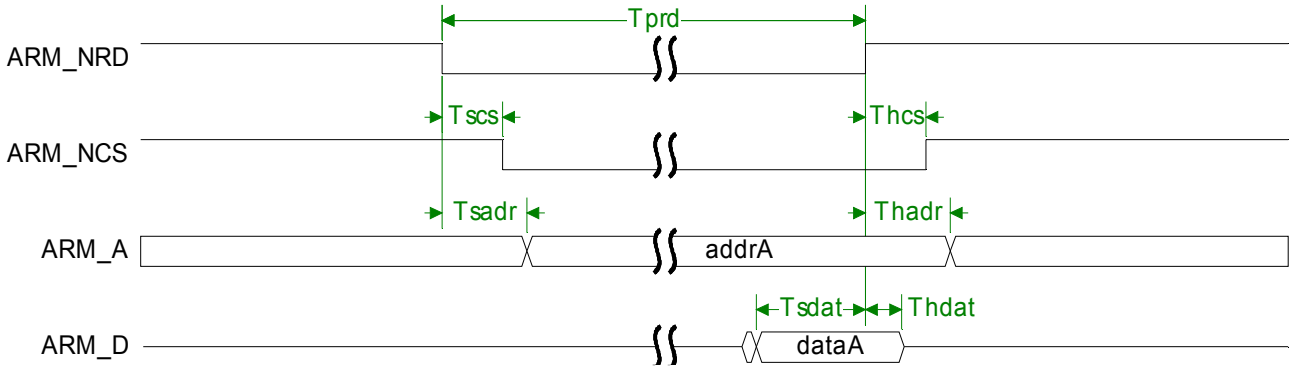


Table 66. ARM External Memory Early Mode Read Times (n= 0 to 8 Wait States)

Symbol	Parameter	Min	Max	Units
T_{PRD}	ARM_NRD low pulse	(n+1) x T_{ACLK}		ns
T_{SCS}	ARM_NCS (1) active to ARM_NRD fall	-1.6	-2.5	ns
T_{HCS}	ARM_NRD rise to ARM_NCS inactive	3.1	4.7	ns
T_{SADR}	ARM_A valid to ARM_NRD fall	-3.4	-5.1	ns
T_{HADR}	ARM_NRD rise to ARM_A invalid	4.0	6.0	ns
T_{SDAT}	ARM_D required setup to ARM_NRD fall	9.7		ns
T_{HDAT}	ARM_D required hold from ARM_NRD fall	0		ns

1. ARM_NCS represents one of the four ARM chip select commands ARM_NCS[0:3].

Figure 70. ARM External Memory Early Mode Read Waveform



ARM External Memory Write

Table 67. ARM External Memory Write Times (0 Wait States)

Symbol	Parameter	Min	Max	Units
T_{PWE}	ARM_NWEB [1] low pulse	0.5 x T_{ACLK}		ns
T_{SCS}	ARM_NCS [2] active to ARM_NWEB rise	T_{ACLK} - 4.1	T_{ACLK} - 2.8	ns
T_{HCS}	ARM_NWEB rise to ARM_NCS inactive	3.1	4.6	ns
T_{SADR}	ARM_A valid to ARM_NWEB fall	0.5 xT_{ACLK} – 6.7	0.5 xT_{ACLK} – 4.5	ns
T_{HADR}	ARM_NWEB rise to ARM_A invalid	4.3	6.3	ns
T_{SDAT}	ARM_D valid to ARM_NWEB end	0.5 xT_{ACLK} – 6.4	0.5 xT_{ACLK} – 4.3	ns
T_{HDAT}	ARM_NWEB rise to ARM_D invalid	0.5	0.7	ns

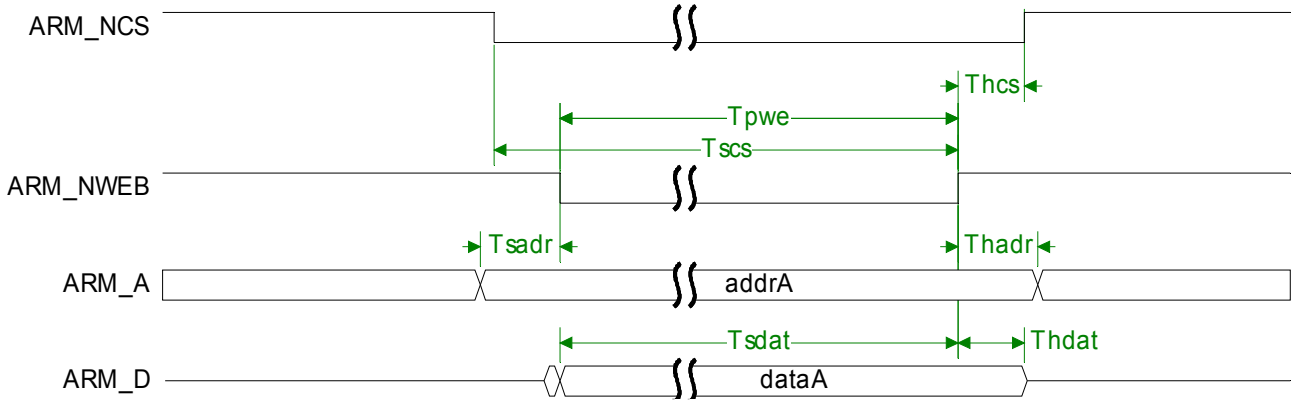
1. ARM_NWEB represents one or both the ARM byte write commands ARM_NWEB[0:1].
2. ARM_NCS represents one of the four ARM chip select commands ARM_NCS[0:3].

Table 68. ARM External Memory Write Times (n= 1 to 8 Wait States)

Symbol	Parameter	Min	Max	Units
T_{PWE}	ARM_NWEB [1] low pulse	n x T_{ACLK}		ns
T_{SCS}	ARM_NCS [2] active to ARM_NWEB rise	(n+0.5) x T_{ACLK} – 3.2	(n+0.5) x T_{ACLK} – 2.1	ns
T_{HCS}	ARM_NWEB rise to ARM_NCS inactive	0.5 xT_{ACLK} + 1.9	0.5 x T_{ACLK} + 2.8	ns
T_{SADR}	ARM_A valid to ARM_NWEB fall	0.5 xT_{ACLK} – 5.8	0.5 xT_{ACLK} – 3.9	ns
T_{HADR}	ARM_NWEB rise to ARM_A invalid	0.5 xT_{ACLK} + 3.6	0.5 xT_{ACLK} + 5.4	ns
T_{SDAT}	ARM_D valid to ARM_NWEB end	n x T_{ACLK} – 5.5	n x T_{ACLK} – 3.7	ns
T_{HDAT}	ARM_NWEB rise to ARM_D invalid	2.9	4.3	ns

1. ARM_NWEB represents one or both the ARM byte write commands ARM_NWEB[0:1].
2. ARM_NCS represents one of the four ARM chip select commands ARM_NCS[0:3].

Figure 71. ARM External Memory Write Waveform



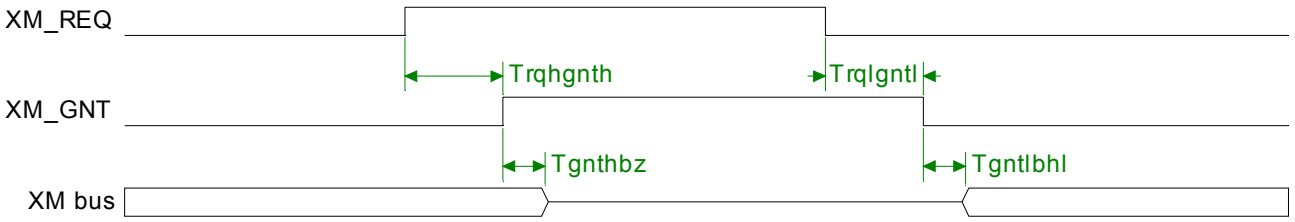
mAgic External Bus Request/Grant

Table 69. XM Bus Request-Grant Waveform

Symbol	Parameter	Min	Max	Units
$T_{RQHGNTH}$	XM_REQ active to XM_GNT active	-	3 x T_{MCLK} + 8.9	ns
T_{GNTHBZ}	XM_GNT active to XM bus (1) in hi-z	-	T_{MCLK} – 3.1	ns
$T_{RQLGNTL}$	XM_REQ inactive to XM_GNT inactive	-	3 x T_{MCLK} + 8.7	ns
$T_{GNTLBHL}$	XM_GNT inactive to XM bus (1) out of hi-z (2)	-	T_{MCLK} – 0.6	ns

1. The slowest line of XM bus that reaches hi-z state.
2. Line load contribute not included

Figure 72. XM Bus Request-Grant Waveform



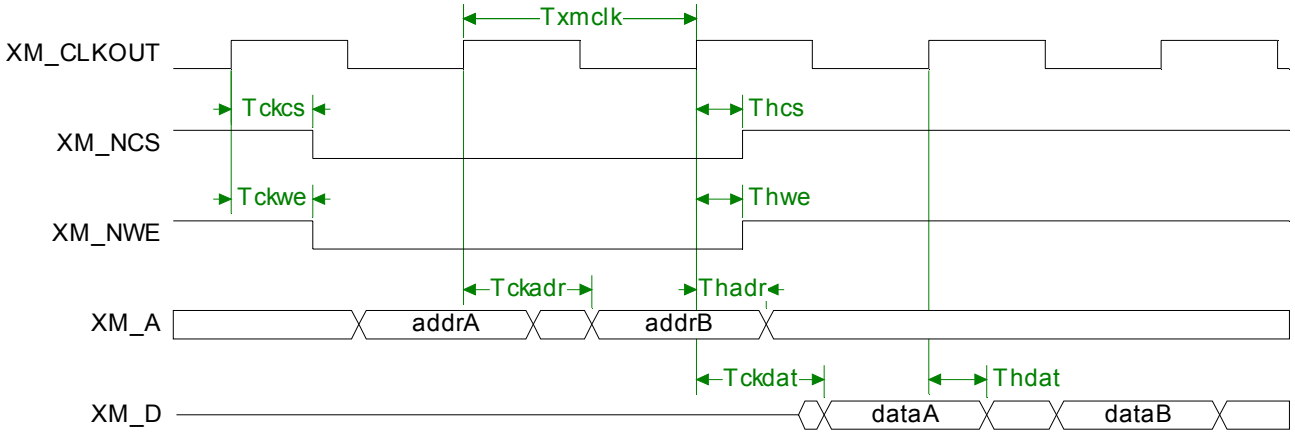
mAgic External Memory Write

Table 70. mAgic External Memory Write Times

Symbol	Parameter	Min	Max	Units
T_{XMCLK}	XM_CLKOUT (1) period	10		ns
	XM_CLKOUT duty cycle	45	55	%
T_{CKCS}	XM_CLKOUT to XM_NCS valid	2.3	3.5	ns
T_{HCS}	XM_CLKOUT to XM_NCS invalid	2.2	3.3	ns
T_{CKWE}	XM_CLKOUT to XM_NEW valid	2.3	3.5	ns
T_{HWE}	XM_CLKOUT to XM_NEW invalid	2.2	3.3	ns
T_{CKADR}	XM_CLKOUT to XM_A valid	2.9	4.3	ns
T_{HADR}	XM_CLKOUT to XM_A invalid	2.7	4.0	ns
T_{CKDAT}	XM_CLKOUT to XM_D valid	3.7	5.5	ns
T_{HDAT}	XM_CLKOUT to XM_D invalid	2.0	3.0	ns

1. XM_CLKOUT represents one of the three available external memory clock XM_CLKOUT[0:2].

Figure 73. mAgic External Memory Write Waveform



mAgic External Memory Read

Table 71. mAgic External Memory Read Times

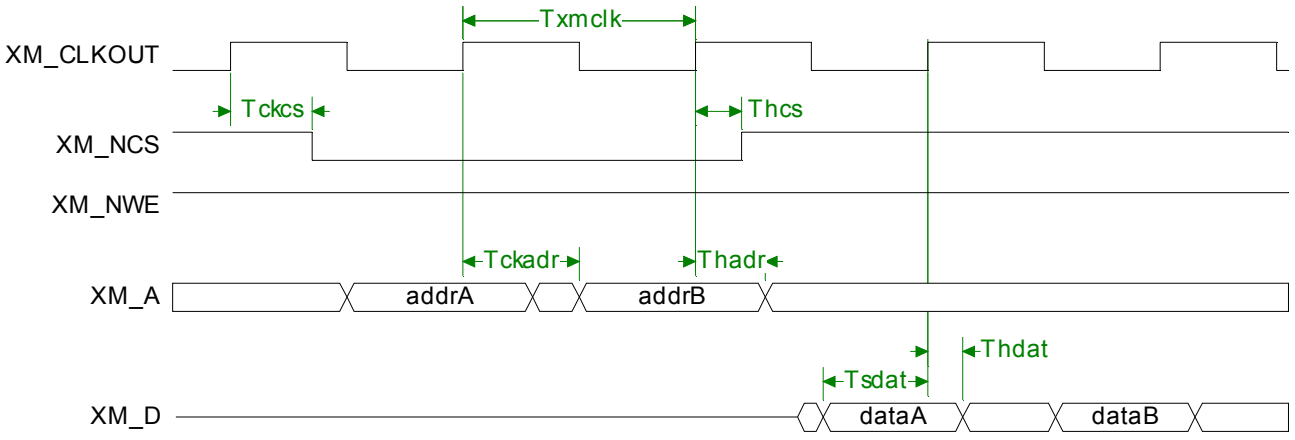
Symbol	Parameter	Min	Max	Units
T_{XMCLK}	XM_CLKOUT (1) period	10		ns
	XM_CLKOUT duty cycle	45	55	%
T_{CKCS}	XM_CLKOUT to XM_NCS valid	2.3	3.5	ns
T_{HCS}	XM_CLKOUT to XM_NCS invalid	2.2	3.3	ns

Table 71. mAgic External Memory Read Times

Symbol	Parameter	Min	Max	Units
T_{CKADR}	XM_CLKOUT to XM_A valid	2.9	4.3	ns
T_{HADR}	XM_CLKOUT to XM_A invalid	2.7	4.0	ns
T_{SDAT}	XM_D required setup time	1.0		ns
T_{HDAT}	XM_D required hold time	0.5		ns

1. XM_CLKOUT represents one of the three available external memory clock XM_CLKOUT[0:2].

Figure 74. mAgic External Memory Read Waveform



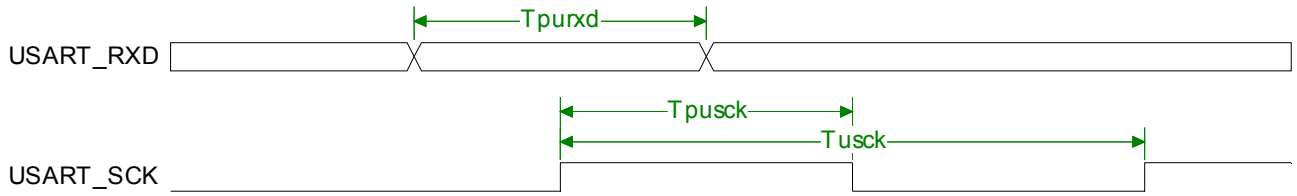
USART

Table 72. USART Sync Mode Input Times

Symbol	Parameter	Min	Max	Units
T_{PURXD}	USART_RXD (1) pulse width	5 x (T_{ACLK} / 2)		ns
T_{PUSCK}	USART_SCK (1) pulse width	5 x (T_{ACLK} / 2)		ns
T_{USCK}	USART_SCK (1) period	9 x (T_{ACLK} / 2)		ns

1. USART_RXD and USART_SCLK represent the lines of both the two USARTs.

Figure 75. USART Waveform



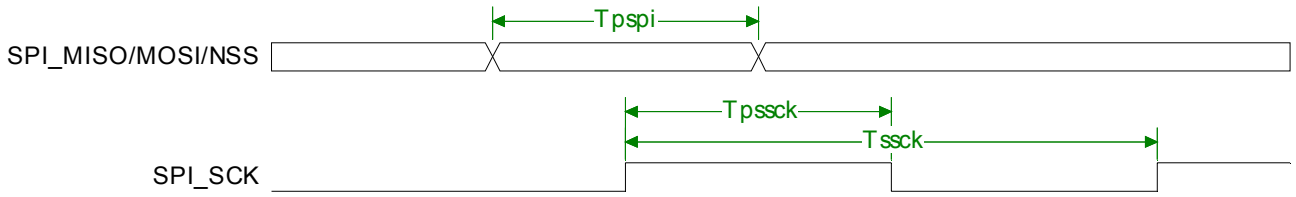
SPI

Table 73. SPI Input Times

Symbol	Parameter	Min	Max	Units
T_{PSPI}	SPI_MISO/MOSI/NSS (1) pulse width	3 x (T_{ACLK} / 2)		ns
T_{PSSCK}	SPI_SCK (1) pulse width	3 x (T_{ACLK} / 2)		ns
T_{SSCK}	SPI_SCK (1) period	5 x (T_{ACLK} / 2)		ns

1. SPI_MISO/MOSI/NSS and SPI_SCK represent the lines of both the two SPIs.

Figure 76. SPI Waveform



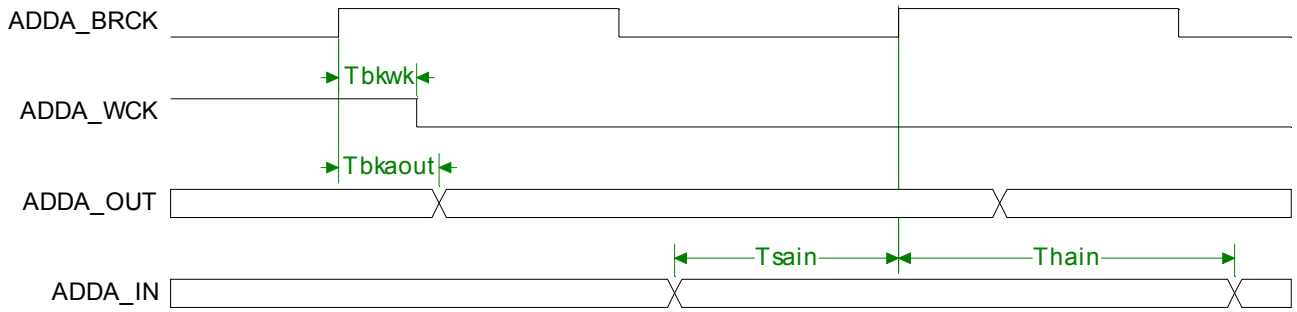
ADDA

Table 74. ADDA Times

Symbol	Parameter	Min	Max	Units
T_{BKWK}	ADDA_BRCK rise/fall (1) to ADDA_WCK rise/fall (1)	-	3 x T_{MCLK} + 8.9	ns
T_{BKAOUT}	ADDA_BRCK rise/fall (1) to ADDA_OUT (2) data valid	-	T_{MCLK} – 3.1	ns
T_{SAIN}	ADDA_IN (2) setup to ADDA_BRCK rise/fall (1)	-	3 x T_{MCLK} + 8.7	ns
T_{HAIN}	ADDA_IN (2) hold ADDA_BRCK rise/fall (1)	-	T_{MCLK} – 0.6	ns

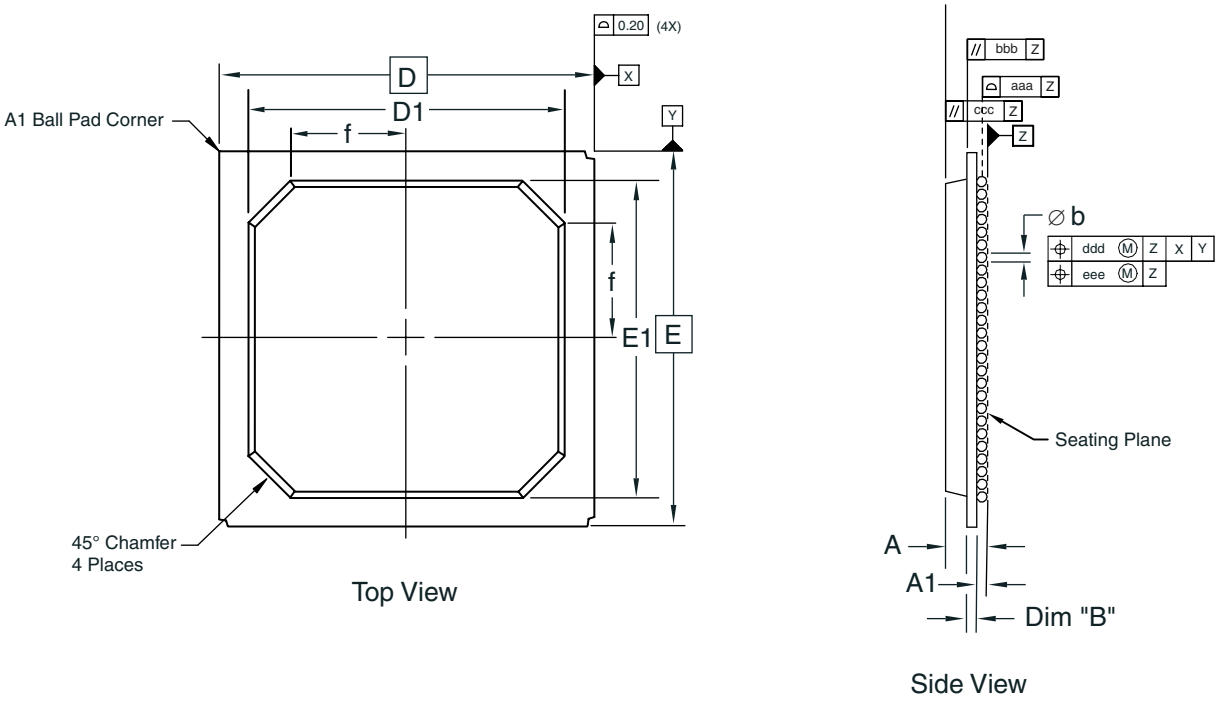
1. Rise or fall depends on ADDA configuration
2. ADDA_IN and ADDA_OUT represent one of the four ADDA input or output lines.

Figure 77. ADDA Waveform



Mechanical Drawing

Figure 78. Mechanical Drawing



COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX
A1	0.50	0.60	0.70
∅ b	0.60	0.75	0.90
aaa		0.30	
bbb		0.25	
ccc		0.35	
ddd		0.30	
eee		0.15	
A	2.12	2.33	2.56
Dim "B"	0.44	0.52	0.60
e REF		1.27	
D/E	34.8	35.0	35.2
D1/E1		30.0	30.7
f REF		11.0	
J/L REF		1.62	

Power Dissipation

The total power dissipation is the sum of two basic contributions:

$P_D = P_{IO} + P_{CORE}$

P_{IO} represents the contribute due to the IO pads current and the output load current.

P_{CORE} represents the contribute due to the internal activity current.

The following table defines the current consumption on different conditions:

Table 75. Power Dissipation

Parameters	Typical Conditions		Worst Conditions	
	Idd IO (3.3V) mA	**Idd CORE (1.8V) mA**	**Idd IO (3.3V) mA**	**Idd CORE (1.8V) mA**
Idd_{peak}	330	460	425	600
Idd_{high}	120	400	155	520
$Idd_{no\ ext}$	25	390	35	500
$Idd_{sys\ mode}$	25	100	35	135
Idd_{rst}	10	160	15	205

Idd_{peak} = mAgic FFT; both mAgic and ARM ext mem written 100% with continuous toggling data

Idd_{high} = mAgic FFT; both mAgic and ARM ext mem read and written alternatively 100% with 50% toggling data

$Idd_{no\ ext}$ = mAgic FFT; ARM FLASH access 100%; no mAgic ext mem access

$Idd_{sys\ mode}$ = mAgic in system mode; ARM FLASH accesses 100%;

Idd_{rst} = D 740 under reset

typical condition = typical process; Tj = 25°; Vdd = nom

worst condition = worst process; Tj = 100°; Vdd = nom + 10%

To estimate power consumption for a specific application use the following equation where % is the amount of time your program spends in that state and each “Idd” contribute corresponds to “IO” or “CORE” columns:

$P_{CORE} = ((\%_{peak} \times Idd_{peak}) + (\%_{high} \times Idd_{high}) + (\%_{no\ ext} \times Idd_{no\ ext}) + (\%_{sys\ mode} \times Idd_{sys\ mode}) + (\%_{rst} \times Idd_{rst})) \times 1.8$

$P_{IO} = ((\%_{peak} \times Idd_{peak}) + (\%_{high} \times Idd_{high}) + (\%_{no\ ext} \times Idd_{no\ ext}) + (\%_{sys\ mode} \times Idd_{sys\ mode}) + (\%_{rst} \times Idd_{rst})) \times 3.3$

Note: Idd_{peak} represents worst-case processor operation (for Idd IO particularly) and it is not considerable for also for hard applications where all data bits do not toggle every cycle.

Thermal Considerations

The power dissipation is given by:

$P_D = (T_J - T_A) / \theta_{JA}$ where

T_J = die temperature (°C)

T_A = ambient temperature (°C)

θ_{JA} = thermal resistance between junction and ambient (°C/W)

for 352 PBGA package (STILL AIR, tolerance 15%): θ ja = 25°C/W

So, estimated the P_D through “Power Dissipation” (See “Power Dissipation” on page 264), it is possible to get the die temperature in order to derate the timing delay values, as defined in Section “Timing Specifications” on page 255, and/or to decide if a cooling device is necessary or not.:

$T_J = T_A + (\theta_{JA} \times P_D)$

Reliability Data

The following table summarizes some basic data that can be used in reliability calculations.

Table 76. Silicon Block Size

Parameters	Data	Unit	Data	Unit
Logic Gates	585	Kgates	10.5	mm^2
Memories	12	M transistors	18	mm^2
Register File	0.3	M transistors	5.1	mm^2
total Device Die Size (pad excluded)			45	mm^2

Ordering Guide

Table 77. Ordering Information

Part Number	Temperature Range	Working Frequency	Operating Supplies	Package
AT572D740	0°C - 70°C	100 MHz	3.3V (I/O) & 1.8V (core)	352PBGA

Related Documents

1. ATMEL: ARM7TDMI (Thumb) Datasheet Rev. B (01/99)
2. ARM AMBA Specification Rev 2.0 - ARM IHI 011 (05/99)
3. ARM AMBA Specification Issue D - ARM IHI 0001D (04/97)
4. ATMEL: mAgic DSP Reference Manual - Rev. 7002A (04/04)
5. ATMEL: ARM7TDMI Bus IF - 32-bit Embedded Core Interface - Rev. 1283B (02/00)
6. ATMEL: Advanced Interrupt Controller 2 (AIC) - Rev. 1796B (03/02)
7. ATMEL: Arbiter - Rev. 1284B (12/99)
8. ATMEL: Peripheral Data Controller 2 (PDC) - Rev. 1734B (02/02)
9. ATMEL: Timer Counter 1 (TC) - Rev. 1753B (11/02/02)
10. ATMEL: Watchdog Timer (WD) - Rev. 1241A (05/99)
11. ATMEL: Parallel Input/Output 1 (PIO) - Rev. 1321C (02/01)
12. ATMEL: USART 2 - Rev. 1733C (26/02/02)
13. ATMEL: Bridge - Rev. 1286C (03/02)
14. ATMEL: External Bus IF (EBI) - Rev. 1759B (05/02)
15. ATMEL: Serial Peripheral IF (SPI) - Rev. 1244C (02/02)
16. ATMEL: ATMEL_PLL_LFT_Filter_CALCULATOR_1v8.xls

Table of Contents

Features ***1***

Description ***2***

Pin Configuration ***3***

Pin Name Conventions 5

Pin Description ***6***

Block Diagram ***10***

Architectural Overview ***11***

System Management 11

mAgic DSP Processor 12

ARM System: ARM7TDMI Processor and Peripherals 17

System 19

mAgic DSP ***20***

Overview 20

Arithmetic Data Format and Operation Types 27

Register File 28

Operators 29

Status Flags and Exceptions 37

Multiple Address Generation Unit (MAGU) 43

Memory Organization 47

Data Memory Hierarchy 52

Direct Memory Access 53

Program Sequencing 55

mAgic-ARM Interface (mAAr) 65

Instruction Set 82

ARM System ***107***

Memory Map 110

ARM7TDMI 111

PDC: Peripheral Data Controller 116

AMC-EBI: Advanced Memory Controller - External Bus Interface 124

AIC: Advanced Interrupt Controller 129

PIO: Parallel I/O Controller 143

USART: Universal Sync/Asynchronous Rx/Tx 157

SPI: Serial Peripheral Interface 189

TC: Timer Counter 205

WD: Watchdog Timer 226

ADDA: Analog-to-digital / Digital-to-analog IF 231

CLKGEN: Clock Generator 238

D740 System 240
Bootstrap 240

Development Tools 244
MADE 244
JTAG-ICE 245
JTST 245

JTAG Boundary Scan 246

Absolute Maximum Ratings 253

DC Characteristics 254
CLOCK 256
RESET 256
ARM External Memory Read 256
ARM External Memory Write 258
mAgic External Bus Request/Grant 259
mAgic External Memory Write 260
mAgic External Memory Read 260
USART 261
SPI 262
ADDA 262

Mechanical Drawing 263

Power Dissipation 264

Thermal Considerations 265

Reliability Data 266

Ordering Guide 267

Related Documents 268

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature

Disclaimer: Atmel Corporation makes no warranty for the use of its products, other than those expressly contained in the Company's standard warranty which is detailed in Atmel's Terms and Conditions located on the Company's web site. The Company assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. No licenses to patents or other intellectual property of Atmel are granted by the Company in connection with the sale of Atmel products, expressly or by implication. Atmel's products are not authorized for use as critical components in life support devices or systems.

© Atmel Corporation 2004. **All rights reserved.** Atmel® is the registered trademarks of Atmel Corporation. Diopsis™ is the trademarks of Atmel Corporation or its subsidiaries. ARM® and Thumb® are the registered trademarks of ARM Ltd.; ARM7TDMI™is the trademark of ARM Ltd.. Other terms and product names may be the trademarks of others.

Printed on recycled paper.

7001A–DPS–09/04