



Technical Data Reference Guide
netX 51/52

Step B

Hilscher Gesellschaft für Systemautomation mbH
www.hilscher.com

DOC120503TRG03EN | Revision 3 | English | 2017-08 | Released | Public

Table of contents

1	Introduction.....	5
1.1	About this document	5
1.2	List of revisions	5
1.3	References to documents	6
1.4	Legal notes.....	7
1.5	Registered trademarks.....	10
2	General description and features	11
2.1	Block diagram	11
2.2	Features	11
2.2.1	Key features	11
2.2.2	Additional features.....	13
2.3	netX 51/52 signal description	15
3	Functional description.....	19
3.1	CPU.....	19
3.1.1	Core CPU	19
3.1.2	Additional CPU xPIC	19
3.2	Memory map	20
3.3	Oscillator	20
3.4	Bootling, security memory and system LED	21
3.4.1	System LED	23
3.4.2	Boot mode and boot sequence.....	24
3.4.3	Host interface mode	25
3.4.4	Extended system information	26
3.5	Reset.....	27
3.5.1	Reset configuration	28
3.6	Watchdog	29
3.6.1	WDGACT signal	29
3.7	Chip mode selection.....	30
3.7.1	Chip mode selection netX 51.....	30
3.7.2	Chip mode selection netX 52.....	30
3.8	Internal memory	31
3.9	External memory interface	31
3.9.1	SRAM/Flash interface to memory interface controller	33
3.9.2	SDRAM interface.....	42
3.9.3	Device connection to HIF-memory	49
3.9.4	Simultaneous operation of SDRAM and parallel Flash memory at the memory interface	50
3.10	Host interface	51
3.10.1	New signal assignment netX 51/52 for extension bus mode	54
3.10.2	New signal assignment netX 51/52 for SDRAM mode	56
3.10.3	New signal assignment netX 51/52 for MII mode	59
3.10.4	New signal assignment netX 51/52 for MII and DPM mode	61
3.10.5	Signal assignment netX 50 pin-compatible mode.....	63
3.10.6	Pinout functions combinations and exclusions	65
3.10.7	Device connection to host interface.....	69
3.10.8	Multiplexing priority.....	86
3.11	Dual-port memory interface.....	87
3.11.1	Main features.....	87
3.11.2	DPM interface block diagram	89
3.11.3	DPM interface mode.....	90
3.11.4	DPM interface structure.....	90
3.11.5	Parallel DPM interface.....	92
3.11.6	Access timing without using the ready signal	105
3.11.7	DPM interrupt signals	106
3.11.8	Data mapping	110
3.11.9	External to internal address transformation	117
3.12	Serial DPM interface	130
3.12.1	Motorola SPI introduction	130
3.12.2	Serial DPM interconnection	132
3.12.3	Serial 1-bit DPM protocol (SDP2).....	133
3.12.4	Serial 4-bit DPM	136
3.12.5	Serial DPM status.....	140
3.12.6	Initializing serial DPM	141

3.13	Extension bus.....	142
3.13.1	Extension bus configuration	142
3.13.2	Extension bus address space and netX memory allocation	142
3.13.3	Address and data byte steering	142
3.13.4	Intel / Motorola data format.....	143
3.13.5	Multiplexed / Non-multiplexed data bus.....	143
3.13.6	Data ready or data acknowledge.....	143
3.13.7	End-of-cycle	143
3.13.8	Pin and signal description of extension bus.....	144
3.13.9	Extension bus component connection.....	144
3.13.10	Extension bus timing without wait states.....	145
3.13.11	Extension bus timing with wait states.....	146
3.14	Using DPM IOs as programmable IOs (PIOs)	147
3.15	Handshake registers	148
3.15.1	Mirrors of handshake cell memory INTRAMHS	151
3.15.2	DPM window mapping for handshake cells	151
3.16	IEEE 1588 system time.....	153
3.17	JTAG and ETM debug interface	154
3.17.1	JTAG interface	154
3.17.2	ETM - Embedded trace macrocell	154
3.18	Vectored interrupt controller	155
3.18.1	Interrupt generation	156
3.18.2	Interrupt priority logic	157
3.18.3	Interrupt flow sequence	157
3.19	DMA controller	158
3.19.1	Functional description	159
3.20	Multiplex matrix	162
3.21	IO-Link controller.....	163
3.21.1	Introduction.....	163
3.21.2	Typical application	164
3.22	CAN controller	165
3.22.1	Features	165
3.23	UART.....	166
3.24	USB.....	169
3.24.1	Overview	169
3.24.2	Functional description	169
3.24.3	USB timing	175
3.25	I2C interface.....	177
3.25.1	Overview	177
3.25.2	Functional description	177
3.26	QSPI/SPI interface	182
3.26.1	QSPI interface (SPI0).....	182
3.26.2	SPI interface (SPI1).....	196
3.27	MII interface	201
3.27.1	MII controller (ETH unit)	201
3.27.2	Pin assignment options	203
3.28	GPIO	204
3.28.1	Timer	206
3.29	PIO	207
3.30	Ethernet interface.....	210
3.31	Fieldbus interface.....	212
4	Electrical specification	213
4.1	Absolute max. ratings.....	213
4.2	Power-up and down sequencing.....	214
4.3	Power consumption / Power dissipation	215
4.3.1	Power consumption of netX 51 and netX 52	215
4.4	AC/DC specifications	216
4.4.1	DC parameters	216
4.4.2	System oscillator / PLL.....	219
4.4.3	Power-on reset/reset input	220
4.4.4	MMIOs.....	221
4.4.5	USB.....	223
4.4.6	PHY.....	223
4.4.7	SDRAM	225
4.4.8	SRAM / FLASH	232
4.4.9	Parallel dual-port memory timing.....	237

4.4.10	Serial dual-port memory timing.....	266
4.4.11	QSPI timing	269
4.4.12	SPI timing	272
4.4.13	I2C.....	278
4.4.14	UART.....	279
4.5	Failure rate (FIT)	280
5	netX 51 package and signal information	281
5.1	Pin table sorted by signals	281
5.2	Pin table sorted by pin number	291
5.3	Pin overview netX 51	309
5.4	Multiplex matrix signals netX 51	310
5.5	netX 51 package	313
5.6	Ordering information	313
5.7	Thermal package specification	314
5.8	Soldering conditions.....	315
5.8.1	Infrared reflow soldering characterization.....	315
5.8.2	Vapor phase reflow soldering (VPS) characterization	316
5.9	General storage conditions	317
5.10	Moisture sensitivity level	317
6	netX 52 package and signal information	318
6.1	Pin table sorted by signals	318
6.2	Pin table sorted by pin number	325
6.3	Pin overview netX 52	341
6.4	Multiplex matrix signals netX 52	342
6.5	netX 52 package	345
6.6	Ordering information	345
6.7	Thermal package specification	346
6.8	Soldering conditions.....	347
6.9	General storage conditions	347
6.10	Moisture sensitivity level	347
7	Appendix	348
7.1	PAD type explanation.....	348
7.2	netX 51/52 SDRAM bank address metal-fix issues	350
7.2.1	Interconnections to SDRAM and SRAM/FLASH	351
7.3	Terms, abbreviations and definitions	353
7.4	List of tables	357
7.5	List of figures	358
7.6	Contacts	362

1 Introduction

1.1 About this document

This document describes netX 51 and netX 52 chip functions.

1.2 List of revisions

Rev	Date	Name	Chapter	Revision
3	2017-08-11	HN/HH	3.4	Figure 2 updated to have the same resistor values for designs with or without Security EEPROM.
			3.4.1	Section <i>System LED</i> added.
			3.10.1	Correction in Table 16, Table 17 und Table 18:
			3.10.2	netX 51: Pin A13 corrected to PIO62, Pin A14 corrected to PIO59,
			3.10.3	netX 52: Pin A11 corrected to PIO62, Pin C11 corrected to PIO59.
			3.11.5	Subsections: <i>Parallel DPM Modes, Connections and Signal Mapping</i> , <i>8-bit Standard SRAM DPM Signal Mapping</i> , <i>16-bit Standard SRAM DPM Signal Mapping</i> , <i>16-bit Multiplexed Mode DPM Signal Mapping</i> moved to Design-In Guide.
			3.11.8.2.2	Two notes added in section <i>Write byte collecting (WBC)</i>
			3.12	Section <i>Serial DPM interface</i> revised.
			3.20	Section <i>Multiplex matrix</i> updated.
			3.22	Subsection <i>CAN controller register</i> moved to Programming Reference Guide.
			3.26.1	Section <i>QSPI interface (SPI0)</i> revised
			3.26.1.3	Section <i>Module organization</i> added.
			4.2	Section <i>Power-up and down sequencing</i> : Sequence added.
			4.4.7.1	Important note added in section <i>IO timing parameters for SDRAM of MEM-eMI</i> .
			4.4	Subsection <i>JTAG</i> removed.
			4.4.11	Section <i>QSPI timing</i> updated.
			5.1, 5.2	Signals SD_Ax and SD_Dx added. Signal MII3_TXCLK added. P12 corrected to OD6.
			5.6, 6.6	Ordering information updated.
			5.7, 6.7	Thermal resistance updated/added.
			5.10, 6.10	Section <i>Moisture sensitivity level</i> added.
			6.1, 6.2	Signals SD_Ax and SD_Dx added. Signal MII3_TXCLK added.
			7.3	Section <i>Terms, abbreviations and definitions</i> extended.

Table 1: List of revisions

1.3 References to documents

This document refers to the following documents:

- [1] Hilscher Gesellschaft für Systemautomation mbH: Programming Reference Guide, netX 51/52, English, 2015.
- [2] Hilscher Gesellschaft für Systemautomation mbH: Design-In Guide, netX 51/52, English, 2017.
- [3] Hilscher Gesellschaft für Systemautomation mbH: Migration Guide, netX 50 to netX 51/52, English, 2015
- [4] Hilscher Gesellschaft für Systemautomation mbH: HAL, CAN, netX 51/52, English, 2012.
- [5] Hilscher Gesellschaft für Systemautomation mbH: HAL, MII, netX 51/52, 2012
- [6] Hilscher Gesellschaft für Systemautomation mbH: HAL, IO-Link, netX 51/52, 2012
- [7] IEEE802.3 – 2002

Table 2: References to documents

1.4 Legal notes

Copyright

© Hilscher Gesellschaft für Systemautomation mbH

All rights reserved.

The images, photographs and texts in the accompanying materials (in the form of a user's manual, operator's manual, Statement of Work document and all other document types, support texts, documentation, etc.) are protected by German and international copyright and by international trade and protective provisions. Without the prior written consent, you do not have permission to duplicate them either in full or in part using technical or mechanical methods (print, photocopy or any other method), to edit them using electronic systems or to transfer them. You are not permitted to make changes to copyright notices, markings, trademarks or ownership declarations. Illustrations are provided without taking the patent situation into account. Any company names and product designations provided in this document may be brands or trademarks by the corresponding owner and may be protected under trademark, brand or patent law. Any form of further use shall require the express consent from the relevant owner of the rights.

Important notes

Utmost care was/is given in the preparation of the documentation at hand consisting of a user's manual, operating manual and any other document type and accompanying texts. However, errors cannot be ruled out. Therefore, we cannot assume any guarantee or legal responsibility for erroneous information or liability of any kind. You are hereby made aware that descriptions found in the user's manual, the accompanying texts and the documentation neither represent a guarantee nor any indication on proper use as stipulated in the agreement or a promised attribute. It cannot be ruled out that the user's manual, the accompanying texts and the documentation do not completely match the described attributes, standards or any other data for the delivered product. A warranty or guarantee with respect to the correctness or accuracy of the information is not assumed.

We reserve the right to modify our products and the specifications for such as well as the corresponding documentation in the form of a user's manual, operating manual and/or any other document types and accompanying texts at any time and without notice without being required to notify of said modification. Changes shall be taken into account in future manuals and do not represent an obligation of any kind, in particular there shall be no right to have delivered documents revised. The manual delivered with the product shall apply.

Under no circumstances shall Hilscher Gesellschaft für Systemautomation mbH be liable for direct, indirect, ancillary or subsequent damage, or for any loss of income, which may arise after use of the information contained herein.

Liability disclaimer

The hardware and/or software was created and tested by Hilscher Gesellschaft für Systemautomation mbH with utmost care and is made available as is. No warranty can be assumed for the performance or flawlessness of the hardware and/or software under all application conditions and scenarios and the work results achieved by the user when using the hardware and/or software. Liability for any damage that may have occurred as a result of using the hardware and/or software or the corresponding documents shall be limited to an event involving willful intent or a grossly negligent violation of a fundamental contractual obligation. However, the right to assert damages due to a violation of a fundamental contractual obligation shall be limited to contract-typical foreseeable damage.

It is hereby expressly agreed upon in particular that any use or utilization of the hardware and/or software in connection with

- Flight control systems in aviation and aerospace;
- Nuclear fission processes in nuclear power plants;
- Medical devices used for life support and
- Vehicle control systems used in passenger transport

shall be excluded. Use of the hardware and/or software in any of the following areas is strictly prohibited:

- For military purposes or in weaponry;
- For designing, engineering, maintaining or operating nuclear systems;
- In flight safety systems, aviation and flight telecommunications systems;
- In life-support systems;
- In systems in which any malfunction in the hardware and/or software may result in physical injuries or fatalities.

You are hereby made aware that the hardware and/or software was not created for use in hazardous environments, which require fail-safe control mechanisms. Use of the hardware and/or software in this kind of environment shall be at your own risk; any liability for damage or loss due to impermissible use shall be excluded.

Warranty

Hilscher Gesellschaft für Systemautomation mbH hereby guarantees that the software shall run without errors in accordance with the requirements listed in the specifications and that there were no defects on the date of acceptance. The warranty period shall be 12 months commencing as of the date of acceptance or purchase (with express declaration or implied, by customer's conclusive behavior, e.g. putting into operation permanently).

The warranty obligation for equipment (hardware) we produce is 36 months, calculated as of the date of delivery ex works. The aforementioned provisions shall not apply if longer warranty periods are mandatory by law pursuant to Section 438 (1.2) BGB, Section 479 (1) BGB and Section 634a (1) BGB [Bürgerliches Gesetzbuch; German Civil Code] If, despite of all due care taken, the delivered product should have a defect, which already existed at the time of the transfer of risk, it shall be at our discretion to either repair the product or to deliver a replacement product, subject to timely notification of defect.

The warranty obligation shall not apply if the notification of defect is not asserted promptly, if the purchaser or third party has tampered with the products, if the defect is the result of natural wear, was caused by unfavorable operating conditions or is due to violations against our operating regulations or against rules of good electrical engineering practice, or if our request to return the defective object is not promptly complied with.

Costs of support, maintenance, customization and product care

Please be advised that any subsequent improvement shall only be free of charge if a defect is found. Any form of technical support, maintenance and customization is not a warranty service, but instead shall be charged extra.

Additional guarantees

Although the hardware and software was developed and tested in-depth with greatest care, Hilscher Gesellschaft für Systemautomation mbH shall not assume any guarantee for the suitability thereof for any purpose that was not confirmed in writing. No guarantee can be granted whereby

the hardware and software satisfies your requirements, or the use of the hardware and/or software is uninterrupted or the hardware and/or software is fault-free.

It cannot be guaranteed that patents and/or ownership privileges have not been infringed upon or violated or that the products are free from third-party influence. No additional guarantees or promises shall be made as to whether the product is market current, free from deficiency in title, or can be integrated or is usable for specific purposes, unless such guarantees or promises are required under existing law and cannot be restricted.

Confidentiality

The customer hereby expressly acknowledges that this document contains trade secrets, information protected by copyright and other patent and ownership privileges as well as any related rights of Hilscher Gesellschaft für Systemautomation mbH. The customer agrees to treat as confidential all of the information made available to customer by Hilscher Gesellschaft für Systemautomation mbH and rights, which were disclosed by Hilscher Gesellschaft für Systemautomation mbH and that were made accessible as well as the terms and conditions of this agreement itself.

The parties hereby agree to one another that the information that each party receives from the other party respectively is and shall remain the intellectual property of said other party, unless provided for otherwise in a contractual agreement.

The customer must not allow any third party to become knowledgeable of this expertise and shall only provide knowledge thereof to authorized users as appropriate and necessary. Companies associated with the customer shall not be deemed third parties. The customer must obligate authorized users to confidentiality. The customer should only use the confidential information in connection with the performances specified in this agreement.

The customer must not use this confidential information to his own advantage or for his own purposes or rather to the advantage or for the purpose of a third party, nor must it be used for commercial purposes and this confidential information must only be used to the extent provided for in this agreement or otherwise to the extent as expressly authorized by the disclosing party in written form. The customer has the right, subject to the obligation to confidentiality, to disclose the terms and conditions of this agreement directly to his legal and financial consultants as would be required for the customer's normal business operation.

Export provisions

The delivered product (including technical data) is subject to the legal export and/or import laws as well as any associated regulations of various countries, especially such laws applicable in Germany and in the United States. The products / hardware / software must not be exported into such countries for which export is prohibited under US American export control laws and its supplementary provisions. You hereby agree to strictly follow the regulations and to yourself be responsible for observing them. You are hereby made aware that you may be required to obtain governmental approval to export, reexport or import the product.

1.5 Registered trademarks

I²C is a registered trademark of NXP Semiconductors, formerly Philips Semiconductors.

CANopen[®] is a registered trademark of CAN in AUTOMATION - International Users and Manufacturers Group e.V. (CiA), Nürnberg.

CC-Link[®] is a registered trademark of Mitsubishi Electric Corporation, Tokyo, Japan.

DeviceNet[®] and EtherNet/IP[®] are trademarks of ODVA (Open DeviceNet Vendor Association, Inc).

EtherCAT[®] is a registered trademark and a patented technology of Beckhoff Automation GmbH, Verl, Germany, formerly Elektro Beckhoff GmbH.

Modbus[®] is a registered trademark of Schneider Electric.

Powerlink is a registered trademark of B&R, Bernecker + Rainer Industrie-Elektronik Ges.m.b.H, Eggelsberg, Austria

Sercos interface[®] is a registered trademark of Sercos International e. V., Suessen, Germany.

All other mentioned trademarks are property of their respective legal owners.

2 General description and features

2.1 Block diagram

Block diagram netX 51/52

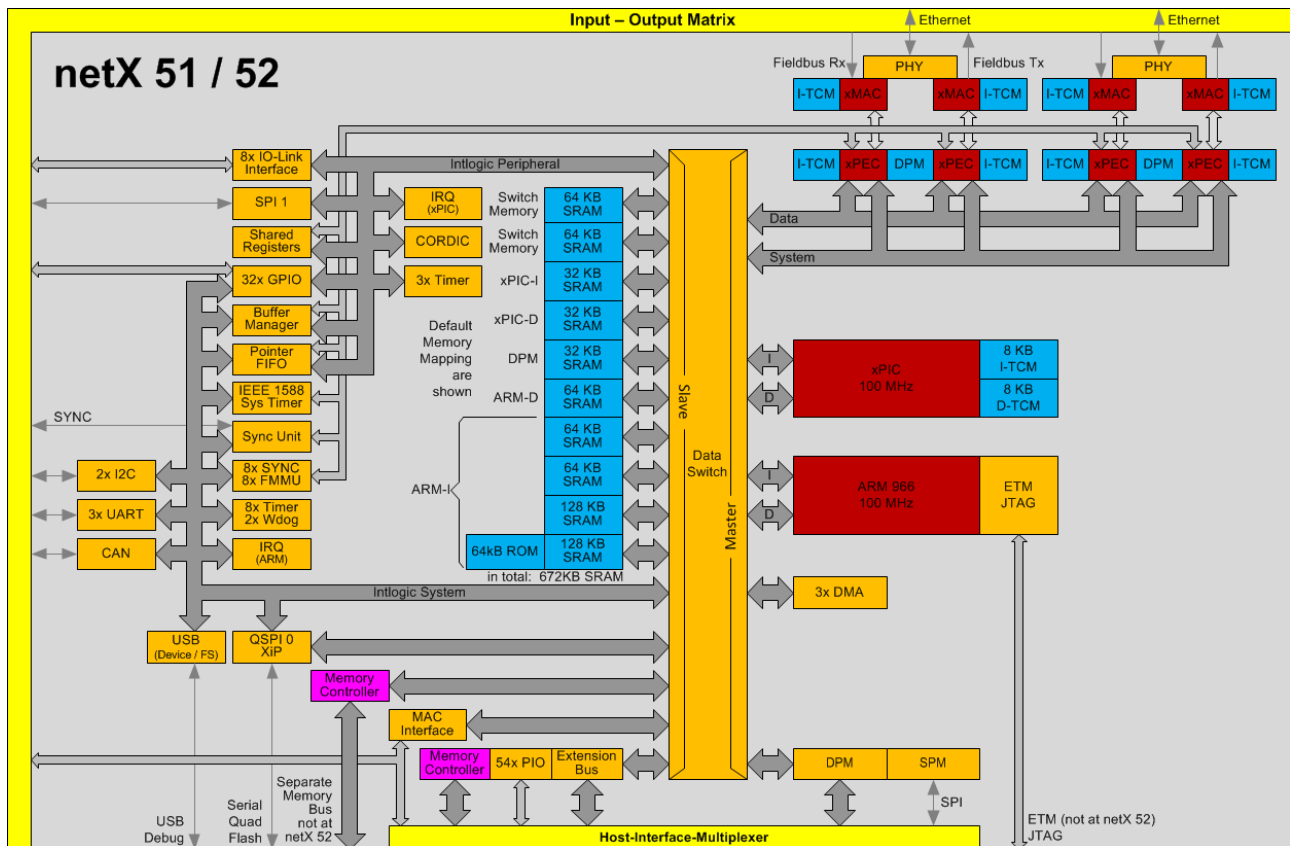


Figure 1: Block diagram of netX 51/52

2.2 Features

2.2.1 Key features

The netX 51 and 52 is an enhancement of the existing netX 50 to fulfil the increasing demands of performance and functionality of industrial networks. These controllers are supporting the PROFINET specification 2.3 with the new option of Dynamic Frame Packaging and the IO-Link version 1.1 with long telegrams.

We followed the successful strategy of the netX 10 controllers and implemented a second RISC CPU. This can be programmed by the user to work parallel with the ARM CPU handling very fast IO signals without interfering the communication tasks.

Furthermore, we increased the application interface with a dedicated CAN controller and an Ethernet MAC. Very often these communication lines were the reason to use the more expensive three channel controller netX 100.

To increase the overall performance of the netX 52 and to allow real single chip solutions the internal memory is dramatically enlarged from 96 KByte to 672 KByte.

netX	51	52
CPU	ARM 966-100 MHz	
Secondary CPU	xPIC-100 MHz	
SRAM / ROM / ITCM / DTCM [KByte]	672 / 64 / - / -	
Separate external memory bus	X	-
DPM parallel [data width]	8 / 16 / 32	
DPM serial	X	
Host interface PIOs	58	
Host interface usable as ExtMemBus/SDRAM / MAC	X / X / X	X / X / X
Communication channels	2	
Internal PHYs	2	
CAN controller/Ethernet MAC	X / X	
IO-Link master controller/channels/IO-Link version	X / 8 / V1.0, V1.1	
USB host/device	- / X	
UARTs/I2C/SPI/QSPI	3 / 2 / 1 / 1	
IEEE 1588 System Time	X	
IOs (without host interface PIOs)	40	24
Pins/package	324 / PBGA	244 / PBGA
Grid/size [mm]	1.0 mm / 19x19	0.8 mm / 15x15
Fieldbus/RTE (w/o PN IRT with DFP and FTS)	X / X	
Support PROFINET IRT with DFP	X	
Support Fast Track Switching (FTS)	X	

Table 3: Key features netX 51/52

- Significant more internal memory available to accelerate tasks
- High performance access to INTRAM blocks via TCM channels of ARM CPU
- Reduced power consumption with higher performance based on lower system clock with ARM CPU high speed memory access via two TCM channels
- Improved ARM performance on SDRAM
- Performance of communication channel doubled to support high performance RTE-systems as PROFINET IRT with dynamic frame packing or fast track switching
- xPIC as additional 100 MHz RISC CPU for time-critical tasks
- Separate CAN controller in addition to two communication channels
- Separate Ethernet MAC in addition to two communication channels (datalink layer realized by xPIC)
- New generation of Renesas' internal PHYs for shortening cut-through delays
- DPM: minimized access times, can run without wait/busy line
- Serial access to internal DPM via SPI/QSPI slave interface without interfering of ARM CPU
- Support IO-Link V1.1 specification
- Dedicated Quad SPI controller instead using internal communication controller for fast loading of program code
- Support of XiP (Execution in Place). Execution of program code directly out of serial Flash

2.2.2 Additional features

Feature	netX 51/52
VIC	32 register
DMAC	3 channels, SDRAM access supported
GPIO	system-time-compare function doubled and moved to new ARM timer/xPIC timer modules
PIO	
MMIO	- netX 51: 40 MMIOs (MMIO0...39) - netX 52: 24 MMIOs (MMIO0...23) - MMIO40...47 multiplexed with DPM pins - MMIO48 multiplexed with CLKOUT 98 functional signals mappable (see section <i>Multiplex matrix signals netX 51</i> on page 310) - Each unused MMIO usable as PIO
Pointer FIFO	32 FIFOs 3200 entries overall - Default depth per FIFO = 100
ARM Timer	2 ARM dedicated 32-bit timers - system-time read and compare - IRQ support
Systime / IEEE1588	2 independent systime units (systime_uc)
FMMU / SyncManager / BufferManager	8 FMMUs, 8 SyncMan, 8 BufMan
Trigger sample unit	New: netX 51 includes improvement for Sercos DIVCLK generation
IO-Link	- Channels: 8, version: V1.0/1.1 - Datalink layer realized via xPIC
SPI	- SPI0: - Master only (SPI or QSPI) - Quad support used for XiP - SPI1: - Master and slave - no Quad support
I2C	- Two I2C modules - PIO mode for I2C pins
UART 0...2	3
USB	- Device only - USB2JTAG integrated
CAN controller	- SJA1000 compatible - Pelican mode only (no Basic CAN mode)
Ethernet MAC	- MII Interface - Supports 10/100 Mbit FD/HD - MAC mode: connect to ext. PHY - PHY mode: direct connection to Host via MII w/o ext. PHY - Datalink layer realized via xPIC - Multiplexed to MMIO and HIF pins
Internal Dual-PHY	round-trip latency 230 ns
MIIMU (MDIO)	Separate MDIO interfaces for internal and external PHYs
CRC unit	Yes
Watchdog	Yes
CORDIC	fast coordinate transformation
Memory controller	- SRAM based device features: - Asynchronous Page Mode (APM) - Optional ready/wait signal for external wait state generation providing signal filtering and timeout logic

Feature	netX 51/52
Host interface	▪ The following netX 50-compatible DPM modes: ▪ INTEL_8BIT_SRAM ▪ INTEL_8BIT_MULTIPLEXED ▪ INTEL_16BIT_SRAM ▪ INTEL_16BIT_BYTE_WRITE ▪ INTEL_16BIT_MUL_NO_BES ▪ INTEL_32BIT_SRAM ▪ MOTOROLA_8BIT_MULTIPLEXED ▪ MOTOROLA_16BIT ▪ MOTOROLA_16BIT_68000 ▪ ▪ Additionally supported DPM modes: ▪ INTEL_16BIT_MUL_BYTE_WRITE ▪ INTEL_16BIT_MUL_2BE ▪ INTEL_16BIT_MUL_BYTE_ADDR ▪ INTEL_32BIT_BYTE_WRITE ▪ INTEL_32BIT_MUL_BYTE_ADDR ▪ INTEL_32BIT_MUL_DWORD_ADDR ▪ INTEL_32BIT_MUL_4BE ▪ INTEL_32BIT_MUL_4BE_BYTE_ADDR ▪ MOTOROLA_8BIT_6800 ▪ MOTOROLA_16BIT_MUL_BYTE_ADDR ▪ MOTOROLA_16BIT_MUL_WORD_ADDR ▪ MOTOROLA_32BIT ▪ MOTOROLA_32BIT_MUL_BYTE_ADDR ▪ MOTOROLA_32BIT_MUL_DWORD_ADDR ▪ TIOMAP_16BIT_NON_MULTIPLEXED ▪ TIOMAP_16BIT_MULTIPLEXED ▪ ISA_8BIT (3V3 signal level) ▪ ISA_16BIT (3V3 signal level) ▪ ▪ Additionally supported serial DPM via SPI/QSPI ▪ SPI modes 0...3, up to 125 MBaud ▪ ▪ Integrated SDRAM controller

Table 4: Additional features

2.3 netX 51/52 signal description

General	
PORn	Power-on reset
RSTINn	Reset input
RSTOUTn	Reset output
RDYn	RDY-LED/boot mode
RUNn	RUN-LED/boot mode
CLKOUT	Clock out
WDGACT	Watchdog active
Oscillator	
OSC_XTI	25 MHz crystal input
OSC_XTO	25 MHz crystal output
OSC_VSS	Oscillator power supply ground
OSC_VDDC	Oscillator power supply core 1.5V
JTAG	
JT_TRSTn	JTAG test reset
JT_TMS	JTAG test mode select
JT_TCLK	JTAG test clock
JT_TDI	JTAG test data input
JT_TDO	JTAG test data output
SPI	
SPI0_CLK	SPI 0 clock
SPI0_CS0n	SPI 0 chip select 0
SPI0_CS1n	SPI 0 chip select 1
SPI0_MISO	SPI 0 master input slave output data
SPI0_MOSI	SPI 0 master output slave input data
QSPI_CLK	XiP/QSPI clock
QSPI_CSn	XiP/QSPI chip select
QSPI_SIO0...3	XiP/QSPI serial IO data 0...3
I2C	
I2C_SCL	I2C serial clock line
I2C_SDA	I2C serial data line
USB	
USB_DNEG	USB D- line
USB_DPOS	USB D+ line
USB_VDDC	USB power supply core 1.5 V
USB_VDDIO	USB power supply IO 3.3 V
Test	
BSCAN_TRST	Reset boundary scan controller
TEST	Activate test mode (left open)
TMC1	Test mode 1 (left open)
TMC2	Test mode 2 (left open)
TACT_TRST	Reset test controller
MEM_IF_OM	Memory interface output mode, connect to GND for normal operating mode

MMIO	
MMIO0...48	Multiplex matrix IO 0...48
Ethernet MAC MII Interface	
MII_RXCLK	Ethernet MAC Rx clock
MII_RXD0...3	Ethernet MAC Rx data 0...3
MII_RXDV	Ethernet MAC Rx data valid
MII_RXER	Ethernet MAC Rx error
MII_TXCLK	Ethernet MAC Tx clock
MII_TXD0...3	Ethernet MAC Tx data 0...3
MII_TXEN	Ethernet MAC Tx enable
MII_TXER	Ethernet MAC Tx error
MII_COL	Ethernet MAC collision
MII_CRS	Ethernet MAC carrier sense
Memory Interface	
MEMSR_CS0...2n	SRAM chip select 0...2
MEMSR_OEn	SRAM output enable
MEMSR_WEn	SRAM write enable
MEMDR_CSn	SDRAM chip select
MEMDR_WEn	SDRAM write enable
MEMDR_RASn	SDRAM RAS
MEMDR_CASn	SDRAM CAS
MEMDR_CKE	SDRAM clock enable
MEMDR_CLK	SDRAM clock
MEM_DQM0	Memory data qualifier mask D0-7
MEM_DQM1	Memory data qualifier mask D8-15
MEM_DQM2	Memory data qualifier mask D16-23
MEM_DQM3	Memory data qualifier mask D24-31
MEM_D0...31	Memory data 0-31
MEM_A0...23	Memory address 0-23
Host interface	
DPM_A0...15	Dual-port memory address 0..15
DPM_BE1n	Dual-port memory byte high enable
DPM_BE3n	Dual-port memory byte enable 3
DPM_CSn	Dual-port memory chip select
DPM_D0...31	Dual-port memory data 0...31
DPM_DIRQn	Dual-port memory data interrupt
DPM_SIRQn	Dual-port memory sync interrupt
DPM_RDn	Dual-port memory read
DPM/RDY/DPM_BUSYn	Dual-port memory ready/busy
DPM_WRn	Dual-port memory write
SPM_MISO	Serial port memory (SPI) master input slave output data
SPM_MOSI	Serial port memory (SPI) master output slave input data
SPM_CSN	Serial port memory (SPI) chip select
SPM_CLK	Serial port memory (SPI) clock
SPM_DIRQ	Serial port memory interrupt source 1
SPM_SIRQ	Serial port memory interrupt source 2

PHY0, PHY1	
PHY0_RXN	PHY 0 receive input negative
PHY0_RXP	PHY 0 receive input positive
PHY0_TXN	PHY 0 transmit output negative
PHY0_TXP	PHY 0 transmit output positive
FO0_RD	Fiberoptic Ethernet channel 0, receive data
FO0_TD	Fiberoptic Ethernet channel 0, transmit data
FO0_EN	Fiberoptic Ethernet channel 0, enable
FO0_SD	Fiberoptic Ethernet channel 0, signal detect
PHY0_VSSAT1	PHY 0 analog ground supply
PHY0_VSSAT2	PHY 0 analog ground supply
PHY0_VSSAR	PHY 0 analog ground supply
PHY0_VDDCART	PHY 0 analog TX/RX power supply 1.5 V
PHY1_RXN	PHY 1 receive input negative
PHY1_RXP	PHY 1 receive input positive
PHY1_TXN	PHY 1 transmit output negative
PHY1_TXP	PHY 1 transmit output positive
FO1_RD	Fiberoptic Ethernet channel 1, receive data
FO1_TD	Fiberoptic Ethernet channel 1, transmit data
FO1_EN	Fiberoptic Ethernet channel 1, enable
FO1_SD	Fiberoptic Ethernet channel 1, signal detect
PHY1_VSSAT1	PHY 1 analog ground supply
PHY1_VSSAT2	PHY 1 analog ground supply
PHY1_VSSAR	PHY 1 analog ground supply
PHY1_VDDCART	PHY 1 analog TX/RX power supply 1.5 V
PHY_EXTRES	Reference resistor 12.4 k/1 %
PHY_ATP	Leave open!
PHY_VSSACP	PHY analog central ground supply
PHY_VDDCAP	PHY analog central power supply 1.5 V
PHY_VDDIOAC	PHY analog central power supply 3.3 V
PHY_VSSAT	PHY analog test ground supply
PHY_VDDIOAT	PHY analog test power supply 3.3 V
Power	
VSS	Ground supply (except PHY and oscillator)
VDDC	Power supply, core 1.5 V (except PHY and oscillator)
VDDIO	Power supply, IO buffer 3.3 V (except PHY and USB)

ETM	
ETM_TCLK	ETM trace clock
ETM_TSYNC	ETM trace synchronization
ETM_DRQ	ETM debug request
ETM_DACK	ETM debug acknowledge
ETM_PSTAT0	ETM pipe status 0
ETM_PSTAT1	ETM pipe status 1
ETM_PSTAT2	ETM pipe status 2
ETM_TPKT00	ETM trace packet 0
ETM_TPKT01	ETM trace packet 1
ETM_TPKT02	ETM trace packet 2
ETM_TPKT03	ETM trace packet 3
ETM_TPKT04	ETM trace packet 4
ETM_TPKT05	ETM trace packet 5
ETM_TPKT06	ETM trace packet 6
ETM_TPKT07	ETM trace packet 7
ETM_TPKT08	ETM trace packet 8
ETM_TPKT09	ETM trace packet 9
ETM_TPKT10	ETM trace packet 10
ETM_TPKT11	ETM trace packet 11
ETM_TPKT12	ETM trace packet 12
ETM_TPKT13	ETM trace packet 13
ETM_TPKT14	ETM trace packet 14
ETM_TPKT15	ETM trace packet 15

Table 5: netX 51/52 signal description

3 Functional description

3.1 CPU

3.1.1 Core CPU

The netX 51/52 is equipped with an ARM966E-S processor core implementing the ARMv5TE instruction set. The CPU runs on a 100 MHz system clock.

Feature	netX 51/52
Core	ARM 966
Speed	100 MHz
I-TCM	Internal SRAMs via TCM interface-accessible without wait states
D-TCM	Internal SRAMs via TCM interface-accessible without wait states
Code execution	▪ Internal SRAM ▪ External memories (via extension bus, SDRAM interface) ▪ Serial Flash via QSPI (Execution in Place)

Table 6: Core CPU

3.1.2 Additional CPU xPIC

netX 51/52 has an additional CPU called xPIC. This CPU runs with a frequency of 100 MHz and is designed to process fast IO signals in parallel to the ARM CPU with a latency time down till five clock cycles.

The xPIC is a flexible peripheral interface controller designed especially for user applications. The xPIC is a general purpose 32-bit RISC CPU with an instruction set optimized (DSP extensions) for fast and deterministic data processing as needed in many real industrial applications. The xPIC can be used as a fast CPU to filter, analyze, collect, convert and process sensor data, ranging from simple IOs to complex encoders and sensors with an analog frontend. Another range of applications is the control of any actuators ranging from simple digital and analog IOs, pumps, valves, or switches to the control of virtually any type of electric motor. Along with the integrated encoder, PWM and CORDIC units, the xPIC can be used to realize many kinds of control applications in the motion or process control field.

The xPIC is also used for the IO-Link controller and the additional third Ethernet MAC channel. In this case the xPIC is not available for user applications.

3.2 Memory map

Memory	netX 51/52 (assigned to ...)	Start address
INTRAM0	128 KByte (ARM)	0x08000000
INTRAM1	128 KByte (ARM)	0x08020000
INTRAM2	64 KByte (ARM)	0x08040000
INTRAM3	64 KByte (ARM)	0x08050000
INTRAM4	64 KByte (ARM)	0x08060000
INTRAM5	32 KByte (xPIC Instr.)	0x08070000
INTRAM6	32 KByte (xPIC data)	0x08078000
INTRAM7	64 KByte (XC0)	0x08080000
INTRAM8	64 KByte (XC1)	0x08090000
INTRAMHS	32 KByte (DPM)	0x080a0000
HIF_SDRAM	25 MByte (ARM, xPIC)	0x40000000
HIF_EXTSRAM0	64 MByte (ARM, xPIC)	0x60000000
HIF_EXTSRAM1	64 MByte (ARM, xPIC)	0x64000000
HIF_EXTSRAM2	64 MByte (ARM, xPIC)	0x68000000
HIF_EXTSRAM3	64 MByte (ARM, xPIC)	0x6c000000
SDRAM	256 MByte (ARM, xPIC)	0x80000000
EXTSRAM0	64 MByte (ARM, xPIC)	0xc0000000
EXTSRAM1	64 MByte (ARM, xPIC)	0xc4000000
EXTSRAM2	64 MByte (ARM, xPIC)	0xc8000000
BOOT-ROM	64 KByte (Bootloader)	0x080f0000
QSPI-ROM (XiP)	16 MByte (ARM, xPIC)	0x0c000000

Table 7: Memory layout

Note: All INTRAM block start addresses shown are addresses via standard ARM966 system interface (segment address 0x08000000). All INTRAM blocks are also accessible via I-TCM (segment address 0x00000000) and D-TCM (segment address 0x04000000) interface of ARM966 core. Note that all INTRAM blocks have mirror start addresses. For example to access INTRAM1 via I-TCM interface the start address is 0x0002 0000 and via D-TCM interface is 0x0402 0000.

3.3 Oscillator

All internal netX clock signals are generated by a PLL which is driven by an internal oscillator that requires an external 25 MHz crystal. Alternatively, an external oscillator can be used.

For a schematic about how to connect an external crystal, see reference [2].

3.4 Booting, security memory and system LED

The general status of a netX-based system is displayed by the System LED(s). It is recommended to use a dual LED here, but two single LEDs can also be used. The general definition of this LED is

RDY yellow the netX with operating system is running

RUN green the user application is running without errors

Be aware that any firmware or specific-application you boot will control the LEDs and, thus, influence their behavior.

The RDY and RUN signals are also used as inputs after a reset to select the boot mode. Applying certain logic levels to these pins during and shortly after the reset results in a preselection of the available boot options and determines where the ROM boot loader looks for executable program code. Moreover, these pins are used to connect a security memory to the netX 51/52 containing licenses and other application-specific information like MAC addresses or SDRAM parameters as well as the desired boot mode and boot options.

In applications without a security memory, the desired boot mode is selected according to the following schematic:

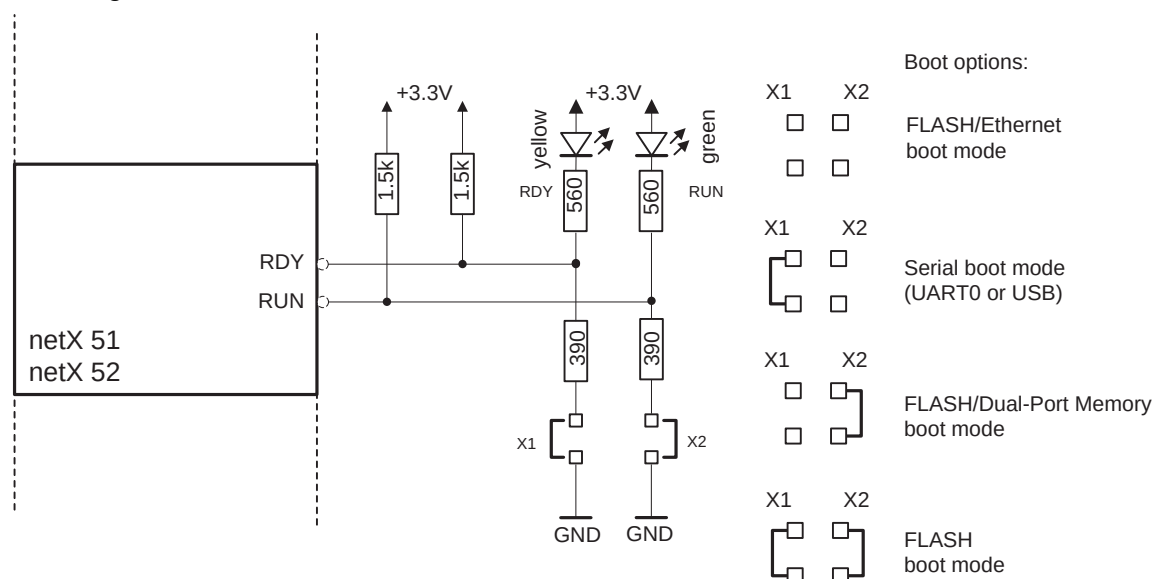


Figure 2: RDY/RUN circuit without security memory

In applications using a security memory (mandatory for master applications), the desired boot mode is selected by a parameter, stored in the security memory. In that case, boot jumpers cannot be used because the security memory will not be accessed unless a high level is applied to both pins, RDY and RUN. However, it is possible and recommended to use a boot mode button to force the serial boot mode.

The following diagram shows

- the proper circuit for attaching the SYS LED,
- a boot mode button,
- a security memory to the netX 51/52:

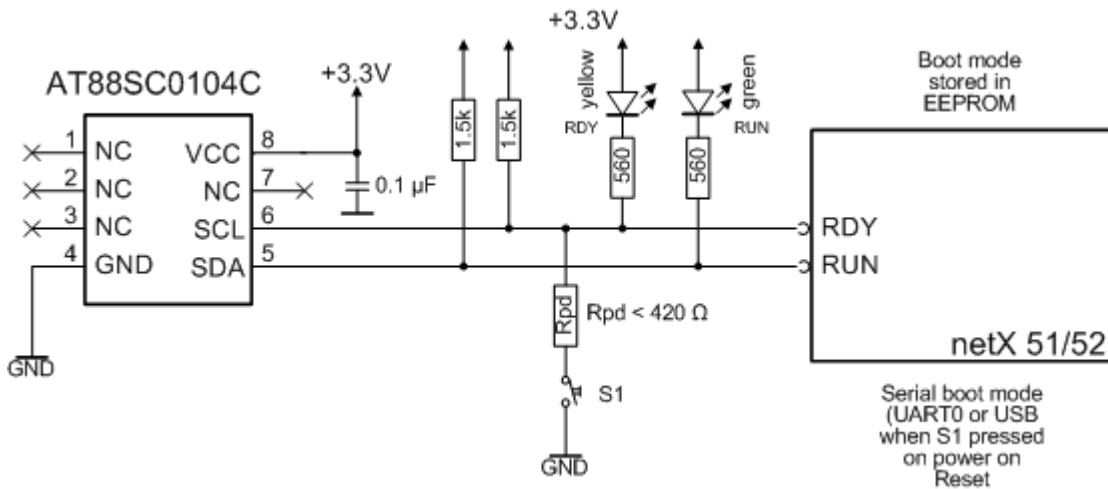


Figure 3: RDY/RUN circuit with security memory

Note: The above circuit differs from the corresponding netX 50 circuit.

3.4.1 System LED

The ROM code uses the System LED with the following states:

LED	Color	State	Meaning
SYS	Yellow	Flashing 0.25 s on 0.25 s off	netX is in serial boot mode : netX is waiting for the software transfer, i.e. for the second stage boot loader. The external device has to start the transfer. The flashing sequence stops as soon as a serial connection is established.
	Yellow	Flashing 0.125 s on 0.125 s off 0.25 s on 0.25 s off	netX is in dual-port memory mode : netX is waiting for the host. The host has to start the software transfer. The flashing sequence stops as soon as the first data have been received.
	Yellow	Flashing 0.25 s on 0.25 s off 0.25 s on 1.75 s off	netX is in Ethernet boot mode : netX is waiting for a link-up on the Ethernet. The flashing sequence stops when an Ethernet link is established.
	Yellow/Green	Alternating flashing 0.125 s yellow on 0.125 s off 0.125 s yellow on 0.125 s off 0.125 s green on 0.125 s off 0.125 s green on 0.125 s off	netX is in Ethernet boot mode : Ethernet boot mode is active and Ethernet link is established. netX is waiting for the software transfer. The external device has to start the software transfer.
	Green	Flashing 0.125 s on 0.125 s off	netX is in Ethernet boot mode : Booting in progress and software transfer is active. The flashing sequence stops when the transfer is finished or aborted.
	Yellow	Flashing 0.875 s on 0.125 s off 0.125 s on 0.125 s off	Failure of booting from source device due to checksum error. As a result, netX changes to serial boot mode and then waits for the software transfer. The flashing sequence stops as soon as a serial connection is established.
	Off	Off	1. After reset and during boot start. 2. During boot from Flash or SD/MMC card. 3. Power supply for the device is missing or hardware defect.

Table 8: States of system LED (used from the ROM code)

The ROM code starts, if available in the netX memory (e.g. in the connected serial Flash memory), an executable software. The netX executes the loaded software which then controls the system LED, i.e. more states are possible than described in Table 8.

3.4.2 Boot mode and boot sequence

The ROM code evaluates the boot mode (boot option) during the booting. The ROM code steps through the boot sequence as listed in the following table. In each step the ROM code 'searches' for a bootable image and if found, executes it.

Boot mode	RDY	RUN	Boot sequence
FLASH/Ethernet boot mode	High	High	1. Boot from parallel FLASH memory (netX 51: at memory interface, netX 52: at host interface*). 2. Boot from serial FLASH using Quad SPI and XIP (CS0). 3. Boot from serial FLASH using SPI (CS0). 4. Boot from MMC/SD (netX 51: CS1; netX 52: CS2). 5. Boot from Ethernet CH0 with DHCP and TFTP. * Select the host interface mode with SPI_MOSI, SPI_CLK and SPI_MISO (described in section <i>Host interface mode</i> on page 25).
Serial boot mode	Low	High	Boot from serial interface: UART0 or USB.
FLASH/Dual-Port Memory boot mode	High	Low	1. Boot from parallel FLASH memory (netX 51 only and at memory interface). 2. Boot from serial FLASH using Quad SPI and XIP (CS0). 3. Boot from serial FLASH using SPI (CS0). 4. Boot from MMC/SD (netX 51: CS1; netX 52: CS2). 5. Boot from dual-port memory. Select the host interface mode with SPI_MOSI, SPI_CLK and SPI_MISO (described in section <i>Host interface mode</i> on page 25).
FLASH boot mode	Low	Low	1. Boot from parallel FLASH memory (netX 51: at memory interface, netX 52: at host interface*). 2. Boot from serial FLASH using Quad SPI and XIP (CS0). 3. Boot from serial FLASH using SPI (CS0). 4. Boot from MMC/SD (netX 51: CS1; netX 52: CS2). * Select the host interface mode with SPI_MOSI, SPI_CLK and SPI_MISO (described in section <i>Host interface mode</i> on page 25).

Table 9: Boot sequence

3.4.3 Host interface mode

For booting netX, the selection of the host interface mode (a new feature of netX 51/52 compared with netX 50) provides the additional features described below.

The host interface mode is selected by the signals SPI_MOSI, SPI_CLK and SPI_MISO. The ROM code reads these signals and uses them to initialize the host interface during the boot process. This means that depending on the selected mode, signals are driven actively from netX or from the host.

To select the right host interface mode:

1. Specify the device that is to be connected physically to the netX host interface.
2. Specify the device netX to be used as a source or destination device during booting.

The tables in the following subsections specify for which signal a pull-up resistor is required to select the host interface mode. The recommended value for the pull-up resistor is 1.5 kΩ.

3.4.3.1 netX boots from the connected memory

netX is typically used with the following boot sequence: netX reads the bootable image from the connected memory device (e. g. Flash device), writes it into the INTRAM and then executes it. That means the device connected to the host interface is not used for booting.

If the device connected to the host interface is not part of the booting process (standard), use the following settings:

Host interface mode	Use if, ... / Remark	SPI_MOSI	SPI_CLK	SPI_MISO
Ignore host interface during booting.	... the device that is connected at the host interface cannot be used as a source or destination device during boot, e. g. the host interface is used as PIO.	Low (open)	Low (open)	Low (open)

Table 10: Host interface mode – Device at host interface is not used for booting

If the bootable image should be written instead of the INTRAM into SDRAM/SRAM device (destination) which is connected to the host interface, select the corresponding setting from the following table. If the bootable image can be read from parallel Flash device (source) which is connected to the host interface, select 'SRAM 16-bit data / 4 MB' from the following table.

Host interface mode	Use if, ... / Remark	SPI_MOSI	SPI_CLK	SPI_MISO
SDRAM 16 bit data / 4 MB	... SDRAM can be used as destination device instead of INTRAM.	Low (open)	Low (open)	High (Pull-up)
SRAM 16 bit data / 4 MB	... parallel Flash (CS0) can be used as source device. - or - ... static RAM (CS0, 1, 2 or 3) can be used as destination device instead of INTRAM.	Low (open)	High (Pull-up)	Low (open)
SDRAM 32 bit data / 4 MB	... SDRAM can be used as destination device instead of INTRAM.	High (Pull-up)	Low (open)	High (Pull-up)

Table 11: Host interface mode – SDRAM/SRAM at host interface

Note: Using a mode of Table 11 requires additionally that the destination device in the boot block of the bootable image is set properly.

3.4.3.2 Host CPU supported: Host CPU boots netX via DPM

The dual-port memory is used for booting netX. The netX waits for the host CPU, which has to transfer the bootable image (e.g. second stage bootloader). Therefore the host CPU has to use the mailboxes of netX for transferring the boot code.

Host interface mode	Use if, ... / Remark	SPI_MOSI	SPI_CLK	SPI_MISO
Serial dual-port memory (SPM) mode	... serial dual-port memory should be used as source device.	Low (open)	High (Pull-up)	High (Pull-up)
Parallel dual-port memory (DPM) 8 bit data / 2 KB	... parallel dual-port memory should be used as source device.	High (Pull-up)	Low (open)	Low (open)
Parallel dual-port memory (DPM) 8 bit data / 64 Kbyte	... parallel dual-port memory should be used as source device.	High (Pull-up)	High (Pull-up)	High (Pull-up)

Table 12: Host interface mode – Dual-port memory is used for booting

3.4.3.3 Reserved host interface mode

This mode is reserved. Do not use it.

Host interface mode	Use if, ... / Remark	SPI_MOSI	SPI_CLK	SPI_MISO
Reserved	Reserved.	High (Pull-up)	High (Pull-up)	Low (open)

Table 13: Host interface mode - Reserved

3.4.4 Extended system information

The two LEDs 'RDY' and 'RUN' described in section *Bootling, security memory and system LED* are controlled by the netX via a special system status configuration register, containing information on the system status of the netX. This status information (DPM_SYS_STA) is set by the netX boot software or firmware. The definition of each status bit and status code is software-specific. Moreover, there are four host status flags which can only be controlled by an external host system connected to the netX 51/52 DPM interface. As soon as the netX performs a write access to the netX status flags, an interrupt request can be generated to notify the host about the changes.

For a detailed description of the register bits, see reference [1].

3.5 Reset

The netX offers several reset sources which can generate a system reset of the chip. Two of them are external inputs, the others are generated by different internal function blocks.

PORn	Power-on reset, input pin (Schmitt trigger) This (active low) signal shall be connected to the output of a voltage supervisor chip which checks the power supply voltages and pulls the power-on reset pin low whenever the voltages are below the minimum specified for netX system operating voltages. The power-on reset signal causes an asynchronous reset of the netX chip and initializes all internal registers and signals to their power-on reset state. Reset timing is specified in section <i>Power-on reset/reset input</i> on page 220.
RSTINn	External reset input pin (Schmitt trigger) This is the second reset input signal (active low) from an external reset source e.g. from the host system. Unlike the PORn, this is a synchronous reset and an optional signal. If not used, this signal may be left unconnected, since the pin is equipped with an internal pull-up resistor (nom. 50 kΩ). However, it is recommended to tie it to VDDIO (+3.3V), since this can improve EMC behavior
WDG_RES	If the internal watchdog counter expires, this reset is generated. It is also possible to generate an interrupt before the watchdog resets the chip. For more details, see section <i>Watchdog</i> on page 29.
HOST_RES	The host system interface initiates this reset by writing a special sequence into a host interface register. The reset will occur 1 ms after starting the write cycle allowing the host to finish the access and to prepare for the netX chip reset.
FIRMW_RES	This reset can be activated by a software command.
XPEC0_RES	This reset is generated by xPEC of communication channel 0.
XPEC1_RES	This reset is generated by xPEC of communication channel 1.

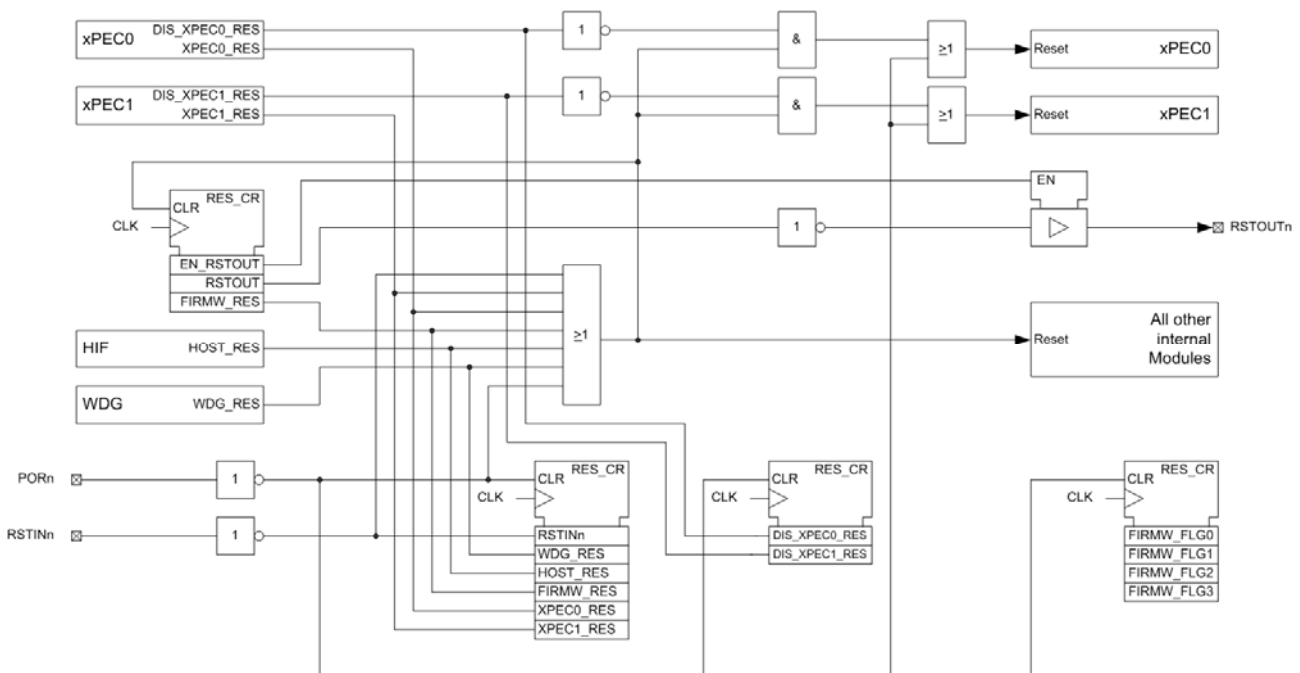


Figure 4: Block diagram of the reset controller

There is also one output signal to reset external connected peripherals:

RSTOUTn This is an output signal to reset connected peripheral devices. It is a tri-stateable signal that can be enabled and set to high or low level by a software command. Any reset (power-on, reset-in or internal reset) will disable the output driver of the signal allowing you to set the desired reset default level to low or high by using a pull-down or pull-up resistor.

A power-on reset is the only reset condition that will clear the value of register RESET_CTRL. All other reset requests are stored in register RESET_CTRL. The firmware can use this information to determine which reset source has activated the last reset signal and act accordingly (e.g. not restart the system after a watchdog-initiated reset).

Moreover, register RESET_CTRL provides 4 bits that can be used to save information, unaffected by any reset, except for the power-on reset.

The firmware can disable the internal system reset signals to the XPEC modules allowing these modules to continue running even while the chip is performing a reset.

3.5.1 Reset configuration

With the rising edge of the last active power-on reset signal (PORn or RSTINn), the levels of I/O pins PIO[85:32] are stored in the corresponding host interface registers. It is possible to read these values for hardware-controlled power-on configuration. For details, see the register description of the host input/output port control and section *Using DPM IOs as programmable IOs (PIOs)* on page 147 (PIO, power-on reset sampling of PIO 32-85 pins).

3.6 Watchdog

For system supervision, the netX is equipped with an internal 2-stage watchdog counter. Once the watchdog is active, the timeout counter has to be triggered continuously.

The watchdog works in two stages: When the IRQ timeout counter has reached zero, an interrupt is generated to indicate that the watchdog will soon perform a reset and needs attention. As soon as the reset timeout counter reaches zero as well, the watchdog will reset the netX.

Whenever the watchdog is triggered, the timeout register values are reloaded to the watchdog timer by setting the watchdog trigger bit. This will also clear an active interrupt request flag (timeout, stage 1 was reached). Writing to the timeout register is possible only if the write enable timeout flag is set. This prevents undesired access to the timeout registers.

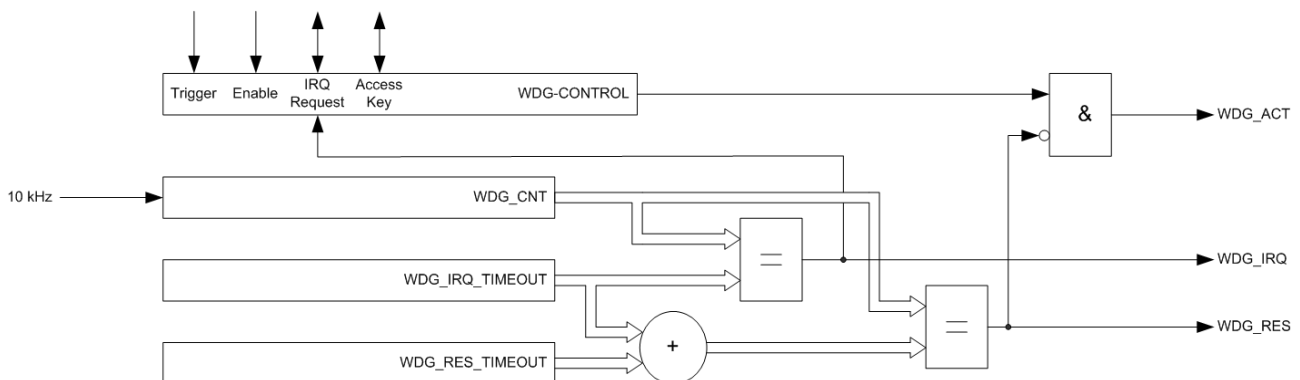


Figure 5: Internal structure of the watchdog logic

3.6.1 WDGACT signal

This signal shows that the watchdog supervision is active and not expired. It can be combined for example with the power stage of a motion control system. After reset this signal is low. When the watchdog will be activated the signal goes into high state. When the watchdog timeout expired this signal is cleared. When the watchdog is not enabled this signal stays low.

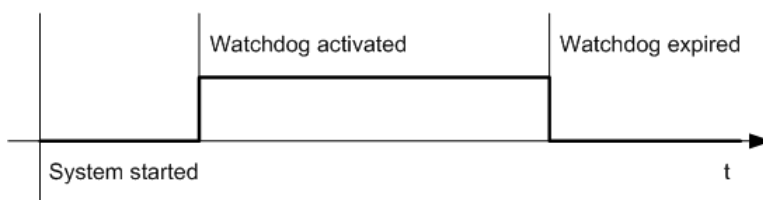


Figure 6: Timing diagram of WDGACT

3.7 Chip mode selection

The ROM code uses pins MEM_A18 and MEM_A19 to read the chip mode. Both pins will be sampled if a negative edge occurs on the reset.

3.7.1 Chip mode selection netX 51

netX 51 offers two chip modes

- netX 51 mode
- netX 51 in netX 50 pin compatibility mode

The figures in the following table show how to use pull-up resistors to select/set the mode.

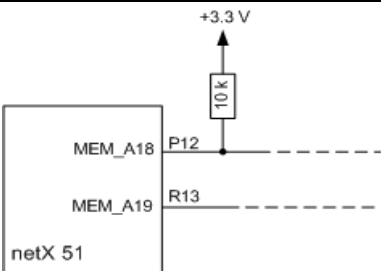
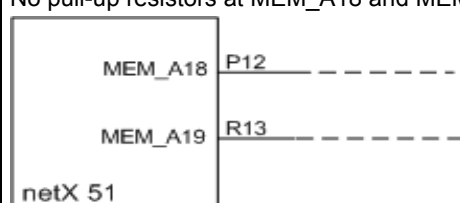
netX 51 mode	netX 51 in netX 50 pin compatibility mode
 Figure 7: netX 51 mode	No pull-up resistors at MEM_A18 and MEM_A19.  Figure 8: netX 51 in netX 50 pin compatibility mode

Figure 9: Chip mode selection netX 51

Address lines MEM_A18 and MEM_A19 can be used to connect a Quad SPI flash by the internal MMIO configuration. For more information, see section *Memory interface for Quad SPI mode* on page 35.

3.7.2 Chip mode selection netX 52

netX 52 has one chip mode. The following figure shows how to use pull-up resistors to select/set the mode.

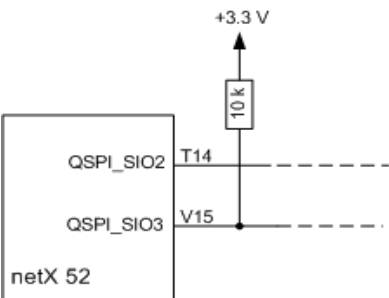


Figure 10: netX 52 chip mode

Address lines MEM_A18 and MEM_A19 can be used to connect a Quad SPI flash by the internal MMIO configuration. For more information, see section *Memory interface for Quad SPI mode* on page 35.

3.8 Internal memory

The netX 51/52 contains 672 KByte static RAM, 64 KByte ROM.

3.9 External memory interface

There are two completely independent external memory interface controllers inside netX 51/52: One is connected to the MEM_-IOs of netX 51 and a second one is connected to the host interface IOs of netX 51/52. However, there is a difference between the SDRAM controllers of the external Memory Interface and the host interface: The external Memory Interface SDRAM controller is a high-performance multi-channel and multi-cache controller (as known from netX 100 and netX 50). Related to netX 50 some performance optimizations have been carried out: Mainly there are 2 separated channels for ARM data and instructions now which will increase ARM performance on SDRAM significantly. The SDRAM controller of the host interface is a single-channel SDRAM-light controller containing 16 or 32 data bits.

Note: Using host interface SDRAM for multi-master purpose or running xPIC or ARM-code there is possible, but will lead to much less performance in comparison to external Memory Interface. Giving a general performance ratio between host interface SDRAM and external Memory Interface SDRAM is not possible as it depends on application. For higher performance it is recommended to use external Memory Interface SDRAM.

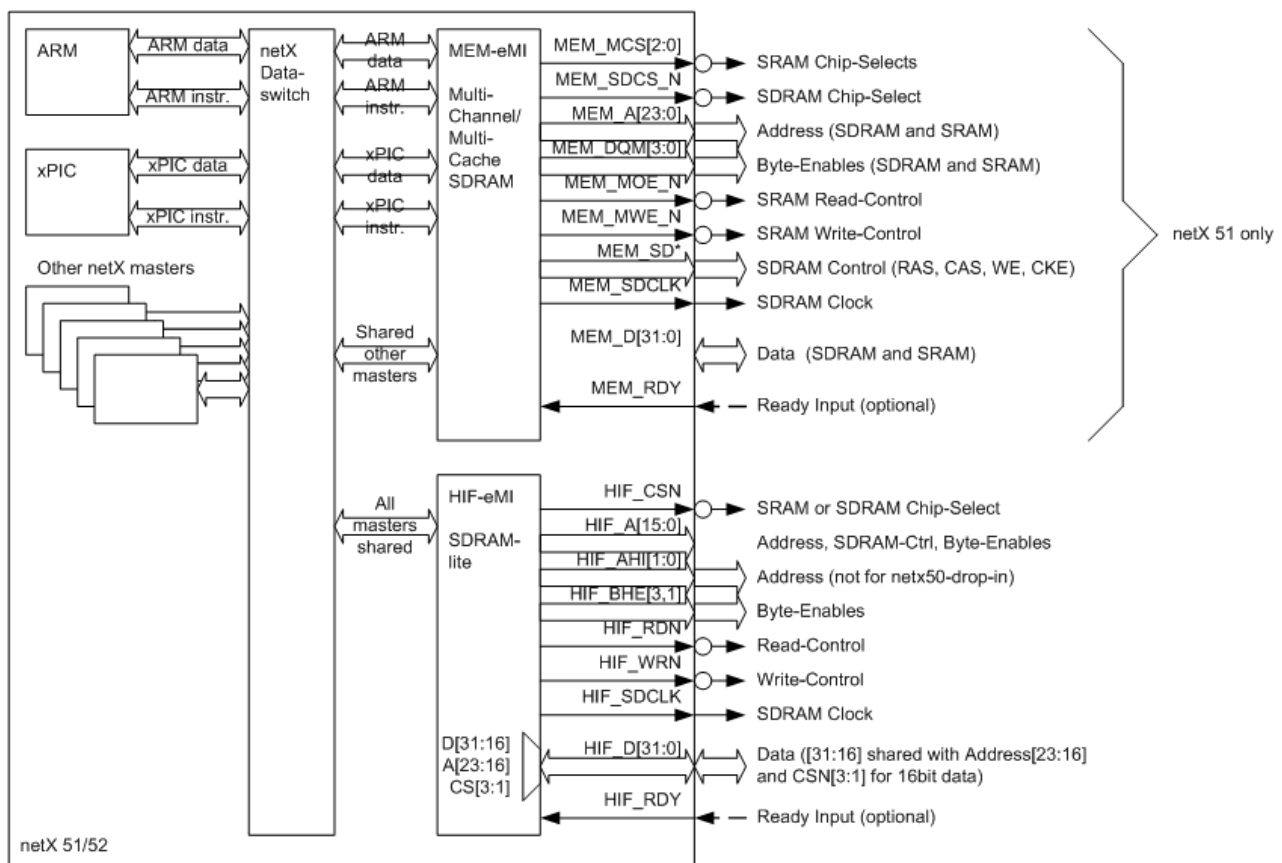


Figure 11: External memory controller of netX 51/52

Note: Host interface IOs provide many different functions (e.g. dual-port memory, MMIOs) and are widely shared. This is described in section *Host interface* on page 51.

The following list provides the key features of netX 51/52 external memory interface:

- Supported devices:
 - SDRAM (single data rate)
 - SRAM based devices like SRAM, NVRAM, Flash with fix access times
 - SRAM based devices with external wait state generation
- 8, 16 or 32 data bits for SRAM and 16 or 32 data bits for SDRAM
- SDRAM can be shared with SRAM based memory (For restrictions see section *Simultaneous operation of SDRAM and parallel Flash memory at the memory interface* on page 50)
- SDRAM features:
 - All typical SDRAM devices supported: 2 or 4 banks, 256 to 4 K columns, 2 K to 16 K rows
 - Widely configurable timing parameters
 - Intelligent refresh controlling
 - Power down function
 - CAS Latency 2 or 3
 - 16 byte read and write caches (external Memory Interface: 5 channels, host interface: single-channel SDRAM light)
- SRAM based device features:
 - External Memory Interface: Up to 24 address lines for 8, 16 or 32 bit data (up to 64 MByte per CS)
 - Host interface: Up to 24 address lines for 8 or 16 bit data (up to 16 MByte per CS)
 - Up to 3 chip-select signals available
 - Widely configurable signal timing: pre, post and wait state pauses
 - Asynchronous Page mode (APM) supported
 - Optional ready/wait signal for external wait state generation providing signal filtering and timeout logic (new)

Note: High address lines of the host interface are shared with upper data lines. Additionally Byte enables of host interface are located on low address lines. Hence, 16 MBytes is limit for 8 or 16 bit SRAM devices per chip-select area there. For 32 bit devices address lines are limited to A0 to A17 (netX 50-drop-in to A15), i.e. there are only 256 KByte (netX 50-drop-in to 64 KByte) available then.

Note: Host interface (memory controller) of netX 51/52 is shared with dual-port memory host interface on netX 51/52 HIF IOs. It is possible to use a serial dual-port memory host connection together with an 8 or 16 bit Host interface memory controller. It is not possible to use a parallel DPM host connection with host interface memory controller and it is also not possible to use serial dual-port memory together with 32 bit host interface memory controller. For details see section *Pinout functions combinations and exclusions* on page 65.

Note: If more than one memory device is connected, the allowed maximum load capacity has to be considered, to ensure stable operation throughout the whole temperature and operating voltage range!

3.9.1 SRAM/Flash interface to memory interface controller

The SRAM/FLASH interface provides a total of three memory areas which provide their own chip select signals and three independent configuration registers. They allow you to set memory bus width and wait state parameters separately for each area. The parameters allow bus width configurations of 8, 16 or 32-bit and wait states of up to 63 clock cycles.

Depending on the bus width of the respective memory component, the 24-bit address bus allows access to 16 MByte, 32 MByte or 64 MByte of static memory per memory area. For 8-bit areas, address lines A23:0 represent the byte address, for 16-bit areas the word address and for 32-bit memory areas the dword address. Hence, no address lines and thus address space are wasted when using 32-bit or 16-bit components. To allow byte or word access in 32-bit and 16-bit mode, four byte lane signals (MEM_DQM3-0) are provided.

Note: The interface does **not** support Burst or Page mode, as provided by many FLASH memory devices.

3.9.1.1 Connecting SRAM and serial Flash

The external memory interface controller provides 3 completely independent chip-select areas for SRAM. Simultaneous usage of different configurations concerning data width or access timing is possible. E.g. chip-select 0 can be connected to an 8-bit SRAM device with 80 ns data access time while chip-select 1 can be connected to a 16-bit device with 200 ns data access time.

Important: Address lines MEM_A18 and MEM_A19 of external memory interface controller are remapped when 4 bit SPI/QSPI is used in non-MMIO mode, e.g. for XiP (XiP is not available on MMIO-SIO 2, 3). This must be considered for signal routing. Figure 12 on page 33 shows this. SDRAM is not affected by this as MEM_A18 and MEM_A19 are never used therefore.

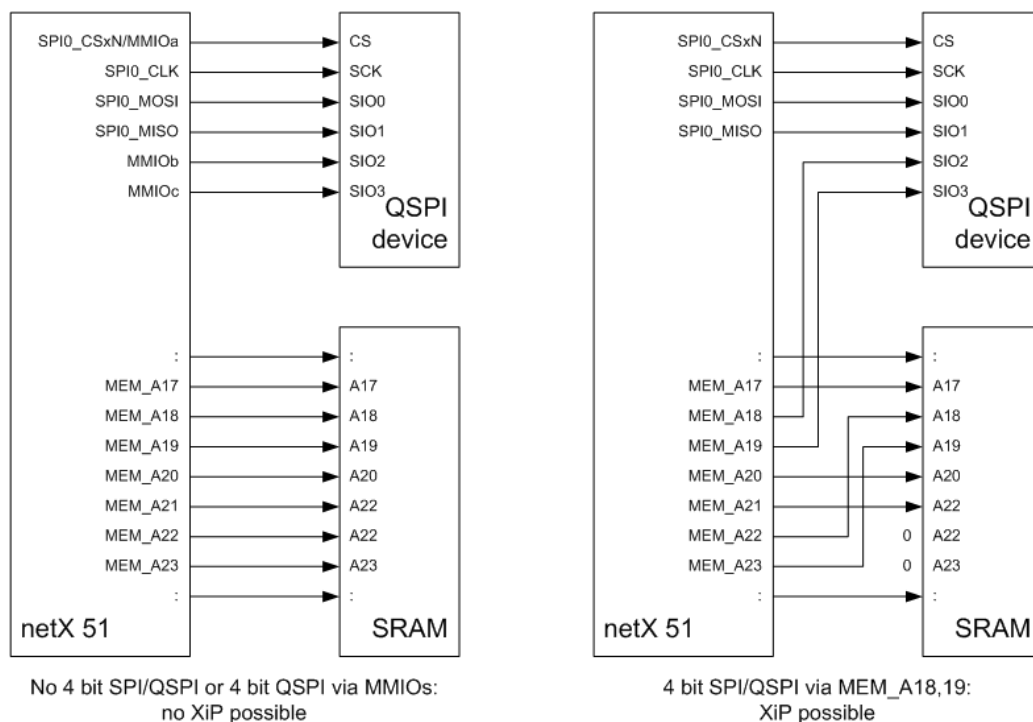


Figure 12: External memory interface controller high address lines when non-MMIO 4 bit SPI/QSPI is used (e.g. XiP)

Note: QSPI/XiP signal mapping is controlled by the global register IO-Config2.

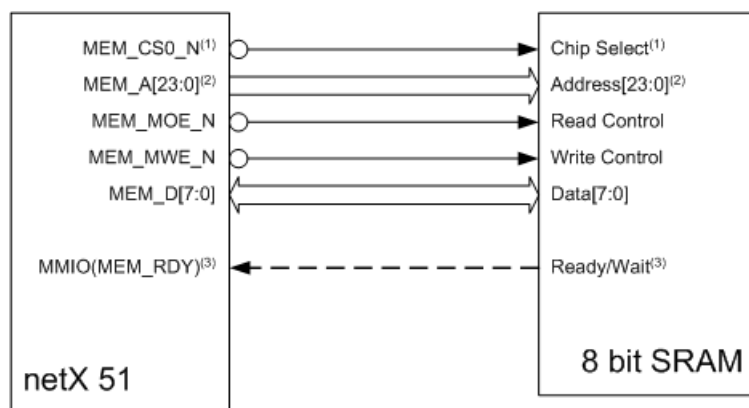


Figure 13: 8-bit standard SRAM interconnection

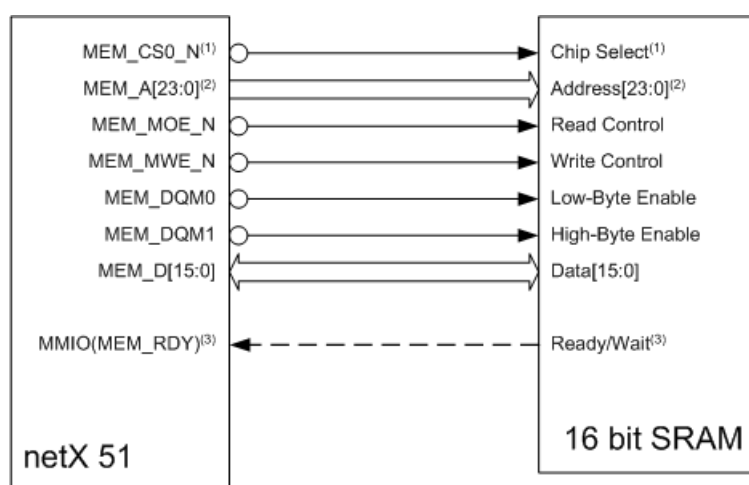


Figure 14: 16-bit standard SRAM interconnection

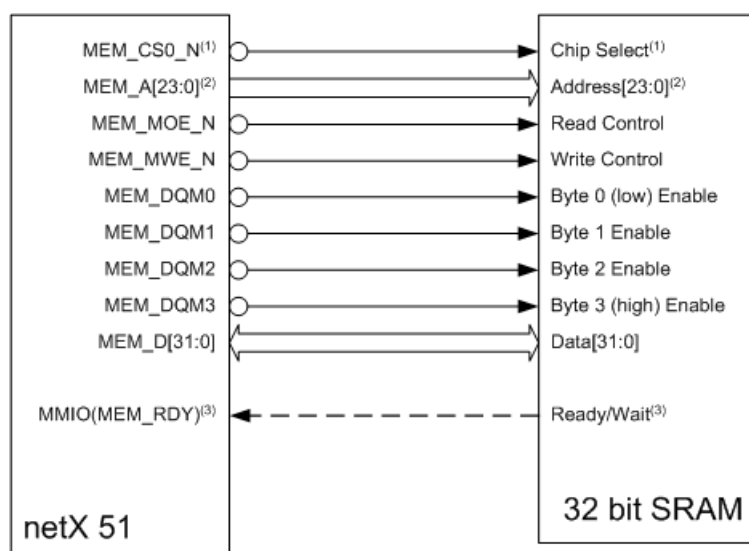


Figure 15: 32-bit standard SRAM interconnection

Notes for SRAM interconnections

1. netX 51 external memory interface provides three completely independent SRAM chip-select signals. MEM_CS1_N or MEM_CS2_N could be used instead. Different configurations can be used simultaneously.
2. 24 address lines are provided by netX 51 external memory interface. Generated address is always right-shifted according to connected device data size. If a device provides less than 16 M (2^{24}) words, upper address lines will be left unconnected.
3. Ready input of the external memory interface is only optional and a multiplex matrix signal (MMIO) for netX 51. It can be mapped to 1 of 49 MMIOs of netX 51. Only 1 ready input is provided for all external memory interface SRAM chip-select areas. It is possible to use it for more than 1 chip-select when only one state (e.g. ready active) is active driven while the other state (busy) is generated by a pull-up or pull-down resistor. However, this is not recommended as slow signal transitions result by doing so.

Note: It is possible to use several small devices instead of a single big one. E.g. it is possible using four 8 bit devices to make up one 32 bit memory.

Important: For netX 51 it is absolutely NOT recommendable to use SRAM together with SDRAM because the max. IO load for valid SDRAM timing is strictly limited. Using both will lead to a swapping of the address lines MEM_A14 to MEM_A17. This is not netX 50-compatible and has a major impact on FLASH devices. See section *netX 51/52 SDRAM bank address metal-fix* issues on page 350.

Memory interface for Quad SPI mode

Two additional signals SIO2 and SIO3 are needed to use a Quad SPI flash instead of a (single) SPI flash. Therefore MEM_A18 and MEM_A19 can be used because normally these two memory signals are not used by SDRAM.

An internal multiplex can be activated to change these address lines into the SIO2 and SIO3 signals for the Quad SPI mode and the MEM_A18 and MEM_A19 functionality moves to the highest address lines.

Ball pos. netX 51	Ball pos. netX 52	Pad type netX 51/52	Signal (default)	Signal (MUX function 1)
P12	T14	IOD6	MEM_A18	QSPI_SIO2
R13	V15	IOD6	MEM_A19	QSPI_SIO3
J14	-	IOD6	MEM_A22	MEM_A18
J15	-	IOD6	MEM_A23	MEM_A19
...	...	IOD6	All memory signals without MEMDR_CLK and MEM_D0-31	Signal unchanged
...	...	IOD6	MEMDR_CLK	MEMDR_CLK
...	...	IOD6	MEM_D0-31	MEM_D0-31

Table 14: Memory interface for Quad SPI mode set by MMIO configuration

3.9.1.2 Connecting parallel FLASH

The following schematics show how to connect SRAM and FLASH to the memory interface of the netX.

Note: These examples assume the use of FLASH components that reserve address line A0 for low/high byte selection (when operating in BYTE mode). However, this does not apply to all FLASH components on the market. Some components always expect 16-bit addresses and thus require a one-to-one connection of the address lines (A0 to A0, A1 to A1, etc.) Users should always consult the datasheet of their FLASH to determine the correct way to hook up the FLASH.

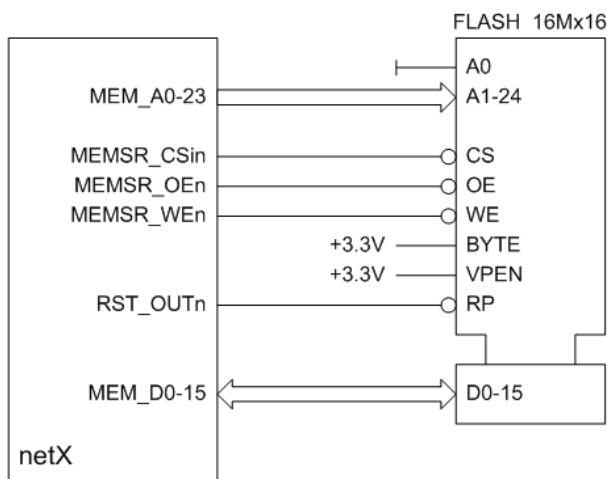


Figure 16: Interface with 16-bit Flash up to 32 MBytes

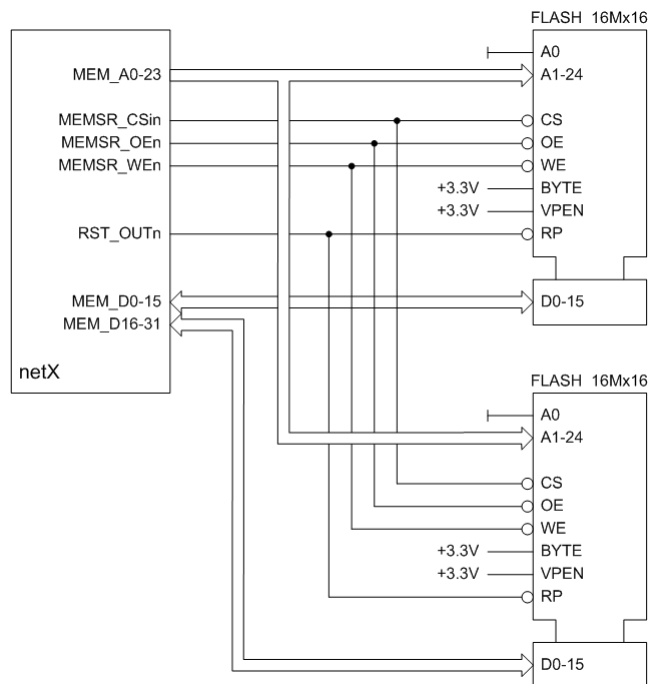


Figure 17: Interface with two 16-bit Flash up to 64 MBytes

3.9.1.3 SRAM access generation

The SRAM part of netX 51/52 external memory interface controller provides all functions to connect 8, 16 or 32-bit devices based on standard SRAM protocol. An optional configurable ready signal allows devices to generate wait states dynamically.

Note: Many devices are based on the SRAM protocol apart from standard SRAM devices – e.g. NVRAM or parallel Flash. In the following text, “SRAM” stands for all of them.

The netX eMI SRAM controller splits an internal access to the SRAM, which is wider than the external SRAM device, into a sequence of external accesses. E.g. a 32-bit access to an 8-bit SRAM device is split into 4 external byte accesses; a 32-bit access to a 16-bit SRAM device is split into 2 external 16-bit accesses.

3.9.1.3.1 SRAM access timing

SRAM access timing is widely configurable for each chip-select individually. Chip-select 0 can additionally be configured for Asynchronous Page Mode (APM, see section *Asynchronous page mode (APM) bursts* on page 40).

Note: APM is only available for chip select 0, not for the other chip-selects. The external memory interface controller and host interface memory controller both support this.

All access timing can be configured in steps of netX 51/52 clock cycle i.e. 10 ns if not stepped down.

The following pauses/wait phases can be configured:

Pause	Comment
pre-pause 0 to 3 cycles	Access setup phase: This phase can be used to match setup timing requirements of the connected device. Read enable and write enable signal will be inactive during this phase. They will be changed to active state when changing to wait state phase or access cycle. All other netX-driven signals are active and valid during this phase.
wait states 0 to 63 cycles (can be extended by ready signal)	Device access time wait phase: During read access this is read enable signal active low phase. During write access this is write enable signal active low phase. All netX-driven signals are active and valid during this phase.
access cycle 1 cycle	Read data is sampled during this cycle and must be valid. Read enable and write enable signal will become inactive at the end of this cycle when followed by a post-pause.
post-pause 0 to 3 cycles	Access hold phase: This phase can be used to match hold (for write) or output disable (for read) timing requirements of the connected device. Read enable and write enable signal will be inactive during this phase. All other netX-driven signals are active and valid during this phase. For write accesses at least 1 cycle post-pause will always be inserted to generate a positive edge on the write enable signal.

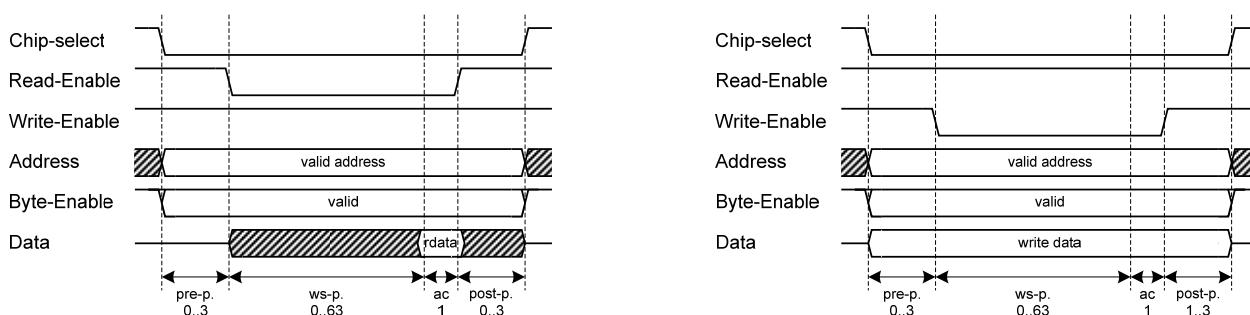


Figure 18: Single read (left) and single write (right) access

Between sequential reads pre and post-pause can be disabled for burst generation. Between sequential writes pre and post-pause will always be performed as configured. For sequential reads and writes, the chip-select signal will remain active between subsequent accesses.

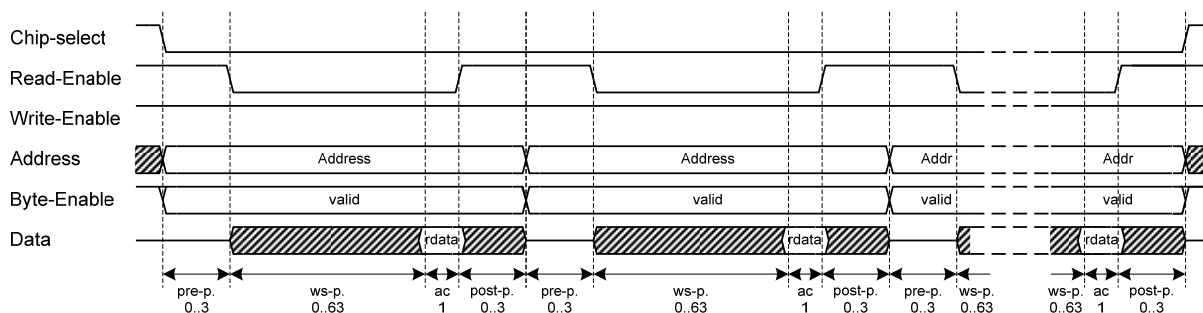


Figure 19: Sequential read pre and post pause not disabled (compatible with netX 50, 100, 500, default)

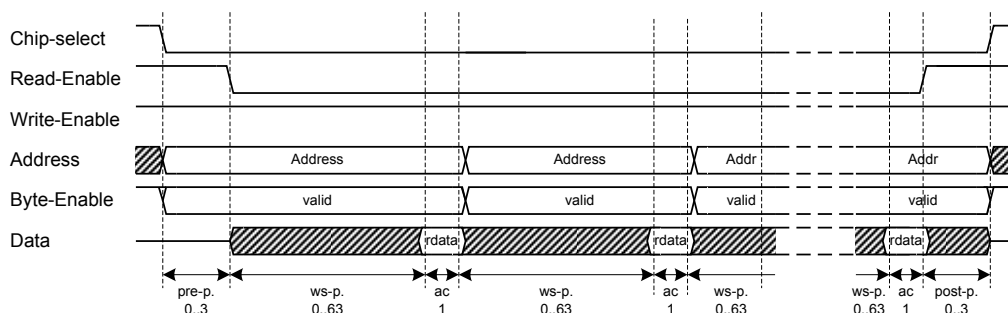


Figure 20: Sequential read pre and post pause disabled (recommended for memory devices)

Note: Memory devices like flash or NVRAM normally do not need pre and post-pauses between sequential reads (for details see device datasheet). To achieve good performance, set the disable-bits (no_p_pre_seq_rd, no_p_post_seq_rd) in the eMI configuration register for the related chip-select. For consecutive reads, addresses can be used in any order, i.e. addresses need not be incrementing.

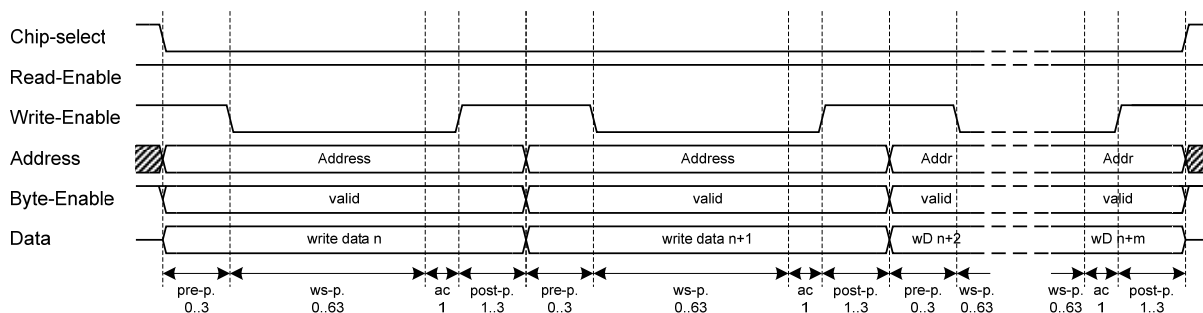


Figure 21: Sequential write

When data direction is changed (read after write or write after read access) at least one cycle pre-pause will be inserted.

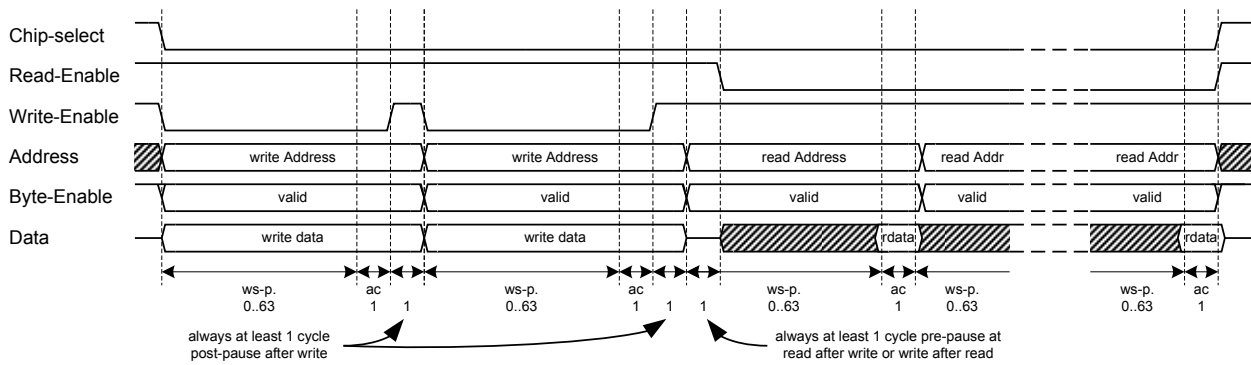


Figure 22: Insertion of pre and post-pause at write and data direction change when 0 is configured

Also at least one pre-pause will be inserted when chip-select is changed. However, access sequences with changing chip-select as described in Figure 23 and Figure 24 are very unlikely. They can be produced only by

- different netX masters running on different external devices simultaneously or
- a single master which, when running burst, exceeds the chip-select address area borders.

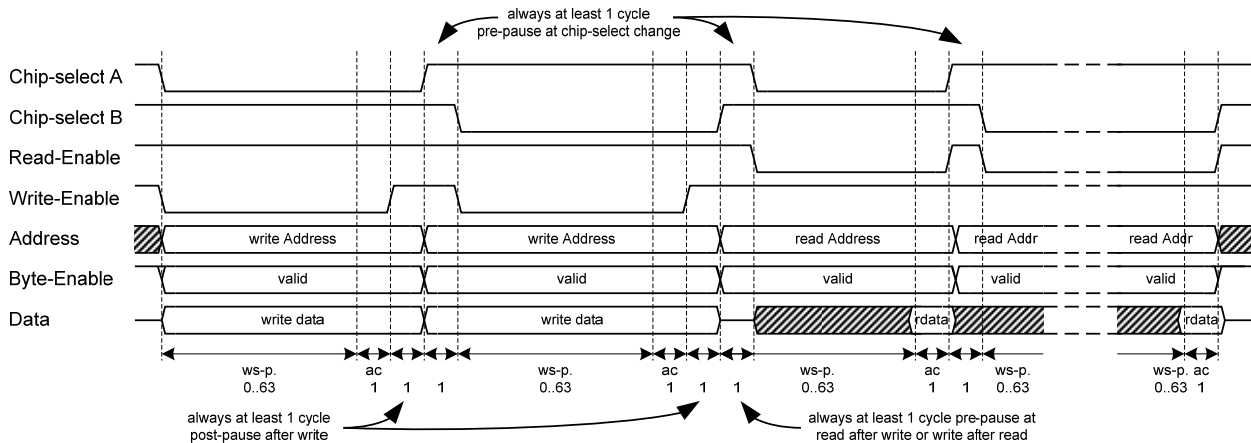


Figure 23: Insertion of pre and post-pause at chip-select change if 0 is configured

Note: If pre or post-pause is not configured to 0, no additional pauses will be inserted. The programmed values will be used if data direction or chip-select is changed.

The following figure shows an example in which:

- Chip-select A is configured for 3 cycle pre-pause, 2 cycle post-pause and disabled pauses during sequential reads.
- Chip-select B is configured for 2 cycle pre-pause, 1 cycle post-pause.

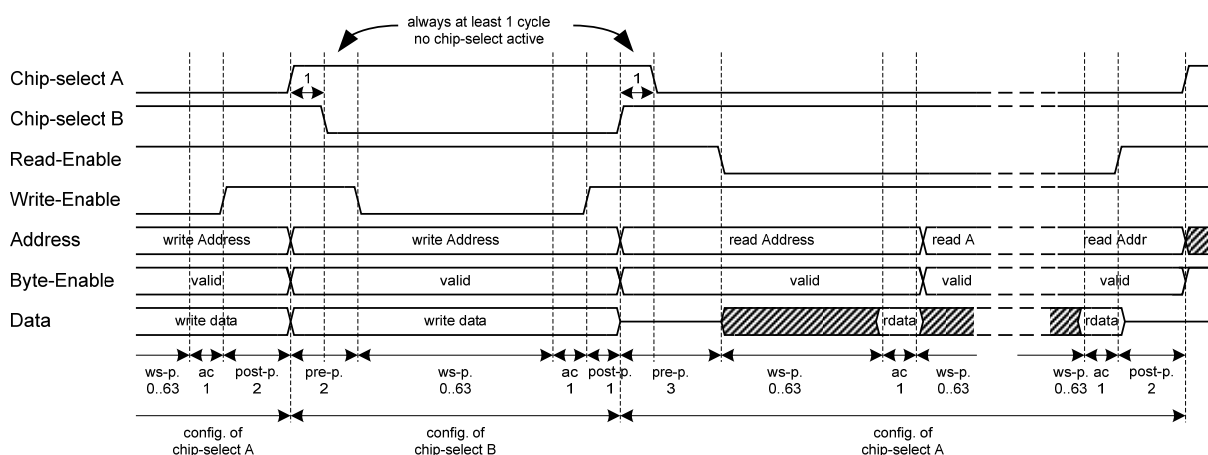


Figure 24: Pre and post-pause at chip-select or data direction change if 0 is not configured

3.9.1.3.2 Asynchronous page mode (APM) bursts

For chip-select 0 APM read bursts can be configured additionally to standard access timing described in section *SRAM access timing* on page 37.

An APM read burst consists of a longer initial access followed by shorter access operations for data inside a certain page. The burst is restarted with the initial access after chip-select has been inactive, after write or when the address changes to another page.

Pages are addressed by higher address lines, data words inside a page by lower address lines. Page boundary can be configured to 4, 8, 16, 32, 64 or 128 for 8-bit devices and to 2, 4, 8, 16, 32 or 64 for 16-bit devices.

The initial access is defined by the wait-phase programmed in chip-select 0 configuration register. For the faster access operations following the initial access an APM-wait-phase of 0 to 15 cycles (i.e. 0 to 150 ns) can be configured.

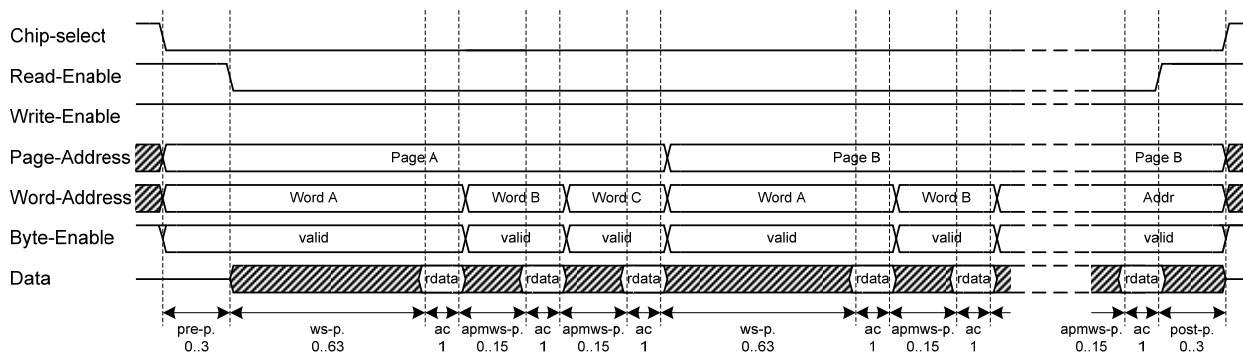


Figure 25: APM read burst

Note: APM is available for read access operations only.

Note: To keep the output enable signal low for the whole APM burst, disable pre and post-pause for sequential reads of chip-select 0 (*no_p_pre_seq_rd*, *no_p_post_seq_rd* configuration bits).

For APM-related performance issues, see section *SRAM burst related performance issues* on page 42.

3.9.1.3.3 Ready signal – external wait state generation

To allow an external device dynamic wait state generation, netX 51/52 provides a ready signal input. Ready signal usage can be configured for each chip-select individually. Ready activity level and signal filtering must be configured collectively for all chip-selects.

Several ready generating devices can be combined by using pull-up or pull-down resistors to generate a ready-state. In this case, devices must drive the signal in wait state only. However, to obtain good timing results, we recommend using only one ready generating device.

Ready status information is available for all chip-selects. A timeout-detection is implemented. Timeout errors can be associated with a system IRQ. In case of an error, the corresponding address and chip-select will be logged for debugging or error handling. Timeout is fixed: 10 μ s (1024 netX cycles).

Note: Ready signal timeout-detection can be disabled, but be careful when doing so.

When enabled; ready signal must be set to wait state before the wait-phase ends. The access cycle will be performed upon detection of the ready-state, but not before the programmed wait phase has passed completely. Ready state detection may take up to 6 cycles depending on the ready filter setting (reference [1], register *ext_rdy_cfg*).

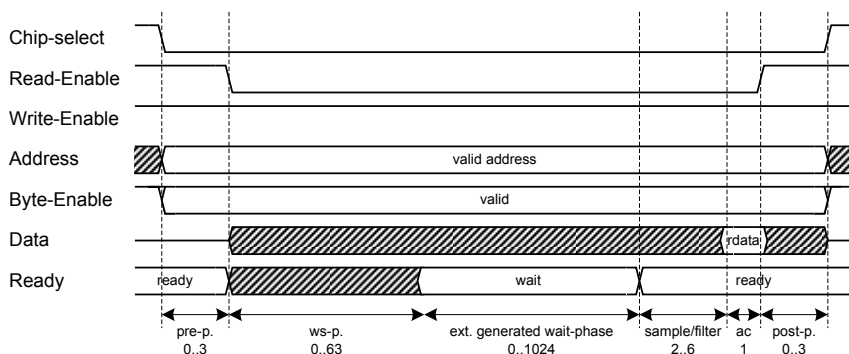


Figure 26: Read access with external wait state generation

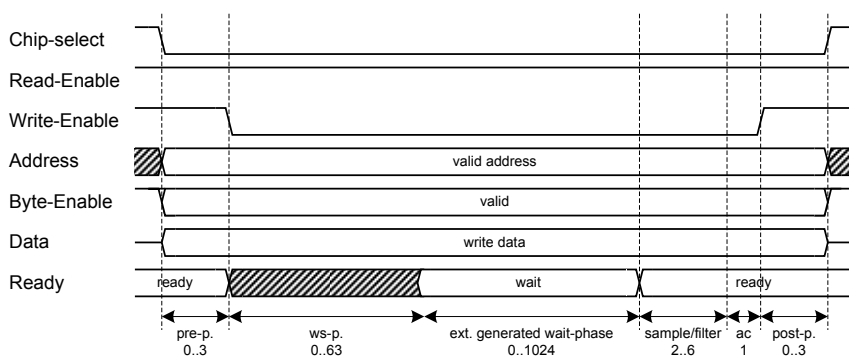


Figure 27: Read access with external wait state generation

Note: Ready signal to extend access operations is a new netX 51/52 feature.

3.9.1.4 SRAM burst related performance issues

Memory devices like Flash or NVRAM normally do not need pre and post-pauses between sequential reads (see section *SRAM access timing* on page 37). To achieve good performance, it is recommended to set the disable-bits (*no_p_pre_seq_rd*, *no_p_post_seq_rd*) in the eMI configuration register for the related chip-select (see reference [1]).

To reach the max. data bandwidth for APM devices, avoid idle cycles between read accesses. If there are idle cycles between subsequent read access operations, the SRAM controller will deselect the external device. That leads to long initial access on the next read start (see section *Asynchronous page mode (APM) bursts* on page 40).

Note: To avoid idle cycles, use the ARM LDM (load multiple) instruction or the *mem_copy* C function.

Note: Memory devices like Flash or NVRAM normally do not need pre and post-pauses between sequential reads (see device datasheet for details). To achieve good performance, set the disable bits (*no_p_pre_seq_rd*, *no_p_post_seq_rd*) in the eMI configuration register of the related chip-select (see reference [1]). For consecutive reads, addresses can be used in any order, i.e. addresses need not be incrementing.

Note: Pre and post-pause disable between sequential reads and APM bursts are new netX 51/52 features.

3.9.2 SDRAM interface

The SDRAM controller can drive all SDRAM single data rate types from 16 MBit to 512 MBit, providing a 1 GByte address space from 0x80000000 to 0xBFFFFFFF.

The following parameters can be set:

- Number of banks 2, 4
- Number of rows 2 k, 4 k, 8 k, 16 k
- Number of columns 256, 512, 1 k, 2 k, 4 k
- Data size 32 and 16-bit
- Refresh mode high and low priority
- Power save mode SDRAM self-refresh mode with disabled clock
switch on/off SDRAM controller

The SDRAM data bus width can be either 16 or 32-bit. To reach max. memory performance, the use of 32-bit is recommended. However, depending on your performance requirements, 16-bit may still be the better choice for some applications due to reduced effort for the PCB layout and smaller board size.

The following table shows all supported memory combinations up to 128 (256) Mbytes total memory.

SDRAM memory size	Organization	Number of chips	Configuration	Total memory size
64 MBit	2 MBit x 32	1	1x32	8 MBytes
	4 MBit x 16	1	1x16	8 MBytes
	4 MBit x 16	2	2x16	16 Mbytes ¹⁾
128 MBit	4 MBit x 32	1	1x32	16 Mbytes
	8 MBit x 16	1	1x16	16 Mbytes
	8 MBit x 16	2	2x16	32 Mbytes ¹⁾
256 MBit	8 MBit x 32	1	1x32	32 Mbytes
	16 MBit x 16	1	1x16	32 Mbytes
	16 MBit x 16	2	2x16	64 Mbytes ¹⁾
512 MBit	16 MBit x 32	1	1x32	64 Mbytes
	32 MBit x 16	1	1x16	64 Mbytes
	32 MBit x 16	2	2x16	128 Mbytes ¹⁾
	64 MBit x 8	4	4x8	256 Mbytes ²⁾

Table 15: SDRAM memory size and organization

Notes

1. If more than one memory device is connected (e.g. two 16-bit SDRAMs instead of one 32-bit component or SDRAM and parallel FLASH or SRAM), the allowed max. load capacity (data signals: max. 15 pF, control signals max. 10 pF) has to be considered to ensure stable operation throughout the whole temperature and operating voltage range!
2. Although the SDRAM controller basically supports 4-chip configurations, such configurations will most likely exceed the max. allowed load capacity!

3.9.2.1 SDRAM component connection

The following schematics show how to connect the SDRAM to netX 51 for 16 bit or 32 bit.

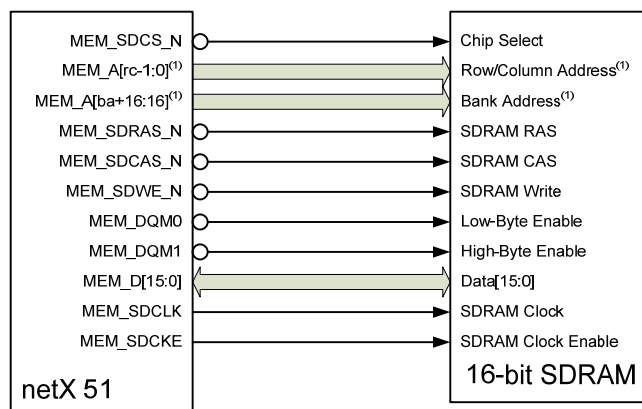


Figure 28: 16-bit SDRAM connection of netX 51

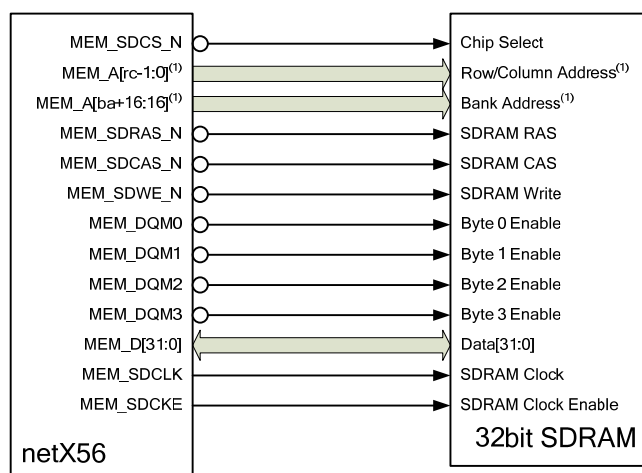


Figure 29: 32-bit standard SDRAM connection of netX 51

For SDRAM component connection 32Mx16 and 16Mx32, see reference [2].

Notes

1. The width of row/column (rc) and bank address (ba) depends on the size of the SDRAM device. The width can be configured in the SDRAM controller configuration registers. For row/column addressing, lines MEM_A0 to MEM_A10 are always used while usage of MEM_A11 to MEM_A13 depends on the device size (rc in figures above). For bank addressing, line MEM_A16 is always used for BA0 and typically (for devices with 4 banks) also MEM_A17 for BA1 (ba in figures above). The appropriate values must be taken from the data sheet of the used SDRAM device.
2. You can use several small SDRAM devices instead of a single big one, e.g. you can use two 16-bit devices to make up one 32-bit memory. If you do this, always observe the max. loads and timing.

Additional note

Note: SDRAM bank I/Os have been changed from MEM_A14, MEM_A15 to MEM_A16, MEM_A17 by netX 51/52 metal-fix for netX 50 compatibility. See section *netX 51/52 SDRAM bank address metal-fix* issues on page 350.

3.9.2.2 SDRAM access optimization

To benefit from the high-performance netX-external memory interface SDRAM controller, software memory mapping of all system masters running applications in SDRAM must be realized very accurately considering SDRAM properties. For netX 51/52, this basically means:

- different masters should not work on SDRAM simultaneously
- separate code and data to different banks if both must be located in SDRAM
- accesses should be optimized (avoid random access, use generated address incrementing or decrementing accesses instead)

3.9.2.2.1 SDRAM access introduction

In SDRAM devices 4 banks can typically be addressed by a bank address (device address MSBs). Each bank contains up to 16 K pages addressable by a row address (1 K = 1024). Each page contains up to 4 K data words.

To access a certain data word of a certain page of a certain bank, the page first must be opened by the ACTIVATE command to the corresponding bank and row address. When the page is open, data of this page can be addressed by the READ or WRITE command. Read data will be delivered from the SDRAM CAS-latency clocks after a READ command was set up.

If a page of a certain bank is already open and access to another page inside this bank is required, the current page must be closed and the new one opened by the PRECHARGE command first.

SDRAM (non DDR, QDR) devices are always connected by an address bus, a data bus, some control signals and one clock signal to a host device e.g. netX. SDRAM action is always performed at the rising edge of the clock signal. I.e. time base of all SDRAM action is SDRAM clock period.

Note: netX family always runs SDRAM on system frequency (100 MHz if not stepped down)

Min. cycle times are specified for SDRAM devices between different SDRAM commands. The SDRAM controller is initialized with these timings and generates appropriate pauses (idle cycles) on the SDRAM interface. These timings depend on used SDRAM device (available timings are given in the table of section *Difference between host interface SDRAM* and external memory interface SDRAM on page 47).

3.9.2.2.2 SDRAM data mapping optimization

Typically one SDRAM page contains 256 to 2048 words (word size is data bus width). As far as the current page is not left (access within an address boundary appropriate to the page size), PRECHARGE and ACTIVATE is not required. This is very important as the SDRAM controller must insert pauses of several clock cycles on the external memory interface to avoid SDRAM timing violations.

The following two figures show the difference between read accesses within one page boundary and accesses exceeding this boundary (write case is similar).

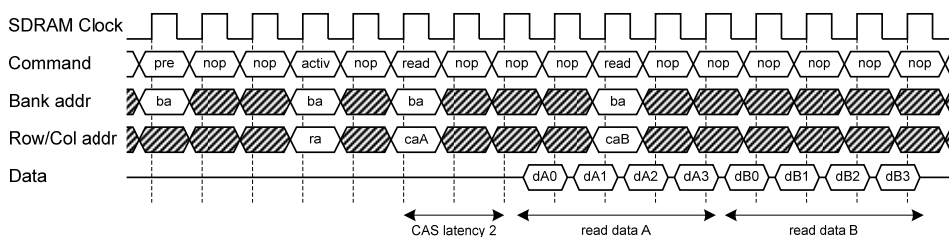


Figure 30: Read access from different SDRAM banks

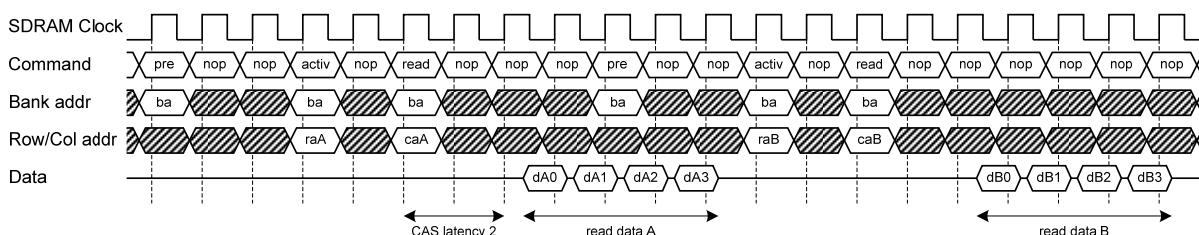


Figure 31: Read access from the same SDRAM banks

To avoid decreasing memory performance by PRECHARGE and ACTIVATE periods the following rules must be attended:

- Applications should keep addressing in page boundary.
- No multi-master SDRAM usage (netX 51/52 only due to reduced SDRAM controller)

3.9.2.3 SDRAM data caches issues

There is a 4 Dword read cache and a 4 Dword write cache implemented in netX 51/52 SDRAM eMI part to avoid burst runaways section *Difference between host interface SDRAM* and external memory interface SDRAM on page 47).

Note: Caches are always enabled and cannot be disabled in the netX 51/52 eMI. Bursts of 4 Dwords are always performed externally i.e. burst length 4 for 32-bit external data bus, burst length 8 for 16-bit external data bus.

Access between 4 Dword caches and external SDRAM devices is always performed by a 4 Dword burst externally. These bursts are never terminated. In the worst case, a whole 4 Dword write burst is performed if only one byte is written to the SDRAM device. All other data is unmasked by the SDRAM DQM-signals (Data Qualifying Mask).

- To reach good performance, data should be read or written with incrementing or decrementing addresses. Random access should be avoided.

Example 1 (bad):

```
load reg0, [addr]
load reg1, [addr+16]
load reg2, [addr+4]
load reg3, [addr+20]
load reg4, [addr+8]
load reg5, [addr+24]
```

Example 2 (good):

```
load reg0, [addr]
load reg2, [addr+4]
load reg4, [addr+8]
load reg1, [addr+16]
load reg3, [addr+20]
load reg5, [addr+24]
```

3.9.2.4 SDRAM performance features

Related to netX 50 some performance optimizations have been carried out: Now, there are mainly 2 separate channels for ARM data and instructions which will significantly increase ARM performance on SDRAM. Additionally, some SDRAM sequence optimization concerning SDRAM page handling has been performed. Also improved netX 10 SDRAM refresh generation is adopted in netX 51/52.

3.9.2.4.1 Difference between host interface SDRAM and external memory interface SDRAM

The external memory interface SDRAM controller is a high-performance multi-channel and multi-cache controller as known from netX 100 and netX 50 (with some further optimization). In contrast to this, the SDRAM controller of the host interface memory controller is a single-channel SDRAM-light controller. All accesses from netX system masters are handled sequentially there while they are handled in parallel by the external memory interface SDRAM controller. Due to this, the external memory interface SDRAM controller is able to interleave the access sequences caused by different masters when accessing SDRAM simultaneously. That leads to a much better performance there.

However, for single master application SDRAM performance of both interfaces is equal (when used same data width, timing etc.).

Note: The ARM-performance on host interface SDRAM is about the same as for netX 51/52 external Memory Interface SDRAM controller (when host interface SDRAM is not shared with other masters at netX 51/52). However, ARM performance is significantly increased on external memory interface SDRAM as ARM is connected by 2 channels there now. This benefit is not available for host interface SDRAM.

3.9.2.4.2 SDRAM timing parameters

The following list provides an overview of timing parameters which can be programmed in netX 51/52 SDRAM controller configuration registers. Programming values depend on the SDRAM device used and must be taken from SDRAM device data sheet.

Note: Compared with earlier netX chips, the existing timing parameters have remained unchanged.

Timing requirement	Pause (100 MHz)
ACTIVATE to READ or WRITE (same bank)	1 ... 3 clocks
ACTIVATE bank A to ACTIVATE bank B	1 ... 3 clocks
WRITE recovery time (last written data to PRECHARGE)	1 ... 3 clocks
CAS latency (read data on bus after READ command setup)	2 ... 3 clocks
PRECHARGE to command time	1 ... 3 clocks
ACTIVATE to PRECHARGE time	3 ... 10 clocks
REFRESH to command time	4 ... 19 clocks

3.9.2.4.3 SDRAM burst data caches

To reach max. data bandwidth by min. access load on the external memory interface, caches for SDRAM accesses are implemented in the netX family eMI controller. There is a 4 Dword read data cache and a 4 Dword write data cache for each SDRAM channel.

Read caches avoid read burst runaway if a master inserts idle cycles (or cycles to other address areas, e.g. instruction when data is in SDRAM) between accesses or in case of subsequent accesses of 8-bit data.

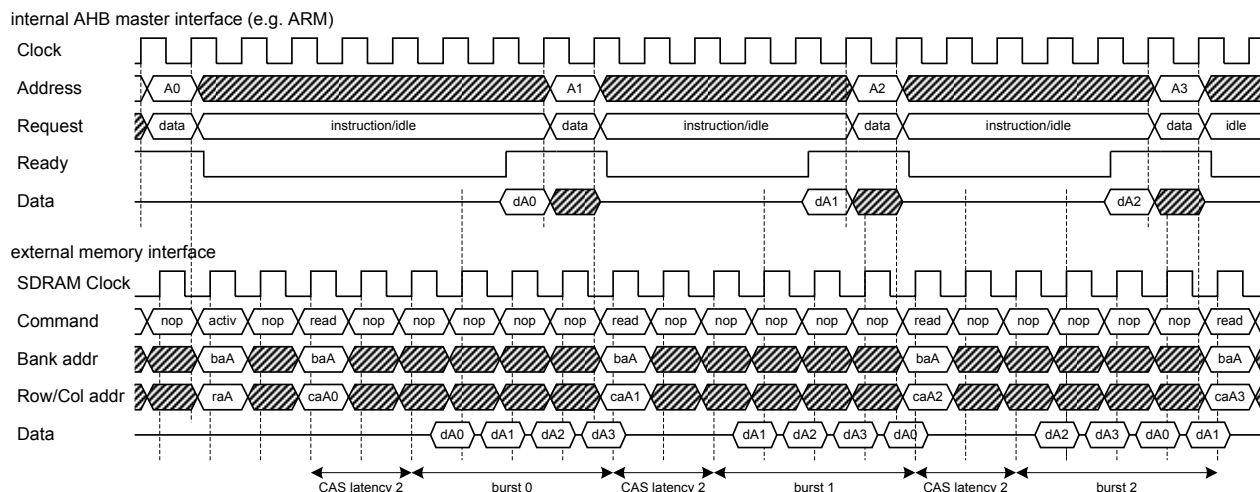


Figure 32: ARM data read burst runaway without caches (avoided by netX MI)

Note: The first data read externally is always related to the current requested address. For max. performance, a continuous read function is implemented in the caches. A burst wrap-around occurs whenever the first requested data is not located at address offset 0 within a burst boundary.

Example:

Burst-length 4, requested data on word-address offset 0 inside a burst boundary:
Access sequence is word 0, word 1, word 2, word 3.

Burst-length 4, requested data on word-address offset 2 inside a burst boundary:
Access sequence is word 2, word 3, word 0, word 1.

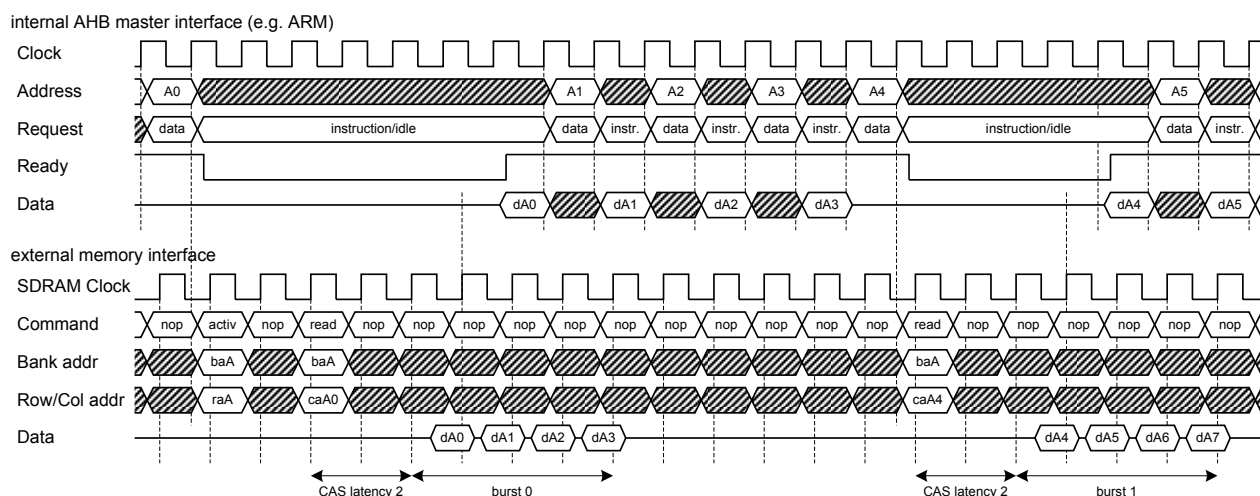


Figure 33: netX ARM cached read access

The comparison of Figure 32 and Figure 33 shows that the external bus load of the memory interface and internal master wait cycles would increase by more than a factor of 3 without read caches. This effect would become even worse if a master were running sub byte accesses.

Interrupted write accesses would produce similar behavior with the only difference that there are no CAS latency cycles externally between write-command and first data. For a better performance, a 4 Dword write cache is also implemented.

Note: External write bursts start at the address to which new internal data should be written to reach max. performance. A burst wrap-around occurs whenever the first requested data is not located at address offset 0 within a burst boundary.

Example:

Burst-length 4, requested data on word-address offset 0 inside a burst boundary:
Access sequence is word 0, word 1, word 2, word 3.

Burst-length 4, requested data on word-address offset 2 inside a burst boundary:
Access sequence is word 2, word 3, word 0, word 1.

3.9.2.4.4 SDRAM refresh generation

SDRAM refresh cycles (PRECHARGE ALL, AUTO REFRESH, ACTIVATE) typically take 120 ns (12 clocks) or more. To minimize worst case access time for SDRAM, refresh priority is lower than access priority. The netX external memory controller can collect up to 2048 refreshes which will be performed when no master requests access.

To avoid refresh errors, refresh will automatically be changed to high priority mode if it is not possible to perform enough refreshes in low-priority mode.

Note: High priority refresh mode is new since netX 10. Due to this mode the refresh status bit need not be checked any more (as necessary for netX 100 and netX 50). However, this bit is still useful for debugging purposes.

4096 refreshes must typically be performed within one all-rows refresh cycle time (64 ms) i.e. refresh period of 15.6 μ s. The SDRAM controller sets the status flag if refresh collecting might become critical in low priority refresh mode. E.g. the memory interface is busy for a long time (typically 2048*(64 ms/4096) i.e. 32 ms). In this case, the hardware will automatically change the refresh generation to high priority mode. As soon as refresh collecting is no longer critical, the mode will change again. The hardware will thus avoid errors caused by missing refreshes.

3.9.3 Device connection to HIF-memory

The host interface memory controller supports 8, 16 and 32-bit SRAM devices and 16 or 32-bit SDRAM devices. However, signal mapping varies and some configurations are mutually exclusive. E.g. using 32-bit SRAM or SDRAM will limit SRAM address area to 256 KB, provides only 2 chip-select areas and no ready input. Additionally, data-lines are ordered differently in 32 and 8 or 16-bit modes. See section *Host interface* on page 51 for host interface setup (HIF_IO_CTRL). Read this carefully before using and designing the HIF-eMI.

3.9.4 Simultaneous operation of SDRAM and parallel Flash memory at the memory interface

For netX 51, the following max. load capacity applies for operation of SDRAM at the **Memory Interface**:

Signal	Max. load capacity
MEM_D0 ... D31	15 pF
All other signals	10 pF

Important: Never connect a parallel Flash memory to the Memory Interface when using SDRAM memory.

Provided that no SDRAM memory is connected at the same time, operating one or more parallel flash memories is possible at the Memory Interface, even if the max. load capacity is thereby exceeded.

The SDRAM controller in the netX must be **deactivated** when operating flash memory to avoid wrong address mapping causing data errors.

Note: The limitations described above do not apply for the host interface. It is possible to operate SDRAM memory and parallel Flash Memory at the host interface at the same time. Max. load capacity for SDRAM at the host interface is 30 pF.

This section provides a detailed description of the structure, applications and configuration of the host interface (HIF) of netX 51/52.

Parallel DPM provides full netX 50 compatibility and is extended by 2 additional optional address lines (HIF_AHI0 and HIF_AHI1). However, these two signals are located on power-ball package positions of netX 50. To avoid damage caused by driving these 2 signals when netX 51 is used for netX 50 drop-in-replacement, they are protected against driving from netX.

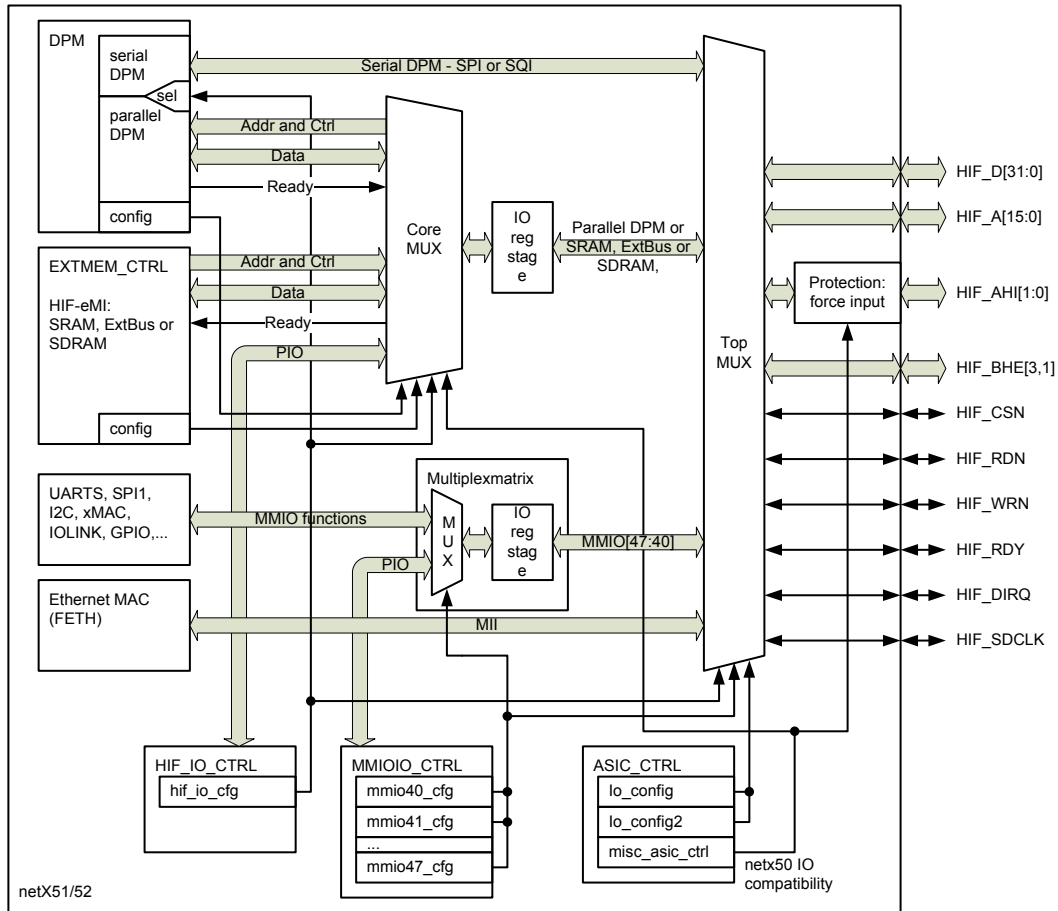


Figure 34: netX 51/52 Host interface IO structure (HIF-IO)

Important: Using netX 51/52 for a netX 50 drop-in-replacement and disabling netX 50 IO compatibility protection may cause permanent damage to netX 51/52.

The following list provides the key functions of netX 51/52 HIF I/Os:

- DPM
 - Parallel DPM
 - Fully compatible with netX 50 function and IO
 - 8, 16 or 32-bit data
 - 11 to 20 address-lines (external address range: 2 KB to 1024 KB)
 - Address-data-multiplexed modes
 - Optional Ready/Busy Output
 - Optional address comparator inputs for address bits A11 to A19
 - Optional address-enable, address-latch or second chip-select inputs
 - Serial DPM
 - 1-bit SPI supporting all 4 SPI modes (clock polarity and phase) up to 125 MHz
 - 4-bit QSPI supporting all 4 SPI modes (clock polarity and phase) up to 33 MHz
 - 2 optional interrupt outputs
 - Optional watchdog-active output
- Extension Bus
 - SRAM and SDRAM devices can be applied together in one design
 - SRAM-like devices:
 - 8 or 16-bit data
 - up to 4 chip-selects
 - up to 25 address-lines (up to 32 MBytes per cs)
 - 32-bit data
 - 1 or 2 chip-selects
 - up to 18 address-lines (up to 256 KBytes per cs)
 - Optional Ready/Busy input
 - wide timing configuration
 - APM (asynchronous page mode) burst support (only chip select 0)
 - special chip-select mode for i80 displays
 - netX 50 compatibility SRAM modes
 - SDRAM (single data rate):
 - 2 or 4 banks
 - 2K to 8K rows
 - 256 to 4K columns
 - 16 or 32-bit data (up to 512 MBytes)
 - wide timing configuration
- 8 MMIO (Multiple matrix signals MMIO40 to MMIO47)
 - Each MMIO can be controlled separately
 - 1 of 98 MMIO functions can be mapped for each MMIO

- Ethernet MII (PHY mode or MAC mode)
 - 3 different signal pinouts selectable to use MII together with:
 - up to 16 bit parallel DPM or serial DPM
 - up to 16 bit ExtBus/SDRAM (reduced additionally with serial DPM)
 - netJACK ETH and serial DPM
 - 6 configuration levels for each pinout
- PIO (programmable IO)
 - Any HIF IO not used for other purpose can be used as PIO in input, output or bidirectional mode.
 - Interrupt generating inputs (configurable for ARM and xPIC separately)
 - Input sampling controllable by enable-input
 - Power-on IO status (sample@nres)
 - Optional signal filtering
 - netX 50-compatibility mode for PIO programming and sampling

Some interfaces can be combined:

- Extension bus 8/16 can be extended by
 - a serial dual-port memory interface or
 - MMIO signals or
 - PIO
- SDRAM 16 can be extended by
 - a serial dual-port memory interface or
 - MMIO signals or
 - MII interface or
 - PIO
- MII Interface can be extended by
 - a serial dual-port memory interface or
 - MMIO signals or
 - PIO
- MII Interface can be extended by
 - a serial dual-port memory interface or
 - a parallel dual-port memory interface or
 - PIO

For this, see section

New signal assignment netX 51/52 for extension bus mode on page 54,

New signal assignment netX 51/52 for SDRAM mode on page 56,

New signal assignment netX 51/52 for MII mode on page 59 or

New signal assignment netX 51/52 for MII and DPM mode on page 61.

3.10.1 New signal assignment netX 51/52 for extension bus mode

The following table shows the new signal assignment for

- Extension bus 8/16/32-bit
- Extension bus 8/16-bit which can be extended by
 - Serial dual-port memory or
 - MMIO
 - PIO

Ball pos. netX 51	Ball pos. netX 52	Extension bus 8/16/32-bit mode signal	Extension bus 8/16-bit mode, extendible			
			signal	SPM extension signal	MMIO extension signal	PIO extension signal
C7	A3	EXT_A0/_BE0n	EXT_A0/_BE0n	-	-	PIO73
B8	B6	EXT_A1/_BE2n	EXT_A1	-	-	PIO70
C8	C6	EXT_A2	EXT_A2	-	-	PIO69
C10	C7	EXT_A3	EXT_A3	-	-	PIO66
A10	B8	EXT_A4	EXT_A4	-	-	PIO65
B9	A7	EXT_A5	EXT_A5	-	-	PIO64
C11	B9	EXT_A6	EXT_A6	-	-	PIO61
D11	C9	EXT_A7	EXT_A7	-	-	PIO60
C13	C13	EXT_A8	EXT_A8	-	-	PIO57
B12	A10	EXT_A9	EXT_A9	-	-	PIO56
C14	B12	EXT_A10	EXT_A10	-	-	PIO53
A17	A14	EXT_A11	EXT_A11	-	-	PIO50
B15	B13	EXT_A12	EXT_A12	-	-	PIO49
A16	A13	EXT_A13	EXT_A13	-	-	PIO48
B14	A12	EXT_A14	EXT_A14	-	-	PIO54
A15	B11	EXT_A15	EXT_A15	-	-	PIO55
E14	A17	EXT_A16	EXT_A22	-	-	-
D14	A18	EXT_A17	EXT_A23	-	-	-
A1	A1	EXT_D0	EXT_D0	-	-	PIO83
B2	C2	EXT_D1	EXT_D1	-	-	PIO82
C2	D2	EXT_D2	EXT_D2	-	-	PIO81
C6	B3	EXT_D3	EXT_D3	-	-	PIO78
A6	B4	EXT_D4	EXT_D4	-	-	PIO77
A3	B2	EXT_D5	EXT_D5	-	-	PIO76
A8	A5	EXT_D6	EXT_D6	-	-	PIO75
B7	B5	EXT_D7	EXT_D7	-	-	PIO74
J16	G18	EXT_D16	extendible by SPM, MMIO or PIO	SPM_MISO	MMIO40	PIO32
H15	G16	EXT_D17	extendible by SPM or MMIO	SPM_MOSI	MMIO41	PIO34
H16	F17	EXT_D18	extendible by SPM or MMIO	SPM_CSn	MMIO42	PIO33
G16	D18	EXT_D19	extendible by SPM or MMIO	SPM_CLK	MMIO43	PIO39
G18	C18	EXT_D26	extendible by SPM or MMIO	SPM_DIRQ	MMIO44	PIO38

Ball pos. netX 51	Ball pos. netX 52	Extension bus 8/16/32-bit mode signal	Extension bus 8/16-bit mode, extendible			
			signal	SPM extension signal	MMIO extension signal	PIO extension signal
G15	B18	EXT_D27	extendible by SPM or MMIO	SPM_SIRQ	MMIO45	PIO37
D18	C17	EXT_D30	extendible by SPM or MMIO	SPM_SIO2	MMIO46	PIO42
C18	B17	EXT_D31	extendible by SPM or MMIO	SPM_SIO3	MMIO47	PIO41
J18	G17	EXT_D8	EXT_D8	-	-	PIO36
H17	F18	EXT_D9	EXT_D9	-	-	PIO35
H18	E18	EXT_D10	EXT_D10	-	-	PIO40
G17	E17	EXT_D11	EXT_D11	-	-	PIOWDGACT
B13	C12	EXT_D20	EXT_A16	-	-	PIO58
A14	C11	EXT_D21	EXT_A17	-	-	PIO59
A13	A11	EXT_D22	EXT_A18	-	-	PIO62
C12	C10	EXT_D23	EXT_A19	-	-	PIO63
B10	A9	EXT_D24	EXT_A20	-	-	PIO67
A11	A8	EXT_D25	EXT_A21	-	-	PIO68
A9	A6	EXT_D12	EXT_D12	-	-	PIO71
C9	B7	EXT_D13	EXT_D13	-	-	PIO72
B6	A2	EXT_D28	EXT_CS2n	-	-	PIO79
A7	A4	EXT_D29	EXT_CS1n	-	-	PIO80
A2	D3	EXT_D14	EXT_D14	-	-	PIO84
J17	H17	EXT_D15	EXT_D15	-	-	PIO85
A18	A16	EXT_BE1n	EXT_BHEn	-	-	PIO43
C15	B16	EXT_BE3n	EXT_CS3n EXT_A24	-	-	PIO44
B16	A15	EXT_CS0n	EXT_CS0n	-	-	PIO51
B17	B15	EXT_RDn	EXT_RDn	-	-	PIO52
C16	B14	EXT_WRn	EXT_WRn	-	-	PIO45
B18	C15	EXT_BUSYn	EXT_BUSYn	-	-	PIO46
C17	D16	EXT_CS1n	EXT_INTn	-	-	PIO47
B11	C8	-	-	-	-	-

Table 16: New signal assignment netX 51/52 for extension bus mode

3.10.2 New signal assignment netX 51/52 for SDRAM mode

The following table shows the new signal assignment for

- SDRAM 32-bit
- SDRAM 16-bit which can be extended by
 - Serial dual-port memory or
 - MMIO
 - MII (Mode 2)
 - PIO

Ball pos. netX 51	Ball pos. netX 52	SDRAM 32-bit mode signal	SDRAM 16-bit mode, extendible				
			signal	SPM extension signal	MMIO extension signal	MI extension signal	PIO extension signal
C7	A3	SD_A0	SD_A0	-	-	-	
B8	B6	SD_A1	SD_A1	-	-	-	
C8	C6	SD_A2	SD_A2	-	-	-	
C10	C7	SD_A3	SD_A3	-	-	-	
A10	B8	SD_A4	SD_A4	-	-	-	
B9	A7	SD_A5	SD_A5	-	-	-	
C11	B9	SD_A6	SD_A6	-	-	-	
D11	C9	SD_A7	SD_A7	-	-	-	
C13	C13	SD_A8	SD_A8	-	-	-	
B12	A10	SD_A9	SD_A9	-	-	-	
C14	B12	SD_A10	SD_A10	-	-	-	
A17	A14	SD_A11	SD_A11	-	-	-	
B15	B13	SD_A12	-	-	-	-	PIO49
A16	A13	SD_DQM0n	SD_DQM0n	-	-	-	
B14	A12	SD_BA0	SD_BA0	-	-	-	
A15	B11	SD_BA1	SD_BA1	-	-	-	
E14	A17	SD_RASn	SD_RASn	-	-	-	
D14	A18	SD_CASn	SD_CASn	-	-	-	
A1	A1	SD_D0	SD_D0	-	-	-	
B2	C2	SD_D1	SD_D1	-	-	-	
C2	D2	SD_D2	SD_D2	-	-	-	
C6	B3	SD_D3	SD_D3	-	-	-	
A6	B4	SD_D4	SD_D4	-	-	-	
A3	B2	SD_D5	SD_D5	-	-	-	
A8	A5	SD_D6	SD_D6	-	-	-	
B7	B5	SD_D7	SD_D7	-	-	-	
J16	G18	SD_D16	extendible by SPM, MMIO, MII or PIO	SPM_MISO	MMIO40	MII2_COL	PIO32
H15	G16	SD_D17	extendible by SPM, MMIO, MII or PIO	SPM_MOSI	MMIO41	MII2_CRS	PIO34

Ball pos. netX 51	Ball pos. netX 52	SDRAM 32-bit mode signal	SDRAM 16-bit mode, extendible				
			signal	SPM extension signal	MMIO extension signal	MII extension signal	PIO extension signal
H16	F17	SD_D18	extendible by SPM, MMIO, MII or PIO	SPM_CS _n	MMIO42	MII2_TXER	PIO33
G16	D18	SD_D19	extendible by SPM, MMIO, MII or PIO	SPM_CLK	MMIO43	MII2_MDIO (MMIO43)	PIO39
G18	C18	SD_D26	extendible by SPM, MMIO, MII or PIO	SPM_DIRQ	MMIO44	MII2_MDC (MMIO44)	PIO38
G15	B18	SD_D27	extendible by SPM, MMIO, MII or PIO	SPM_SIRQ	MMIO45	MII2_RXD1	PIO37
D18	C17	SD_D30	extendible by SPM, MMIO, MII or PIO	SPM_SIO2	MMIO46	MII2_RXD2	PIO42
C18	B17	SD_D31	extendible by SPM, MMIO, MII or PIO	SPM_SIO3	MMIO47	MII2_RXD3	PIO41
J18	G17	SD_D8	SD_D8	-	-	-	-
H17	F18	SD_D9	SD_D9	-	-	-	-
H18	E18	SD_D10	SD_D10	-	-	-	-
G17	E17	SD_D11	SD_D11	-	-	-	-
B13	C12	SD_D20	extendible by MII or PIO	-	-	MII2_TXCLK	PIO58
A14	C11	SD_D21	extendible by MII or PIO	-	-	MII2_TXD0	PIO59
A13	A11	SD_D22	extendible by MII or PIO	-	-	MII2_TXD1	PIO62
C12	C10	SD_D23	extendible by MII or PIO	-	-	MII2_TXD2	PIO63
B10	A9	SD_D24	extendible by MII or PIO	-	-	MII2_TXD3	PIO67
A11	A8	SD_D25	extendible by MII or PIO	-	-	MII2_TXEN	PIO68
A9	A6	SD_D12	SD_D12	-	-	-	-
C9	B7	SD_D13	SD_D13	-	-	-	-
B6	A2	SD_D28	extendible by MII or PIO	-	-	MII2_RXCLK	PIO79
A7	A4	SD_D29	extendible by MII or PIO	-	-	MII2_RXER	PIO80
A2	D3	SD_D14	SD_D14	-	-	-	
J17	H17	SD_D15	SD_D15	-	-	-	
A18	A16	SD_DQM1 _n	SD_DQM1 _n	-	-	-	
C15	B16	SD_DQM3 _n	extendible by MII or PIO	-	-	MII2_RXDV	PIO44
B16	A15	SD_CS _n	SD_CS _n	-	-	-	-
B17	B15	SD_DQM2 _n	extendible by PIO	-	-	-	PIO52
C16	B14	SD_WEn	SD_WEn	-	-	-	-

Ball pos. netX 51	Ball pos. netX 52	SDRAM 32-bit mode signal	SDRAM 16-bit mode, extendible				
			signal	SPM extension signal	MMIO extension signal	MII extension signal	PIO extension signal
B18	C15	SD_CKE	SD_CKE	-	-	-	-
C17	D16		extendible by MII or PIO	-	-	MII2_RXD0	PIO47
B11	C8	SD_CLK	SD_CLK	-	-	-	-

Table 17: New signal assignment netX 51/52 for SDRAM mode

3.10.3 New signal assignment netX 51/52 for MII mode

The following table shows the new signal assignment for

- MII (Mode 3) which can be extended by
 - Serial dual-port memory or
 - MMIO or
 - PIO

Ball pos netX 51	Ball pos netX 52	MII mode, extendible			
		signal	SPM extension signal	MMIO extension signal	PIO extension signal
C7	A3	MII3_TXER	-	-	PIO73
B8	B6	MII3_COL	-	-	PIO70
C8	C6	MII3_CRS	-	-	PIO69
C10	C7	MII3_RXD0	-	-	PIO66
A10	B8	MII3_RXD1	-	-	PIO65
B9	A7	MII3_RXD2	-	-	PIO64
C11	B9	MII3_RXD3	-	-	PIO61
D11	C9	MII3_RXDV	-	-	PIO60
C13	C13	MII3_TXD0	-	-	PIO57
B12	A10	MII3_TXD1	-	-	PIO56
C14	B12	MII3_TXD2	-	-	PIO53
A17	A14	MII3_TXD3	-	-	PIO50
B15	B13	MII3_TXEN	-	-	PIO49
A16	A13	MII3_TXCLK	-	-	PIO48
B14	A12	-	-	-	PIO54
A15	B11	-	-	-	PIO55
E14	A17	-	-	-	-
D14	A18	-	-	-	-
A1	A1	-	-	-	PIO83
B2	C2	-	-	-	PIO82
C2	D2	-	-	-	PIO81
C6	B3	-	-	-	PIO78
A6	B4	-	-	-	PIO77
A3	B2	-	-	-	PIO76
A8	A5	-	-	-	PIO75
B7	B5	-	-	-	PIO74
J16	G18	-	SPM_MISO	MMIO40	PIO32
H15	G16	-	SPM_MOSI	MMIO41	PIO34
H16	F17	-	SPM_CS _n	MMIO42	PIO33
G16	D18	-	SPM_CLK	MMIO43	PIO39
G18	C18	-	SPM_DIRQ	MMIO44	PIO38
G15	B18	-	SPM_SIRQ	MMIO45	PIO37
D18	C17	MII3_MDIO (MMIO46)	-	MMIO46	PIO42
C18	B17	MII3_MDC (MMIO47)	-	MMIO47	PIO41
J18	G17	-	-	-	PIO36
H17	F18	-	-	-	PIO35

Ball pos netX 51	Ball pos netX 52	MII mode, extendible			
		signal	SPM extension signal	MMIO extension signal	PIO extension signal
H18	E18	-	-	-	PIO40
G17	E17	-	-	-	PIOWDGACT
B13	C12	-	-	-	PIO58
A14	C11	-	-	-	PIO59
A13	A11	-	-	-	PIO62
C12	C10	-	-	-	PIO63
B10	A9	-	-	-	PIO67
A11	A8	-	-	-	PIO68
A9	A6	-	-	-	PIO71
C9	B7	-	-	-	PIO72
B6	A2	-	-	-	PIO79
A7	A4	-	-	-	PIO80
A2	D3	-	-	-	PIO84
J17	H17	-	-	-	PIO85
A18	A16	MII3_RXER	-	-	PIO43
C15	B16	-	-	-	PIO44
B16	A15	-	-	-	PIO51
B17	B15	-	-	-	PIO52
C16	B14	-	-	-	PIO45
B18	C15	MII3_RXCLK	-	-	PIO46
C17	D16	-	-	-	PIO47
B11	C8	-	-	-	-

Table 18: New signal assignment netX 51/52 for MII mode

3.10.4 New signal assignment netX 51/52 for MII and DPM mode

The following table shows the new signal assignment for

- MII (Mode 1) which can be extended by
 - parallel dual-port memory or
 - serial dual-port memory or
 - PIO

Ball pos netX 51	Ball pos netX 52	MII mode, extendible			
		signal	Parallel Dual-Port Memory 8/16-bit extension signal	SPM extension signal	PIO extension signal
C7	A3	-	DPM_A0	-	PIO73
B8	B6	-	DPM_A1	-	PIO70
C8	C6	-	DPM_A2	-	PIO69
C10	C7	-	DPM_A3	-	PIO66
A10	B8	-	DPM_A4	-	PIO65
B9	A7	-	DPM_A5	-	PIO64
C11	B9	-	DPM_A6	-	PIO61
D11	C9	-	DPM_A7	-	PIO60
C13	C13	-	DPM_A8	-	PIO57
B12	A10	-	DPM_A9	-	PIO56
C14	B12	-	DPM_A10	-	PIO53
A17	A14	-	DPM_A11	-	PIO50
B15	B13	-	DPM_A12	-	PIO49
A16	A13	-	DPM_A13	-	PIO48
B14	A12	-	DPM_A14	-	PIO54
A15	B11	-	DPM_A15	-	PIO55
E14	A17	MII1_COL	-	-	-
D14	A18	MII1_CRS	-	-	-
A1	A1	-	DPM_D0	-	PIO83
B2	C2	-	DPM_D1	-	PIO82
C2	D2	-	DPM_D2	-	PIO81
C6	B3	-	DPM_D3	-	PIO78
A6	B4	-	DPM_D4	-	PIO77
A3	B2	-	DPM_D5	-	PIO76
A8	A5	-	DPM_D6	-	PIO75
B7	B5	-	DPM_D7	-	PIO74
J16	G18	-	DPM_D8	SPM_MISO/SIO1	PIO32
H15	G16	-	DPM_D9	SPM_MOSI/SIO0	PIO34
H16	F17	-	DPM_D10	SPM_CS _n	PIO33
G16	D18	-	DPM_D11	SPM_CLK	PIO39
G18	C18	-	DPM_D12	SPM_DIRQ	PIO38
G15	B18	-	DPM_D13	SPM_SIRQ	PIO37
D18	C17	-	DPM_D14	SPM_SIO2	PIO42
C18	B17	-	DPM_D15	SPM_SIO3	PIO41
J18	G17	MII1_TXER	-	-	-
H17	F18	-	DPM_ALE16	-	PIO35

Ball pos netX 51	Ball pos netX 52	MII mode, extendible			
		signal	Parallel Dual-Port Memory 8/16-bit extension signal	SPM extension signal	PIO extension signal
H18	E18	MII1_RXDV	-	-	-
G17	E17	-	WDGACT	-	-
B13	C12	MII1_TXCLK	-	-	-
A14	C11	MII1_TXD0	-	-	-
A13	A11	MII1_TXD1	-	-	-
C12	C10	MII1_TXD2	-	-	-
B10	A9	MII1_TXD3	-	-	-
A11	A8	MII1_TXEN	-	-	-
A9	A6	MII1_RXD0	-	-	-
C9	B7	MII1_RXD1	-	-	-
B6	A2	MII1_RXCLK	-	-	-
A7	A4	MII1_RXER	-	-	-
A2	D3	MII1_RXD2	-	-	-
J17	H17	MII1_RXD3	-	-	-
A18	A16	-	DPM_BHE _n	-	PIO43
C15	B16	-	DPM_WRH _n	-	PIO44
B16	A15	-	DPM_CS _n	-	PIO51
B17	B15	-	DPM_RD _n	-	PIO52
C16	B14	-	DPM_WRL _n	-	PIO45
B18	C15	-	DPM_RDY	-	PIO46
C17	D16	-	DPM_DIRQ	-	PIO47
B11	C8	-	DPM_SIRQ	-	-

Table 19: New signal assignment netX 51/52 for MII mode

The signal MII_MDIO and MII_MDC have to be assigned to two MMIO signals.

3.10.5 Signal assignment netX 50 pin-compatible mode

The following table shows the signal assignment for

- Parallel dual-port memory 8/16 bit
- Parallel dual-port memory 32 bit
- Extension bus 8/16 bit

These modes are compatible with netX 50.

Ball pos. netX 51	Ball pos. netX 52	Parallel Dual-port Memory 8/16-bit mode (netX 50 pin-compatible) signal	Parallel Dual-port Memory 32-bit mode (netX 50 pin-compatible) signal	Extension bus 8/16-bit mode (netX 50 pin-compatible) signal
C7	A3	DPM_A0/BE0n	DPM_BE0n	EXT_A0 EXT_BE0n
B8	B6	DPM_A1/BE2n	DPM_BE2n	EXT_A1
C8	C6	DPM_A2	DPM_A2	EXT_A2
C10	C7	DPM_A3	DPM_A3	EXT_A3
A10	B8	DPM_A4	DPM_A4	EXT_A4
B9	A7	DPM_A5	DPM_A5	EXT_A5
C11	B9	DPM_A6	DPM_A6	EXT_A6
D11	C9	DPM_A7	DPM_A7	EXT_A7
C13	C13	DPM_A8	DPM_A8	EXT_A8
B12	A10	DPM_A9	DPM_A9	EXT_A9
C14	B12	DPM_A10	DPM_A10	EXT_A10
A17	A14	DPM_A11	DPM_A11	EXT_A11
B15	B13	DPM_A12	DPM_A12	EXT_A12
A16	A13	DPM_A13	DPM_A13	EXT_A13
B14	A12	DPM_A14	DPM_A14	EXT_A14
A15	B11	DPM_A15	DPM_A15 DPM_ALE32_NX50	EXT_A15
E14	A17		DPM_A16	
D14	A18		DPM_A17 DPM_ALE32_NX51	
A1	A1	DPM_D0	DPM_D0	EXT_D0
B2	C2	DPM_D1	DPM_D1	EXT_D1
C2	D2	DPM_D2	DPM_D2	EXT_D2
C6	B3	DPM_D3	DPM_D3	EXT_D3
A6	B4	DPM_D4	DPM_D4	EXT_D4
A3	B2	DPM_D5	DPM_D5	EXT_D5
A8	A5	DPM_D6	DPM_D6	EXT_D6
B7	B5	DPM_D7	DPM_D7	EXT_D7
J16	G18	DPM_D8	DPM_D8	EXT_D8_NX50
H15	G16	DPM_D9	DPM_D9	EXT_D9_NX50
H16	F17	DPM_D10	DPM_D10	EXT_D10_NX50
G16	D18	DPM_D11	DPM_D11	EXT_D11_NX50
G18	C18	DPM_D12	DPM_D12	EXT_D12_NX50
G15	B18	DPM_D13	DPM_D13	EXT_D13_NX50
D18	C17	DPM_D14	DPM_D14	EXT_D14_NX50

Ball pos. netX 51	Ball pos. netX 52	Parallel Dual-port Memory 8/16-bit mode (netX 50 pin-compatible) signal	Parallel Dual-port Memory 32-bit mode (netX 50 pin-compatible) signal	Extension bus 8/16-bit mode (netX 50 pin-compatible) signal
C18	B17	DPM_D15	DPM_D15	EXT_D15_NX50
J18	G17		DPM_D16	
H17	F18	DPM_ALE16	DPM_D17	
H18	E18	DPM_SELA19	DPM_D18	EXT_A24_NX50
G17	E17		DPM_D19	
B13	C12	DPM_A16	DPM_D20	EXT_A16
A14	C11	DPM_A17	DPM_D21	EXT_A17
A13	A11	DPM_A18	DPM_D22	EXT_A18
C12	C10	DPM_A19	DPM_D23	EXT_A19
B10	A9	DPM_SELA12	DPM_D24	EXT_A20
A11	A8	DPM_SELA13	DPM_D25	EXT_A21
A9	A6	DPM_SELA14	DPM_D26	EXT_A22_NX50
C9	B7	DPM_SELA15	DPM_D27	EXT_A23_NX50
B6	A2	DPM_SELA16	DPM_D28	EXT_CS2n
A7	A4	DPM_SELA17	DPM_D29	EXT_CS1n
A2	D3	DPM_SELA18	DPM_D30	EXT_CS3n_NX50
J17	H17	DPM_SELA11	DPM_D31	
A18	A16	DPM_BHEn/CS2	DPM_BE1n	EXT_BHEn
C15	B16	DPM_WRHn	DPM_BE3n	
B16	A15	DPM_CSn	DPM_CSn	EXT_CS0n
B17	B15	DPM_RDn	DPM_RDn	EXT_RDn
C16	B14	DPM_WRn/WRLn	DPM_WRn	EXT_WRn
B18	C15	DPM_BUSYn	DPM_BUSYn	EXT_BUSYn
C17	D16	DPM_DIRQn	DPM_DIRQn	EXT_INTn
B11	C8	DPM_SIRQn	DPM_SIRQn	

Table 20: Signal assignment netX 50 pin-compatible mode

3.10.6 Pinout functions combinations and exclusions

HIF-functions are not mutually exclusive, e.g. Serial DPM can be used with 16-bit HIF-SDRAM. However, parallel DPM cannot be used together with HIF-SDRAM (as both functions use the same IOs). The following table provides information on all functions HIF-IOs can be used for. It shows which functions can be combined and which functions cannot be combined with other functions.

Important: Several HIF functions require signal mapping that does not match naming of HIF-IOs. E.g. second data-byte of SDRAM-function is not applied to HIF_D[15:8]. To avoid trouble, carefully read the detailed interconnection section concerning your application in *Device connection to host interface* on page 69.

Note: In the following table sometimes function-classes are used in the columns '*Can be used with*' or '*Cannot be used with*' instead of listing all functions of a certain class. E.g. 'pDPM' means the class of all parallel DPM function, i.e. pDPM8, pDPM16 or pDPM32. A function class encloses all functions starting with the same name-pattern. E.g. HIF-ETH2 encloses HIF-ETH2 and HIF-ETH2r.

Note: The number of PIOs that is max. available is calculated without any optional signal (e.g. only 12 address-lines and no ready signal for parallel DPM). For typical applications, less PIOs are available. The correct number of PIOs results from the customer's application requirements. Available PIOs are additionally reduced if multiple functions are used together (e.g. sDPM together with 16 bit SDRAM).

Function (notes)	Features	max PIOs or other function	netJACK	netX50-compatible	can be used with	cannot be used with
pDPM8 (2)	8-bit data parallel DPM. Up to 20 address-lines. Optional address-compare input for A11 and second IRQ (SIRQ) are provided for host.	36	DPM	yes	HIF-ETH1 HIF-MMIO	sDPM nx50ExtBus ExtBus HIF-SDRAM HIF-ETH2 HIF-ETH3
pDPM16 (2)	16-bit data parallel DPM. Up to 20 address-lines. Optional address-compare input for A11 and second IRQ (SIRQ) are provided for host.	27	DPM	yes	HIF-ETH1	sDPM nx50ExtBus ExtBus HIF-SDRAM HIF-ETH2 HIF-ETH3 HIF-MMIO
pDPM32 (2)	32-bit data parallel DPM. Up to 18 address-lines. 2 IRQs (SIRQ, DIRQ) for host. No address-compare inputs.	10	no	no	nothing	any other
sDPM1 (1), (2), (10)	Serial 4-wire 1-bit SPI DPM. 2 IRQs (SIRQ, DIRQ) for host.	54	SPM	no	ExtBus8 ExtBus16 HIF-SDRAM16 HIF-ETH1 HIF-ETH2r HIF-ETH3 HIF-MMIO46 HIF-MMIO46	pDPM nx50ExtBus ExtBus32 HIF-SDRAM32 HIF-ETH2f other HIF-MMIO
sDPM4 (10)	Serial 6-wire 4-bit QSPI DPM. 2 IRQs (SIRQ, DIRQ) for host.	52	SPM	no	ExtBus8 ExtBus16 HIF-SDRAM16 HIF-ETH1 HIF-ETH3	pDPM nx50ExtBus ExtBus32 HIF-SDRAM32 HIF-ETH2 HIF-MMIO
nx50ExtBus8 (3), (11)	8-bit data SRAM-like. Up to 4 chip-selects. Up to 25 address-lines. netX 50 IO-compatible, but special netX 50 Intel or Motorola modes no longer available. Optional Ready input.	34	no	yes but	nothing	could be shared, but it is strongly recommended not to do so for new designs. Use ExtBus8 instead.
nx50ExtBus16 (3), (11)	16-bit data SRAM-like. Up to 4 chip-selects. Up to 25 address-lines. netX 50 IO-compatible, but special netX 50 Intel or Motorola modes no longer available. Optional Ready input.	25	no	yes but	nothing	any other

Function (notes)	Features	max PIOs or other function	netJACK	netX50-compatible	can be used with	cannot be used with
ExtBus8 (1), (3), (4), (5), (7)	8-bit data SRAM-like. Up to 4 chip-selects. Up to 25 address-lines. Optional Ready input.	36	no	no	sDPM ExtBus HIF-SDRAM HIF-ETH2 HIF-MMIO	pDPM nx50ExtBus HIF-ETH1 HIF-ETH3
ExtBus16 (1), (3), (4), (5), (7)	16-bit data SRAM-like. Up to 4 chip-selects. Up to 25 address-lines. Optional Ready input.	27	no	no	sDPM ExtBus HIF-SDRAM HIF-ETH2 HIF-MMIO	pDPM nx50ExtBus HIF-ETH1 HIF-ETH3
ExtBus32 (3), (4), (5), (6), (7)	32-bit data SRAM-like. Up to 2 chip-selects. Up to 18 address-lines. Optional Ready input.	9	no	no	ExtBus HIF-SDRAM	pDPM sDPM nx50ExtBus HIF-ETH HIF-MMIO
HIF-SDRAM16 (1), (4),	16-bit data SDRAM 1 chip-select. Up to 15 address-lines: up to 13 for row, column, 1 or 2 for banks.	20	no	no	sDPM ExtBus HIF-SDRAM HIF-ETH2 HIF-MMIO	pDPM nx50ExtBus HIF-ETH1 HIF-ETH3
HIF-SDRAM32 (4), (5)	32-bit data SDRAM 1 chip-select. Up to 15 address-lines: up to 13 for row, column, 1 or 2 for banks	2	no	no	ExtBus	pDPM sDPM nx50ExtBus HIF-ETH HIF-MMIO
HIF-ETH1 (8)	Ethernet MII pinout 1	42	no	no	pDPM8 pDPM16 sDPM MMIO	pDPM32 nx50ExtBus ExtBus HIF-SDRAM
HIF-ETH2f (8), (10)	full Ethernet MII pinout 2	42	no	no	ExtBus8 ExtBus16 HIF-SDRAM16 HIF-MMIO43 HIF-MMIO44	pDPM sDPM nx50ExtBus ExtBus32 HIF-SDRAM32 other HIF-MMIO
HIF-ETH2r (1), (8), (10)	reduced Ethernet MII pinout 2.	42	no	no	sDPM1 ExtBus8 ExtBus16 HIF-SDRAM16 HIF-MMIO40 to HIF-MMIO44	pDPM sDPM4 nx50ExtBus ExtBus32 HIF-SDRAM32 other HIF-MMIO
HIF-ETH3f (8)	full Ethernet MII pinout 3.	42	ETH	no	sDPM HIF-MMIO	pDPM nx50ExtBus ExtBus HIF-SDRAM

Function (notes)	Features	max PIOs or other function	netJACK	netX50-compatible	can be used with	cannot be used with
HIF-ETH3r (8), (9)	reduced Ethernet MII pinout 3. Minimalistic ExtBus possible (e.g. i80 display).	42	ETH	no	sDPM HIF-MMIO i80-ExtBus	pDPM nx50ExtBus other ExtBus HIF-SDRAM
i80-ExtBus (9)	Minimalistic 8 or 16 bit ExtBus e.g. for i80 displays)			no	HIF-ETH3r	all other
HIF-MMIO (10)	MMIO40 to 47 function on IOs HIF_D[15:8].	50	no	no	pDPM8 nx50ExtBus8 ExtBus8 ExtBus16 HIF-SDRAM16 HIF-ETH1 HIF-ETH3	sDPM4 nx50ExtBus16 ExtBus32 HIF-SDRAM32

Table 21: Pinout functions combinations and exclusions

Notes

- All three sDPM1, HIF-ETH2r and ExtBus8/ExtBus16/HIF-SDRAM16 can be applied together. However, sDPM1 provides only 1 IRQ (SIRQ) then.
- Attention: IRQ signal mapping of sDPM and pDPM modes differ:
 - pDPM-SIRQ: HIF_SDCLK, pDPM-DIRQ: HIF_DIRQ
 - sDPM-SIRQ: HIF_D13, sDPM-DIRQ: HIF_D12
- HIF_DIRQ can be used as IRQ input. This is compatible with netX 50 PIO47 IRQ.
- There is no Ready/Busy signal available when ExtBus is used together with HIF-SDRAM (as HIF_RDY is SDRAM CKE).
- There are the following restrictions concerning ExtBus8, ExtBus16 or ExtBus32 when shared with HIF-SDRAM32:
 - There is only 1 chip-select available for ExtBus. CS1 must be used, but this is remapped to HIF_DIRQ then.
 - There are only up to 18 address-lines available.
 - There is no IRQ input available on HIF-IOs (as HIF_DIRQ is chip-select).
- ExtBus32 can be used with IRQ and Ready/Busy input when CS0 is used as single chip-select. HIF-SDRAM is not possible then.
- ExtBus provides up to 4 chip-selects (CS0...3). There is no requirement using the same data width for all chip-selects. However, there could be further restrictions concerning the amount of available chip-selects or address lines then.
- There are 6 different configuration levels for each HIF-ETH pinout providing more or less signals depending on required MII signals. 'Reduced' means here configuration level 4 or lower (i.e. without 'collision', 'carrier sense' and 'transmit error').
- i80 ExtBus mode is only for combination with netJACK-ETH/HIF-ETH3r. It is a standard ExtBus mode where only 3 address lines are available. For other sharing i80 devices could be connected to standard ExtBus.
- Each HIF-MMIO can be enabled separately on IO. That allows using some HIF-MMIO when HIF_D[15:8] are not completely used by another function. E.g. MMIO46 and 47 could be

used together with sDPM. Or MMIO45 could be used together with sDPM4 when sDPM4 requires only 1 IRQ and does not use DPM_SPI_SIRQ on HIF_D13.

11. The functions nx50EtxBus8 and nx50EtxBus16 are dedicated for replacement of netX 50 by netX 51 of existing designs. It is strongly recommended not to use them for new designs.

3.10.7 Device connection to host interface

3.10.7.1 Parallel DPM functions (pDPM)

Pin-functions of parallel DPM:

Pin function	direction	parallel DPM function	related config. registers	notes
dpm_a[19:0]	input	address lines	dpm_addr_cfg.addr_range dpm_cfg0x0.mode dpm_if_cfg.addr_sh dpm_if_cfg.be_sel	1, 2
dpm_d[31:0]	inout	data lines	dpm_cfg0x0.mode	
dpm_csn	input	chip-select	dpm_if_cfg.cs_ctrl	3
dpm_cs2n	input	second chip-select	dpm_if_cfg.cs_ctrl	3
dpm_rdn	input	read or data direction control	dpm_if_cfg.dir_ctrl	
dpm_wrn	input	write control (or low byte write strobe)	dpm_if_cfg.dir_ctrl dpm_if_cfg.be_sel	
dpm_be[3:0]	input	byte enables (or byte write strobes for pDPM32)	dpm_if_cfg.be_sel dpm_if_cfg.be_wr_dis dpm_if_cfg.be_rd_dis	4
dpm_be8	input	byte enable or strobe for pDPM8	dpm_if_cfg.be_sel dpm_if_cfg.be_wr_dis dpm_if_cfg.be_rd_dis	4
dpm_be16l	input	alternative netX 50-compatible low byte enable or strobe for pDPM16 (can be used for netX 50 dpm_wrln)	dpm_if_cfg.be_sel dpm_if_cfg.be_wr_dis dpm_if_cfg.be_rd_dis	4
dpm_be16h	input	alternative netX 50-compatible high byte enable or strobe for pDPM16 (can be used for netX 50 dpm_wrhcn)	dpm_if_cfg.be_sel dpm_if_cfg.be_wr_dis dpm_if_cfg.be_rd_dis	4
dpm_ale	input	address latch or enable for pDPM8 and pDPM16	dpm_cfg0x0.mode dpm_if_cfg.aen_pol	
dpm_ale32_nx50	input	address latch or enable for pDPM32 netX 50-compatible	dpm_cfg0x0.mode dpm_if_cfg.aen_sel dpm_if_cfg.aen_pol	
dpm_ale32	input	address latch or enable for pDPM32 not netX 50-compatible, +2 address line on HIF_AH10,1	dpm_cfg0x0.mode dpm_if_cfg.aen_sel dpm_if_cfg.aen_pol	
dpm_sela[19:11]	input	compare address inputs for internal address decoding (only pDPM8 or pDPM16)	dpm_addr_cfg.addr_range dpm_addr_cfg.addr_cmp_a*	5
dpm_rdy	output	ready/busy for host	dpm_rdy_cfg dpm_pio_cfg1.sel_rdy_pio	6
dpm_dirq	output	IRQ 1 for host (e.g. for data-IRQ)	dpm_io_cfg_misc.irqoc dpm_io_cfg_misc.irqpol dpm_pio_cfg1.sel_dirq_pio	6, 7
dpm_sirq	output	IRQ 2 for host (e.g. for service-IRQ)	dpm_io_cfg_misc.fiqoc dpm_io_cfg_misc.fiqpol dpm_pio_cfg1.sel_sirq_pio	6, 7

Pin function	direction	parallel DPM function	related config. registers	notes
dpm_isamcs16n	output	ISA memcs16n	dpm_if_cfg. isa_bhe3_is_memcs16n	
wdgact	output	watchdog active for host	hif_io_cfg. en_hif_wdg_sys_hif_d19 netx_sys_wdg_ctrl	

Table 22: Pin functions of parallel DPM functions (pDPM)

Notes

1. Address lines 0 to 10 are mandatory for non-multiplexed DPM modes, higher lines are optional. For multiplexed modes (i.e. address and data are applied to dpm_d lines, selection is made by address latch signal) address lines 8 to 10 are mandatory for pDPM8 and unused for pDPM16 and pDPM32.
2. DPM address bus can be byte-oriented with byte enables on low address lines or word-oriented (i.e. byte address is right-shifted according to used data size) without byte enables.
3. DPM can be configured with 1 or 2 or even without chip select signals, but omitting chip select is not recommended.
4. Byte enables have many different functions. There are various different signal mappings possible, they can be used as byte strobes, active-polarity can be configured etc. They can even be ignored, but this is not recommended and should be used for netX 50-compatibility only. For details, see description of related configuration bits.
5. Participation of each dpm_sela input can be controlled separately. An internal programmable address compare value can be used alternatively.
6. Signal is in PIO mode by default and must be set to functional mode first.
7. Attention: HIF IOs used for IRQs differ for serial and parallel DPM mapping, but IRQ function itself is the same.

IO functions for parallel DPM. Functions in “()” brackets are only optional, e.g. (dpm_a15).

HIF-IO	pDPM8	pDPM16	pDPM32	notes
hif_a0	dpm_a0	dpm_a0 dpm_be0	dpm_a0 dpm_be0	1 2
hif_a1	dpm_a1	dpm_a1	dpm_a1 dpm_be2	1 2
hif_a2	dpm_a2	dpm_a2	dpm_a2	1
hif_a3	dpm_a3	dpm_a3	dpm_a3	1
hif_a4	dpm_a4	dpm_a4	dpm_a4	1
hif_a5	dpm_a5	dpm_a5	dpm_a5	1
hif_a6	dpm_a6	dpm_a6	dpm_a6	1
hif_a7	dpm_a7	dpm_a7	dpm_a7	1
hif_a8	dpm_a8	dpm_a8	dpm_a8	1
hif_a9	dpm_a9	dpm_a9	dpm_a9	1
hif_a10	dpm_a10	dpm_a10	dpm_a10	1
hif_a11	(dpm_a11)	(dpm_a11)	(dpm_a11)	
hif_a12	(dpm_a12)	(dpm_a12)	(dpm_a12)	
hif_a13	(dpm_a13)	(dpm_a13)	(dpm_a13)	
hif_a14	(dpm_a14)	(dpm_a14)	(dpm_a14)	
hif_a15	(dpm_a15)	(dpm_a15)	(dpm_a15) (dpm_ale32_nx50)	3
hif_ahi0	-	-	(dpm_a16)	4
hif_ahi1	-	-	(dpm_a17) (dpm_ale32)	4 3
hif_d0	dpm_d0	dpm_d0	dpm_d0	
hif_d1	dpm_d1	dpm_d1	dpm_d1	
hif_d2	dpm_d2	dpm_d2	dpm_d2	
hif_d3	dpm_d3	dpm_d3	dpm_d3	
hif_d4	dpm_d4	dpm_d4	dpm_d4	
hif_d5	dpm_d5	dpm_d5	dpm_d5	
hif_d6	dpm_d6	dpm_d6	dpm_d6	
hif_d7	dpm_d7	dpm_d7	dpm_d7	
hif_d8	-	dpm_d8	dpm_d8	
hif_d9	-	dpm_d9	dpm_d9	
hif_d10	-	dpm_d10	dpm_d10	
hif_d11	-	dpm_d11	dpm_d11	
hif_d12	-	dpm_d12	dpm_d12	
hif_d13	-	dpm_d13	dpm_d13	
hif_d14	-	dpm_d14	dpm_d14	
hif_d15	-	dpm_d15	dpm_d15	
hif_d16	-	-	dpm_d16	
hif_d17	(dpm_ale)	(dpm_ale)	dpm_d17	3
hif_d18	(dpm_sela19)	(dpm_sela19)	dpm_d18	
hif_d19	(wdgact)	(wdgact)	dpm_d19	
hif_d20	(dpm_a16)	(dpm_a16)	dpm_d20	
hif_d21	(dpm_a17)	(dpm_a17)	dpm_d21	
hif_d22	(dpm_a18)	(dpm_a18)	dpm_d22	
hif_d23	(dpm_a19)	(dpm_a19)	dpm_d23	

HIF-IO	pDPM8	pDPM16	pDPM32	notes
hif_d24	(dpm_sela12)	(dpm_sela12)	dpm_d24	
hif_d25	(dpm_sela13)	(dpm_sela13)	dpm_d25	
hif_d26	(dpm_sela14)	(dpm_sela14)	dpm_d26	
hif_d27	(dpm_sela15)	(dpm_sela15)	dpm_d27	
hif_d28	(dpm_sela16)	(dpm_sela16)	dpm_d28	
hif_d29	(dpm_sela17)	(dpm_sela17)	dpm_d29	
hif_d30	(dpm_sela18)	(dpm_sela18)	dpm_d30	
hif_d31	(dpm_sela11)	(dpm_sela11)	dpm_d31	
hif_bhe1	(dpm_be8) (dpm_cs2n)	dpm_be1 (dpm_cs2n)	dpm_be1 (dpm_cs2n)	2
hif_bhe3		(dpm_be16h) (dpm_isamcs16n)	dpm_be3	2
hif_csn	dpm_csn	dpm_csn	dpm_csn	
hif_rdn	dpm_rdn	dpm_rdn	dpm_rdn	5
hif_wrn	dpm_wrn	dpm_wrn (dpm_be16l)	dpm_wrn	5
hif_rdy	(dpm_rdy)	(dpm_rdy)	(dpm_rdy)	6
hif_dirq	(dpm_dirq)	(dpm_dirq)	(dpm_dirq)	
hif_sdclk	(dpm_sirq)	(dpm_sirq)	(dpm_sirq)	

Table 23: IO functions for parallel DPM

Notes

1. For multiplexed modes address is located on data lines. Low address lines are not used then. However, address lines exceeding data bus width (e.g. dpm_a[10:8] for pDPM8) and byte enables are used.
2. Byte enables and byte strobes can be configured as 'to be ignored' and are not really mandatory. However, this is intended for netX 50-compatibility only and is not recommended for new designs.
3. Address latch signal becomes mandatory for multiplexed modes. Address latch signal location varies depending data width and netX 50-compatibility.
4. Attention: netX 50 power pad location. Not for netX 50-drop-in-replacement.
5. Using hif_rdn as data direction line (e.g. old Motorola or Intel CPU like) instead of using dedicated read and write control signals (SRAM like) is possible. hif_wrn could be omitted then. For details, see the register description of dpm_if_cfg.dir_ctrl.
6. Basically no ready/busy signal needs to be used. However, netX access times vary widely depending on target address. Hence, it is recommended to use ready signal to reach good performance.

3.10.7.2 Serial DPM functions (sDPM)

Pin-functions of serial DPM:

pin-function	direction	parallel DPM function	related config. registers	Notes
dpm_spi_csn	input	chip select		
dpm_spi_clk	input	serial clock	hif_io_cfg.sel_dpm_serial_spo hif_io_cfg.sel_dpm_serial_sph	
dpm_spi_mosi	input	1-bit SPI netX receive data		
dpm_spi_miso	output	1-bit SPI netX send data	dpm_timing_cfg. sdpm_miso_early	
dpm_sio[3:0]	inout	4-bit QSPI data	dpm_timing_cfg. en_dpm_serial_sqi	
dpm_spi_dirq	output	IRQ 1 for host (e.g. for data-IRQ)	dpm_io_cfg_misc.irqoec dpm_io_cfg_misc.irqppl dpm_pio_cfg1.sel_dirq_pio	1
dpm_spi_sirq	output	IRQ 2 for host (e.g. for service-IRQ)	dpm_io_cfg_misc.fiqoec dpm_io_cfg_misc.fiqppl dpm_pio_cfg1.sel_sirq_pio	1
wdgact	output	watchdog active for host	hif_io_cfg. en_hif_wdg_sys_hif_d19 netx_sys_wdg_ctrl	

Table 24: Pin functions of serial DPM

Note

1. Attention: HIF IOs used for IRQs differ for serial and parallel DPM mapping, but IRQ function itself is the same.

IO functions for serial DPM. Functions in “()” brackets are only optional, e.g. (dpm_spi_dirq). All HIF IOs not listed below can be used for other function.

HIF-IO	sDPM1	sDPM4	Notes
hif_d8	dpm_spi_miso	dpm_sio1	
hif_d9	dpm_spi_mosi	dpm_sio0	
hif_d10	dpm_spi_csn	dpm_spi_csn	
hif_d11	dpm_spi_clk	dpm_spi_clk	
hif_d12	(dpm_spi_dirq)	(dpm_spi_dirq)	
hif_d13	(dpm_spi_sirq)	(dpm_spi_sirq)	
hif_d14	-	dpm_sio2	
hif_d15	-	dpm_sio3	
hif_d19	(wdgact)	(wdgact)	1

Table 25: IO functions for serial DPM

Note

1. Watchdog function is for serial DPM identical to parallel DPM watchdog.

3.10.7.3 Extension bus functions (nx50ExtBus, ExtBus, HIF-SDRAM)

Pin-functions of HIF-extension bus and HIF-SDRAM functions

Pin function	direction	parallel DPM function	related config. registers	notes
ext_a[24:0]	output	address lines	hif_io_cfg.sel_hif_a_width sdram_general_ctrl.columns sdram_general_ctrl.rows	1, 5
ext_d[31:0]	inout	data lines	hif_io_cfg.hif_mi_cfg sdram_general_ctrl.dbus32 extsram*_ctrl.dwidth	5
ext_cs[3:0]	output	SRAM chip-selects	extsram*_ctrl.dwidth	2, 5
ext_rd	output	SRAM read control		
ext_wr	output	SRAM write control		
ext_be[3:0]	output	byte enables		2
ext_bhe	output	high byte enables for 16-bit data SRAM/ExtBus		2, 3
ext_rdy	input	ready/busy signal from device	hif_io_cfg.en_hif_rdy_pio_mi ext_rdy_cfg	5
ext_int	input	interrupt signal from device		6
sd_cs	output	SDRAM chip-selects	hif_io_cfg.en_hif_sdram_mi	2
sd_a[1:0]	output	SDRAM low address lines		4
sd_ba[1:0]	output	SDRAM bank address	sdram_general_ctrl.banks	
sd_ras	output	SDRAM row access control		2
sd_cas	output	SDRAM column access control		2
sd_we	output	SDRAM write control		2
sd_dqm[3:0]	output	SDRAM data qualifiers (byte enables)		2
sd_cke	output	SDRAM clock enable		
sd_clk	output	SDRAM clock		

Table 26: Pin functions of HIF extension bus and HIF SDRAM functions

Notes

1. Address lines 0 to 10 are mandatory for non-multiplexed DPM modes, higher lines are optional. For multiplexed modes (i.e. address and data are applied to dpm_d lines; selection is made by address latch signal) address lines 8 to 10 are mandatory for pDPM8 and unused for pDPM16 and pDPM32.
2. Signal is low active. enough
3. ext_bhe and ext_be1 are identical.
4. ext_a2 and higher are used for higher SDRAM address lines
5. Use configuration registers of HIF-eMI (address area hif_asyncmem_ctrl or hif_sdram_ctrl), not of MEM-eMI (not address area ext_asyncmem_ctrl or ext_sdram_ctrl).
6. IRQ input ext_int is a HIF-PIO function rather than an ExtBus function. However, ext_int is listed here because this IRQ input can be used for all ExtBus signal mappings. For more information, see 3.10.7.6.

IO-functions of HIF-Extension bus and HIF-SDRAM:

HIF-IO	nx50ExtBus8	ExtBus8	nx50ExtBus16	ExtBus16 HIF-SDRAM16	ExtBus32 HIF-SDRAM32	Notes
hif_a0	ext_a0	ext_a0	ext_a0	ext_a0	ext_be0 sd_a0	1
hif_a1	ext_a1	ext_a1	ext_a1	ext_a1	ext_be2 sd_a1	1
hif_a2	ext_a2	ext_a2	ext_a2	ext_a2	ext_a2	
hif_a3	ext_a3	ext_a3	ext_a3	ext_a3	ext_a3	
hif_a4	ext_a4	ext_a4	ext_a4	ext_a4	ext_a4	
hif_a5	ext_a5	ext_a5	ext_a5	ext_a5	ext_a5	
hif_a6	ext_a6	ext_a6	ext_a6	ext_a6	ext_a6	
hif_a7	ext_a7	ext_a7	ext_a7	ext_a7	ext_a7	
hif_a8	ext_a8	ext_a8	ext_a8	ext_a8	ext_a8	
hif_a9	ext_a9	ext_a9	ext_a9	ext_a9	ext_a9	
hif_a10	ext_a10	ext_a10	ext_a10	ext_a10	ext_a10	
hif_a11	(ext_a11)	(ext_a11)	(ext_a11)	(ext_a11)	(ext_a11)	2
hif_a12	(ext_a12)	(ext_a12)	(ext_a12)	(ext_a12)	(ext_a12)	
hif_a13	(ext_a13)	(ext_a13)	(ext_a13)	(ext_a13) sd_dqm0	(ext_a13) sd_dqm0	6
hif_a14	(ext_a14)	(ext_a14)	(ext_a14)	(ext_a14) sd_ba0	(ext_a14) sd_ba0	6
hif_a15	(ext_a15)	(ext_a15)	(ext_a15)	(ext_a15) sd_ba1	(ext_a15) sd_ba1	6
hif_a16	!	(ext_a22)	!	(ext_a22) sd_ras	(ext_a16) sd_ras	3, 6
hif_a17	!	(ext_a23)	!	(ext_a23) sd_cas	(ext_a17) sd_cas	3, 6
hif_d0	ext_d0	ext_d0	ext_d0	ext_d0	ext_d0	
hif_d1	ext_d1	ext_d1	ext_d1	ext_d1	ext_d1	
hif_d2	ext_d2	ext_d2	ext_d2	ext_d2	ext_d2	
hif_d3	ext_d3	ext_d3	ext_d3	ext_d3	ext_d3	
hif_d4	ext_d4	ext_d4	ext_d4	ext_d4	ext_d4	
hif_d5	ext_d5	ext_d5	ext_d5	ext_d5	ext_d5	
hif_d6	ext_d6	ext_d6	ext_d6	ext_d6	ext_d6	
hif_d7	ext_d7	ext_d7	ext_d7	ext_d7	ext_d7	
hif_d8	-	-	ext_d8	-	ext_d16	4
hif_d9	-	-	ext_d9	-	ext_d17	4
hif_d10	-	-	ext_d10	-	ext_d18	4
hif_d11	-	-	ext_d11	-	ext_d19	4
hif_d12	-	-	ext_d12	-	ext_d26	4
hif_d13	-	-	ext_d13	-	ext_d27	4
hif_d14	-	-	ext_d14	-	ext_d30	4
hif_d15	-	-	ext_d15	-	ext_d31	4
hif_d16	-	-	-	ext_d8	ext_d8	4
hif_d17	-	-	-	ext_d9	ext_d9	4
hif_d18	(ext_a24)	-	(ext_a24)	ext_d10	ext_d10	4
hif_d19	-	-	-	ext_d11	ext_d11	4
hif_d20	(ext_a16)	(ext_a16)	(ext_a16)	(ext_a16)	ext_d20	

HIF-IO	nx50ExtBus8	ExtBus8	nx50ExtBus16	ExtBus16 HIF-SDRAM16	ExtBus32 HIF-SDRAM32	Notes
hif_d21	(ext_a17)	(ext_a17)	(ext_a17)	(ext_a17)	ext_d21	
hif_d22	(ext_a18)	(ext_a18)	(ext_a18)	(ext_a18)	ext_d22	
hif_d23	(ext_a19)	(ext_a19)	(ext_a19)	(ext_a19)	ext_d23	
hif_d24	(ext_a20)	(ext_a20)	(ext_a20)	(ext_a20)	ext_d24	
hif_d25	(ext_a21)	(ext_a21)	(ext_a21)	(ext_a21)	ext_d25	
hif_d26	(ext_a22)	-	(ext_a22)	ext_d12	ext_d12	4
hif_d27	(ext_a23)	-	(ext_a23)	ext_d13	ext_d13	4
hif_d28	(ext_cs2)	(ext_cs2)	(ext_cs2)	(ext_cs2)	ext_d28	
hif_d29	(ext_cs1)	(ext_cs1)	(ext_cs1)	(ext_cs1)	ext_d29	
hif_d30	(ext_cs3)	-	(ext_cs3)	ext_d14	ext_d14	4
hif_d31	-	-	-	ext_d15	ext_d15	4
hif_bhe1	-	-	ext_bhe	ext_bhe sd_dqm1	ext_be1 sd_dqm1	6
hif_bhe3	-	(ext_cs3) (ext_a24)	-	(ext_cs3) (ext_a24)	ext_be3 sd_dqm3	6, 7
hif_csn	ext_cs0	ext_cs0	ext_cs0	ext_cs0 sd_cs	ext_cs0 sd_cs	5, 9
hif_rdn	ext_rd	ext_rd	ext_rd	ext_rd sd_dqm2	ext_rd sd_dqm2	6
hif_wrn	ext_wr	ext_wr	ext_wr	ext_wr sd_we	ext_wr sd_we	6
hif_rdy	(ext_rdy)	(ext_rdy)	(ext_rdy)	(ext_rdy) sd_cke	(ext_rdy) sd_cke	8
hif_dirq	(ext_int)	(ext_int)	(ext_int)	(ext_int)	(ext_cs1) (ext_int)	9
hif_sdclk	-	-	-	sd_clk	sd_clk	

Table 27: IO functions of HIF extension bus and HIF SDRAM

Notes

1. For SRAM-ExtBus access byte address is always applied to address bus, i.e. address on bus is not right-shifted according to data size. For 32-bit SRAM devices hif_a0 and hif_a1 carry byte enables. ext_a2 (hif_a2) is address bit 0 of 32-bit SRAM device then. For SDRAM: access always word address is generated, i.e. address on bus is right-shifted according to data size. Address bit 0 is located on hif_a0 and bit 1 on hif_a1 then. SDRAM byte enable (data qualifiers, sd_dqm-signals) location differs from SRAM byte enables.
2. Address lines 0 to 10 are always driven by netX when ExtBus is selected. Higher address lines can be disabled by software and available for PIO-usage then.
3. Attention: netX 50 power pad location. Not for netX 50-drop-in-replacement.
4. Attention: ExtBus data lines are reordered when netX 50 compatibility mode is not used in order to share ExtBus16 or HIF-SDRAM16 with sDPM or MMIOs in one design. This is not possible with netX 50 signal mapping. Be careful: ext_d ordering differs from hif_d ordering (e.g. hif_d16 is used as ext_d8 instead of ext_d16).
5. Must be used as SDRAM chip-select when SDRAM function is desired. SRAM chip select 0 is not available then. Chip selects are mutually exclusive, i.e. it is not allowed to connect hif_csn to SRAM and SDRAM in one design.
6. Must be connected to both: SRAM (ExtBus) and SDRAM device when both types are desired. SDRAM function will be applied to this IO during SDRAM access while SRAM

function will be applied during SRAM/ExtBus access. This is switched dynamically by internal netX memory controller.

7. Usage of ext_cs3 and ext_a24 are mutually exclusive for when not netX 50-compatible. I.e. when chip select 3 is used address is limited to 24 bits or when 25 address bits are used there are only 3 chip selects available.
8. ExtBus ready/busy function is not available when SDRAM is used. sd_cke is a mandatory SDRAM signal.
9. Only 2 chip select signals are available when using 32-bit data. The following applications are possible when either SDRAM or ExtBus/SRAM are 32-bit data:
 - A. HIF-SDRAM32 stand alone (sd_cs on hif_csn)
 - B. ExtBus32 stand alone (ext_cs0 on hif_csn). Both ready/busy and IRQ input are available then.
 - C. SDRAM (sd_cs on hif_csn) + ExtBus(ext_cs1 on hif_dirq). Both ready/busy and IRQ input are not available then.
 - D. ExtBus (ext_cs0 on hif_csn) + ExtBus(ext_cs1 on hif_dirq). Ready/busy is available, but IRQ input is not available then.

3.10.7.4 MMIO functions (MMIO40 to MMIO47)

Pin-functions of HIF-MMIOs:

Pin function	direction	MMIO function	related config. registers	notes
mmio40	input, output or in out	One of 97 netX 51/52 MMIO core functions		1
mmio41	input, output or in out	One of 97 netX 51/52 MMIO core functions		
:	:	:	:	
mmio47	input, output or in out	One of 97 netX 51/52 MMIO core functions		

Table 28: Pin functions of HIF MMIOs

Note

1. MMIO signal direction depends on selected netX 51/52 MMIO core function.

The following table provides netX 51/52 MMIO core functions:

no.	core function	signal direction	description
0	xm0_io0	in out	Fieldbus0
1	xm0_io1	in out	Fieldbus0
2	xm0_io2	in out	Fieldbus0
3	xm0_io3	in out	Fieldbus0
4	xm0_io4	in out	Fieldbus0
5	xm0_io5	in out	Fieldbus0
6	xm0_rx	input	Fieldbus0
7	xm0_tx_out	tristatable output	Fieldbus0
8	xm1_io0	in out	Fieldbus1
9	xm1_io1	in out	Fieldbus1
10	xm1_io2	in out	Fieldbus1
11	xm1_io3	in out	Fieldbus1
12	xm1_io4	in out	Fieldbus1
13	xm1_io5	in out	Fieldbus1
14	xm1_rx	in out	Fieldbus1
15	xm1_tx_out	tristatable output	Fieldbus1
16	gpio0	in out	GPIO/IO-Link
17	gpio1	in out	GPIO/IO-Link
18	gpio2	in out	GPIO/IO-Link
19	gpio3	in out	GPIO/IO-Link
20	gpio4	in out	GPIO/IO-Link
21	gpio5	in out	GPIO/IO-Link
22	gpio6	in out	GPIO/IO-Link
23	gpio7	in out	GPIO/IO-Link
24	gpio8	in out	GPIO/IO-Link
25	gpio9	in out	GPIO/IO-Link
26	gpio10	in out	GPIO/IO-Link
27	gpio11	in out	GPIO/IO-Link
28	gpio12	in out	GPIO/IO-Link
29	gpio13	inout	GPIO/IO-Link
30	gpio14	in out	GPIO/IO-Link
31	gpio15	in out	GPIO/IO-Link
32	gpio16	in out	GPIO/IO-Link
33	gpio17	in out	GPIO/IO-Link
34	gpio18	in out	GPIO/IO-Link
35	gpio19	in out	GPIO/IO-Link
36	gpio20	in out	GPIO/IO-Link
37	gpio21	in out	GPIO/IO-Link
38	gpio22	in out	GPIO/IO-Link
39	gpio23	in out	GPIO/IO-Link
40	gpio24	in out	GPIO/IO-Link
41	gpio25	in out	GPIO/IO-Link
42	gpio26	in out	GPIO/IO-Link
43	gpio27	in out	GPIO/IO-Link
44	gpio28	in out	GPIO/IO-Link
45	gpio29	in out	GPIO/IO-Link

no.	core function	signal direction	description
46	gpio30	in out	GPIO/IO-Link
47	gpio31	in out	GPIO/IO-Link
48	phy0_led0	always driven output	INT_PHY_CTRL0 link
49	phy0_led1	always driven output	INT_PHY_CTRL0 trans
50	phy0_led2	always driven output	INT_PHY_CTRL0 speed100
51	phy0_led3	always driven output	INT_PHY_CTRL0 duplex
52	phy1_led0	always driven output	INT_PHY_CTRL1 link
53	phy1_led1	always driven output	INT_PHY_CTRL1 trans
54	phy1_led2	always driven output	INT_PHY_CTRL1 speed100
55	phy1_led3	always driven output	INT_PHY_CTRL1 duplex
56	mii_mdc	always driven output	MDIO
57	mii_mdio	in out	MDIO
58	spi0_cs2n	in out	SPI0 3rd chip select
59	spi0_sio2_mmio	in out	SPI0
60	spi0_sio3_mmio	in out	SPI0
61	spi1_clk	in out	SPI1
62	spi1_cs0n	in out	SPI1
63	spi1_cs1n	in out	SPI1
64	spi1_cs2n	in out	SPI1
65	spi1_miso	in out	SPI1
66	spi1_mosi	in out	SPI1
67	i2c0_scl_mmio	in out	I2C
68	i2c0_sda_mmio	in out	I2C
69	i2c1_scl	in out	I2C
70	i2c1_sda	in out	I2C
71	xc_sample0	input	Trigger/Latch Unit
72	xc_sample1	input	Trigger/Latch Unit
73	xc_trigger0	tristatable output	Trigger/Latch Unit
74	xc_trigger1	tristatable output	Trigger/Latch Unit
75	uart0_ctsn	input	UART 0
76	uart0_rtsn	tristatable output	UART 0
77	uart0_rxd	input	UART 0
78	uart0_txd	tristatable output	UART 0
79	uart1_ctsn	input	UART 1
80	uart1_rtsn	tristatable output	UART 1
81	uart1_rxd	input	UART 1
82	uart1_txd	tristatable output	UART 1
83	uart2_ctsn	input	UART 2
84	uart2_rtsn	tristatable output	UART 2
85	uart2_rxd	input	UART 2
86	uart2_txd	tristatable output	UART 2
87	can_rx	input	CAN
88	can_tx	always driven output	CAN
89	mem_rdy	input	MEM IF ready/busy input
90	pio0	in out	PIO
91	pio1	in out	PIO
92	pio2	in out	PIO

no.	core function	signal direction	description
93	pio3	in out	PIO
94	pio4	in out	PIO
95	pio5	in out	PIO
96	pio6	in out	PIO
97	pio7	in out	PIO

Table 29: MMIO core functions

IO functions for MMIOs. Functions in “()” brackets are only optional. All HIF IOs not listed below can be used for other function.

HIF-IO	MMIO	Notes
hif_d8	(MMIO40)	1
hif_d9	(MMIO41)	1
hif_d10	(MMIO42)	1
hif_d11	(MMIO43)	1
hif_d12	(MMIO44)	1
hif_d13	(MMIO45)	1
hif_d14	(MMIO46)	1
hif_d15	(MMIO47)	1

Table 30: IO functions for MMIOs

Note

1. Each MMIO can be controlled and activated separately.

3.10.7.5 Ethernet functions of HIF (HIF-ETH)

Ethernet can be mapped to HIF IOs in 6 different programmable configuration levels providing more or less signals. Level 2 is for PHY mode (netX behaves as PHY and is connected to a host via MII). Levels 3 and higher are for MAC mode (netX is connected to a PHY).

Note: Configuration level 1 provides a reduced MII signal set which is not supported by netX 51/52 MAC.

Three different HIF-ETH pinout options are provided to share ETH with different other HIF functions (see table in section *Pinout functions combinations and exclusions* on page 65). Each HIF-ETH pinout option can be used in one of the 6 configuration levels. This must be programmed by register `io_config2.sel_eth_mii`.

The following table provides an overview of pin-functions of HIF Ethernet:

Pin function	direction	function	configuration level						notes
			(1)	2	3	4	5	6	
rxclk	input	receive clock			X	X	X	X	
rx[1:0]	input	receive data	(x)	X	X	X	X	X	
rx[3:2]	input	receive data		X	X	X	X	X	
rx[4:3]	input	receive data valid	(x)	X	X	X	X	X	
rx[5:4]	input	receive error	(x)			X	X	X	
txclk	input or output	transmit clock	(x)	X	X	X	X	X	1
tx[1:0]	output	transmit data	(x)	X	X	X	X	X	
tx[3:2]	output	transmit data		X	X	X	X	X	
tx[4:3]	output	transmit data enable	(x)	X	X	X	X	X	
tx[5:4]	output	transmit error						X	
col	input	collision					X	X	
crs	input	carrier sense					X	X	

Table 31: Ethernet functions of HIF (HIF-ETH)

Note

1. txclk is input in MAC mode and output in PHY mode. It is used as global clock for receive and transmit when only one clock is used (e.g. PHY mode).

IO functions for HIF-ETH. Functions in “()” brackets are only optional due to configuration level 3 or higher. Configuration level 1 is not respected as this function (reduced MII) is not supported by MAC internally.

HIF-IO	HIF-ETH1	HIF-ETH2	HIF-ETH3	Notes
hif_a0	-	-	(txer)	1
hif_a1	-	-	(col)	2
hif_a2	-	-	(crs)	2
hif_a3	-	-	rxd0	
hif_a4	-	-	rxd1	
hif_a5	-	-	rxd2	
hif_a6	-	-	rxd3	
hif_a7	-	-	rxdv	
hif_a8	-	-	txd0	
hif_a9	-	-	txd1	
hif_a10	-	-	txd2	
hif_a11	-	-	txd3	
hif_a12	-	-	txen	
hif_a13	-	-	txclk	
hif_a14	-	-	-	
hif_a15	-	-	-	
hif_ahi0	(col)	-	-	2
hif_ahi1	(crs)	-	-	2
hif_d0	-	-	-	
hif_d1	-	-	-	
hif_d2	-	-	-	
hif_d3	-	-	-	
hif_d4	-	-	-	
hif_d5	-	-	-	
hif_d6	-	-	-	
hif_d7	-	-	-	
hif_d8	-	(col)	-	2
hif_d9	-	(crs)	-	2
hif_d10	-	(txer)	-	1
hif_d11	-	-	-	
hif_d12	-	-	-	
hif_d13	-	rxd1	-	
hif_d14	-	rxd2	-	
hif_d15	-	rxd3	-	
hif_d16	(txer)	-	-	1
hif_d17	-	-	-	
hif_d18	rxdv	-	-	
hif_d19	-	-	-	
hif_d20	txclk		-	
hif_d21	txd0		-	
hif_d22	txd1		-	
hif_d23	txd2		-	
hif_d24	txd3		-	
hif_d25	txen		-	

HIF-IO	HIF-ETH1	HIF-ETH2	HIF-ETH3	Notes
hif_d26	rxcl0	-	-	
hif_d27	rxcl1	-	-	
hif_d28	(rxclk)		-	3
hif_d29	(rxer)		-	4
hif_d30	rxcl2	-	-	
hif_d31	rxcl3	-	-	
hif_bhe1	-	-	(rxer)	4
hif_bhe3	-	rxclv	-	
hif_csn	-	-	-	
hif_rdn	-	-	-	
hif_wrn	-	-	-	
hif_rdy	-	-	(rxclk)	3
hif_dirq	-	rxcl0	-	
hif_sdclk	-	-	-	

Table 32: IO functions for HIF-ETH

Notes

1. txer (transmit error) is an optional signal of configuration level 6.
2. col (collision) and crs (carrier sense) are optional signals of configuration level 5 and 6.
3. rxclk (receive clock) is an optional signal of configuration level 3 or higher.
4. rxer (receive error) is an optional signal of configuration level 4 or higher

3.10.7.6 PIO functions of HIF

Basically each HIF IO provides a PIO function, but it is available only if the related HIF IO is not programmed for any other function (e.g. HIF is configured as ExtBus8. This does not require hif_d8. Hence, hif_d8 can be used as PIO).

Each IO can be set to one of the following modes:

- driven mode (output mode)
 - drive high
 - drive low
- undriven mode (input mode)

Monitoring HIF IO states and input sampling can always be realized and do not depend on drive mode or non-PIO function. Sampling can be controlled by an enable input to realize sample-and-hold. There are two inputs providing sample enable function, but they cannot be used at the same time.

Available sampling enable inputs:

- hif_d16 (netX 50 naming: pio36), or
- hif_d4 (netX 50 naming: pio77)

Note: Sampling enable properties can be configured by software: sampling when enable is low, sampling when enable is high, optional hazard suppression (signal filtering) on enables.

HIF IOs are sampled once at netX 51/52 power-on release when HIF IOs are in a netX-undriven state. Pull-up or down resistor configuration will be sampled then. This can be used for system configuration ('strapping options').

Important: HIF IOs are never driven by netX during power-on reset release. However, netX cannot guarantee that an external device will not drive these IOs then. In this case resistor configuration will be overdriven. E.g. HIF-SDRAM data lines will remain in a driven state when netX 51/52 receives a reset during a HIF-SDRAM read access as the SDRAM clock will be stalled immediately. Or HIF is used as DPM and host is performing an access at the moment of netX 51/52 reset release.

There are 5 HIF IOs that can be used for IRQ generation:

- hif_dirq (netX 50 naming: pio47)
- hif_d16 (netX 50 naming: pio36)
- hif_d17 (netX 50 naming: pio35)
- hif_d18 (netX 50 naming: pio40)
- hif_d27 (netX 50 naming: pio72)

Each can be configured to level sensitive IRQs (high or low) or to edge sensitive IRQs (rising or falling) and are netX 50-compatible. Hazard suppression (input filtering) can be enabled for IRQ inputs. Each IRQ can be associated to ARM or xPIC (or both) individually.

Note: Hazard suppression can only be enabled globally for IRQ and sample-enable inputs. This cannot be configured for single inputs separately. Spikes up to 10 ns will be suppressed. However, this leads to an additional input latency of 10 ns

The following table provides an overview of HIF PIO functions and related netX 50 naming.

HIF-IO	netX 50 name	Alternative netX 50 name	Sample enable	IRQ	notes
hif_a0	PIO73	hif_reg_data8			
hif_a1	PIO70	hif_reg_data12			
hif_a2	PIO69	hif_reg_data11			
hif_a3	PIO66	hif_reg_data46			
hif_a4	PIO65	hif_reg_data43			
hif_a5	PIO64	hif_reg_data15			
hif_a6	PIO61	hif_reg_data49			
hif_a7	PIO60	hif_reg_data52			
hif_a8	PIO57	hif_reg_data44			
hif_a9	PIO56	hif_reg_data48			
hif_a10	PIO53	hif_reg_data18			
hif_a11	PIO50	hif_reg_data23			
hif_a12	PIO49	hif_reg_data22			
hif_a13	PIO48	hif_reg_data21			
hif_a14	PIO54	hif_reg_data19			
hif_a15	PIO55	hif_reg_data20			
hif_ahi0					1
hif_ahi1					1
hif_d0	PIO83	hif_reg_data24			
hif_d1	PIO82	hif_reg_data25			
hif_d2	PIO81	hif_reg_data47			
hif_d3	PIO78	hif_reg_data2			
hif_d4	PIO77	hif_reg_data1	X		
hif_d5	PIO76	hif_reg_data0			
hif_d6	PIO75	hif_reg_data6			
hif_d7	PIO74	hif_reg_data5			
hif_d8	PIO32	hif_reg_data4			
hif_d9	PIO34	hif_reg_data10			
hif_d10	PIO33	hif_reg_data9			
hif_d11	PIO39	hif_reg_data53			
hif_d12	PIO38	hif_reg_data38			
hif_d13	PIO37	hif_reg_data39			
hif_d14	PIO42	hif_reg_data34			
hif_d15	PIO41	hif_reg_data35			
hif_d16	PIO36	hif_reg_data31	X	X	
hif_d17	PIO35	hif_reg_data30		X	
hif_d18	PIO40	hif_reg_data29		X	
hif_d19		hif_reg_data36			2
hif_d20	PIO58	hif_reg_data37			
hif_d21	PIO59	hif_reg_data32			
hif_d22	PIO62	hif_reg_data33			
hif_d23	PIO63	hif_reg_data16			
hif_d24	PIO67	hif_reg_data17			
hif_d25	PIO68	hif_reg_data51			
hif_d26	PIO71	hif_reg_data50			

HIF-IO	netX 50 name	Alternative netX 50 name	Sample enable	IRQ	notes
hif_d27	PIO72	hif_reg_data41		X	
hif_d28	PIO79	hif_reg_data40			
hif_d29	PIO80	hif_reg_data13			
hif_d30	PIO84	hif_reg_data14			
hif_d31	PIO85	hif_reg_data7			
hif_bhe1	PIO43	hif_reg_data42			
hif_bhe3	PIO44	hif_reg_data3			
hif_csn	PIO51	hif_reg_data54			
hif_rdn	PIO52	hif_reg_data28			
hif_wrn	PIO45	hif_reg_data45			
hif_rdy	PIO46	hif_reg_data27			
hif_dirq	PIO47	hif_reg_data26		X	3
hif_sdclk	pci_clk				4

Table 33: PIO functions

Notes

1. Attention: netX 50 power pad location. Not for netX 50-drop-in-replacement.
2. Watchdog active output, no PIO function in netX 50. For netX 51/52 PIO function available.
3. IRQ on hif_dirq is suitable best for ExtBus usage and is available there in all ExtBus modes.
4. netX 51/52 provides no PCI function.

3.10.8 Multiplexing priority

Multiplexing priority becomes important if more than one function is activated on one HIF IO. Basically this should be avoided, but it could become possible e.g. if an i80 ExtBus interface is applied together with HIF-ETH3r (netJACK). In this case HIF-ETH3 signals have higher priority on HIF_A IOs than ExtBus address lines.

The following list provides the ascending priority of HIF IO functions. sDPM has the highest, PIO function the lowest priority:

1. sDPM
2. HIF-MMIO
3. HIF-ETH
4. pDPM
5. HIF-SDRAM
6. HIF-ExtBus
7. PIO

3.11 Dual-port memory interface

The DPM interface allows data transfer between the netX and an external host system. Unlike standard DPM, the netX 51/52 DPM is a virtual DPM which appears as a linear memory to the host side while access operations to the DPM are redirected to up to four different memory areas making all netX functions accessible.

The netX DPM interface can be a parallel DPM interface (e.g. an SRAM-like interface) or a high speed serial DPM interface. Any configuration can be performed by software. The netX must select the basic mode. Once the DPM is accessible from the host, the host can make the fine mode adjustment directly.

DPM interface is located on HIF-IOs for netX 51/52. These IOs are widely shared (e.g. could also be used as memory interface). For details on possible combinations of DPM with other HIF functions, see section *Pinout functions combinations and exclusions* on page 65.

3.11.1 Main features

The following list provides an overview of the Dual-port memory features:

- Parallel or serial DPM modes widely configurable by software.
- Parallel DPM
 - IO compatibility with netX 50 HIF-DPM.
 - 8, 16 or 32-bit data width.
 - Separated address and data busses or multiplexed A/D bus.
 - SRAM modes (two separated read and write control signals).
 - Motorola and Intel modes (byte strobes and one data direction signal).
 - Programmable active levels of control signals (as byte, address or chip enables).
 - 55 ns read access time for random INTRAM access.
 - 35 ns burst read access time to INTRAM using read ahead.
 - Configurable address setup time for read (0 ns to 30 ns in steps of 10 ns).
 - Configurable read data setup time (0 ns to 70 ns in steps of 10 ns).
 - Optional widely configurable ready/busy/acknowledge signal.
 - Optional input signal filtering (tolerating hazards < 10 ns).
 - Optional integrated address comparator.
 - Read burst support (address change without chip-select/read signal toggling).
- Serial DPM
 - Standard 1 bit SPI (CLK, CS#, MOSI, MISO).
 - 4-bit Quad-SPI/QSPI (CLK, CS#, SIO[3:0]).
 - All four clock modes (polarity high or low, phase 0 or 1) supported for SPI and QSPI.
 - Clock rates up to 125 MHz for SPI and 33 MHz for QSPI.
 - New serial frame format (SDP2, QDP2) optimized for CPU or DMA interaction
- Configurable external visible address ranges
 - 2 K (A[10:0])
 - 4 K (A[11:0])
 - 8 K (A[12:0])
 - 16 K (A[13:0])
 - 32 K (A[14:0])

- 64 K (A[15:0])
- 128 K (A[16:0])
- 256 K (A[17:0])
- 512 K (A[18:0])
- 1024 K (A[19:0])
- Configurable byte collecting/latching for 32-bit data consistency in serial, 8 or 16-bit modes.
- Hardware endianness modes
 - little endian (e.g. ARM)
 - 16-bit big endian (e.g. Motorola 16-bit CPUs)
 - 32-bit big endian (e.g. Motorola 32-bit CPUs)
- Configurable DPM configuration/status register area
 - On first 256 bytes of programmed external address range.
 - On last 256 bytes of programmed external address range (netX 50-compatible).
 - Configuration/status register area hidden.
- Four configurable DPM data windows (compatible with netX 500/100/50/10)
 - Programmable address mapping external to netX-internal 32-bit address.
 - Window size and address mapping can be adjusted in steps of 128 bytes.
 - Read ahead function configurable for each window
 - Write byte collecting configurable for each window
 - Each window mapping can be associated to one of the two netX 51/52 triple buffer managers of the HANDSHAKE_CTRL unit.
- One configurable DPM access tunnel: 64-byte window to overlay special small netX-internal address area for host (e.g. for handshake cells)
- Enhanced access error detection
- Interrupt generation for external host device
 - Two interrupt signals (normal, fast IRQ)
 - Interrupt active level programmable
 - Interrupt drive programmable (always-active, pull-up, pull-down or inverted active level)
 - 1 software interrupt e.g. for test purpose
 - All IRQs can be masked separately for ARM or xPIC
- All unused DPM signals can be used as programmable IOs (PIOs)
- Wide set of netX 50-compatible firmware-related configuration of status registers
- netX 51/52 version readable from DPM configuration register area

Note: Like all other digital I/Os of the netX 51/52, the DPM interface uses 3.3 V signaling voltage only. 5 V signals cannot be used!

3.11.2 DPM interface block diagram

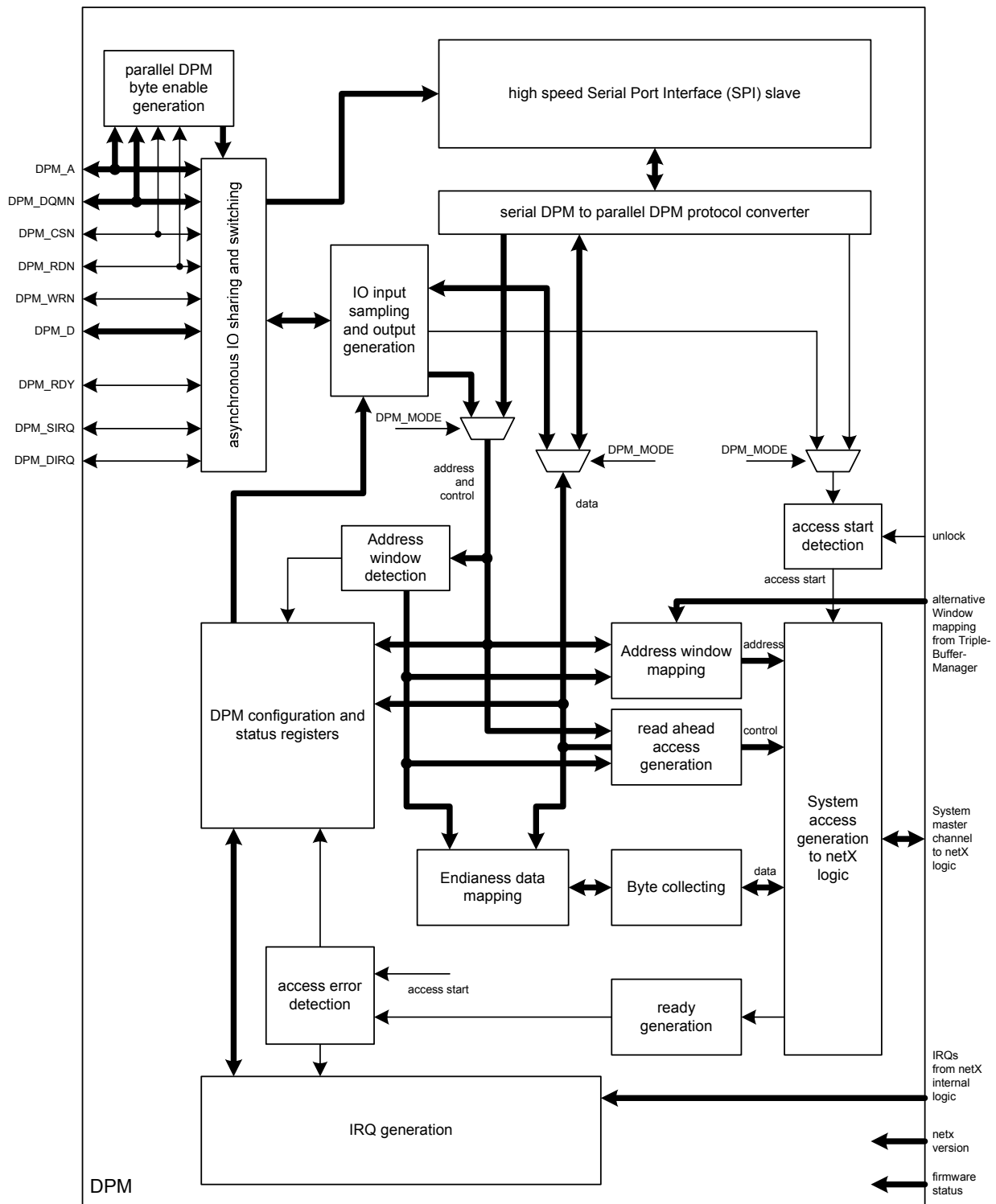


Figure 35: netX 51/52 DPM interface block diagram

3.11.3 DPM interface mode

The netX 51/52 DPM interface supports many different parallel DPM modes (section *Parallel DPM interface* on page 92) and a high speed serial DPM interface based on Motorola SPI (section *Serial DPM interface* on page 130). Selection between serial and parallel DPM must be programmed in the HIF_IO_CTRL area of netX 51/52.

The host CPU can set all further configurations:

- parallel mode data width
- data endianness
- address mapping
- signal timing
- interrupt handling
- PIO controlling

For all DPM modes, the configuration registers are located by default in the first 256 bytes (address 0x0 to 0xFF) of the external DPM. For netX 50 compatibility this configuration area can be relocated to the end of the external address range (last 256 bytes) or switched off completely.

Naturally DPM modes differ in IO-usage and data bandwidth. Unused DPM IOs can be used as programmable IOs (PIO).

For applications requiring additional PIOs, a smaller DPM interface can be selected. On the one hand a smaller interface typically provides lower performance than a large one, but on the other hand it offers more PIOs for external functions (e.g.: LEDs, switches, jumper). Moreover, data bandwidth is not only defined by data width: A fast serial connection could provide even higher performance than a slow parallel connection (especially when DPM ready signal is not used).

3.11.4 DPM interface structure

The internal structure of netX 51/52 is a synchronous design running on netX system clock (typically 100 MHz). Each external DPM access is converted to an internal netX system access using synchronous 100 MHz logic.

For parallel DPM modes, all input signals are synchronized by at least two stages of sample flip-flops to avoid propagating invalid signal states into the internal logic of the netX.

In serial DPM mode, serial clock input is directly used for serial to parallel input and parallel to serial output shift register stages. That allows serial clock rates even above netX-internal 100 MHz system clock, but requires high signal quality. All netX-internal data paths between serial clock and netX system clock domain are synchronized by two stage synchronization flip-flops.

Important: Signals of serial DPM, especially serial clock (DPM_SPI_CLK) and chip select (DPM_SPI_CSN) signals must be free of hazards as they are directly used for clock and reset signals.

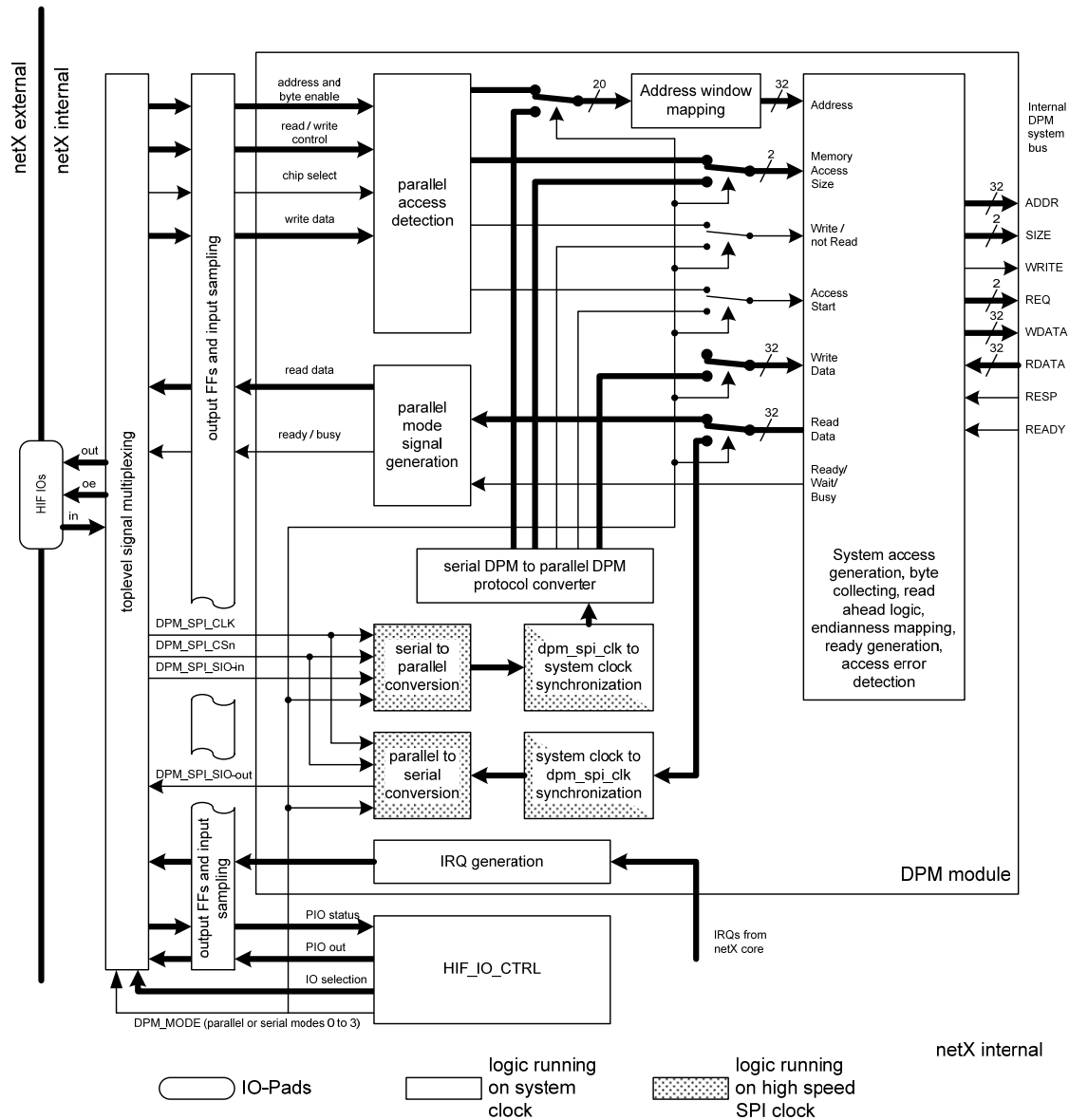


Figure 36: DPM interface structure diagram

Serial and parallel DPM I/Os are shared. Serial communication is performed by a special serial protocol containing address, data direction and data size information (section *Serial 1-bit DPM protocol* on page 133). A protocol converter transforms serial data frames to standard parallel accesses.

Selection between serial and parallel DPM and PIO-controlling of unused DPM signals must be programmed inside netX 51/52 HIF_IO_CTRL area.

netX transforms DPM addresses of the external access into the 32-bit address range prior to the netX-internal system access.

3.11.5 Parallel DPM interface

This section provides an overview of netX 51/52 parallel function and features. This includes a guide (section *Parallel DPM mode and setup guide* on page 95) to determine a required setting if a host CPU cannot be found in the list of *Supported parallel DPM modes* on page 94.

For detailed information on parallel DPM modes, connections, signal mapping, access timing depending on type of access, and internal netX 51/52 properties, see reference [2].

Important: There are several programmable timing parameters for parallel DPM (e.g. signal filtering, address setup time, read data setup time). By default these parameters are set in such a way that the function of the DPM is guaranteed even with unusual host timings (e.g. unstable address at access start). However, this leads to longer access times than typically required. For good performance it is absolutely necessary to change default timing to timings according to the application and host used. This is explained in section *Programmable parallel DPM timing parameters and signal filtering* on page 103.

Important: DPM provides an optional ready/busy/acknowledge signal. Not using this signal could lead to reduced performance especially when host is accessing netX address areas beyond internal memories (INTRAMs). Non-INTRAM-areas (e.g. peripheral registers or XC memories) require longer access times which must be observed to avoid data loss.

Note: In the following description, pDPM stands for parallel DPM, pDPM8 for parallel DPM with 8-bit data, pDPM16 for parallel DPM with 16-bit data, and so on.

3.11.5.1 Pin functions of parallel DPM

Pin functions of parallel DPM support all DPM modes known from netX 50 DPM. They are mapped to netX 51/52 HIF-IOs netX 50-compatible to allow drop-in-replacement. Which pin function comes out on which HIF IO, see section *Parallel DPM functions (pDPM)* on page 69.

Pin-function	Direction	Parallel DPM function	Related config. registers	Notes
dpm_a[19:0]	input	address lines	dpm_addr_cfg.addr_range dpm_cfg0x0.mode dpm_if_cfg.addr_sh dpm_if_cfg.be_sel	1, 2
dpm_d[31:0]	inout	data lines	dpm_cfg0x0.mode	
dpm_csn	input	chip-select	dpm_if_cfg.cs_ctrl	3
dpm_cs2n	input	second chip-select	dpm_if_cfg.cs_ctrl	3
dpm_rdn	input	read or data direction control	dpm_if_cfg.dir_ctrl	
dpm_wrn	input	write control (or low byte write strobe)	dpm_if_cfg.dir_ctrl dpm_if_cfg.be_sel	
dpm_be[3:0]	input	byte enables (or byte write strobes for pDPM32)	dpm_if_cfg.be_sel dpm_if_cfg.be_wr_dis dpm_if_cfg.be_rd_dis	4
dpm_be8	input	byte enable or strobe for pDPM8	dpm_if_cfg.be_sel dpm_if_cfg.be_wr_dis dpm_if_cfg.be_rd_dis	4

Pin-function	Direction	Parallel DPM function	Related config. registers	Notes
dpm_be16l	input	alternative netX 50-compatible low byte enable or strobe for pDPM16 (can be used for netx50 dpm_wrln)	dpm_if_cfg.be_sel dpm_if_cfg.be_wr_dis dpm_if_cfg.be_rd_dis	4
dpm_be16h	input	alternative netX 50-compatible high byte enable or strobe for pDPM16 (can be used for netX 50 dpm_wrhn)	dpm_if_cfg.be_sel dpm_if_cfg.be_wr_dis dpm_if_cfg.be_rd_dis	4
dpm_ale	input	address latch or enable for pDPM8 and pDPM16	dpm_cfg0x0.mode dpm_if_cfg.aen_pol	
dpm_ale32_nx50	input	address latch or enable for pDPM32 netX 50-compatible	dpm_cfg0x0.mode dpm_if_cfg.aen_sel dpm_if_cfg.aen_pol	
dpm_ale32	input	address latch or enable for pDPM32 not netX 50-compatible, +2 address line on HIF_AH10,1	dpm_cfg0x0.mode dpm_if_cfg.aen_sel dpm_if_cfg.aen_pol	
dpm_sela[19:11]	input	compare address inputs for internal address decoding (only pDPM8 or pDPM16)	dpm_addr_cfg.addr_range dpm_addr_cfg.addr_cmp_a*	5
dpm_rdy	output	ready/busy for host	dpm_rdy_cfg dpm_pio_cfg1.sel_rdy_pio	6
dpm_dirq	output	IRQ 1 for host (e.g. for data-IRQ)	dpm_io_cfg_misc.irqoec dpm_io_cfg_misc.irqpol dpm_pio_cfg1.sel_dirq_pio	6, 7
dpm_sirq	output	IRQ 2 for host (e.g. for service-IRQ)	dpm_io_cfg_misc.fiqoec dpm_io_cfg_misc.fiqpol dpm_pio_cfg1.sel_sirq_pio	6, 7
dpm_isamcs16n	output	ISA memcs16n	dpm_if_cfg. isa_bhe3_is_memcs16n	
wdgact	output	watchdog active for host	hif_io_cfg. en_hif_wdg_sys_hif_d19 netx_sys_wdg_ctrl	

Table 34: Pin functions of parallel DPM

Notes

1. Address lines 0 to 10 are mandatory for non-multiplexed DPM modes, higher lines are optional. For multiplexed modes (i.e. address and data are applied to dpm_d lines; selection is made by address latch signal) address lines 8 to 10 are mandatory for pDPM8 and unused for pDPM16 and pDPM32.
2. DPM address bus can be byte-oriented with byte enables on low address lines or word-oriented (i.e. byte address is right-shifted according to used data size) without byte enables.
3. DPM can be configured with 1 or 2 or even without chip select signals, but omitting chip select is not recommended.
4. Byte enables have many different functions. Various different signal mappings are possible, they can be used as byte strobes, active-polarity can be configured etc. They can even be ignored, but this is not recommended and should be used for netX 50-compatibility only. For more information, see description of related configuration bits.
5. The participation of each dpm_sela input can be controlled separately. Alternatively, an internal programmable address compare value can be used.
6. Signal is in PIO mode by default and must be set to functional mode first.
7. Attention: HIF IOs used for IRQs differ for serial and parallel DPM mapping, but the IRQ function is the same.

3.11.5.2 Supported parallel DPM modes

Modes are selected by configuring the function of control signals e.g. byte enables, read and write controlling or chip-selects. This leads to a wide range of supported DPM modes:

The following modes are supported by netX 51/52 parallel DPM without any external glue logic.

- Intel, 8-bit (SRAM-like)
- Intel, 16-bit, byte-write
- Intel, 16-bit, byte enable (SRAM-like)
- Intel, 32-bit, byte-write
- Intel, 32-bit, byte enable (SRAM-like)
- Intel, 8-bit multiplexed address-data-bus
- Intel, 16-bit multiplexed address-data-bus netX 50-compatible: no byte enables
- Intel, 16-bit multiplexed address-data-bus byte-write
- Intel, 16-bit multiplexed address-data-bus 2 byte enables, byte address
- Intel, 16-bit multiplexed address-data-bus 2 byte enables, word address
- Intel, 32-bit multiplexed address-data-bus netX 50-compatible: byte address
- Intel, 32-bit multiplexed address-data-bus byte-write Dword address
- Intel, 32-bit multiplexed address-data-bus 4 byte enables, byte address
- Intel, 32-bit multiplexed address-data-bus 4 byte enables, Dword address
- TI OMAP, 16-bit (non-multiplexed)
- TI OMAP, 16-bit multiplexed address-data-bus
- Motorola, 8-bit (6800)
- Motorola, 16-bit
- Motorola, 16-bit (68000)
- Motorola, 32-bit
- Motorola, 8-bit multiplexed address-data-bus
- Motorola, 16-bit multiplexed address-data-bus netX 50-compatible: byte address
- Motorola, 16-bit multiplexed address-data-bus word address
- Motorola, 32-bit multiplexed address-data-bus netX 50-compatible: byte address
- Motorola, 32-bit multiplexed address-data-bus Dword address
- ISA, 8-bit (3V3 signal level)
- ISA, 16-bit (3V3 signal level)

3.11.5.3 Parallel DPM mode and setup guide

The following steps must be performed to configure the netX 51/52 DPM according to the host requirements. The description of register bits *dpm_if_cfg* provides an overview of the settings for the modes listed above. If a desired mode is not in the list, step through the following guide to determine whether the netX 51/52 DPM is applicable at all.

1. Determine the required data width (i.e. 8, 16 or 32-bit data) and the mode to be used, multiplexed or non-multiplexed (register bits *dpm_cfg_0x0.mode*).
2. Determine the required data direction protocol of the interface used (register bits *dpm_if_cfg.dir_ctrl*). There are 3 possibilities:
 - A. SRAM-like interface with 2 separated low-active lines: Read enable or output enable (RDn, OEn) and write enable (WRn, WEn).
 - B. Intel-like interface with 1 data direction line. High level flags a write access; low level flags a read access.
 - C. Motorola-like interface with 1 data direction line. High level flags a read access; low level flags a write access.
3. Determine the use of chip-select signals and the use of internal address-decoding (address comparator). 1 or 2 chip-select signals can be used. For details, see description of the register bits *dpm_if_cfg.cs_ctrl*.
4. Determine the active-level of byte enable, byte strobe or byte-write signals. The enable for each data-byte can be controlled separately by the register bits *dpm_if_cfg.be_pol*. If no byte enables are required (e.g. for 8-bit data interfaces), this step can be ignored.
5. Select the IO allocation of byte enable or byte strobe signals. For details, see the description of register bits *dpm_if_cfg.be_sel*. If no byte enables are required (e.g. for 8-bit data interfaces), this step can be ignored.
6. Determine the use of byte enable or byte strobe during read access; they can be ignored (e.g. for netX 50-compatibility) by register bit *dpm_if_cfg.be_rd_dis*. When ignored, a read access is always performed according to the full data bus width and all data bus lines are always driven.
7. Determine the use of byte enable or byte strobe during write access; they can be ignored (e.g. for netX 50-compatibility) by the register bit *dpm_if_cfg.be_wr_dis*. When ignored, a write access is always performed according to the full data-bus width. It is not possible to write single bytes when using a 16 or 32-bit data bus then.
8. Determine whether byte addresses or word addresses are used, i.e. whether an address which is right-shifted according to the used data-width is used or not. For details, see description of register bit *dpm_if_cfg.addr_sh*.
9. If an address-enable or address-latch signal is used (i.e. always for multiplexed modes), select this signal by register bits *dpm_if_cfg.aen_sel* and configure the active level by register bit *dpm_if_cfg.aen_pol*.
10. For ISA mode, signal MEMCS16n can be activated by register bit *dpm_if_cfg.isa_bhe3_is_memcs16n*.

3.11.5.4 High address lines in multiplexed modes

For multiplexed modes, the address is generated by the host on the multiplexed Address/Data bus (A/D bus). However, especially for 8-bit data, the A/D bus width is not sufficient for addressing (smallest programmable DPM address range is 2 kB, i.e. 11 address lines are required). Hence, to generate valid addresses, additional high address lines are required.

Important: For high address lines always DPM_A inputs are used. The first high address line is the one above the highest A/D bus line.

Address bits A0 to A11 are mandatory because they are required for the smallest programmable address range (i.e. 2 kB). Address lines A12 to A19 are optional, but they must be used if a larger address range is desired.

The following table describes the address generation in multiplexed modes.

A/D bus	optional high address	mandatory high address	low address on A/D bus
8-bit	DPM_A[19:11]	DPM_A[10:8]	DPM_D[7:0]
16-bit	DPM_A[19:16]		DPM_D[15:0]
32-bit	-		DPM_D[19:0]

Note: The table above is related to non-shifted byte addresses. Addresses can be shifted for word addressing, see section *Shifted addresses – Word addressing* on page 96.

Important: Both, A/D bus addresses and high addresses on DPM_A lines are latched at the same moment: They must be stable and valid around the change for address to data phase of the DPM_ALE signal. Chip-select need not to be active then.

3.11.5.5 Shifted addresses – Word addressing

Address shifting could become important especially in multiplexed modes: E.g. many hosts are in a non-multiplexed mode (e.g. 2 k address range with 16-bit data) by default. To reach a larger address range, they can switch to multiplexed mode. Some hosts use byte addresses while others use word-oriented addresses. Word-oriented means that the address applied is related to data words according to the programmed data width and not a standard byte address. I.e. the access address (byte address) is right-shifted according to the used data width by host before it is applied to the A/D bus. The word address is left-shifted according to the data width inside netX DPM to receive the originally byte address.

To allow mode switching including change from unshifted to shifted addresses without additional external glue logic, netX DPM can perform an inverse address shifting.

Note: For non-multiplexed modes, word addressing is possible, but unlikely. Low address lines are typically used as byte enables there. They are not available for word addressing.

Note: Address shifting is irrelevant for 8-bit data width since the word width is one byte.

Note: The largest DPM address range is 1 MByte, i.e. 20 address bits. Using shifted addresses will not extend this range even if more address lines are physically available then. All address bits above A[19] are ignored.

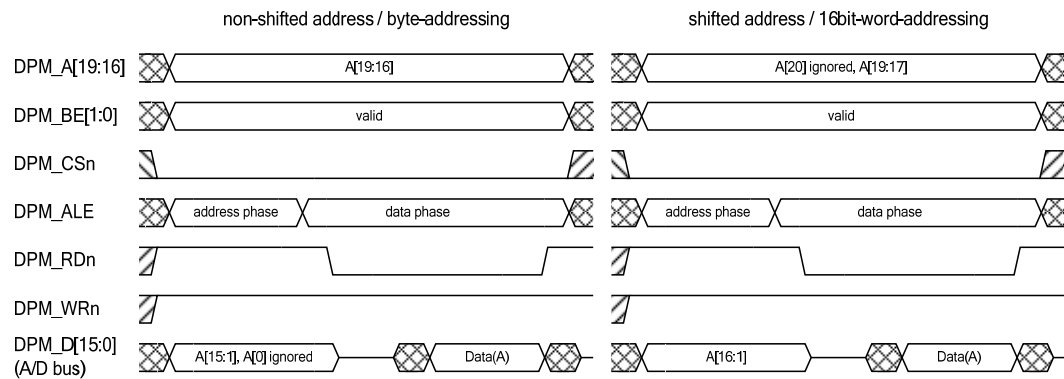


Figure 37: Word and byte addressing for 16-bit data width (read)

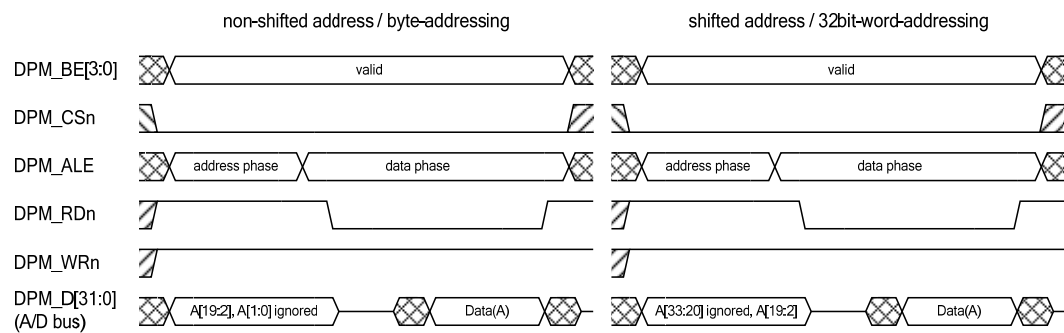


Figure 38: Word and byte addressing for 32-bit data width (read)

Note: For multiplexed modes with 32-bit data width all address bits are stored on the A/D bus (DPM_D lines). DPM_A lines are not used for addressing then.

Note: Address shifting can be applied similar to multiplexed SRAM modes (Figure 37 and Figure 38) for multiplexed Intel and Motorola-like interfaces.

3.11.5.6 Parallel DPM ready generation

DPM access time could vary e.g. depending on internal state of the DPM or netX target address. To avoid errors, the worst case access time must always be observed by the host devices. However, running all DPM access with worst case timing could lead to a significantly lower performance than running all access with an optimal access length. To avoid wasting performance there is a widely configurable ready signal. During an access it is set to not-ready state when netX is not able to receive or deliver new data. The host must insert wait cycles then and stretch the current access until netX releases it to ready state. When this signal is set to ready-state again, host device can finish the access.

The read data setup time t_{RDS} is the time before data becomes valid. This timing parameter can be programmed by DPM timing configuration register (*dpm_timing_cfg*), see section *Read data setup time - t_{RDS}* on page 103.

Important: The read data setup time is set by default to 70 ns, designed for very slow hosts. This results in a bad performance for faster hosts because with each read access time is wasted. To improve performance, set the t_{RDS} parameter according to the host used.

The ready signal (DPM_RDY) behavior of the netX DPM interface is widely programmable:

- programmable signal polarity:
 - ready/acknowledge/not-wait high, wait/busy low
 - ready/acknowledge/not-wait low, wait/busy high
- 2 programmable signal modes:
 - wait/busy mode
 - ready/acknowledge mode
- 4 programmable driving modes
 - High-impedance output (mode 0)
 - Push-pull output (mode 1)
 - Sustained-tristate output (mode 2)
 - Open-drain/open-source output (depending on the configured polarity, mode 3)

Note: netX 100, netX 500 and netX 50 all have a different ready driving mode coding. Modes 2 and 3 are reordered for better timings.

In wait/busy mode, the DPM_RDY signal is set to active state (i.e. netX is busy) immediately after the start of access. The host must wait and extend the current access while the DPM_RDY remains active. After release to inactive or high impedance state the access can be finished by host device. The signal then remains inactive until the next access.

In ready/acknowledge mode DPM_RDY does not changed at access start. The active-state rather generates an acknowledge pulse at access end. A host must wait and extend the current access until the signal becomes active.

Note: The DPM_RDY signal is always set to busy state for a read access. In contrast single write accesses are usually accepted without setting DPM_RDY to busy state.

Note: In multiplexed DPM modes, DPM_RDY is never set to wait/busy state during address phase, only during data phase.

Note: DPM_RDY signal is configured as PIO after netX reset by default. Before usage as ready signal this must be changed in DPM PIO configuration register (*dpm_pio_cfg1*).

Note: The ready/acknowledge mode is not only an inverted version of the wait/busy mode. The figures further down show this.

In High impedance mode (default after netX reset) ready generation is disabled and DPM_RDY signal is never driven by netX. In Push-pull mode, the signal is always driven. In Open drain/open source mode, the signal will be driven only when active, i.e. it is necessary to use external pull-up or pull-down resistors to maintain a valid inactive signal level. The sustained tristate output mode works similar to the Open drain/open source output mode. However, the signal will be driven when entering the inactive state for one cycle before entering the high impedance state. This provides a signal edge which is faster than it could be achieved by a pull-down or pull-up resistor while the signal may still be shared with other components (Figure 39).

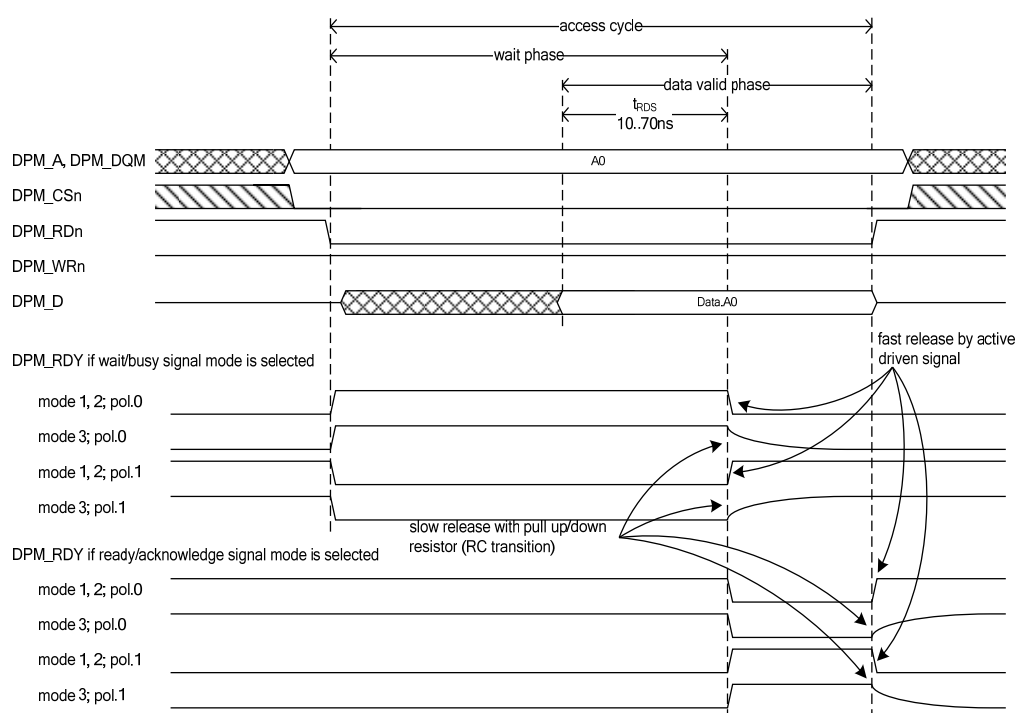


Figure 39: Ready signal transitions (non-multiplexed SRAM mode read example)

The following figures give an overview of ready signal behavior and different access types supported by the netX DPM interface. In wait/busy mode the drive active level is the inverted programmed ready signal polarity (as the active phase is the wait-phase).

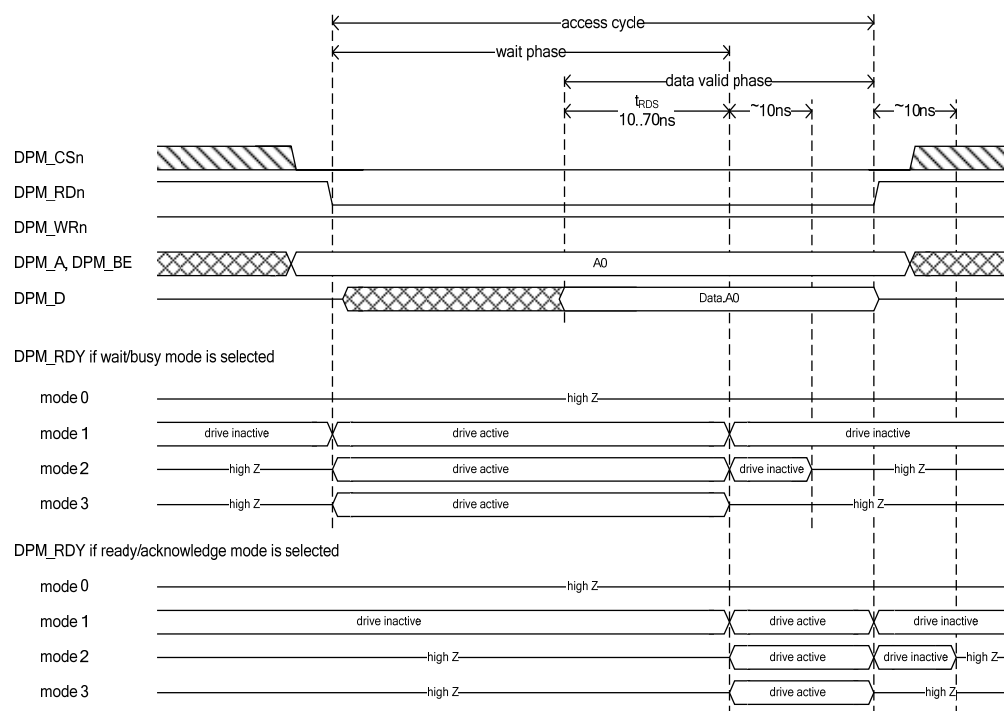


Figure 40: Non-multiplexed mode read access and ready generation (SRAM mode example)

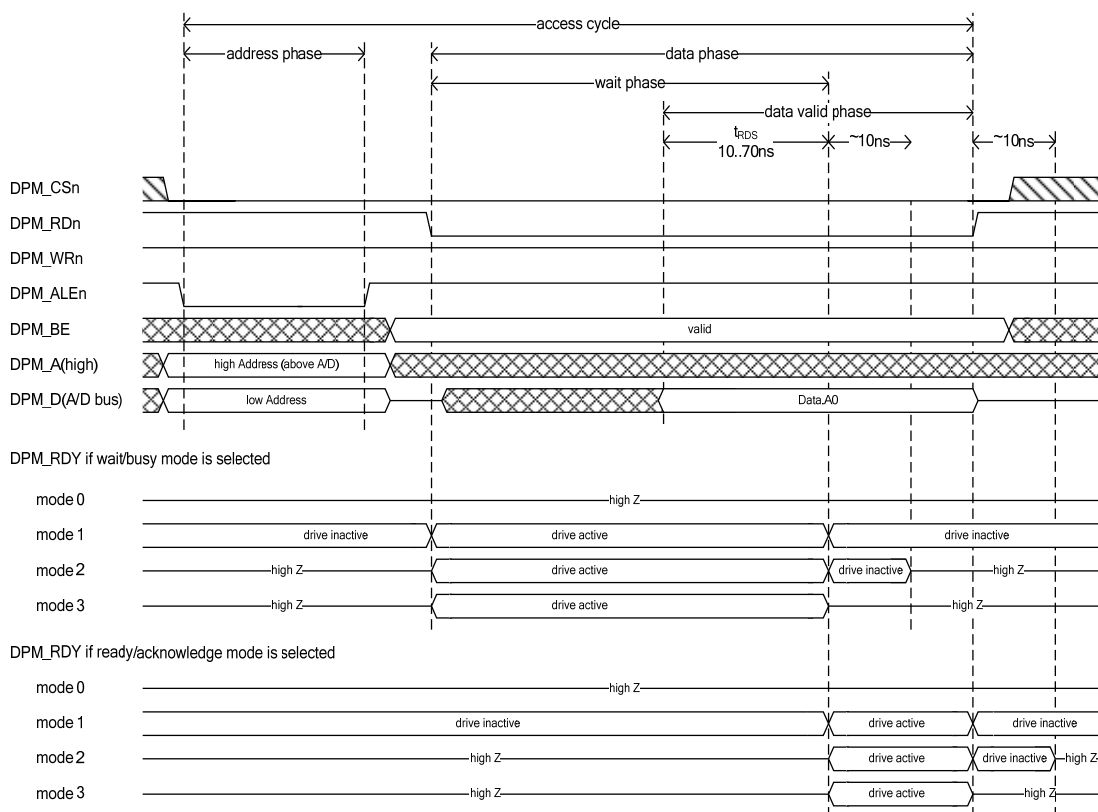


Figure 41: Multiplexed mode read access and ready generation (SRAM mode example)

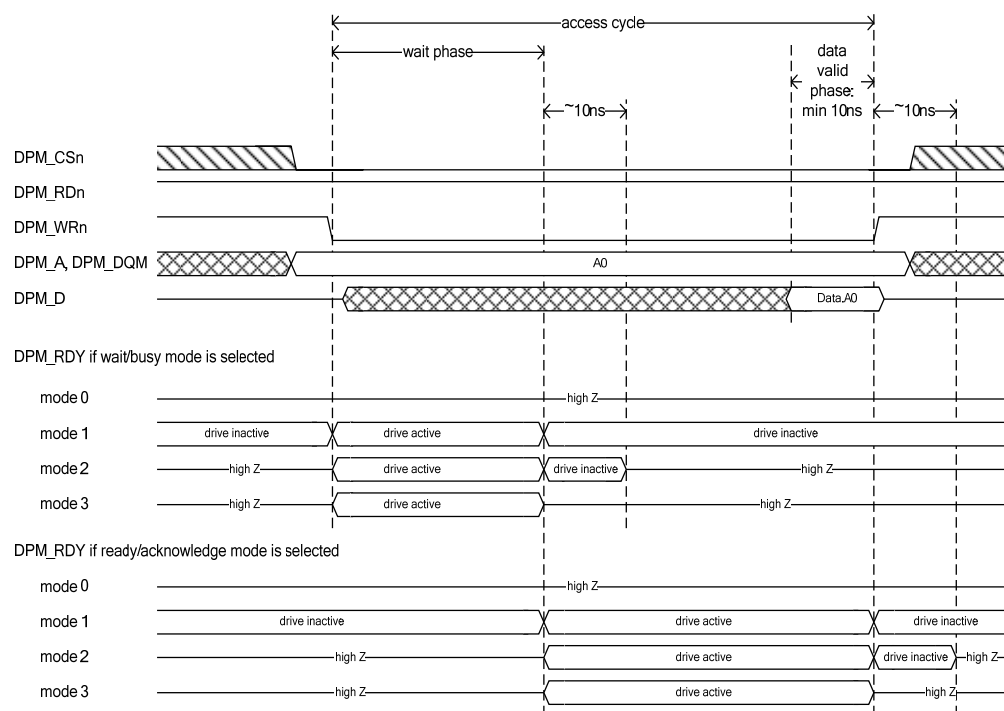


Figure 42: Non-multiplexed mode write access ready generation (SRAM mode example)

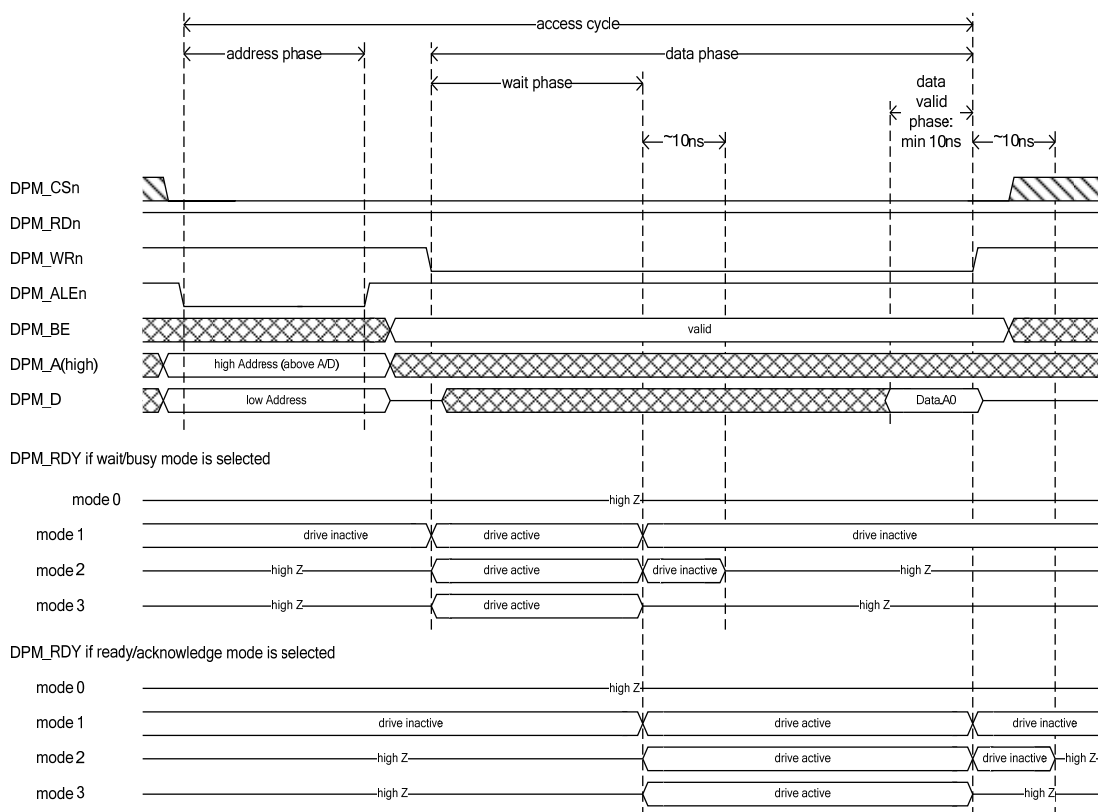


Figure 43: Multiplexed mode write access ready generation (SRAM mode example)

Note: Single write accesses are usually accepted without setting DPM_RDY to busy state. In wait/busy mode the signal does not become active then. In ready/acknowledge mode the signal becomes active state immediately at access start then.

Under abnormal conditions netX could be internally busy for an unpredictable time. To avoid stalling of the host, a busy-timeout prevents the access from being blocked.

In case of a timeout, the current access will be aborted by setting the ready signal to ready state. The host is now able to terminate the access and prevents system lock up conditions. The data of the aborted access is invalid. A flag in the netX DPM status registers indicates an access termination. This indication requires an action of the host. Additionally, an interrupt can be generated.

Note: Long access times and timeouts can generally be avoided by a DPM window setup that allows access to zero wait state address areas only.

Note: The timeout can be configured to 256 netX clock cycles (i.e. 2.56 us, netX 50-compatible), to 2048 clock cycles (i.e. 20.48 us) or completely disabled (which is not recommended).

3.11.5.7 Programmable parallel DPM timing parameters and signal filtering

The timing parameters read data setup time (t_{RDS}) and address setup time (t_{OSA}) can be set in register *dpm_timing_cfg*. Signal filtering can be enabled or disabled.

Important: Changing timing configuration will have impact on parallel DPM access timing. After power on reset, the timing parameter values are set by default to work with very slow hosts. For good performance these values have to be set according to the host access generation (see host data sheet) and signal quality which depends on PCB wiring and operating environment.

The following figures show the configurable timing parameters:

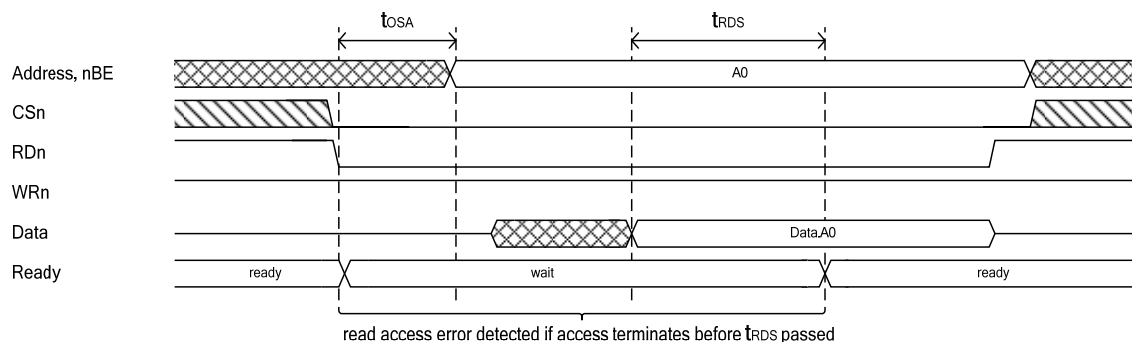


Figure 44: Read data setup time and read address setup time (SRAM read)

3.11.5.7.1 Read data setup time - t_{RDS}

The read data setup time parameter can control the read access timing. In ready signal mode, the read data setup time parameter specifies the time that elapses before the read data become valid on the external data bus. As soon as they have become valid, the ready signal changes from wait to ready state. If the host device terminates an access before the read data setup time has passed, the netX DPM interface will detect a read access error.

Read data setup time can be set in steps of 10 ns (1 system clock cycle) between 0 ns and 70 ns. Default after power-on reset is 20 ns.

Important: The set read data setup time will extend the total read cycle time.

Note: The read data setup timing parameter has no impact on any write access.

Note: Read access error detection does not depend on the use of the ready signal, see section *Initializing the DPM interface* on page 126).

3.11.5.7.2 Address setup time - t_{OSA}

In SRAM modes address lines and byte enables (data qualifier signals, nBE) are typically stable at access start (chip-select and read or write enable have both become low). However, some host devices simultaneously generate control signals and the address. In this case address lines on external bus can become valid later than the control signals (e.g. due to the higher capacitive load by PCB wiring).

Invalid (or unstable) address or byte enables (data qualifier signals) at start of a read access will lead to a false internal access address when internal access is started immediately after read and chip select signals are detected active. To avoid that, read address setup time can be configured. Internal access will not be started until address setup time has passed.

Address setup time can be programmed in steps of 10 ns (1 system clock cycle) between 0 ns and 30 ns. Default after power-on reset is 30 ns.

Important: Programmed address setup time will extend total read and write cycle time.

Note: In Motorola and Intel modes (i.e. modes using 1 data direction signal instead of 2 separated SRAM-like read and write enable signals), the byte enables must not become active while the direction signal is not stable. To prevent the byte enable from becoming active although the direction signal is still unstable, increase the address setup time. This will force the DPM to wait for a stable direction signal.

3.11.5.7.3 Input signal filtering

Signal filtering is implemented in netX DPM to suppress short term hazards (e.g. caused by crosstalk on PCB). When enabled, hazards shorter than 10 ns (1 netX system clock cycle) are filtered off. When filtering is disabled, all signals must be clear of hazards or spikes.

Signal filtering can be enabled (default after power-on reset) or disabled.

Important: Enabling signal filtering will cause 1 additional system clock cycle for each signal sampling. One system clock period must be added to all access cycle times (including min. write cycle time, min. idle time between access operations and min. address or data phase in multiplexed mode) when filtering is enabled.

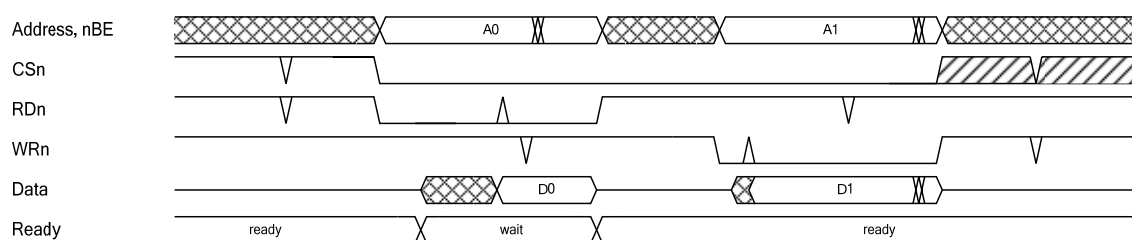


Figure 45: Hazards and spikes which will be suppressed when signal filtering is enabled

Note: Signal filtering cannot always eliminate all hazards, especially if overall signal quality is bad.

3.11.6 Access timing without using the ready signal

If the host does not use the DPM ready signal, the host must not access the netX address areas with internal wait states.

Note: Accessing netX wait state areas without using the ready signal is possible, but access timing calculation becomes very complex and could result in unacceptable timings.

To avoid internal wait states, access must be limited to the following netX target areas:

- INTRAMs
- Handshake cells
- DPM configuration and status window 0

Important: The netX software has to configure the handshake cells located in the HANDSHAKE_CTRL address area which is a wait state area. IRQs – also for handshake cells - can be handled by dpm_firmware registers. These registers are located in the DPM configuration and status window 0 (zero wait state).

Note: To always avoid access to the wait state areas, the DPM window can be set up in such a way that it provides access to zero wait state areas only.

Access timing always depends on the programmed setting of t_{OSA} and on whether input filtering is enabled (3.11.5.7). The access timing can be calculated by:

$$t_{AC} = 4.5 \text{ system clock cycles} + 10 \text{ ns} + t_{OSA} + t_{filter}$$

That leads to the following table:

DPM access time for 0 wait state areas INTRAM, handshake cells, DPM cfg Win0	Register setting of <i>dpm_timing_cfg</i>	
	filter	t_osa
55 ns (4.5 system clock cycles + approx. 10 ns)	0	0
65 ns (5.5 system clock cycles + approx. 10 ns)	0	1
75 ns (6.5 system clock cycles + approx. 10 ns)	0	2
85 ns (7.5 system clock cycles + approx. 10 ns)	0	3
65 ns (5.5 system clock cycles + approx. 10 ns)	1	0
75 ns (6.5 system clock cycles + approx. 10 ns)	1	1
85 ns (7.5 system clock cycles + approx. 10 ns)	1	2
95 ns (8.5 system clock cycles + approx. 10 ns)	1	3

Note: The values are approximate values. Real timings are a little faster (about 2 ns). For exact values, see timing characteristics in section *Parallel dual-port memory timing* on page 237.

3.11.7 DPM interrupt signals

The netX DPM interface provides two IRQ signals:

- DPM_SIRQ / DPM_FIQ: Service IRQ, fast IRQ
- DPM_DIRQ / DPM_IRQ: Data IRQ, normal IRQ

Note: Signal naming (“fast”, “normal”) does not imply any priority. Both signals are generated in the same way. Naming the signals “fast” and “normal” IRQ is adapted from the ARM naming convention.

The polarity and drivability of each IRQ signal can be configured. Figure 46 shows configurable IRQ driving modes.

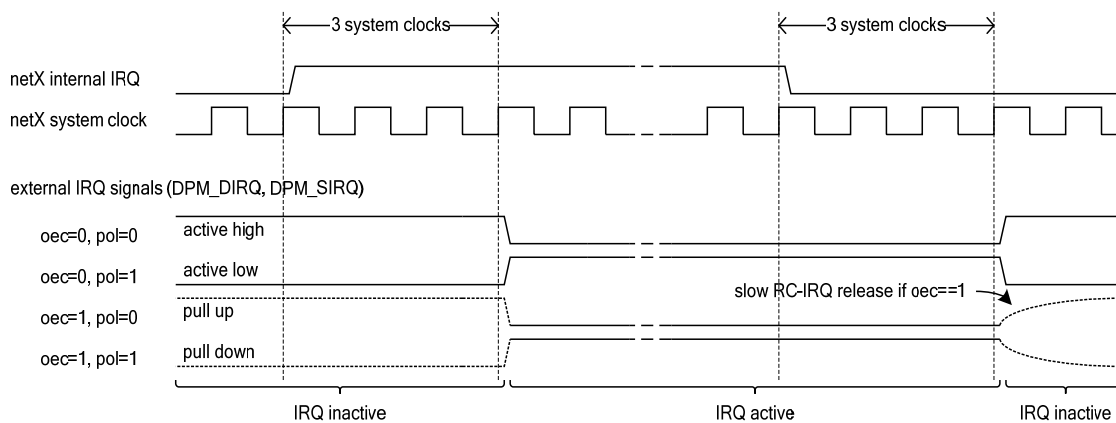


Figure 46: IRQ signal driving modes

Output enable controlled IRQ generation allows combining netX IRQ signals with other IRQ generating devices (wired OR). As IRQ inactive states are realized by pull up or down resistors, IRQ release will be rather slow (depending on RC characteristics).

Any IRQ event takes three system clock cycles (i.e. 30 ns at 100 MHz) being propagated from DPM internal interface part to external IRQ signals. IRQ signals are always level-oriented. The DPM interface simply propagates netX-internal IRQ levels to external IRQ signals after masking them. For clearing the IRQ signal, the requesting IRQ source has to be cleared inside the IRQ generating module or disabled by clearing its mask inside DPM interface.

Important: IRQ signals are mapped to different netX 51/52 IO depending on whether serial or parallel DPM mode is selected.

IRQ signal	netX 51/52 IO for parallel DPM	netX 51/52 IO for serial DPM
DPM_SIRQ / DPM_FIQ	HIF_SDCLK (i.e. HIF_SIRQ)	HIF_D13 (i.e. DPM_SPI_SIRQ)
DPM_DIRQ / DPM_IRQ	HIF_DIRQ	HIF_D12 (i.e. DPM_SPI_DIRQ)

The two independent IRQ signals provide IRQ handling on two different priority levels on the host CPU. E.g. system status events like DPM errors or watchdog timeouts can be mapped on one IRQ signal while functional IRQs for data flow control can be mapped on the other IRQ signal.

Apart from these two DPM host IRQs, the DPM modules provides two further netX-internal DPM-IRQs: One is dedicated to the netX ARM CPU, the other one to the netX XPIC CPU. In total there are 4 DPM-IRQs:

1. DPM-host-SIRQ
2. DPM-host-DIRQ
3. DPM-ARM-IRQ
4. DPM-XPIC-IRQ)

These IRQs could be used for netX-internal DPM error handling; for debug or for early software development (netX-internal ARM could replace the external host e.g. because a complete host-netX system is not available yet).

Internal DPM-IRQs for ARM and XPIC are implemented exactly in the same way as host-DPM-IRQs.

Note: Drivability and polarity programming is available for external IRQs only.

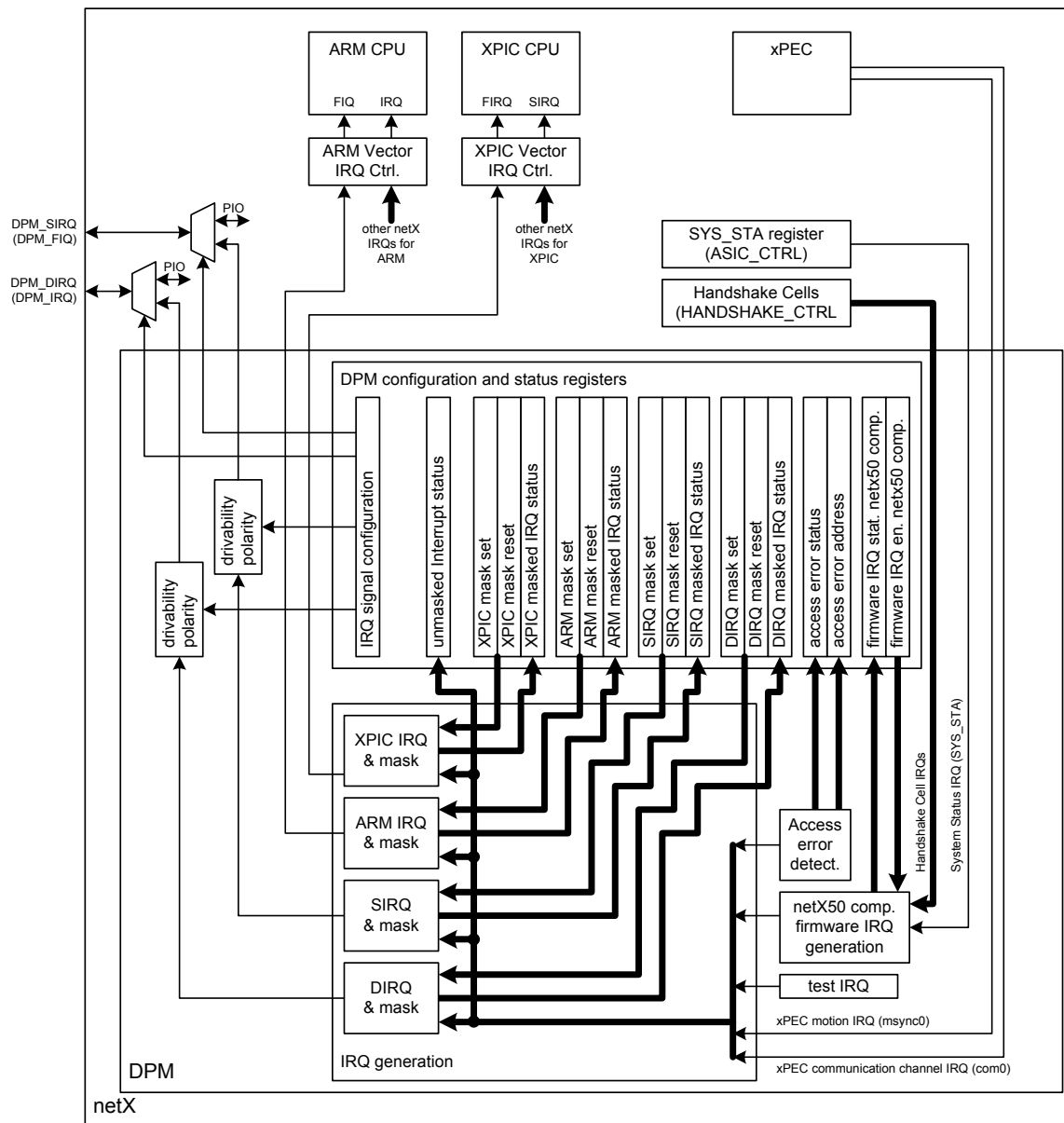


Figure 47: DPM IRQ generation

There are 4 functional IRQ sources inside netX and an IRQ for test purposes.

No.	IRQ	Description
31	-	reserved
30	osac	OSAC interrupt
29	can	CAN interrupt
28	trigger_lt	Trigger-Latch interrupt
27	dmac	DMA-controller interrupt
26	sysstate	combined licence error and MI timeout interrupt
25	int_phy	combined internal PHY 0 / PHY 1 interrupt
24	-	reserved
23	-	reserved
22	msync1	motion synchronization channel 1 (= xpec1_irq[15:12]) interrupt
21	msync0	motion synchronization channel 0 (= xpec0_irq[15:12]) interrupt
20	xpic_debug	xPIC Debug interrupt
19	-	reserved
18	com1	communication channel 1 (= xpec1_irq[11:0]) interrupt
17	com0	communication channel 0 (= xpec0_irq[11:0]) interrupt
16	gpio	combined GPIO 0-30 / IOLINK interrupt
15	firmware	combined handshake cell and SYS_STA firmware interrupt
14	-	reserved
13	i2c	combined I2C 0 and I2C 1 interrupt
12	spi	combined SPI 0 / QSPI, SPI 1 interrupt
11	usb	USB interrupt
10	uart2	UART 2 interrupt
9	uart1	UART 1 interrupt
8	uart0	UART 0 interrupt
7	watchdog	combined Watchdog from WDG_SYS and XPIC_WDG module interrupt
6	gpio31	external GPIO 31 interrupt
5	sys_time_s	ARM_TIMER sys_time_s/sys_time_uc_s interrupt
4	sys_time_ns	ARM_TIMER sys_time_ns/sys_time_uc_ns interrupt
3	gpio_timer	GPIO Timer 0..4 interrupt
2	-	reserved
1	dpm_err	DPM access error interrupt
0	dpm_sw	software IRQ for netX IRQ targets (e.g. ARM, xPIC) interrupt

Table 35: IRQ table, netX 51/52

Each IRQ can be AND-masked.

Each mask can be programmed individually by one mask-set configuration register

- dpm_irq_arm_mask_set
- dpm_irq_xpic_mask_set
- dpm_irq_fiq_mask_set
- dpm_irq_irq_mask_set

and one mask-reset configuration register

- dpm_irq_arm_mask_reset
- dpm_irq_xpic_mask_reset
- dpm_irq_fiq_mask_reset
- dpm_irq_irq_mask_reset

Note: Read modify write sequences which could be interrupted are not needed to change IRQ masks. To disable an IRQ from mask, simply write the respective *mask_reset* register. To enable it, simply write the respective *mask_set* register.

IRQ states from netX-internal logic before masking can be read from register *dpm_irq_raw*. Masked IRQ states of each DPM-IRQ can be read from the status registers *dpm_irq_arm_masked*, *dpm_irq_xpic_masked*, *dpm_irq_fiq_masked* and *dpm_irq_irq_masked*.

If an IRQ is generated inside netX and its related mask bit is set, the respective DPM IRQ signal will be set. Resetting the mask will immediately change the DPM-IRQ signal from active to inactive state (3 system clocks latency). The internal IRQ, however, remains active and can be detected by reading register *dpm_irq_raw*. Clearing the IRQ internally, requires IRQ handling according to the requesting netX-internal logic.

The DPM firmware IRQ

The firmware IRQ is generated netX 50-compatibly inside the DPM module. Firmware IRQs have multiple sources, e.g. handshake cell status and system status register (SYS_STA) changes.

Each IRQ source has a flag (and a special IRQ vector) in register *dpm_firmware_irq_raw* (*DPM_HOST_INT_STAT0*). They can be enabled in register *dpm_firmware_irq_mask* (*DPM_HOST_INT_EN0*) (3.11.9.5).

To use the DPM IRQ signals netX 50-compatibly, configure them as follows:

1. Disable all IRQs for DPM by writing 0xffffffff to register *dpm_irq_irq_mask_reset* and *dpm_irq_fiq_mask_reset*.
2. Configure the DPM IRQ IOs for IRQ usage instead of default PIO function by register *dpm_pio_cfg1*.
3. Configure drivability and active level as required by register *dpm_io_cfg_misc*.
4. Set up the firmware IRQ by the netX 50-compatibility registers *dpm_firmware_irq_mask*, *mask2* (*DPM_HOST_INT_EN0*, *EN2* as known from netX 50).
5. Enable the firmware IRQ mask by writing bit *firmware* in register *dpm_irq_irq_mask_set* or *dpm_irq_fiq_mask_set*.

DPM software IRQ and testing of IRQ handling

Since netX 51/52 a software-IRQ is provided. Programming can be performed by register new *dpm_sw_irq* in DPM configuration window 0 (or DPM-INTLOGIC address area). To avoid read-modify-write accesses, a set function and a reset function, are provided separately. They can be set and cleared separately for each DPM-IRQ-target (*DPM_DIRQ*, *DPM_FIRQ*, *ARM*, *XPIC*).

Note: The test IRQ known from netX 10 no longer exists. This function can be replaced by using the software-IRQ.

3.11.8 Data mapping

For high performance data handling netX DPM interface provides hardware endianness data mapping. For access to 32-bit, netX data by an external data bus with less than 32-bit data width byte collecting feature is implemented.

3.11.8.1 Endianness modes and DPM data width setting

Data endianness means orientation of single byte inside a 16 or 32-bit data word. Most common endianness modes are:

Little endian:

The most significant byte is stored on the highest address inside the word, the least significant byte on the lowest address inside the word.

Example for a 32-bit word at address A:

32-bit word at A	Byte at A+0	Byte at A+1	Byte at A+2	Byte at A+3
0x12345678	0x78	0x56	0x34	0x12

Big endian:

The most significant byte is stored on the lowest address inside the word, the least significant byte on the highest address inside the word.

Example for a 32-bit word at address A:

32-bit word at A	Byte at A+0	Byte at A+1	Byte at A+2	Byte at A+3
0x12345678	0x12	0x34	0x56	0x78

Note: 8-bit CPUs using bytes and byte addresses only do not require endianness mapping.

netX data is 32-bit organized with little endian format. Data exchange with host devices with different data orientation (e. g. big endian oriented Intel CPUs) requires data mapping. netX DPM hardware supports endianness data mapping. Data width setting and endianness configuration is carried out by access to register *dpm_cfg0x0*.

The netX DPM interface provides three data mapping modes:

- netX connection to all 8-bit host devices or little endian 16 or 32-bit host device
- netX connection to 16-bit big endian host device
- netX connection to 32-bit big endian host device

Note: There is no relation between endianness width and DPM interface data width. E. g. it is possible to connect a 32-bit big endian host to netX by an 8-bit DPM interconnection.

The following table explains endianness data mapping and byte orientation. The byte number equals byte significance. Byte significance inside a 32-bit word is byte number module 4:

Address	Byte number little endian			Byte number big endian		
	8-bit	16-bit	32-bit	8-bit	16-bit	32-bit
0x0	1	1	1	1	2	4
0x1	2	2	2	2	1	3
0x2	3	3	3	3	4	2
0x3	4	4	4	4	3	1
0x4	5	5	5	5	6	8
0x5	6	6	6	6	5	7
0x6	7	7	7	7	8	6
0x7	8	8	8	8	7	5
0x8	9	9	9	9	10	12
0x9	10	10	10	10	9	11
0xa	11	11	11	11	12	10
0xb	12	12	12	12	11	9
:	:	:	:	:	:	:

3.11.8.2 Byte collecting and 32-bit address area handling

Byte collecting of netX DPM interface provides 32-bit data consistence for external DPM interfaces of less than 32 bit and it minimizes netX-internal data traffic.

Byte collecting basically means: Several external DPM accesses of less than 32 bit are combined to one netX-internal 32-bit access.

Two sorts of byte collecting are implemented in netX DPM interface:

- Read Byte Latching (RBL)
- Write Byte Collecting (WBC)

RBL and WBC are configurable for each DPM data window separately by registers *dpm_win*_map*. They are also available for DPM access tunnel. Both are enabled by default. For DPM configuration window 0, RBL and WBC are disabled and cannot be enabled.

Neither RBL nor WBC data buffers/latches are affected by any access (neither read nor write) to DPM configuration window 0 (e.g. for reading DPM access error status or IRQ handling).

Note: There are two differences concerning byte collecting between netX 10 and netX 51/52:

Since netX 51/52 RBL is configurable for each DPM data window separately (by registers *dpm_win*_map*). RBL of netX 10 data windows was always enabled and could not be disabled. For DPM configuration window 0 there is no change: RBL is always disabled there.

Since netX 51/52 WBC buffer is cleared when address changes the current 4 byte boundary.

3.11.8.2.1 Read data latching (RDL) / Read byte collecting

Internally netX is 32-bit data organized. Therefore external DPM read access typically causes a 32-bit read access netX inside, even if the external access width is only 8 or 16-bit (due to external interface width, access size or using serial DPM). However, always transforming each external to an internal read access could lead to problems - e.g. assuming an 8-bit DPM interface and reading 32-bit data FIFOs (by 4 subsequent 8-bit reads) would lead to false FIFO fill level when each external read is transformed to an internal one (the FIFO level will be decreased by 4 while the host reads only one FIFO entry). Another example would be quick changing system status like system timers (read value would become inconsistent when data changes netX inside between the single 8 or 16-bit reads of the 32-bit system timer). To avoid these problems a RDL is implemented.

RDL means: An external read access will initiate a 32-bit access netX inside. Therefore the internal address is truncated at its 4-byte boundary i.e. the 2 lowest address bits are ignored. The read-data returned from this address is latched. The following accesses to the same 4-byte boundary will receive their data from the latch as long as no byte is repeated (each latched byte has an internal flag whether it was really read by host yet or not). There are no requirements concerning access order for this: Bytes or 16-bit words could be read with incrementing addresses, decrementing addresses or in any random order.

To avoid problems when polling for data a repeated access to already read data leads to a new internal access and a complete latch update.

Important: RDL is not affected by accesses to DPM configuration window 0. Normal data window access can always be interrupted by a window 0 access (read or write) without clearing or changing the RDL buffer.

The following circumstances lead to a new internal read access and a complete latch update (configuration window 0 accesses are ignored):

- The external address leaves the current 4-byte address boundary.
- The external access is a repeated access to the same byte or half-word inside the current 4-byte address boundary.
- The prior access was a write access.

Important: RDL could lead to aged (old) read data when related address is updated netX inside. An example for this is given further down.

For 8-bit DPM interfaces that means: Reading all 4 bytes (without repeating one byte) of a 32-bit word causes only one 32-bit access netX inside. The internal access is performed before first byte is stored on external bus. Data of the following 3 external byte accesses is delivered from data latch inside netX DPM interface.

For 16-bit DPM interfaces there will be only two accesses for reading a 32-bit word. The second comes from the latched data then.

For 32-bit DPM interfaces the latch typically is never used.

Furthermore, RDL reduces the access time when already latched data is addresses (and access is not repeated) as no access netX inside is required as shown in the following figure.

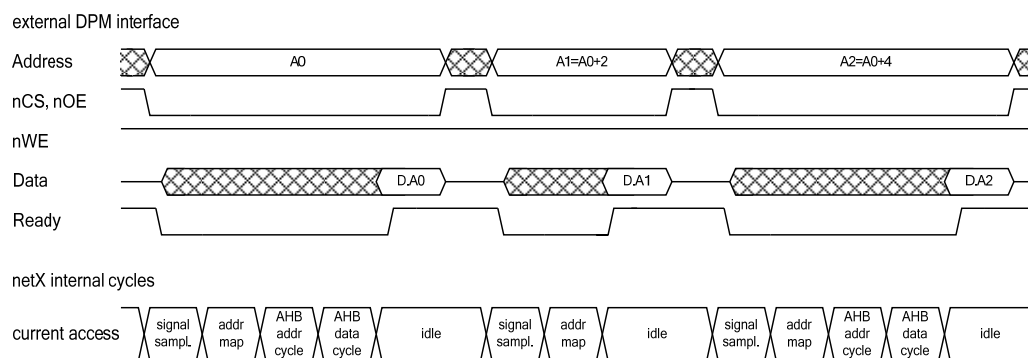


Figure 48: Example: Address incrementing 16-bit read accesses (read data latching)

Host devices supporting APM read burst could also benefit from read data latching as shown in the following figure.

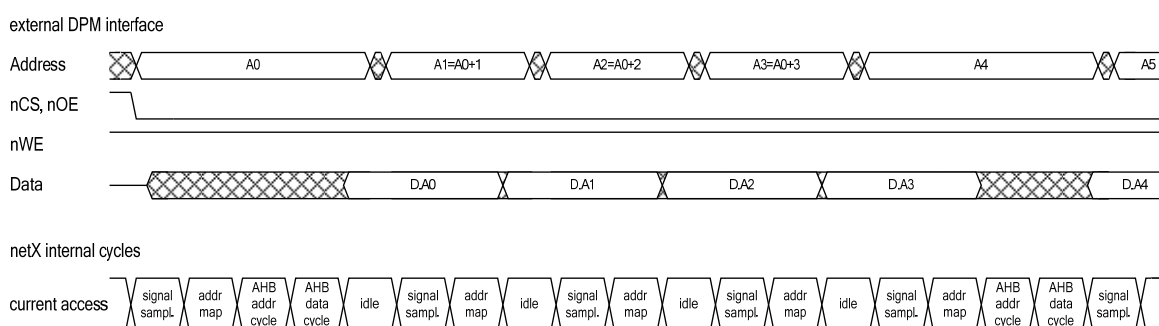


Figure 49: Example: Address incrementing 8-bit APM read burst (read data latching)

3.11.8.2.2 Write byte collecting (WBC)

Without WBC each external DPM access initiates an appropriate write access (of the same width) inside netX. However, many netX register and memory areas are 32-bit accessible only and cannot be byte or 16-bit accessed. WBC allows accessing these areas using serial DPM or DPM interfaces with 8 or 16-bit data anyhow.

If WBC is enabled, external write accesses will be collected. When all 4 bytes within a 32-bit word have been written, the whole 32-bit word is stored at once netX inside.

The following table provides an overview of netX byte-accessible address areas. All other address areas are 32-bit accessible only and WBC must be enabled there.

netX byte-accessible areas	Instances
DPM configuration registers (fix external DPM window 0)	DPM
netX system memories	INTRAM0 to INTRAM5
xPEC data memory	xPEC0 DRAM

WBC buffer will be cleared under the following conditions (configuration window 0 accesses are ignored):

- WBC buffer is full (all 4 bytes were written) and 32-bit write access is performed internally.
- On write access to DPM data window with disabled WBC (byte area).
- When the external address leaves the current 4-byte address boundary.
- On read access.

Important: WBC buffer remains unchanged when current access produces a write error. This access must be repeated.

Important: WBC is not affected by accesses to DPM configuration window 0. Normal data window access can always be interrupted by a window 0 access (read or write) without clearing or changing the WBC buffer.

Important: Using WBC, repeated write access to the same byte within the 32-bit word will replace bytes written before (last write data will be written to netX).

3.11.8.2.3 Byte collecting examples

Data consistence read example:

Considering 32-bit timer register netX inside changing each system clock cycle and 8-bit DPM interface: Reading the whole 32 bit of the register takes 4 external 8-bit DPM read accesses. If the timer register is re-read each external access, data consistence will be lost as the timer register will change between subsequent external accesses (even an overflow could occur). To avoid this, RDL must be enabled for the timer address area. Then, the DPM interface internally reads the timer register only at the first requested byte. The data of the following external reads will be delivered from the RDL latch which contains the whole consistent 32-bit timer value. The register will not be re-read internally until an external access is repeated to an already read byte.

Data polling example:

Considering 16-bit status information which is polled (until a required pattern matches) and 8-bit DPM interface: Reading the whole 16 bit takes 2 external 8-bit accesses. The status register is internally read when the first byte is requested, the second byte will be delivered from the DPM interface data buffer. If the host device detects no pattern match, the first byte access will be repeated. The DPM module detects that this data has already been delivered to the host device and will repeat the internal access to the netX status register. Updated data will be delivered to the host device and stored in the RDL latch. This can be repeated until the host device receives the correct pattern.

Read byte collecting pitfall (aging) example:

Considering byte streams (e.g. strings) being stored by netX ARM CPU and read by DPM: Host device reads one byte from netX (e.g. from address 0x1000). Thereafter, the ARM application running on netX writes one byte to address 0x1001 (same 32-bit address boundary as already read). If the host device now reads the byte at address 0x1001, the old value before the ARM write access to 0x1001 will be delivered from DPM data buffer as this is neither a repeated read nor an address change to a new 32-bit address. To ensure that current data is delivered to the host device in such a case, disable RDL for the related data window by its register *dpm_win_map*.

Example: Write 32 bit in 4 bytes with enabled byte collecting:

external DPM accesses	DPM write buffer ⁽¹⁾	netX-internal access / state
write 0xaa to 0x1004 (8-bit)	0xUUUUUUaa	buffer byte 0
write 0xbb to 0x1005 (8-bit)	0xUUUUbbaa	add byte 1 to buffer
write 0xcc to 0x1006 (8-bit)	0xUUccbbaa	add byte 2 to buffer
write 0xdd to 0x1007 (8-bit)	0xddccbbaa	store buffer to 0x1004, clear buffer

Note: (1) "U" means invalid.

Example: Write 32-bit in 8 and 16-bit accesses with enabled byte collecting:

external DPM accesses	DPM write buffer	netX-internal access / state
write 0xaa to 0x1005 (8 bit)	0xUUUUaaUU	buffer byte 1
write 0xbbcc to 0x1006 (16 bit)	0xbbccaaUU	buffer bytes 2 and 3
write 0xdd to 0x1004 (8 bit)	0xbbccadd	store buffer to 0x1004, clear buffer

Example: Replace byte / one byte missing for internal 32-bit access:

external DPM accesses	DPM write buffer	netX-internal access / state
write 0xaa to 0x1004 (8-bit)	0xUUUUUUaa	buffer byte 0
write 0xbb to 0x1005 (8-bit)	0xUUUUbbaa	add byte 1 to buffer
write 0xcc to 0x1005 (8-bit)	0xUUUUccaa	!!! replace byte 1 in buffer !!!
write 0xdd to 0x1007 (8-bit)	0xddUUccaa	add byte 3 to buffer, wait for byte 2

Example: Clear write buffer due to change of 32-bit boundary (since netX 51/52):

external DPM accesses	DPM write buffer	netX-internal access / state
write 0xaa to 0x1004 (8-bit)	0xUUUUUUaa	buffer byte 0
write 0xbb to 0x1005 (8-bit)	0xUUUUbbaa	add byte 1 to buffer
write 0xcc to 0x1006 (8-bit)	0xUUccbbaa	add byte 2 to buffer
write 0xdd to 0x1003 (8-bit)	0xddUUUUUU	!!! clear data written before !!!

Example: Write interrupted by read to other data window (not window 0) with enabled byte collecting:

external DPM accesses	DPM write buffer	netX-internal access / state
write 0xaa to 0x1004 (8-bit)	0xUUUUUUaa	buffer byte 0
write 0xbb to 0x1005 (8-bit)	0xUUUUbbaa	add byte 1 to buffer
write 0xcc to 0x1006 (8-bit)	0xUUccbbaa	add byte 2 to buffer
DPM read access (0x1000)	0xUUUUUUUU	!!! read access: clear buffer !!!
write 0x11 to 0x1004 (8 bit)	0xUUUUUU11	buffer byte 0
write 0x22 to 0x1005 (8-bit)	0xUUUU2211	add byte 1 to buffer
write 0x4433 to 0x1006 (16-bit)	0x44332211	store buffer to 0x1004, clear buffer

Example: Write interrupted by DPM configuration register accesses (window 0) with enabled byte collecting:

external DPM accesses	DPM write buffer	netX-internal access/state
write 0xbbaa to 0x1004 (16-bit)	0xUUUUbbaa	buffer byte 2 and 3
read from address 0x1c	0xUUUUbbaa	read DPM status register
write to address 0x84	0xUUUUbbaa	write DPM IRQ FIQ mask register
write 0xddcc to 0x1006 (16-bit)	0xddccbbaa	store buffer to 0x1004, clear buffer

3.11.9 External to internal address transformation

Internal netX memory map makes up a full 32-bit address range. For straight access from a host to all netX addresses, a 32-bit address bus between host and netX would be necessary. However, hosts typically do not provide an external 32-bit address bus. To reach all netX addresses from a host by a smaller external address range, address mapping and windowing is implemented inside the DPM module.

Address mapping expands the external access address to an internal 32-bit wide netX address. To make the external netX 51/52 address range more flexible, the DPM can be divided into 5 sub-regions (DPM windows): A special window 0 for DPM status and configuration and 4 general purpose windows (“*data windows 1 to 4*”) for data exchange and access to netX logic. External to internal address mapping and several features (e.g. byte collecting or read ahead) can be programmed for data windows individually. Moreover, there is the DPM access tunnel. This is an additional small (64 bytes) DPM window which can be laid over the normal DPM windows.

The address transformation of the netX 51/52 provides the following features:

- Configurable external address ranges: from 2 kB (A[10:0]) up to 1024 kB (A[19:0])
- Configurable DPM window mapping and address transformation
 - Window 0 containing DPM configuration and status registers, DPM IRQ and handshake
 - Byte accessible (no RDL and no WBC available)
 - Can be mapped to the first 256 external bytes
 - Can be mapped to the last 256 external bytes
 - Can be removed from the external address range
 - 4 configurable DPM data windows to reach full netX-internal 32-bit address area, each with individually programmable
 - window size in 128-byte steps
 - address mapping in 128-byte steps
 - byte collecting (RDL and WBC)
 - read ahead
 - Association to one of the two triple buffer managers of the HANDSHAKE_CTRL unit
- One configurable DPM access tunnel: 64 bytes window to overlay special small netX-internal address area for host (e.g. for handshake cells).

The following figure shows an example of how a host could get access to netX by an external 64 kB address range using an external memory interface (i.e. using parallel DPM).

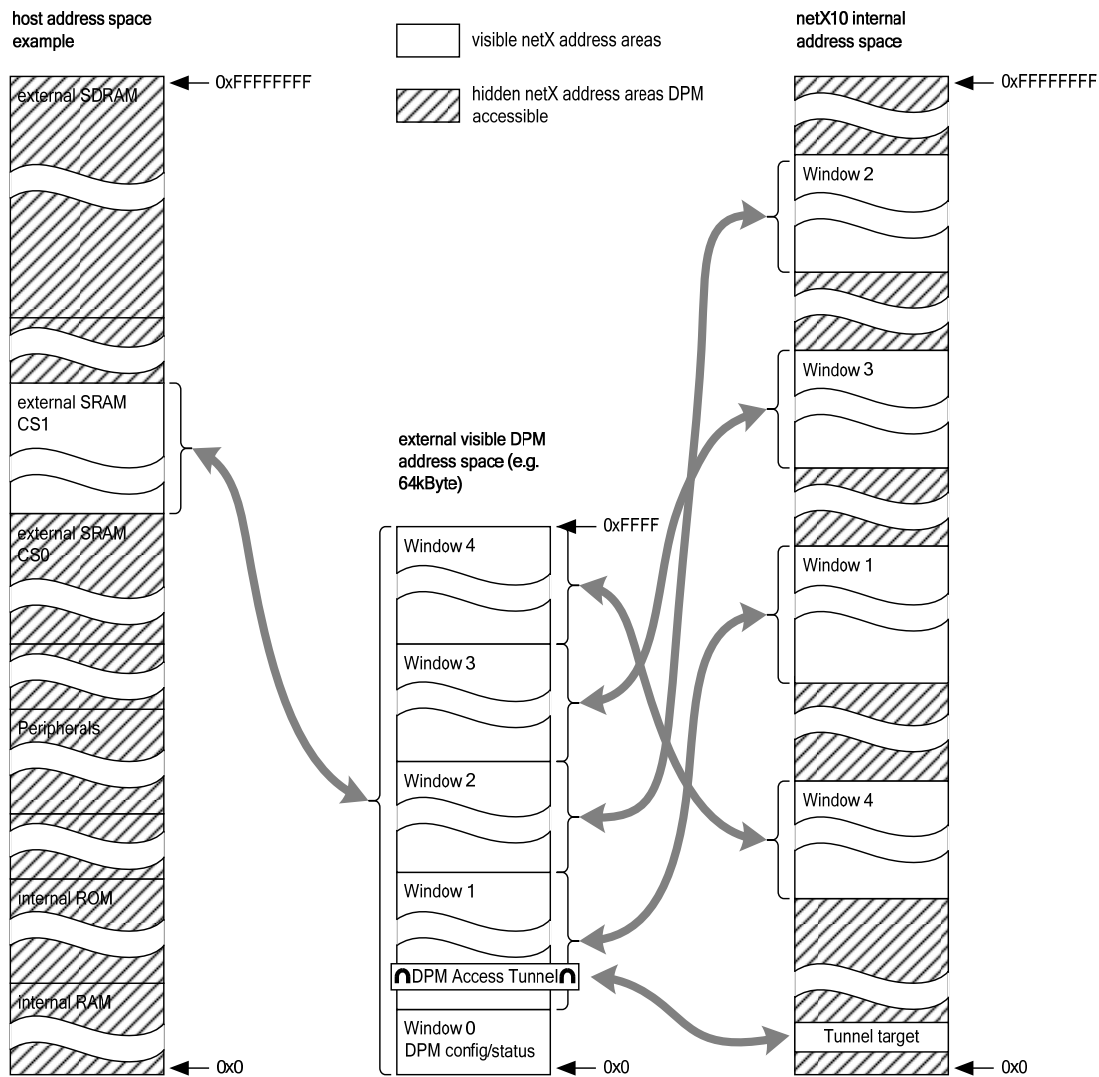


Figure 50: Address transformation example

3.11.9.1 External visible address range

External DPM address range can be configured by register `dpm_addr_cfg`, see reference [1]. After netX reset the smallest address range (2 kB) is set as default. This can be changed to any power of 2 up to 1024 kB (i.e. possible address ranges are: 2 kB, 4 kB, 8 kB, 16 kB, 32 kB, 64 kB, 128 kB, 256 kB, 512 kB and 1024 kB).

Note: If the external address range is configured smaller than it really is (e.g. during setup), the address lines exceeding the configured address range will be masked to 0. This leads to address mirroring if host targets addresses beyond the external address range. E.g. when the external address range is 2 kB (0x0 to 0x7FF) and the host accesses address 0x900, the netX will receive address 0x100.

Note: For netX 10 and netX 5 the programmed external address range had impact on the serial DPM protocol. This is obsolete since the introduction of the new SDP2 frame format with netX 51/52.

3.11.9.2 DPM configuration and status window 0

DPM configuration and status window 0 provides direct access to all registers of DPM module without limitation. It is similar to the *Global Control Block* of netX 50.

Configuration and status window 0 can be located to the first or last 256 bytes of the programmed external address range. Alternatively it can be disabled completely. All this can be programmed by register *dpm_adr_cfg*.

The following table shows possible locations of window 0:

External address range	low window 0		high window 0	
	start	end	start	end
1024 kB (20 bits, lines)	0x00000	0x000FF	0xFFFF0	0xFFFFF
512 kB (19 bits, lines)	0x00000	0x00FF	0x7FF00	0x7FFFF
256 kB (18 bits, lines)	0x00000	0x000FF	0x3FF00	0x3FFFF
128 kB (17 bits, lines)	0x00000	0x000FF	0x1FF00	0x1FFFF
64 kB (16 bits, lines)	0x0000	0x00FF	0xFF00	0xFFFF
32 kB (15 bits, lines)	0x0000	0x00FF	0x7F00	0x7FFF
16 kB (14 bits, lines)	0x0000	0x00FF	0x3F00	0x3FFF
8 kB (13 bits, lines)	0x0000	0x00FF	0x1F00	0x1FFF
4 kB (12 bits, lines)	0x000	0x0FF	0xF00	0xFFF
2 kB (11 bits, lines)	0x000	0x0FF	0x700	0x7FF

All DPM configuration and status registers are also available inside netX INTLOGIC register areas (area DPM). External DPM configuration window 0 is only a mirror of internal INTLOGIC DPM register area. There are no differences and no restrictions concerning register function between those areas. Due to this, DPM configuration registers could also be accessed by the host using a DPM data window (window 1 to 4) targeting INTLOGIC DPM area (e.g. when window 0 is disabled externally). However, we do not recommend doing this:

Important: It is not recommended using DPM data windows 1 to 4 for access to DPM configuration and status registers via INTLOGIC DPM address area. Be careful when doing so: Window 0 access behavior differs in detail, e.g. bypasses byte collecting, (WBC and RDL) and read ahead logic. This has impact on many issues and could lead to problems when INTLOGIC area is used. Additionally for INTLOGIC DPM address area netX inserts wait states which could lead to problems especially when not using the ready signal.

In contrast to DPM data windows (1 and higher), DPM window 0 has the following fix properties that cannot be changed by software:

- No Write Byte Collecting (WBC)
- No Read Data Latching (RDL)
- Always 0 internal wait states
- No read ahead

3.11.9.3 DPM data window mapping (window 1 and higher)

DPM window 1 and higher (“data windows” – for netX 51/52: window 1 to 4) can be used to access all netX-internal addresses. Each of these windows can be programmed individually for size and address target area inside netX. Additional features as byte collecting and read ahead can also be configured for them.

Window size is defined by the programmable external window end address. The first external address of a window is the one following the end address of the preceding window. Hence, window 2 is the external address area following window 1, window 3 the external address area following window 2 and so on. The start address of window 1 is either external address 0 or 0x100 depending on the programmed location of configuration window 0.

Window end addresses, as well as the window size, are programmable in steps of 128 bytes. The size of window 1 can be programmed by register `dpm_win1_end`, the size of window 2 by register `dpm_win2_end` and so on.

The value that must be programmed as window end is the start address of the next window (not the last address of the related window). E.g.: Window 1 end address programmed to 0x400 means that the last external address of window 1 is 0x3FF and the next window starts at 0x400. Due to 128-byte granularity the lower 6 bits of window end address are always 0 and read-only.

Important: Single windows can be disabled by setting their window end address to 0. However, if window *n* is disabled, the end address of window *n-1* defines the start of window *n+1*.

Unused external address areas (unmapped addresses) cannot be programmed between windows. However, there could be an unused external address area after the last window if the last window end address is programmed to a value being smaller than the external address range.

Important: Changing (or disabling it) the end address of a window has always impact on the size of the next enabled window.

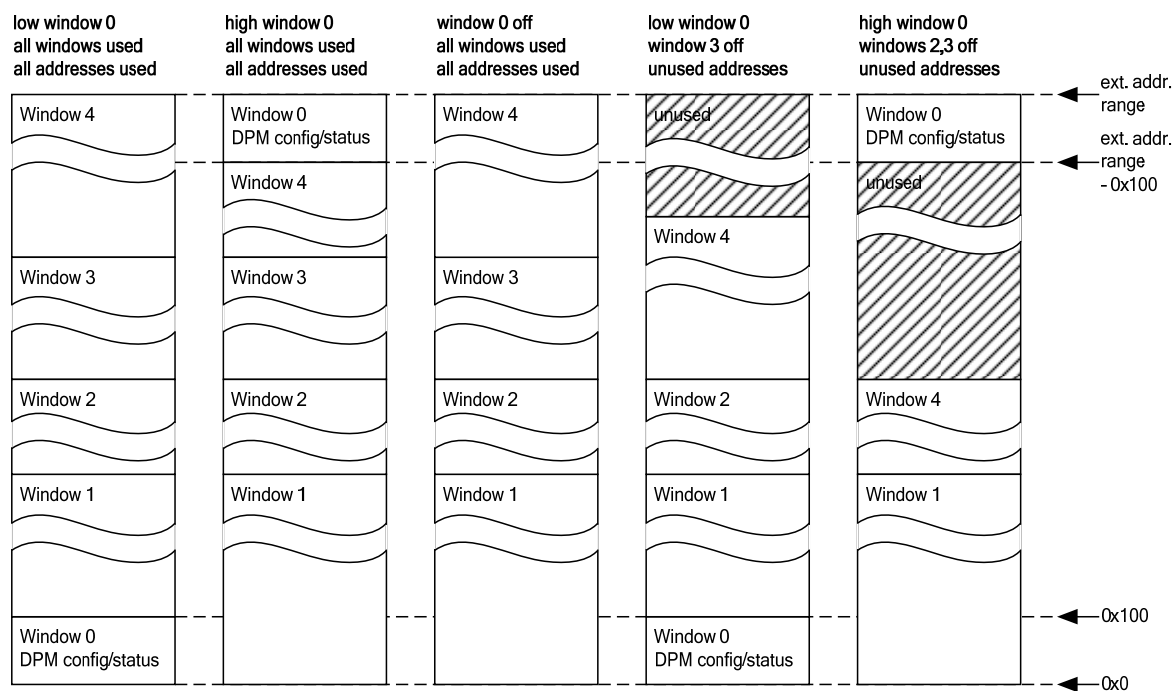


Figure 51: Examples of external address windowing

DPM data windows can be mapped to the whole 32-bit netX-internal address area. To reach this, each DPM window can be dynamically mapped inside a 1024 kB (1 MB) page. Both, mapping value and page base can be programmed.

External to internal address transformation is made by the following formula for a host access to window n: $\text{netx\_addr} = \text{win\_page}(n) + ((\text{win\_map}(n) + \text{ext\_addr}) \& 0x000FFFFF)$

where

- `ext_addr` is the external address to data window n
- `win_page(n)` is the base address off the 1 MB page programmed for window n
- `win_map(n)` is the mapping value programmed for window n
- `netx_addr` is the resulting 32-bit netX target address.

Important: The addition “`win_map(n) + ext_addr`” is only 20 bit wide, i.e. it is performed only for address bits 0 to 19 (no carry to `netx_addr` bit 20). This is the reason for “`& 0x000FFFFF`” in the formula above.

Transformation (window page and window map value) and all further programmable properties of window 1 can be programmed by register `dpm_win1_map`. For window 2 this must be programmed by register `dpm_win2_map` and so on.

Important: Changing the mapping of a window never has impact on the size or mapping of other windows (in contrast to changing its end address).

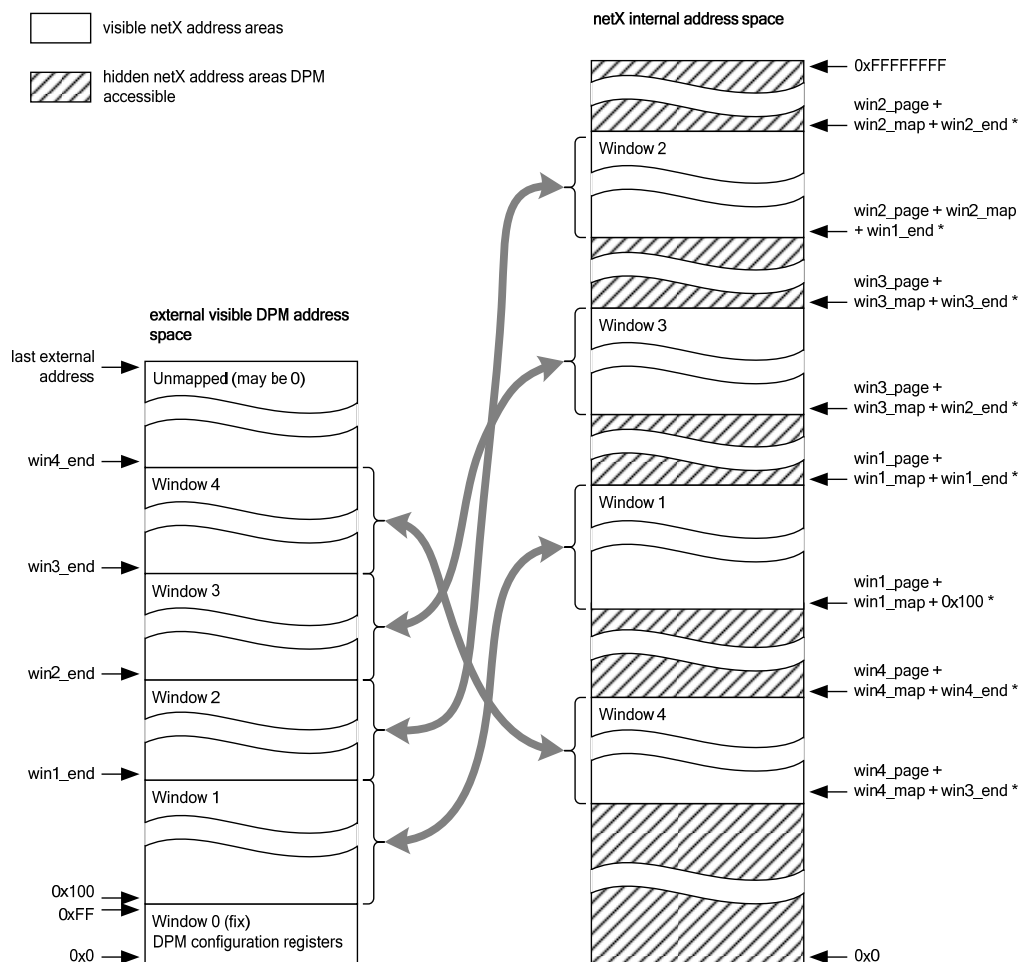


Figure 52: DPM address mapping (example with low configuration window 0)

Address mapping calculation and rules

The following basic rules must be observed when configuring netX DPM window mapping:

- Used (not disabled) data windows must be configured with ascending addresses:

$$win\_end1 < win\_end2 < win\_end3 < win\_end4$$
- A data window can be disabled by setting the respective *win_end* register value to 0.
- Data window mapping and size can be configured in steps of 128 byte.
- First external address of window n: *win_end(n-1)*
- Last external address of window n: *win_end(n) – 1*
- Window size calculation of window n:

$$win\_size(n) = win\_end(n) - win\_end(n-1) \text{ [bytes]}$$
- First internal address of window n:

$$win\_page(n) + ((win\_map(n) + win\_end(n-1)) \& 0x000FFFFF)$$
- Last internal address of window n:

$$win\_page(n) + ((win\_map(n) + win\_end(n) - 1) \& 0x000FFFFF)$$
- If low configuration window 0 is used, the start of the first enable data window is 0x100
- If high configuration window 0 is used, the last 256 bytes of the external address range will be mapped there. They will be mapped with a higher priority if a data window end address is programmed to the external address range (i.e. is programmed for highest addresses).
- Remapping of one window (changing the respective *dpm_win_map* register value) without changing its size (changing the respective *dpm_win_end* register value) does not affect the other windows.
- Changing the size of one window (changing the respective *dpm_win_end* register value) will change location and mapping of all other windows above.

Note: In the rules above window(n-1) is the preceding enabled window of window n.

Data window mapping example

External DPM interface contains 14 bit. The following netX address areas shall be accessible by host device: Handshake-INTRAM5 (by *intram5_dpm_mirror*), INTRAM0, xPEC0 registers and xPEC0 data RAM. INTRAM5 and INTRAM0 shall be byte-accessible and read ahead shall be enabled. Data RAM of xPEC0 shall also be byte-accessible, but without read ahead. For registers, 32-bit write accesses shall be performed internally and read ahead shall be disabled. DPM configuration window 0 shall be visible at low external addresses.

External address range is limited to 8 Kbyte by 14-bit address (0x0 to 0x2000). Moreover, first 256 bytes will be needed for configuration window 0. Less than 8 Kbyte are available for data windows, but INTRAMs and xPEC RAM exceed this. The xPEC0 register area needs 260 bytes.

A good solution could be to provide 2 Kbyte windows for each RAM and a small window of 384 bytes (due to 128 byte granularity) for xPEC0 registers. A block of the first 2 Kbytes of each RAM can be accessed by a host then. If access to another block of a RAM is requested, the related window can be remapped to it without impact on any other window.

An appropriate DPM window setup can be programmed as described by the following table.

DPM window n	win 0	win 1(data)	win 2 (data)	win 3 (data)	win 4(data)
usage	DPM status	INTRAMHS	INTRAM0	-	-
size (bytes)	256 ⁽¹⁾	2048	2048	-	-
ext. start addr.	0x0000 ⁽¹⁾	0x0100	0x0900	-	-
last ext. addr. ⁽²⁾	0x00FF ⁽¹⁾	0x08FF	0x10FF	-	-
window end addr.	0x0100 ⁽¹⁾	0x0900	0x1100	-	-
netx addr. start	-	0x1A008000	0x08000000	-	-
netx page addr. ⁽²⁾	-	0x1A000000	0x08000000	-	-
win. map value ⁽³⁾	-	0x07F00	0xFF700	-	-
read byte area ⁽⁴⁾	yes ⁽¹⁾	yes	yes	-	-
write byte area ⁽⁴⁾	yes ⁽¹⁾	yes	yes	-	-
read ahead	no ⁽¹⁾	no	yes	-	-
win*_end reg. value	n.a.	0x00000900	0x00001100	-	-
win*_map reg. value ⁽⁴⁾	n.a.	0x1A007F11	0x080FF713	-	-

Notes:

1. Fix configuration window 0 ends at 0x100. No end and mapping register are available here. Location is programmed in register *dpm_adr_cfg*.
2. Window page value is derived from the highest 12 bits of the netX address start
3. Window mapping value can be derived from the formulas above, but observe the bit widths: The mapping value is 20-bit wide. It can be calculated then by

$$\text{win\_map} = \text{netx\_addr\_start} - \text{netx\_page} - \text{ext\_start\_addr}$$
4. Since netX 51/52, byte accessibility is split off for read and write. To support internal byte write, WBC must be enabled by setting the *byte_area* bit of register *win_map*. To support internal byte read, RDL must be disabled by setting bit *dis_rd_latch* of register *win_map*.

Window remapping example

Considering DPM address mapping as described in the example above.

If the host requires access to INTRAM0 data which is not located in first 2 kB block, window 2 must be remapped. To change window 2 from the first 2 kB INTRAM0 block (netX addresses 0x08000000 to 0x080007FF) to the third 2 kB block (0x08001000 to 0x080017FF), window mapping value must be changed to 0x00700. The new value of the whole register *dpm_win2_map* must then be 0x08000713. As the window size is not changed, the other DPM windows remain unchanged.

3.11.9.4 Access to uninitialized DPM addresses

Read access from unmapped external addresses will return 0x0BAD0BAD (or a part of this if 8 or 16 bits are read only). Write access to such addresses will be ignored. The ready signal will be set to ready state immediately if ready signal usage is enabled.

Important: The DPM module must already be enabled and configured according to the host requirements before 0x0BAD0BAD pattern can be read. While the DPM is disabled no read data will be driven. The host will receive data determined by states of data lines coming from pull-up/down resistor configuration or states driven in PIO modes then.

Note: 0x0BAD0BAD pattern could be used for DPM wakeup polling (3.11.9.6).

Note: There is no further detection for access to unmapped addresses (no status bit, IRQ or the like).

3.11.9.5 netX 50-compatible registers

The netX 50/100 DPM interface uses a 512-byte configuration window in 64 kB mode. However, the lower half is marked as reserved. All host-accessible DPM registers of the netX 50/100 DPM are located in the last 256 bytes. The netX 51/52 DPM interface provides compatible registers for some netX 50/100 functions typically required by host applications.

Note: The netX 50-compatibility registers used to be part of the netX 50/100 *global control block*.

The following table shows the set of netX 50-compatible registers (according to “netX 50 Program Reference Guide Rev.10 – 2008/11”). All entries marked by “-“, are reserved. Register details are provided in register description, see reference [1].

Address ⁽¹⁾	Register name (netX 50)	Register name (netX 51/52)	Compatible
0xc0	DPM_HOST_WDG_HOST_TIMEOUT	_(2)	no
0xc4	DPM_HOST_WDG_HOST_TRIG	_(2)	no
0xc8	DPM_HOST_WDG_ARM_TIMEOUT	_(2)	no
0xcc	-		no
0xd0	DPM_HOST_TMR_CTRL	_(2)	no
0xd4	DPM_HOST_TMR_START_VAL	_(2)	no
0xd8	DPM_HOST_SYS_STAT	dpm_sys_sta	yes
0xdc	DPM_HOST_RESET_REQ	dpm_reset_request	yes
0xe0	DPM_HOST_INT_STAT0	dpm_firmware_irq_raw	yes ⁽³⁾
0xe4	-	-	no
0xe8	DPM_HOST_INT_STAT2	dpm_firmware_irq_raw2	yes ⁽³⁾
0xec	-	-	no
0xf0	DPM_HOST_INT_EN0	dpm_firmware_irq_mask	yes ⁽³⁾
0xf4	-	-	no
0xf8	DPM_HOST_INT_EN2	dpm_firmware_irq_mask2	yes ⁽³⁾
0xfc	-	dpm_netx_version	no

Notes

1. The given address is the address offset inside the external DPM configuration window 0. This window must be mapped to the last 256 bytes of the external DPM address space for netX 50-compatibility (or must be disabled).
2. The netX 50/100 DPM timer and watchdog registers are obsolete.
3. Some IRQ events became obsolete for netX 51/52 and are omitted.

3.11.9.5.1 netX 50-compatible DPM interrupts

The netX 51/52 firmware IRQ covers the IRQ function of the host side of the netX 50 global control block. Possible interrupt sources are the 16 handshake cells and a system status change event. The DPM watchdog timeout and host timer event are no longer available.

Each interrupt request can be enabled or disabled independently. Moreover, each IRQ is “OR-ed” with any other enabled DPM interrupt to the global interrupt request for the host interface which is routed to the netX VIC (Vectored Interrupt Controller), i.e. to the netX host interrupt output signal.

While all DPM interrupt sources have separate flag bits, their status can additionally be read by a common 8-bit interrupt vector that always reflects the active (and enabled) interrupt with the highest priority. As soon as this interrupt is cleared, the vector displays the next active (and enabled) interrupt with the next lower priority. If no (enabled) interrupts are active, the vector has the value 0x00. Writing a 1 to the status flag, will clear the interrupt status flag. Writing a 0 has no effect. Reading a handshake register will clear the corresponding interrupt automatically.

One global interrupt request bit signals that one or more interrupt request is/are active. While all interrupt status flags will always be set when the corresponding source is active (regardless of whether the interrupt has been enabled or not), the global bit is set only if at least one of the active interrupts (which correspondingly applies to the reset vector) has also been enabled.

3.11.9.5.2 netX 50-compatible DPM software reset

The DPM software reset register allows the host processor to reset the netX chip via a special access procedure. After initiation, the chip will be reset. The software reset is comparable to any other reset except for the power-on reset.

For initiation of the DPM software reset, the host has to write the sequence 0x00, 0x01, 0x03, 0x07, 0x0F, 0x1F, 0x3F and 0x7F to register *dpm_reset_request*.

3.11.9.5.3 netX 50-compatible system status

Register SYS_STA of the netX 50 DPM global control block is located in the ASIC_CTRL area for netX 51/52. SYS_STA function is still netX 50-compatible: It is determined for exchanging status information between netX and host processor chip. Some bits of this register can only be written from netX side and some bits are only writeable from host side. This register also controls the state of the status LED 'RDY' and 'RUN'. To provide status information to the host, the netX software can use an 8-bit field for the NETX_STATUS_CODE. If enabled, a system status interrupt (external interrupt to the host system) will be generated on any write access to the data bits 8 to 15 of the system status register.

3.11.9.5.4 netX 50-compatible host timer

The host timer is no longer supported for netX 51/52.

3.11.9.5.5 netX 50-compatible process data watchdog timer

The process data watchdog timer is no longer supported for netX 51/52.

3.11.9.6 Initializing the DPM interface

An entire DPM initialization is usually not required since certain functions can be reconfigured customized by a host later on (e.g. host could change from 8-bit SRAM mode to 16-bit SRAM mode; expand external address area or enable DPM-ready signal itself). However, after reset the basic DPM function must be programmed by software running on netX-side first (typically by netX boot loader running on netX ARM CPU). The information required on the desired basic DPM mode may come from the netX IO-pull-up/down environment sampled at reset (netX `sample@nres` function) or other sources (e.g. external FLASH).

This basic DPM initialization must set up the DPM as far as necessary to provide a DPM suitable for the min. requirements of the host. E.g. if the host requires multiplexed mode and does not support standard SRAM mode, netX must initialize DPM for multiplexed mode. Basically that means: Initializing the netX software must setup DPM in a way that provides accessibility to the DPM configuration window 0 for the host.

Important: The host can reconfigure the DPM only if DPM configuration window 0 (or DPM area inside INTLOGIC address area) is available and not disabled by DPM initialization. If it is disabled, the netX software must completely set up the DPM.

Before the host can access netX by DPM, the host must ensure that the DPM is ready for use, i.e. netX power-up or reset and basic DPM configuration have been carried out, e.g. by:

- A special wakeup signals: E.g. a netX output that is switched from low to high by netX software after DPM is ready. However, this will use an extra IO on netX and host and lead to additional wiring on PCB.
- Wakeup polling: Host polls on a special address and waits for a significant pattern. Examples:
 - netX ARM initializes the DPM interface and sets up window mapping so that external address 0x100 targets INTRAM. When ready the value that is target of 0x100 is changed to a significant pattern (e.g. 0x12345678).
 - Should the host perform the window mapping, the 0x0BAD0BAD pattern of unmapped addresses can be used (3.11.9.4).
 - Some registers of DPM configuration window 0 can be used for polling, e.g. register `dpm_win0_map`, even if the netX-internal address area is still locked (*unlocked-bit* of register `dpm_status`).

Important: DPM access to DPM data windows could be blocked during netX booting even if DPM is already configured and configuration window 0 is reachable. This can be determined inside window 0 by host by *unlocked-bit* of register `dpm_status`. If the DPM configuration window 0 is not available on the external address range, handshake signal or polling target must not flag DPM-ready while the internal address area is still locked by the software running on netX.

Important: It is not recommended to simply check for a worst case netX power-up time before first DPM access: After electrical power-on and stable external clock generation, internal netX power-up procedure lasts about 12 ms (netX-internal PLL power-up after release of netX reset input). After that, internal components could run boot procedures before configuring the DPM module. Time for this could vary widely (e.g. compare booting of a small image from fast parallel flash with booting of a large image from a slow serial interface) and could even be unpredictable (considering possible firmware or boot image changes). Simply checking for a worst-case netX power-up time before the first DPM access is therefore not recommended.

Note: The configuration registers for global DPM enable and selection between parallel and serial DPM are not part of the DPM register area (neither window 0 nor internal DPM area of INTLOGIC). For netX 51/52 the global enable and mode selection must be configured by register *hif_io_cfg* of HIF_IO_CTRL area (inside INTLOGIC area).

Note: During netX reset and while not configured by software all host interface pins are in undriven input mode and pulled high by internal pull-up resistors (exception: HIF_SDCLK/HIF_SIRQ is pulled down). Users should make sure that these high level signals will not cause an undesired behavior of their host system (e.g. active high interrupt signal) and provide the appropriate pull-down resistors where necessary.

Initializing parallel DPM

Setting up parallel DPM to a desired mode typically requires several DPM configuration registers to be changed: The basic DPM interface mode must be configured in register *dpm_cfg0x0*. However, further programming of other registers (e.g. for special signal properties or address range) is required additionally. An instable interface behavior may result if the whole DPM setup cannot be applied at once. Even damage may result if a host tries to access the DPM e.g. for wakeup-polling or if netX is reset while the host still tries to access the netX.) The netX 51/52 DPM provides a solution for this:

To avoid instable configurations, major changes of DPM interface mode must be confirmed by writing the bit *mode* in register *dpm_cfg0x0*. Before confirmation, new settings are stored in shadow registers only. The following registers are concerned:

- All bits of register *dpm_if_cfg* (i.e. global signal configuration)
- The bit fields *addr_cmp_a** in register *dpm_addr_cfg* (address comparator configuration).

Important Confirmation is required even if the value written to register *dpm_cfg0x0* does not change the register.

Important New settings are never applied before confirmation. They are stored in shadow registers only. Reading will always deliver the last written data to allow read modify write, i.e. read data comes from the shadow registers before confirmation.

Note: The table of section 3.11.5.1 provides the relations between DPM signals and DPM configuration registers. For more information, see the detailed register descriptions.

Note: DPM must be enabled and parallel mode must be selected in register *hif_io_cfg* of INTLOGIC-HIF_IO_CTRL. This has not been changed since netX 10.

The following procedure is recommended for parallel DPM initialization

1. Setup basic DPM function from netX side (e.g. by netX ARM). In the meantime the host may monitor the wake-up signal or poll for the wake-up pattern.
 - A. Determine the desired DPM interface function (e.g. “use parallel multiplexed mode”) for configuration e.g. by IO-states, sample@nres states or configuration found in non-volatile devices (e.g. FLASH).
 - B. Enable the DPM and select the parallel mode by register *hif_io_cfg* of HIF_IO_CTRL area.
 - i. *hif_io_cfg.sel_hif_dpm*: Set the bit to enable DPM
 - ii. *hif_io_cfg.sel_dpm_serial*: Do not set the bit, select parallel DPM
 - C. Set up the DPM for the basic host requirements to make configuration window 0 accessible for the host. This requires a least:
 - i. Register *dpm_if_cfg*: Program the signal behavior according to the desired parallel DPM mode (e.g. 16-bit for Motorola 68000 CPU, see 3.11.5.2). This will be stored in the shadow registers only!
 - ii. Register *dpm_addr_cfg*: If required, set up the external address range and the internal address comparator. This will be stored in the shadow registers only! Also select or disable the low or high DPM configuration window 0. However, if disabled, netX must perform all further configurations.
 - iii. *dpm_cfg0x0.mode*: Select the basic parallel DPM and confirm the settings. This will apply the shadow register settings to the DPM.
 - D. As soon as the “unlocked bit” is set in register “dpm_status”, the netX can flag “DPM ready” to the host. This occurs either via a host-monitored netX-PIO signal (wake-up signal) or a wake-up pattern. The wake-up pattern is a definite value at a definite address in the netX, typically in the DPM. If the host finds the expected value while polling this address, the netX is “awake”. If the wake-up pattern is to be found in the INTRAM, an additional or temporary window mapping is required which can be overwritten later by the host or netX.
2. Customizing the DPM from host side: DPM configuration window 0 is available now so host can program DPM features directly according to its requirements. The advantage of this is that netX needs no further information on host properties (e.g. timing).

Important: Customizing DPM for host is absolutely necessary for good performance although basic access is also possible without it. If configuration window 0 is disabled, the netX has to customize the DPM.

- A. *dpm_cfg0x0.endian*: Program host endianness
 - B. *dpm_timing_cfg*: Program timing parameters for host requirements
 - C. Should the ready signal be used, perform the following steps (step order is important)
 - i. *dpm_rdy_cfg*: program ready behavior
 - ii. *dpm_pio_cfg1.sel_rdy_pio*: switch from PIO to ready mode.
 - iii. enable the ready usage on the host CPU (typically the host timing must also be reconfigured)
 - D. The host can expand the external address range of the default DPM configuration:
 - i. *dpm_addr_cfg:addr_range* Program external address range (address bus width). This will be stored in the shadow-registers only!
-

- ii. Rewrite (read-write) *dpm_cfg0x0.mode* for confirmation. If a high configuration window is used, its location will change from host view.
- E. Program DPM data windows and address mapping
 - i. *dpm_win1_end* to *dpm_win4_end*: Program window size and active windows
 - ii. *dpm_win1_map* to *dpm_win4_map*: Program window mapping for target addresses and properties (byte area, read ahead).
- F. Program DPM tunnel, if used:
 - i. *dpm_tunnel_cfg*: Enable tunnel and program external tunnel base and tunnel properties.
 - ii. *dpm_itbaddr*: Program internal tunnel target
- G. Program IRQs for host, if used:
 - i. Program the registers *dpm_irq_irq_mask_set / _reset*; *dpm_irq_fiq_mask_set / _reset* and *dpm_firmware_irq_mask*
 - ii. *dpm_io_cfg_misc*: Set up IRQ behavior (drivability and activity level)
 - iii. *dpm_pio_cfg1*: Change from PIO to IRQ mode.

3.12 Serial DPM interface

Apart from standard parallel DPM interface modes (3.11.4) netX DPM supports a high speed serial DPM interface based on Motorola SPI (Serial Peripheral Interface). All four Motorola standard SPI frame formats are supported (SPI clock polarity 0 or 1, SPI clock phase 0 or 1).

A hardware protocol converting unit inside netX DPM interface transforms serial data to netX-internal accesses.

Note: The serial DPM mode must be configured in the register HIF_IO_CTRL.

netX 51/52 supports the following features:

- 4-bit Quad SPI (sDPM4) mode, clock rates up to 33 MHz (i.e. data-rates up to 133 Mbit/s, 16.6 Mbyte/s) are supported
- 1-bit serial SPI (DPM-SPI) mode, clock rates of up to 125 MHz
- all 4 SPI clock modes differing in phase or polarity are supported
- timing of serial read-data from netX can be configured
- serial status byte for quick error handling

3.12.1 Motorola SPI introduction

SPI (Serial Peripheral Interface) is a full-duplex 4-wire interface defined by Motorola. Transfers are serial, 8-bit oriented and bidirectional between master and slaves. Slave devices are selected by a chip-select signal. Data is transferred MSB first.

Note: Motorola SPI standard does not specify 8-bit data frames.
Though netX serial DPM always works byte-oriented.

Motorola SPI signals are:

- Clock signal (SPI_CLK or SPI_SCK), master driven
- Master transmit, slave receive data signal (SPI_MOSI), master driven
- Master receive, slave transmit data signal (SPI_MISO), slave driven
- Active low chip-select signal (SPI_CS_N), master driven

Note: Chip-select signal is sometimes also called slave-select signal (SPI_SS or SPI_FSS)

3.12.1.1 Frame formats

Four different Motorola SPI frame formats are provided. For netX DPM they can be configured by software.

SPI clock polarity (SPO)

SPI clock polarity is the idle state of SPI clock signal SPI_CLK when no data is transferred.

SPI clock phase (SPH)

SPI clock phase controls SPI data generation and sample timing. If SPH 0 is selected, data on SPI_MOSI and SPI_MISO will be sampled on the first clock edge of SPI_CLK. If SPH 1 is selected, the signal will be sampled on the second edge. Data will be generated one edge before.

SPI_MOSI read data generation

To realize fast SPI_MISO generation, read data is always generated early by netX DPM interface. That means SPI_MISO is set to next bit's state at master sampling edge i.e. one edge before master data (SPI_MOSI) generation. This is necessary to compensate runtime latencies of SPI_CLK and SPI_MISO signal. These latencies exceed several nano seconds. Read data on SPI_MISO will thus always be stable at the host device MISO input when the sampling clock edge is generated.

Note: SPI_MISO changes are always related to the sampling clock edge. The following figures (Figure 53 to Figure 56) show fast SPI_CLK clock behavior where the next SPI_CLK edge has been generated before SPI_MISO change propagated to the host device. For slow clock frequencies SPI_MISO state changes a short time after the sampling clock edges.

3.12.1.2 SPI frame format with SPO=0 and SPH=0

Clock idle state is low and data is sampled on the first (rising) edge of any SPI_CLK period. MSB data is generated when chip-select is activated. SPI_MOSI changes on the falling SPI_CLK edge.

Note: SPI_MISO generation is explained above.

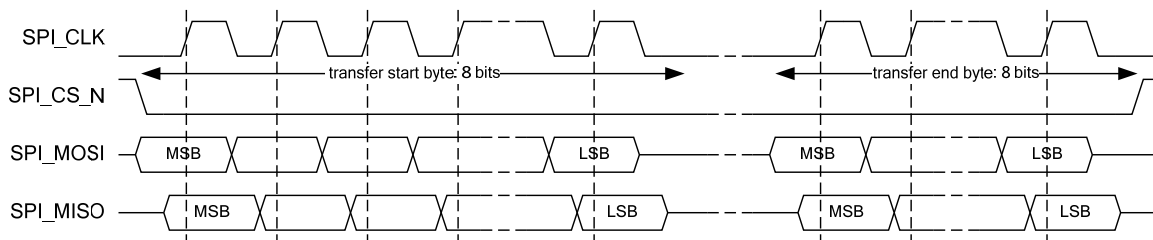


Figure 53: SPI SPO=0 and SPH=0 transfer

3.12.1.3 SPI frame format with SPO=0 and SPH=1

Clock idle state is low and data is sampled on the second (falling) edge of any SPI_CLK period. MSB data is generated on the first (rising) SPI_CLK edge. SPI_MOSI changes on the rising SPI_CLK edge.

Note: SPI_MISO generation is explained above.

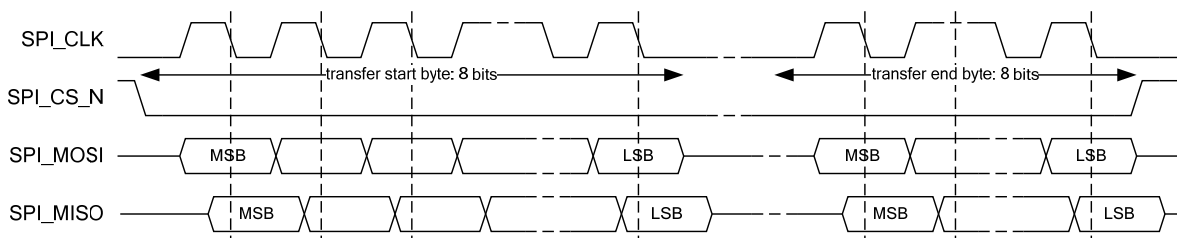


Figure 54: SPI SPO=0 and SPH=1 transfer

3.12.1.4 SPI frame format with SPO=1 and SPH=0

Clock idle state is high and data is sampled on the first (falling) edge of any SPI_CLK period. MSB data is generated when chip-select is activated. SPI_MOSI changes on the rising SPI_CLK edge.

Note: SPI_MISO generation is explained above.

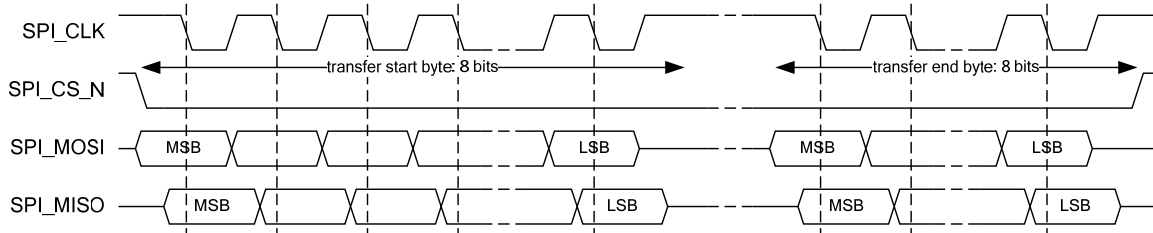


Figure 55: SPI SPO=1 and SPH=0 transfer

3.12.1.5 SPI frame format with SPO=1 and SPH=1

Clock idle state is high and data is sampled on the second (rising) edge of any SPI_CLK period. MSB data is generated on the first (falling) SPI_CLK edge. SPI_MOSI changes on the falling SPI_CLK edge.

Note: SPI_MISO generation is explained above.

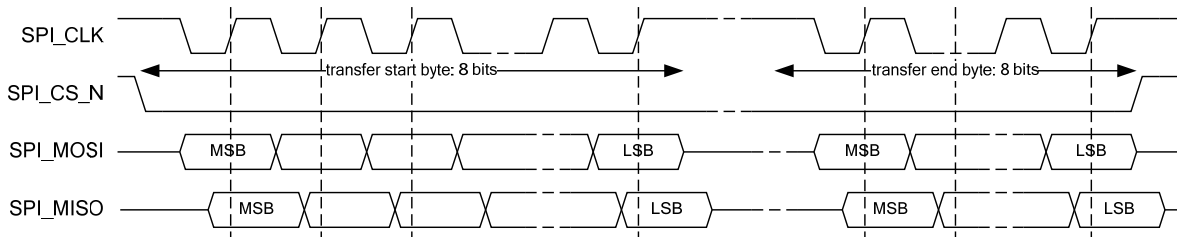


Figure 56: SPI SPO=1 and SPH=1 transfer

3.12.2 Serial DPM interconnection

Serial connections of netX 51/52 DPM interface consist of 4 standard SPI signals for data transfers. Serial mode is configured in the HIF_IO_CTRL registers, not in DPM registers, see section *Initializing the DPM interface* on page 126. Clock polarity and phase can also be configured there according to the SPI mode provided by the host device used.

Interface signal	netX 51/52 connection	Comment
DPM_SPI_MISO	HIF_D8	SPI master in slave out data output
DPM_SPI_MOSI	HIF_D9	SPI master out slave in data input
DPM_SPI_CS _n	HIF_D10	SPI chip-select input
DPM_SPI_CLK	HIF_D11	SPI serial clock input
DPM_SPI_DIRQ _n	HIF_D12	Optional DIRQ for host (e.g. for data IRQ)
DPM_SPI_SIRQ _n	HIF_D13	Optional SIRQ for host (e.g. for service IRQ)

For connecting a host CPU, see reference [2].

3.12.3 Serial 1-bit DPM protocol (SDP2)

Basically all serial DPM transfers of protocol SDP2 consist of a header (3 to 5 bytes) followed by a byte-oriented address-incrementing data stream. All bytes (header and data) are serial transmitted MSB first (i.e. standard SPI). Chip-select signal must remain active all time during an access sequence (inactive chip-select will terminate a transfer).

The transfer header always consists of a direction bit (read/not-write bit), three command (cmd) bits and a 20-bit address.

Note: The serial DPM protocol was completely revised for netX 51/52 and differs to netX 10.

For max. performance, 3 different types of streams are implemented which can be selected for each transfer individually by the cmd-bits:

- Type 0: Straight stream without any dummy cycles or ready polling.
- Type 1: Stream with dummy cycles before each data byte to allow longer netX-internal access times without any error checking or ready polling by a host (e.g. for DMA transfers). The number of dummy cycles before the first and each later data byte (or Dword) can be programmed for each transfer individually by an additional 4th command byte in the header.
- Type 2: Stream with ready polling before each data byte (or Dword).

The following stream types are provided for protocol SDP2. Other settings for cmd are reserved and must not be used.

stream type	cmd	transfer length for n bytes read	for n bytes write
0: straight	000	4+n	3+n
1: byte with dummy	010	5+n+dummy.start+(n-1)*dummy.inter	4+n+d.st+(n-1)*d.int
1: Dword with dummy	011	5+n+d.st+(n>>2-1)*d.int	4+n+d.st+(n>>2-1)*d.int
2: byte ready polling	100	min.: 3+2n	min.: 3+n
2: Dword ready polling	101	min.: 4+n+n>>2	min.: 3+n

All headers of a read-sequence are extended by an additional length-byte before the first data byte. This is necessary to avoid problems with read-sensitive addresses (e.g. FIFOs) as read accesses must be performed as read ahead internally. If a read sequence is not terminated after <length> bytes, invalid read data will be returned. No read access will be performed inside netX then.

Stream types 1 and 2 can be additionally configured as byte streams or Dword-streams. A byte stream requires dummy cycles (or ready-polling) before each data byte. A Dword stream requires dummy cycles (or ready polling) before the first data byte, later they are required only if the address of the related data changes to the next 32-bit word. This can be used to optimize access to DPM windows where Write-Byte-Collecting (WBC) or Read-Data-Latching (RDL) is activated.

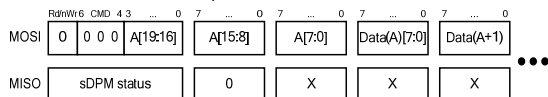
Note: Byte streams can always be performed, even if they target WBC or RDL windows. In that case, dummy cycles or ready polling inside a Dword are/is not required. This helps decrease overhead. In contrast to that, applying Dword-streams to byte areas will fail. When addressing byte areas, byte streams must be used.

MISO line will be driven as soon as chip-select is active (never floating), but the driving state is not always predictable. During dummy cycles of stream type 1, MISO shows the current ready-state of the internal DPM part. The current ready-state can be used for debugging or trying out access timing.

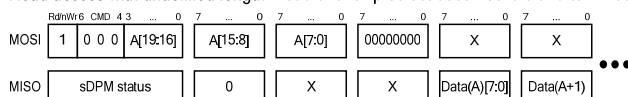
During the first byte of the header the current serial DPM status (3.12.5) will always be clocked out on MISO. The current serial DPM status provides information on the current DPM error states and can be used if access timing is unpredictable.

3.12.3.1 1-bit SPI stream type 0: SDP2 straight stream

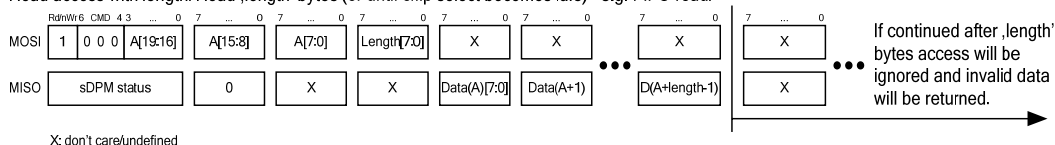
Write access: Write until chip-select becomes idle and terminates sequence:



Read access with undefined length: Read until chip-select becomes idle and terminates sequence, send 0 for length:



Read access with length: Read 'length' bytes (or until chip-select becomes idle) - e.g. FIFO read:



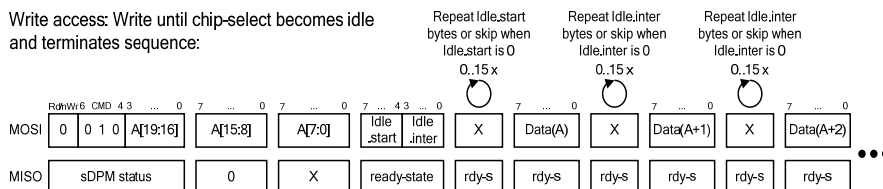
X: don't care/undefined

Figure 57: Serial DPM protocol: straight stream, type 0

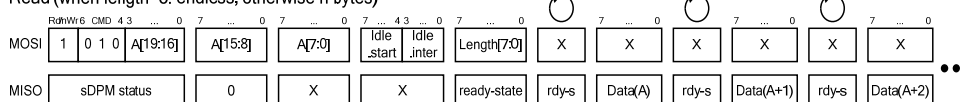
3.12.3.2 1-bit SPI stream type 1: SDP2 stream with dummy cycles

Note: Program the read access length as shown in Figure 57: Set 0 for endless or specify the number of bytes to be read.

Write access: Write until chip-select becomes idle and terminates sequence:



Read (when length=0: endless, otherwise n bytes)



X: don't care/undefined

Figure 58: Serial DPM protocol: dummy cycle stream, byte sequence, type 1

Note: Both, idle.start (dummy.start) and idle.inter (dummy.inter) can be set to 0. Related dummy loops will be skipped then.

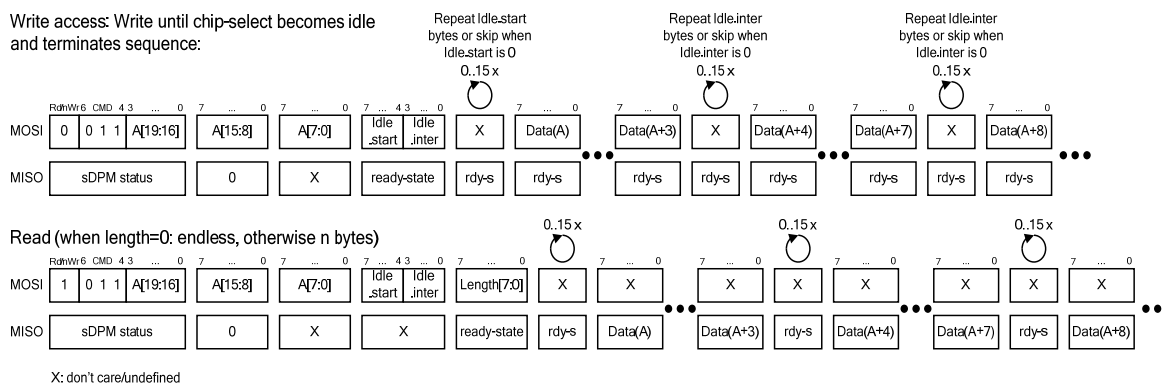


Figure 59: Serial DPM protocol: dummy cycle stream, Dword sequence, type 1

Note: Idle.inter cycle loops are likely to be inserted when changing to a new Dword (i.e. leaving current 4-byte-boundary). If you do not start with an address being dividable by 4, there will be less than 4 bytes before first idle.inter loop.

3.12.3.3 1-bit SPI stream type 2: SDP2 stream with ready-polling

The current netX DPM ready-state is returned by bit 0 of ready polling bytes. Other bits are not always predictable.

Important: Internal netX ready state is simply clocked out while ready polling bytes and can be used for timing analysis. However, ready state after write bytes could overlap with the following read transfer. Hence, simply comparing 'equal 0' of ready polling bytes could fail – this comparison must be masked by “AND 1” (bit 0 must be used).

For read transfers, polling is required before each data byte (or Dword). Once ready has been received as active, the next byte transferred is a data byte.

For write transfers, ready state is related to current write data on MOSI.

For byte stream, each data byte must be written repeatedly until ready is received active on MISO.

For Dword stream, ready-active polling is required only with bytes leaving the current 4-byte address boundary.

Note: Program the read access length as shown in Figure 57: Set 0 for endless or specify the number of bytes to be read.

Note: Ready status of the first read data is already returned when length is set. Sending the length byte need not be repeated (even though it is a write byte) if ready is inactive during the first polling.

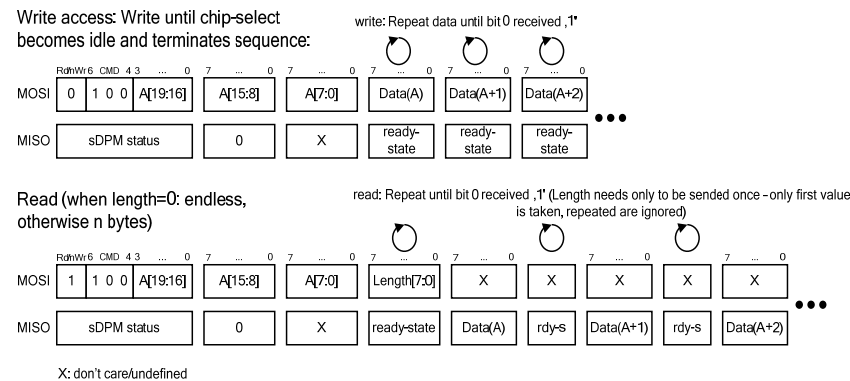


Figure 60: Serial DPM protocol: ready-polling stream, byte sequence, type 2

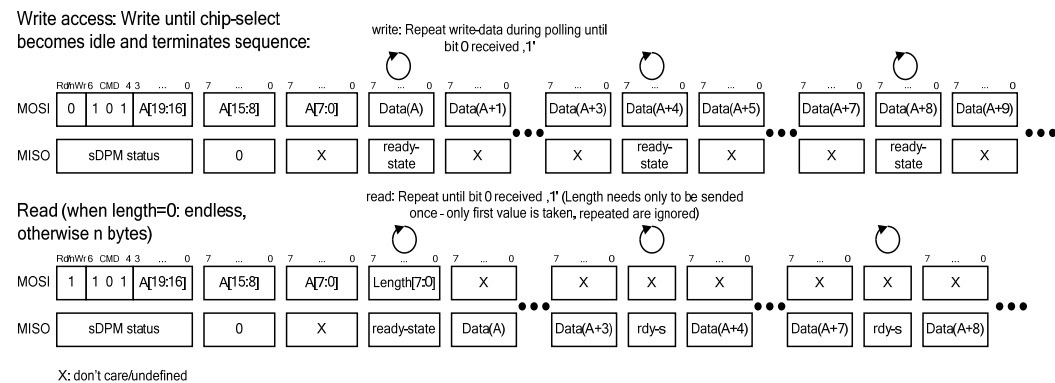


Figure 61: Serial DPM protocol: ready-polling stream, Dword sequence, type 2

3.12.4 Serial 4-bit DPM

netX 51/52 DPM provides a 4-bit SPI (Quad-SPI, SQI) mode consisting of the 4 standard SPI signals plus 2 additional data lines. The way in which signal mapping is performed allows a default 1-bit SPI interconnection which can be switched to 4-bit.

Note: The 4-bit serial SPI DPM mode is also called “SQI-DPM” or “sDPM4”.

SQI Signal	SPI Signal	netX51/52 IO	Comment
DPM_SIO1	DPM_SPI_MISO	HIF_D8	SPI: master in slave out data output SQI: serial data bit 0
DPM_SIO0	DPM_SPI_MOSI	HIF_D9	SPI: master out slave in data input SQI: serial data bit 0
DPM_SQI_CSn	DPM_SPI_CSN	HIF_D10	SPI or SQI: chip-select input
DPM_SQI_CLK	DPM_SPI_CLK	HIF_D11	SPI or SQI: serial clock input
DPM_SPI_DIRQ	DPM_SPI_DIRQ	HIF_D12	Optional DIRQ for host (e.g. for data IRQ)
DPM_SPI_SIRQ	DPM_SPI_SIRQ	HIF_D13	Optional SIRQ for host (e.g. for service IRQ)
DPM_SIO2	-	HIF_D14	SQI only: serial data bit 2
DPM_SIO3	-	HIF_D15	SQI only: serial data bit 3

For connecting a host CPU, see reference [2].

4-bit QSPI/SQI is only half-duplex (in contrast to SPI which is full-duplex): Data uses the same lines for receive and transmit. Data can be sent or received, but not simultaneously (as it is possible with full-duplex SPI). Therefore all data lines are bidirectional. Data direction could change during one transfer (e.g. for read a command is usually sent first, then the data direction is changed and read data is received). All other QSPI/SQI behavior like clock-phase, clock-polarity or chip-select is the same as standard SPI.

Max. clock rates of SQI-DPM are 33 MHz which are limited by netX-internal access times and not by IO-timing.

3.12.4.1 QSPI frame formats

QSPI needs only 2 clock periods to transfer one byte (as there are 4 data lines). Data is organized MSB-first like standard SPI. Figure 62 shows this for serial mode 0 (clock polarity is low and master generating clock-phase is 0).

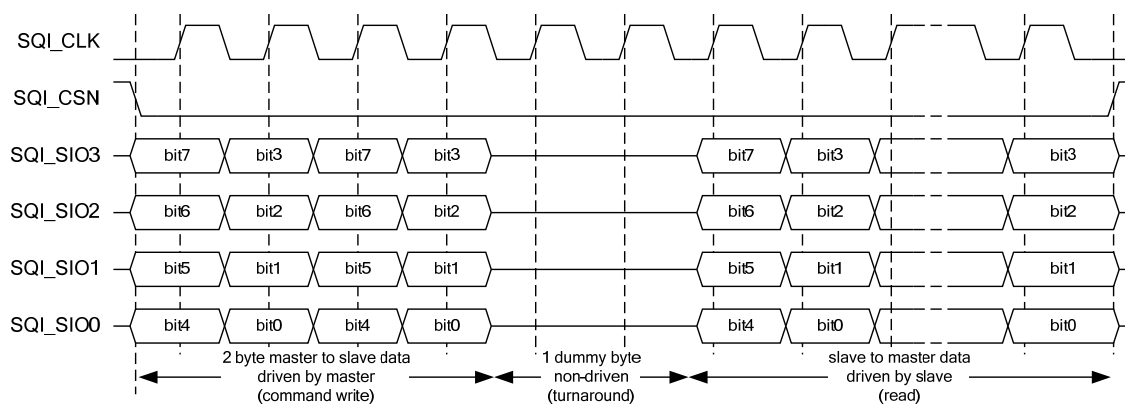


Figure 62: Serial mode 0 QSPI frame format example

Note: Due to latencies of clock signal and data generating logic inside slave device, slave-driven data typically appears later on external lines as master-driven data. Slave data must be valid before the next sampling clock edge.

Due to SPI mode 0 clock idle state is low. The first data is generated when chip-select is activated, all further data is generated by the falling clock edges of any clock period. Data is sampled (received) on each rising clock edge.

Note: Relation between clock polarity/phase and data generation/sampling is the same for SPI and QSPI. For details, see the description of the SPI frame formats in netX DPM documentation.

Due to half-duplex character of QSPI a transfer typically is started by a command sent by master (the first 2 bytes in the example of Figure 62). This informs the slave about the further data direction i.e. whether it is a read or a write transfer. In case of a read transfer (as shown by Figure 62) a turnaround dummy byte is typically inserted before the slave starts driving read data to avoid collisions on data lines (which could cause permanent hardware damage).

3.12.4.2 Serial 4-bit DPM protocol (QDP2)

Protocol QDP2 (4-bit serial DPM) is very similar to protocol SDP2 (1-bit serial DPM). However, the half-duplex character of QSPI/SQI causes some differences between SDP2 and QDP2:

- DPM status cannot be returned at the first transfer-byte for 4-bit serial DPM. A special status read can be performed instead. A new stream type needing only a 1-byte header is provided therefore.
- A turnaround-byte is inserted for read-transfers between header and first byte of read data to avoid collisions on data lines.

The following issues are identical for protocol QDP2 and SDP2:

- All data is transferred MSB first.
- Chip-select signal must remain active all time during a transfer (inactive chip-select will terminate it).
- All data transfers (not status read stream) start with a header (3 to 5 bytes) followed by a byte-oriented address-incrementing data stream. The transfer header always consists of a direction bit (read/not write bit), three command (cmd) bits, and a 20-bit address. Headers of read transfers are extended by a length byte before the first data byte.

Only stream type 0 (straight-stream) is supported for QDP2. Ready-polling and streams with dummy cycles are not supported. SDP2 can be used for access to netX address areas requiring these features.

The following stream types are provided for protocol QDP2. Other settings for cmd are reserved and must not be used.

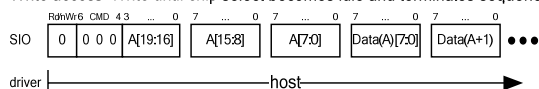
stream type	cmd	transfer length for n bytes read	for n bytes write
0: straight	000	5+n	3+n
3: status read	110	2+n	-

Note: Byte streams can always be performed, even if they target WBC or RDL windows. In that case, dummy cycles or ready-polling in a Dword are/is not required. That helps decrease overhead. In contrast to that, applying Dword-streams to byte areas will fail. When addressing byte areas, byte streams must be used.

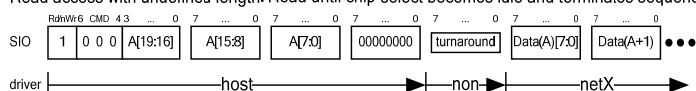
3.12.4.3 4-bit serial DPM stream type 0: Straight stream

4-bit serial DPM straight stream is similar to 1-bit serial DPM straight stream, but a non-driven turnaround byte is inserted between header and data for read and serial status cannot be returned (due to half-duplex character).

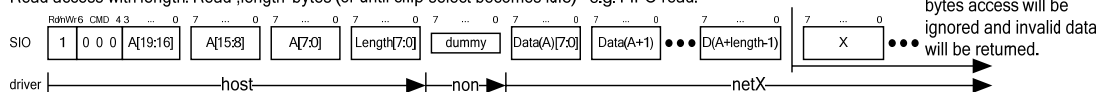
Write access: Write until chip-select becomes idle and terminates sequence:



Read access with undefined length: Read until chip-select becomes idle and terminates sequence, send 0 for length:



Read access with length: Read 'length' bytes (or until chip-select becomes idle) - e.g. FIFO read:



X: don't care/undefined

Figure 63: QSPI Serial DPM protocol: straight stream, type 0

3.12.4.4 Stream type 3: Status read

4-bit serial DPM stream type 3 provides only status read yet. No write transfers can be performed by this stream type. This function is reserved and can be used in future.

Read access: Read current DPM status after command byte until chip-select becomes idle and terminates sequence:

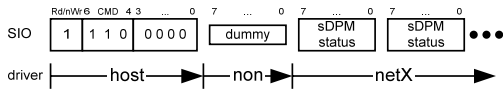


Figure 64: QSPI serial DPM protocol: status-read, type 3, address 0

Serial status of 4-bit serial DPM is identical to serial status of 1-bit serial DPM.

3.12.4.5 Initializing 4-bit serial DPM

Typically QSPI devices also support 1-bit SPI mode. They are in a SPI mode by default after power-up. If a master supports 4-bit QSPI the master can switch itself and the device to 4-bit QSPI mode by a special command later on. This activates the 2 additional data lines and changes to QSPI mode. The host can change directly from SPI to QSPI mode. However, netX DPM must already be programmed to SPI mode from netX side first and DPM configuration window 0 must be available for host.

Important: The netX has to set the serial DPM mode by register HIF_IO_CFG of HIF_IO_CTRL area (not inside DPM configuration window 0 or DPM register area). Switching from SPI to QSPI mode must be performed by register DPM_TIMING_CFG: (DPM_ACCESS_CFG) of DPM configuration window 0 or DPM register area.

Note: DPM configuration window 0 can also be disabled, but then switching to QSPI mode (as well as any further DPM configuration) must be performed by the netX via a special handshake protocol (e.g. using netX handshake cells for special commands).

To initialize the 4-bit SPI, proceed as follows:

1. Set up serial DPM on netX and host side:
 - A. From netX side (e.g. by netX ARM): Enable DPM and select serial DPM by register hif_io_cfg of HIF_IO_CTRL area.
 - i. hif_io_cfg.sel_hif_dpm: Set bit to enable DPM
 - ii. hif_io_cfg.sel_dpm_serial: Set bit for serial DPM
 - iii. Program clock phase and clock polarity by hif_io_cfg.sel_dpm_serial_spo and hif_io_cfg.sel_dpm_serial_sph according to required SPI mode.
 - B. From netX side (e.g. by netX ARM): Make the DPM configuration window 0 accessible for the host. This is required for direct SPI to QSPI switching from the host:
 - i. dpm_addr_cfg.cfg_win_addr_cfg:
Select low or high DPM configuration window 0.
 - C. From host side: Configure host-netX connection for standard SPI and wait for netX DPM wakeup (e.g. by polling for a special data-pattern or by signal handshaking). Use serial DPM SPI protocol (section *Serial 1-bit DPM protocol* on page 133).
2. Switch to QSPI from host side:

- A. Enable 4-bit QSPI mode by setting bit `en_dpm_serial_sqi` bit in register `dpm_timing_cfg` (`dpm_access_cfg`). Terminate the SPI-transfer immediately after the byte containing this bit was written (necessary as netX changes to QSPI mode immediately after the byte was written).
- B. Change host to QSPI mode, but be careful: Make sure that register `dpm_timing_cfg` is really written yet. E.g. FIFOs inside host SPI/QSPI interface could cause delays. Register `dpm_timing_cfg` must be written in SPI mode.
3. Continue transfers in QSPI modes using QSPI serial DPM protocol (Serial 4-bit DPM protocol (QDP2)).

Note: Initializing DPM completely typically requires further steps (e.g. programming address mapping).

Fallback from QSPI to SPI mode

Falling back to SPI mode could be useful to benefit from SPI-DPM protocol features including stream with dummy cycles or with ready polling. Proceed as follows:

1. Switch from QSPI to SPI from host side:
 - A. Disable 4-bit QSPI mode by clearing bit `en_dpm_serial_sqi` in register `dpm_timing_cfg` (`dpm_access_cfg`). Terminate the QSPI-transfer immediately after the byte containing this bit is written (necessary as netX changes to QSPI mode immediately after writing the byte).
 - B. Change host to SPI mode, but be careful: Make sure that register `dpm_timing_cfg` is really written yet. E.g. FIFOs in the host SPI/QSPI interface could cause delays. Register `dpm_timing_cfg` must be written in SPI mode.
2. Continue transfers in SPI modes using SPI-DPM protocol.

3.12.5 Serial DPM status

Serial DPM status provides information on the current DPM status and can be used if access-timing is unpredictable. The status is directly taken from register `dpm_status`. However, the bits inside the serial DPM status are reordered and some bits which are not relevant for serial DPM are omitted there.

Important: For DPM-SPI the current serial DPM status will always be clocked out on MISO while the first byte of the header is transmitted. However, unused (reserved) bits of serial status are not defined and returned value could be unpredictable. These bits must be masked by software before further use.

```
For SPI-DPM do not simply compare the whole status-byte for an expected value. Do not:
if (sdpm_status!=0x11) {
    // error handling
}
AND-mask bits before compare:
#define SDPM_AND_MASK 0x1F
if ((sdpm_status & SDPM_AND_MASK)!=0x11) {
    // error handling
}
```

Serial DPM status

Bits	Name	Description	R/W	Default
7:5	-	reserved (for DPM-SPI read data will be unpredictable)	R	0x0
4	SEL_DPM_SERIAL	DPM_MODE configuration input state. 0: DPM is in parallel mode (DPM_MODE configuration input is low). 1: DPM is in serial mode (DPM_MODE configuration input is high).	R	0x0
3	RDY_TO_ERR	DPM_RDY timeout error status flag. This error may occur if the host device tries to access a permanently busy netX address area (e.g. netX xPEC program RAM while xPEC is running). To avoid the stalling of the host device, the DPM_RDY signal is released to ready state after 2048 system clock cycles (i.e. 20.48 us) at least. 1: The last access to netX busy address was aborted to prevent the host device from stalling. 0: Access was finished successfully by DPM_RDY setting to ready state.	R	0x0
2	WR_ERR	DPM write error status flag. Write errors will occur if the host device ignores the ready signal (DPM_RDY) and the external DPM write access is terminated before the data could be stored. In some cases certain netX address areas may be busy for an unpredictable time. If DPM_RDY is not used, check for a write error after a write access to these areas. In case of a write error, this bit is set immediately after the respective write access. Repeat the write access until no error occurs. 1: The external DPM write access was too fast to store the write data. Repeat the write access. 0: Write access is terminated without error.	R	0x0
1	RD_ERR	DPM read error status flag. Read errors occur if the host device ignores the ready signal (DPM_RDY) and the external DPM read access is terminated before read data could be set on the external DPM data bus (see also <i>t_rds</i> in register <i>dpm_timing_cfg</i>). In case of a read error, this bit is set immediately after the respective read access. Repeat the read access until no error occurs. 1: The external DPM read access was too fast. Repeat the read access. 0: Read data OK.	R	0x0
0	UNLOCKED	DPM is locked during netX power-up and boot phase. DPM access to other addresses than these DPM control address areas is not possible before this bit is set to 1. Poll for 1 after power-up or reset.	R	0x0

3.12.6 Initializing serial DPM

Serial DPM mode must be set by netX using register *hif_io_cfg* of HIF_IO_CTRL area (not inside DPM configuration window 0 or DPM register area).

The following procedure is recommended for serial DPM initialization:

1. Setup DPM-SPI on netX and host side:
 - A. From netX side (e.g. by netX ARM): Enable DPM and select serial DPM by register *hif_io_cfg* of HIF_IO_CTRL area.
 - i. *hif_io_cfg.sel_hif_dpm*: Set bit to enable DPM
 - ii. *hif_io_cfg.sel_dpm_serial*: Set bit for serial DPM
 - iii. Program clock-phase and clock-polarity by *hif_io_cfg.sel_dpm_serial_spo* and *hif_io_cfg.sel_dpm_serial_sph* according to the required SPI mode.
 - B. From netX side (e.g. by netX ARM): Make DPM configuration window 0 accessible for host.
 - i. *dpm_addr_cfg.cfg_win_addr_cfg*: Select low or high DPM configuration window 0.

- C. From host side: Configure host-netX connection for standard SPI and wait for netX DPM wake-up (e.g. by polling for a special data pattern or by signal handshaking, see section 3.11). Use DPM-SPI protocol.

Important: For fast SPI clock rates (typically above 33 MHz), early MISO generation must be enabled (bit *sdpm_miso_early* of register *dpm_timing_cfg*). When enabled, serial read data is clocked out with the rising edge of the SPI clock. This leads to better read data setup times for serial data, but may cause hold-timing problems for slow serial clocks.

Note: Initializing the DPM entirely typically requires further steps (e.g. programming address mapping, endianness or IRQs). Proceed as with parallel DPM (3.11.9.6).

3.13 Extension bus

The extension bus mode is one of the basic modes the host interface of the netX can be operated with. It is used for the direct access of external peripherals or static memory from the netX. The following list gives an overview of the features of the extension bus:

- up to 4 chip select signals with independent programmable bus width size and timing
- 8 / 16 bit data bus width
- multiplexed or non multiplexed data bus
- 25 bit address range (32 MByte) for each chip select, max. address range 4 x 32 MByte
- programmable bus timing with wait states, setup and hold times
- support for external ready or wait signal
- unused interface pins can be used as general purpose input/output pins

3.13.1 Extension bus configuration

To use the extension bus, the netX host interface must be configured for 'Extension Bus Mode' in register *DPM_ARM_IF_CFG0*. It is important to also configure each required signal line of the extension bus for host interface mode in the registers *DPM_ARM_IO_MODE0* and *DPM_ARM_IO_MODE1*. Unused extension bus signal lines may be left configured for I/O mode (e.g. unused upper address lines or data lines 15-8 when using an 8 bit device only) allowing their use as additional PIO signals.

When the extension bus is enabled, a host system cannot access the virtual DPM of the netX chip. Each extension bus memory area is configured via register 'Extension Bus Configuration Chip Select'. For a detailed register description, see reference [1].

3.13.2 Extension bus address space and netX memory allocation

The extension bus address space is accessible by netX via 4 different memory base addresses *HIF_EXTSRAM0*, *HIF_EXTSRAM1*, *HIF_EXTSRAM2* and *HIF_EXTSRAM2*, see Table 7 on page 20. ARM and xPIC CPU can access these memory areas.

3.13.3 Address and data byte steering

The extension bus logic can handle any 8, 16 or 32-bit netX access from netX to the external peripherals or memory devices. According to the bus width configuration of the extension bus an access from netX side is converted to external 8 or 16-bit access operations. For example a 32-bit access to an 8-bit device is converted to four 8-bit access operations.

Note: When performing an access to an extension bus memory location by the ARM CPU, the CPU will be stopped until the access is completed. Hence, a 32-bit access to a slow 8-bit device may have a significant impact on the netX performance and will increase interrupt latency or DMA transfer times.

The following table shows the address and data steering between netX and external devices.

Extension bus data width	Address / data byte steering			
8-bit	netX data 0-7	→	Extension bus data 0-7	ADR[1:0] = 00
	netX data 8-15		Extension bus data 0-7	ADR[1:0] = 01
	netX data 16-23		Extension bus data 0-7	ADR[1:0] = 10
	netX data 24-31		Extension bus data 0-7	ADR[1:0] = 11
16-bit (Byte access)	netX data 0-7	→	Extension bus data 0-7	ADR[1:0] = 00
	netX data 8-15		Extension bus data 8-15	ADR[1:0] = 01
	netX data 16-23		Extension bus data 0-7	ADR[1:0] = 10
	netX data 24-31		Extension bus data 8-15	ADR[1:0] = 11
16-bit (Word access)	netX data 0-15	→	Extension bus data 0-15	ADR[1:0] = 00
	netX data 16-31		Extension bus data 0-15	ADR[1:0] = 10

Table 36: Address and data byte steering

3.13.4 Intel / Motorola data format

The netX system works with Intel memory format (little endian). There is no special mode for connection to Motorola type (big endian) memory systems. Any necessary data conversions must be performed by software when accessing big endian components.

3.13.5 Multiplexed / Non-multiplexed data bus

Apart from devices with a separate address and data bus, the extension bus also supports devices with a multiplexed data bus. The timing and polarity of the address latch enable signal and data hold times can be programmed in a wide range.

3.13.6 Data ready or data acknowledge

When the external wait/ready function is enabled, the EXT_RDY signal is sampled at the rising system clock edge during the active access. It must be set at least 2.5 clock cycles before the rising edge of read or write strobe signal to add wait states to the current cycle. The polarity of the signal can be programmed supporting devices with ready signal as well as devices with wait signal generation.

3.13.7 End-of-cycle

When the end of the access cycle specified by the Trdwrcyc parameter is reached, the cycle will be stopped immediately and all active signals will be deactivated.

Users should be careful when setting timing parameters because the versatility of the interface even allows settings that do not make sense. If, for example, the write delay time Twron exceeds the complete cycle time specified by the Trdwrcyc parameter, signal EXT_WRn will never be set!

3.13.8 Pin and signal description of extension bus

The following table lists all extension bus signals.

Pin name	Intel	Motorola	Description
EXT_D0-15	DATA[15:0]	DATA[15:0]	Extension bus data 0-15 Bi-directional data bus. When an 8 bit data bus is selected the high byte of the data bus can be used as programmable input / output pins.
EXT_A0-24	ADDR[24:0]	ADDR[24:0]	Extension bus address 0-24 25-bit address bus. Each address line can be enabled or disabled individually. Higher address lines can thus be used as programmable input/output pins if the complete address range is not required. The address line EXT_A0 is low when an even byte or word is accessed. It will be active for 8-bit and 16-bit data bus width.
EXT_CS0n EXT_CS1n EXT_CS2n EXT_CS3n	CS0n CS1n CS2n CS3n	CS0n CS1n CS2n CS3n	Extension bus chip select 0 Extension bus chip select 1 Extension bus chip select 2 Extension bus chip select 3 External chip selects for extension bus. These signals are decoded using internal address lines EXT_A27 and EXT_A28.
EXT_ALE	ALE	ASn	Extension bus address latch enable Address latch enable used for multiplexed address/data-bus access. If active, it indicates a valid address driven on the data bus. Used for Intel and Motorola multiplexed bus modes.
EXT_BHEn	BHEn	BHEn	Extension bus high enable Byte High Enable. This signal is activated when an odd byte or a word is accessed. It will be active for 8-bit and 16-bit data width.
EXT_RDn	RDn	RD/WRn	Extension bus read Active low read signal can be also configured as a direction signal EXT_RD/WRn for Motorola type interfaces.
EXT_WRLn	WRLn	DSLn	Extension bus write low Active low write or write low signal, depending on configuration. This signal can be configured as the data strobe or low byte data strobe signal for Motorola type interfaces.
EXT_WRHn	WRHn	DSHn	Extension bus write high This is the high byte write strobe signal and can also be configured as the high byte data strobe signal for Motorola type interfaces.
EXT_RDY	WAITn	READYn	Extension bus ready External data ready or data acknowledge input for access cycle extension. The polarity of this signal can be configured. The input signal is internally connected to high level when used as input / output pin.
EXT_IRQ	IRQn	IRQn	Extension bus interrupt request External interrupt request pins, level triggered. The interrupt lines are internally connected to a low level when the pins are programmed as input/output pins.

Table 37: Extension bus interface signals

3.13.9 Extension bus component connection

For extension bus component connection, see reference [2].

3.13.10 Extension bus timing without wait states

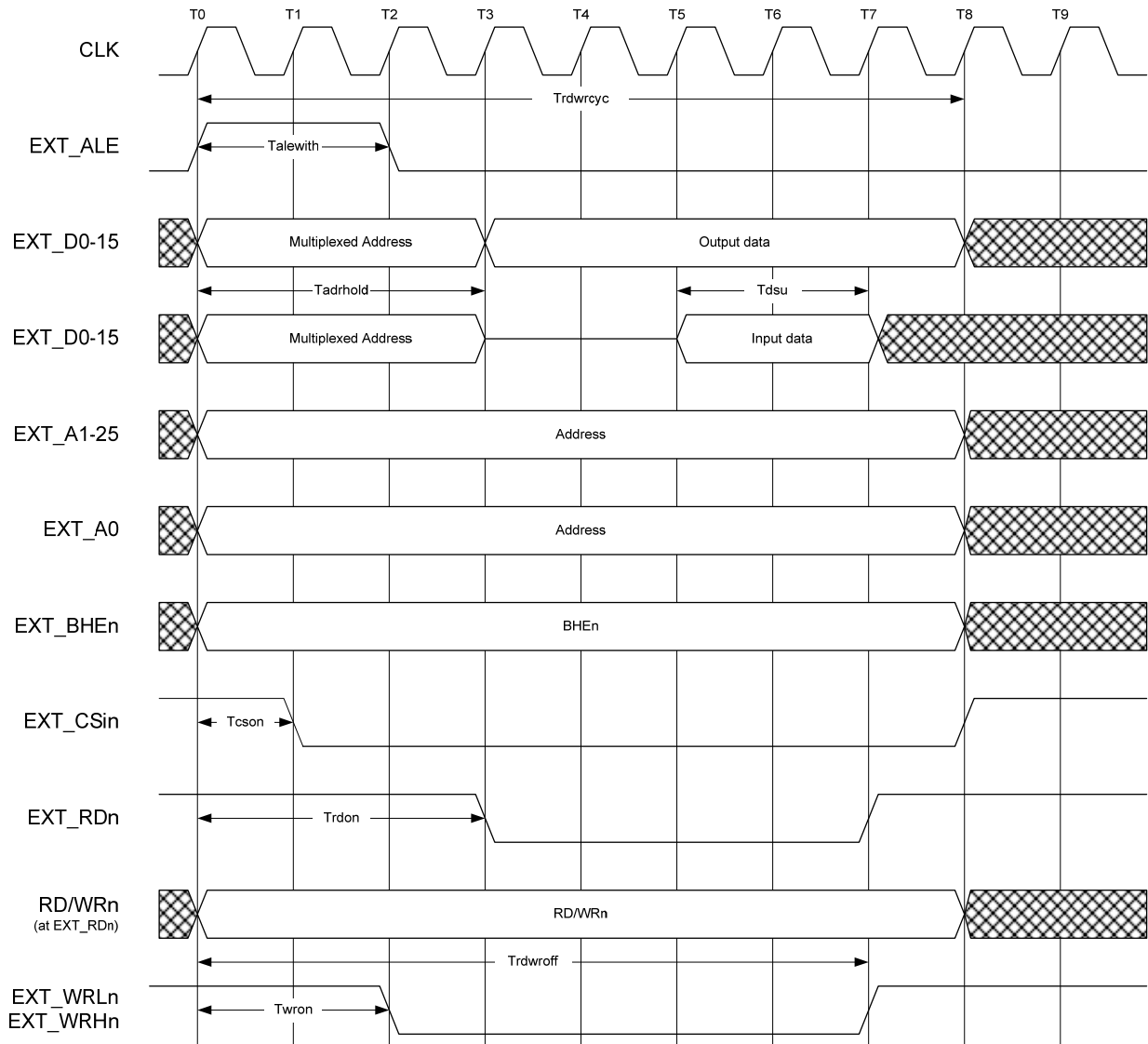


Figure 65: Extension bus timing without wait states

Parameter	Description	Value	Dimension
Talewidth	Delay time from start cycle until ALE inactive	0-7	CYC
Tdrhold	Delay time from start cycle until invalid address at the data bus.	0-7	CYC
Tcson	Delay time from start cycle until chip select insertion	0-7	CYC
Trdon	Delay time from start cycle until RD low in system clocks	0-7	CYC
Twron	Delay time from start cycle until WR low in system clocks	0-7	CYC
Trdwroff	Delay time from start cycle until RD or WR inactive delay for accesses in system clocks	0-31	CYC
Trdwrcyc	Set the end of an access cycle in system clocks The values 0x00 and 0x01 are interpreted as 0x20	2-32	CYC
Tdsu	Data setup time for read access	1	CYC

3.13.11 Extension bus timing with wait states

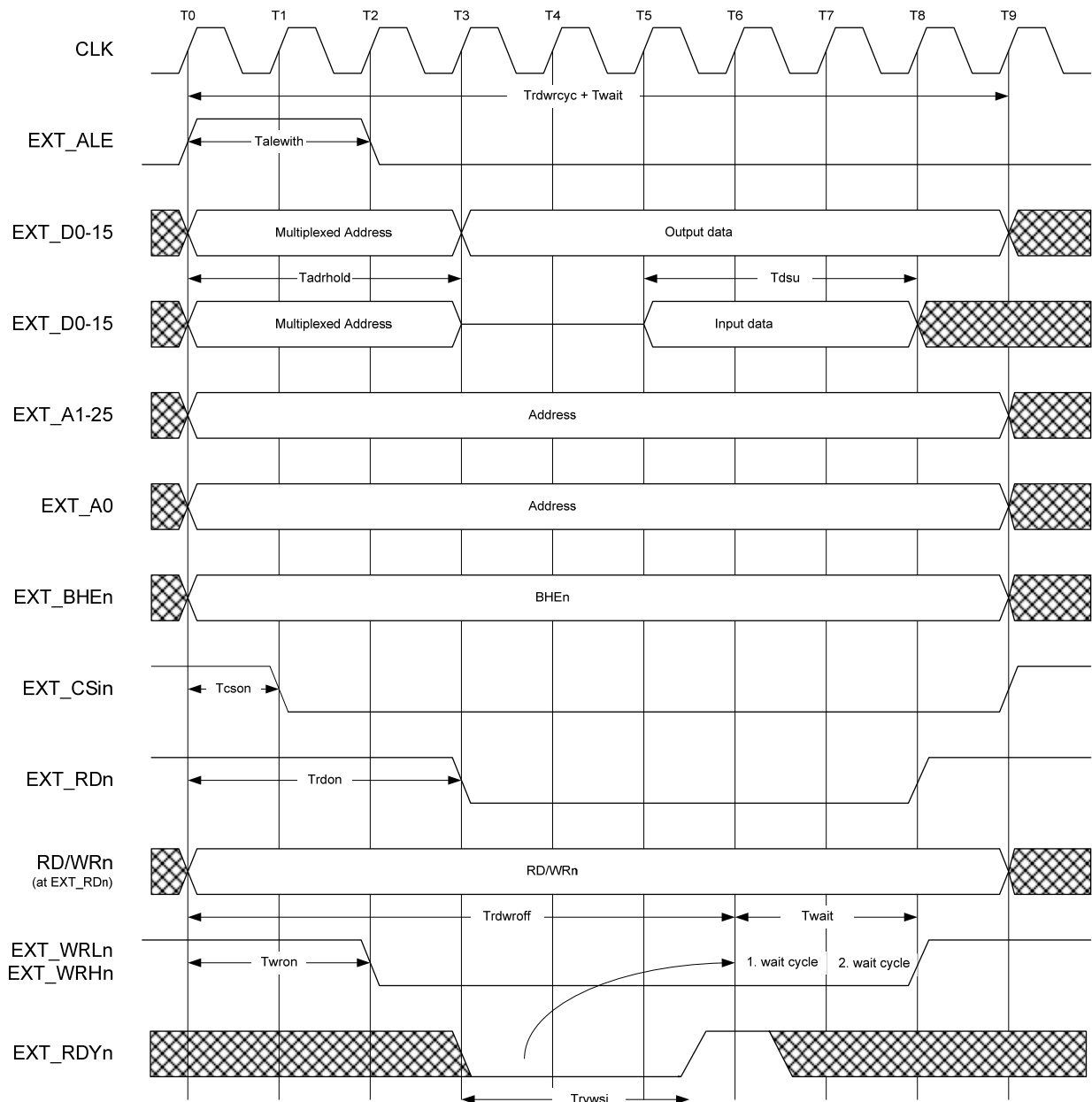


Figure 66: Extension bus timing with wait states

Parameter	Description	Value	Dimension
Trwsi	Ready valid to wait state insertion delay	2.5	CYC

3.14 Using DPM IOs as programmable IOs (PIOs)

Since netX 51/52 PIO mode for each DPM signal must be enabled explicitly by software before the IO can be used as PIO (i.e. before drivability and drive level can be programmed directly).

Note: Before netX 51/52 only few DPM modes were supported and unused DPM IOs could be directly derived from programmed DPM configuration (e.g. for 8-bit data width, data lines 8 to 15 were directly set to PIO mode). This became too complex due to the wide range of DPM modes which is supported now.

The PIO mode of DPM signals can be enabled by 2 registers (*dpm_pio_cfg0* and *1*) in DPM configuration window 0 (or in INTLOGIC-DPM area). If a signal is not in PIO mode, DPM handles it according to its function. E.g. for parallel DPM signal DPM_A7 is an address line that will always be in input mode and never be driven. In PIO mode, registers *hif_pio_out* and *hif_pio_oe* of *HIF_IO_CTRL* area control the signal behavior.

PIO mode is disabled by default for all DPM signals with the following exceptions:

- The 2 DPM-IRQ signals DPM_DIRQ and DPM_SIRQ are in PIO mode by default
- The DPM-ready signal is in PIO mode by default.

Important: Configuring a signal for PIO mode does not automatically exclude it from DPM function. E.g. when parallel DPM is used, signal DPM_A7 is always regarded as DPM address line regardless of whether it is configured as PIO or not.

Important: Configuring a signal for PIO mode which is used for DPM could cause permanent damage to netX or host if the signal is driven by netX and host. E.g. when parallel DPM is used, signal DPM_A7 is an address line which is normally always driven by the host. As far as PIO function for DPM_A7 is disabled, the signal is never driven by netX. If PIO function and signal driving were enabled by software, netX could drive the signal, but doing so could cause damage as host and netX would drive against each other.

Note: Since netX 51/52, PIO usage of DPM_SIRQ, DPM_DIRQ and DPM_RDY has moved from *dpm_io_cfg_misc* to register *dpm_pio_cfg1*.

Note: PIO programming registers are still located in the area INTLOGIC-HIF_IO_CTRL.

Example for PIO usage:

Assuming a parallel DPM with 8 kB address range (using only DPM_A0 to DPM_A12): In this case DPM_A13 to DPM_A15 can be used as PIOs because they are not required for DPM.

At first these signals must be configured for PIO use by programming the related bits of bit-field *sel_a_pio* of register *dpm_pio_cfg1* to 1. Then, the signals can be programmed using registers *hif_pio_out*, *hif_pio_oe* and *hif_pio_in* of area *HIF_IO_CTRL*.

3.15 Handshake registers

To allow the synchronization of the data transfer between host system and netX, a total of 16 handshake cell pairs with interrupt capability are available. Handshake cells can be 8 or 16-bit wide and are thus available for all selectable DPM bus widths. A cell pair is always located inside a 32-bit DWORD.

While both cells of a pair can always be read by host and netX, the upper cell of the pair can only be written by the host and the lower cell can only be written by the netX. In 8-bit mode the lower 16 bits of the 32-bit DWORD behave like a standard SRAM.

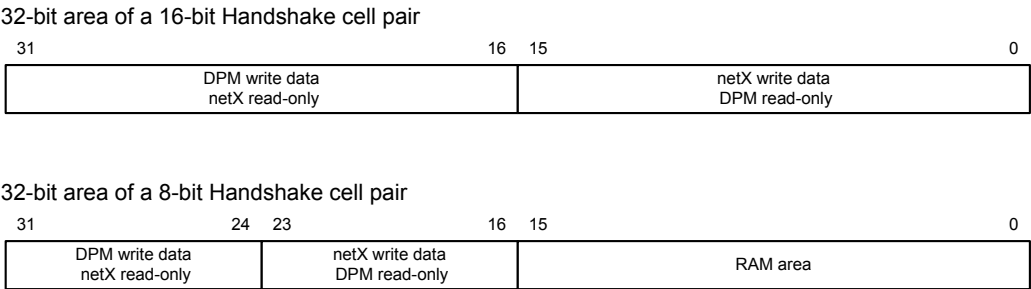


Figure 67: Handshake cell DWORD

Writing to a handshake cell from the host side can generate an interrupt to the netX side. Writing from the netX side can generate an interrupt to the host (by the signals DPM_DIRQ or DPM_SIRQ).

After receiving an interrupt the processor (host CPU resp. netX) will read the changed data of the IRQ generating cell. This will automatically clear the IRQ.

Before reading the changed data, the IRQ generating cell must be determined by the status registers of the HANDSHAKE_CTRL unit or for host CPUs additionally by the registers *firmware_irq* of DPM configuration and status window 0. An interrupt vector and a status flag for each handshake cell are available there.

Note:

Managing the handshake cell IRQs by the *firmware_irq* registers of DPM configuration and status window 0 is compatible with netX 50. All handshake cell management of the netX 100/50 global control block can be found there bit-compatible. When using these registers a host needs no direct access to the HANDSHAKE_CTRL unit.

Note:

Instead of reading the cell, a handshake IRQ can also be cleared by writing ‘1’ to the flag bit.

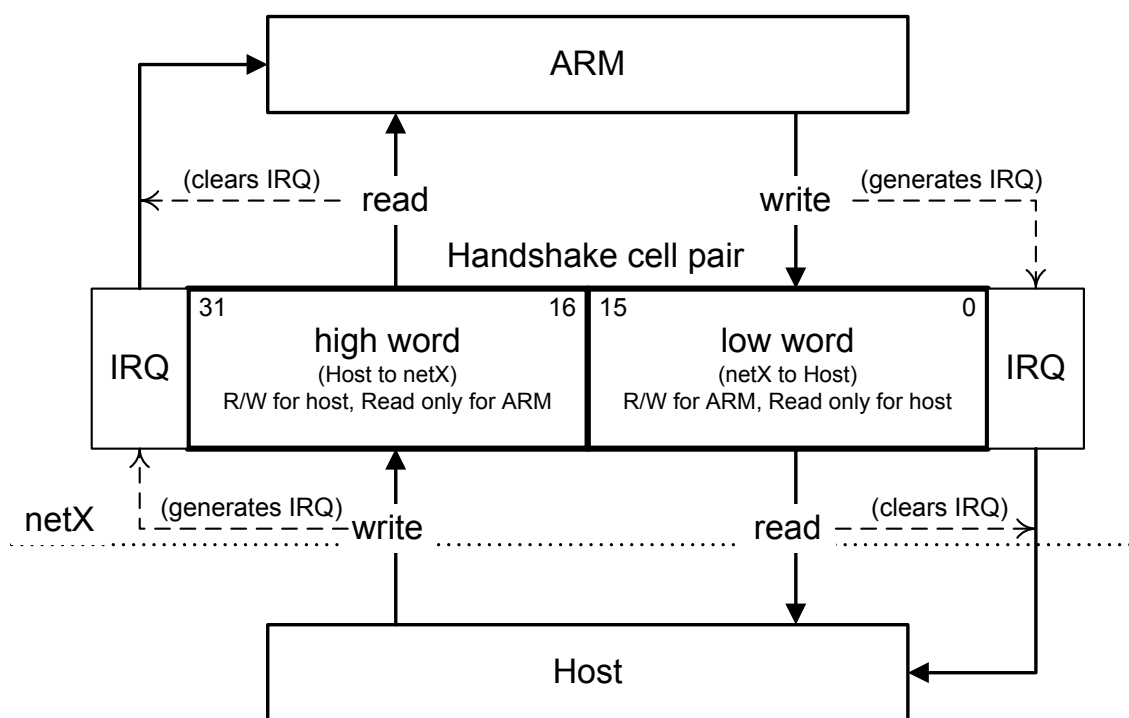


Figure 68: Handshake cell dataflow (16-bit)

Handshake cells must be configured in the HANDSHAKE_CTRL unit. Each handshake cell pair can be enabled individually. The width (8 or 16-bit) can be programmed for each pair separately.

Important: There is a difference to netX 50: For netX 50 each handshake cells pair could be mapped individually inside the external DPM address range. This is changed: The 16 DWORDS of the handshake cell pairs can only be mapped address-incrementingly to a single 64 byte block now.

Note: There is no default handshake cell mapping after a power-on reset, i.e. the netX firmware must configure the desired handshake cells before they can be used.

The handshake cells are physically located in the internal netX memory INTRAMHS (no longer in special registers). INTRAMHS (32 KB for netX 51/52) is dedicated to the data exchange between netX and host. Due to this, INTRAMHS is typically mapped to external DPM address space and available for a host. The location of the 16 handshake cell pairs inside INTRAMHS can be programmed to any 256 byte offset (bit-field *base256* of register *handshake_base_addr* of HANDSHAKE_CTRL area).

The address offset inside INTRAMHS for handshake cell *N* is:

8-bit cell pair	Host → netX: handshake cell offset = $256 * base256 + 4 * N + 3$
	netX → Host: handshake cell offset = $256 * base256 + 4 * N + 2$
16-bit cell pair	Host → netX: handshake cell offset = $256 * base256 + 4 * N + 2$
	netX → Host: handshake cell offset = $256 * base256 + 4 * N + 0$

Note: If a cell pair is not enabled, the related DWORD behaves like a standard SRAM. All bytes can be written by ARM and host then.

Note: An 8-bit or serial DPM typically requires 8-bit handshake cells as otherwise two access operations are needed to read or write 16-bit cells. However, if Read-Data-Latching

(RDL) and Write-Byte-Collecting (WBC) of the DPM module are used, 8-bit or serial DPM can also behave like a 32-bit interface.

Note: With netX 100/50 the overlapping of handshake cells was possible and required a priority order, but this is obsolete since all handshake cells are en-block now.

Handshake IRQs are set and cleared by a special logic which monitors any access to INTRAMHS. If a write access to the address of an enabled cell is detected, an IRQ will be generated. If a read access is detected, an IRQ will be cleared.

The selection of the correct IRQ target for setting or clearing an IRQ is not directly derived from the component (ARM or host) which initiated an access. Instead of deriving it directly, there are several mirrors of INTRAMHS netX inside, at least one for each IRQ target. An access to INTRAMHS via one of these mirrors behaves like the related component which initiated the access.

Important: The IRQ target of the handshake cell access is derived only from the INTRAMHS mirror used. E.g. if ARM uses a DPM mirror of INTRAMHS, IRQs will be generated as if DPM initiated the access. If DPM uses an ARM mirror, IRQs behave as if ARM initiated the access. This could be used during development e.g. to test host functions on the ARM. Usually the ARM should use ARM mirrors and the host should use DPM mirrors for handshake cell access.

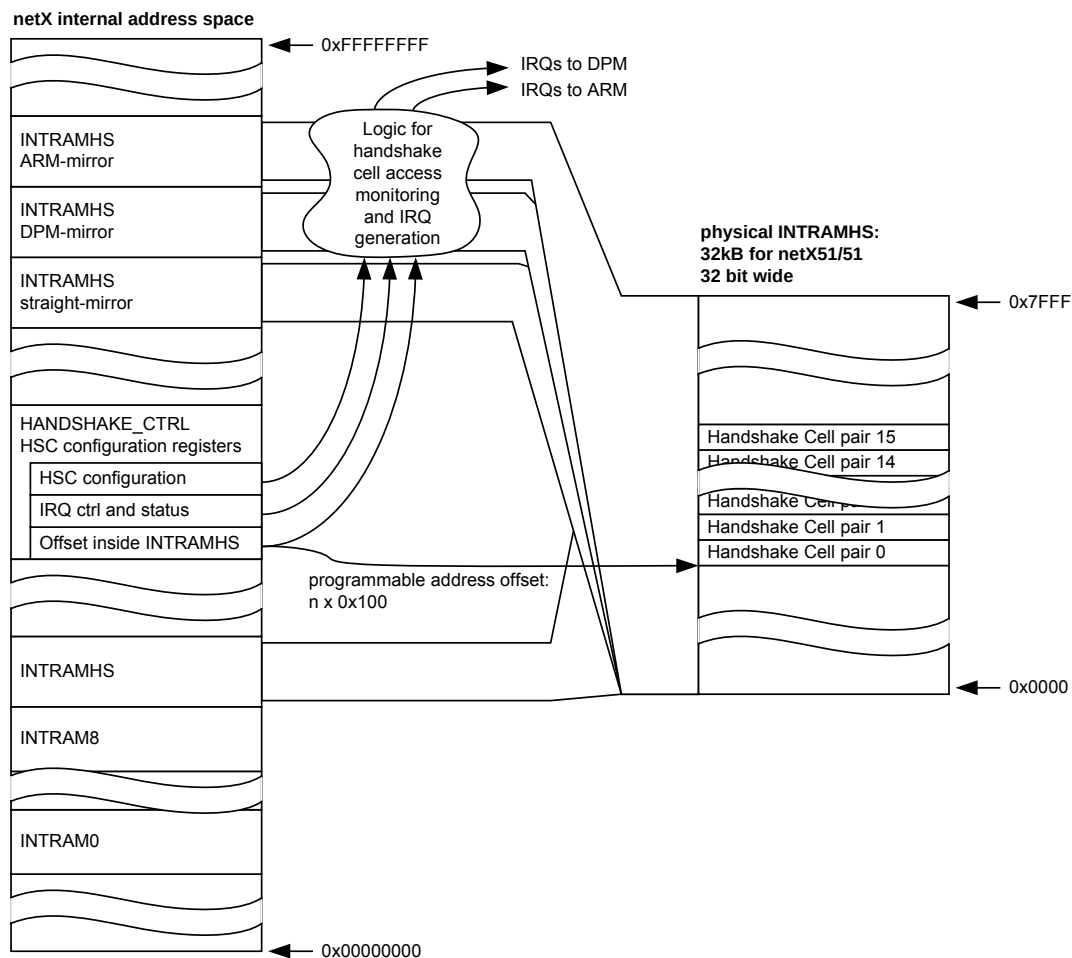


Figure 69: Handshake cell monitoring of INTRAMHS

3.15.1 Mirrors of handshake cell memory INTRAMHS

The following table shows all address mirrors of INTRAMHS.

Important: Physically there is only one INTRAMHS with 32 KB. Writing to the same offset via a mirror (no matter which one) will overwrite the corresponding cell of the physical INTRAM.

Important: Several INTRAMHS mirrors are not assigned to any IRQ target. In that case, INTRAMHS will completely behave like a standard memory, i.e. there are no read-only bytes (even if handshake cells are enabled) and there is no further access monitoring when these mirrors are used.

INTRAMHS mirrors	Start address	End address	Assigned IRQ
intramhs_mirror_itcm	0x000A0000	0x000A7FFF	-
intramhs_mirror_dtcmm	0x040A0000	0x040A7FFF	-
intramhs	0x080A0000	0x080A7FFF	-
intramhs_buf_straight_mirror	0x0A000000	0x0A007FFF	-
intramhs_buf_dpm_mirror	0x0A008000	0x0A00FFFF	DPM/host
intramhs_buf_arm_mirror	0x0A010000	0x0A017FFF	ARM
intramhs_buf_xpic_mirror	0x0A018000	0x0A01FFFF	reserved
intramhs_straight_mirror	0x1A000000	0x1A007FFF	-
intramhs_dpm_mirror	0x1A008000	0x1A00FFFF	DPM/host
intramhs_arm_mirror	0x1A010000	0x1A017FFF	ARM
intramhs_xpic_mirror	0x1A018000	0x1A01FFFF	reserved
intramhs_mirror_hi	0xFFEA0000	0xFFEA7FFF	-

Note: Using *intramhs_arm_mirror* for netX ARM is recommended. Access via *intramhs_buf_arm_mirror* could use ARM write buffering and thus lead to undesired behavior. For DPM there is no difference between using *intramhs_dpm_mirror* or *intramhs_buf_dpm_mirror*.

3.15.2 DPM window mapping for handshake cells

To provide handshaking with a host, the part of INTRAMHS which contains the handshake cells must be reachable for the host. Using *intramhs_dpm_mirror* is therefore recommended.

Note: The handshake cell block can be mapped to 128-byte boundaries for a host by DPM window mapping (although the cells can be mapped only to 256-byte boundaries inside INTRAMHS). Using DPM access tunnel even allows mapping to 64-byte address boundaries. Since netX 51/52, all 16 DWORDS can be reached by the DPM access tunnel.

DPM window setup example:

Only 3 of the 4 programmable DPM data windows are used in this example. The external address range is 16 kB (from 0x0 to 0x3FFF):

Window 1 provides access to 4864 (0x1300) bytes of general purpose memory area. The handshake cells shall be located from 0x200 (pair 0) to 0x240 (pair 15) inside this window. The

external address range of this window is 0x0 to 0x12FF. The internal target of this window shall be the first 4864 bytes of INTRAMHS (offset 0x0 to 0x12FF).

Window 2 is dedicated to data from host to netX and has 5632 (0x1600) bytes. The external address range of this window is 0x1300 to 0x28FF. Internally there shall be 3 memory areas of 5632 bytes each. Each area is a buffer of netX output data. A buffer can be updated by changing the mapping of DPM window 2 from one area to another (which allows handling the other areas in the background). All 3 buffers shall also be located inside INTRAMHS: buffer 0 from offset 0x1300 to 0x28FF, buffer 1 from offset 0x2900 to 0x3EFF and buffer 2 from offset 0x3F00 to 0x54FF. After DPM initialization, buffer 0 shall be mapped.

Window 3 is dedicated to data from netX to host and has also 5632 (0x1600) bytes. The external address range of this window is 0x2900 to 0x3EFF. There shall be only one netX-internal memory area for this window. The offset from 0x5500 to 0x6AFF inside INTRAMHS shall be used (INTRAMHS is too small for 3 further buffers, but using another INTRAM would be possible).

The last 256 bytes of the external address range cover DPM configuration and status window.

DPM window n	win 0	win 1(data)	win 2 (data)	win 3 (data)	win 4 (data)
usage	handshake cell status	INTRAMHS general	INTRAMHS outbuf0..2	INTRAMHS inbuf	disabled
size (bytes)	256 ⁽¹⁾	4864	5632	5632	0
ext. start addr.	0x3F00 ⁽¹⁾	0x0	0x1300	0x2900	-
last ext. addr.	0x3FFF ⁽¹⁾	0x12FF	0x28FF	0x3EFF	-
window end addr.	0x4000 ⁽¹⁾	0x1300	0x2900	0x3F00	0x0
netx addr. start	-	0x1A008000	0x1A009300	0x1A00D500	-
netx page addr.	-	0x1A000000	0x1A000000	0x1A000000	-
win. map value	-	0x08000	0x08000	0x0AC00	-
read byte area	yes ⁽¹⁾	yes	yes	yes	-
write byte area	yes ⁽¹⁾	yes	yes	yes	-
Read ahead	no ⁽¹⁾	no	no	no	-
win*_end reg. value	n.a.	0x00001300	0x00002900	0x00003F00	0x00000000
win*_map reg. value	n.a.	0x1A008011	0x1A008011	0x1A00AC11	0x00000000

Note

1. Fix configuration window 0 ends at 0x100. There is neither an end address register, nor a mapping register. Location is programmed in register *dpm_adr_cfg*.

To locate the handshake cells from 0x200 to 0x240 of the external address space, map them to the address offset 0x200 in INTRAMHS by programming bit-field *base256* of register *handshake_base_addr* of HANDSHAKE_CTRL area to 0x2. Moreover, activate the *enable* bit in this register. Thereafter the single handshake cells can be configured by registers *handshake_hsc0_ctrl* to *handshake_hsc15_ctrl*.

3.16 IEEE 1588 system time

The precision system time derives from a counter clocked with the 100 MHz system clock. From the view of the application it has a 10 ns resolution. Due to drift, aging or failure of the crystal, this time can differ from a system-wide master clock which is very often needed in real-time Ethernet system.

The system time is not realized by a standard counter. It uses an adder which increases the current time value by a programmable number (nominally 10) every clock period. If the 100 MHz clock deviates from the master clock, the added value will slightly differ by 10 with a resolution of 2^{-28} ns to compensate the deviation. This can be calculated based on the protocol of IEEE 1588 or other real-time Ethernet functions.

The system time is provided in two 32-bit registers. Register 1 represents the seconds, register 2 the nanoseconds from time zero. The application has to read the seconds value first because this will freeze the nanoseconds register to get a consistent system time.

netX 51/52 offers 2 timers.

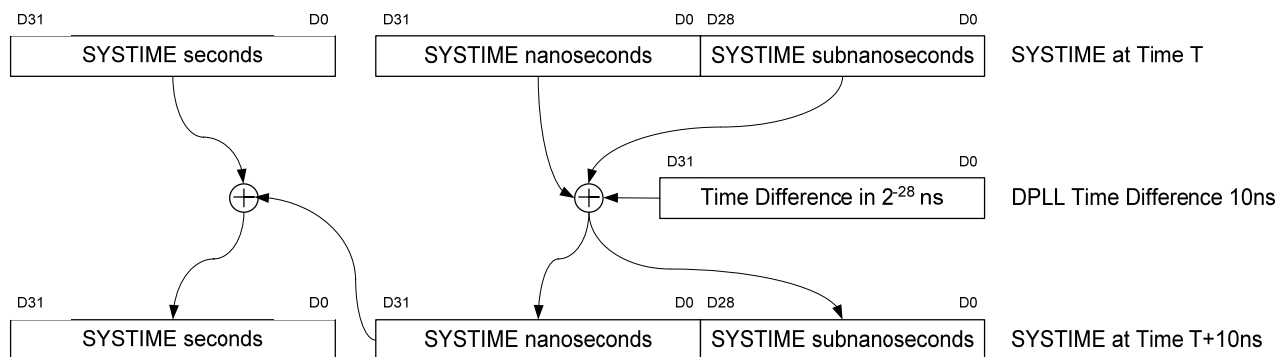


Figure 70: System time calculation

The following diagram shows how the time clock compensation works. With a clock period of $\Delta T = 10$ ns the value $\Delta \text{CNT} = 10$ will be added continuously to the system time CNT1 to reach CNT 2 exactly at T2. If the clock runs too fast, CNT2 will be reached after $T2_{\text{fast}}$. If the clock is too slow, CNT2 will be reached after $T2_{\text{slow}}$. If ΔCNT is calculated exactly, CNT2 will be reached at T2. The procedure of an ongoing correction prevents the problems of a one-step correction resulting in a large step of the system time.

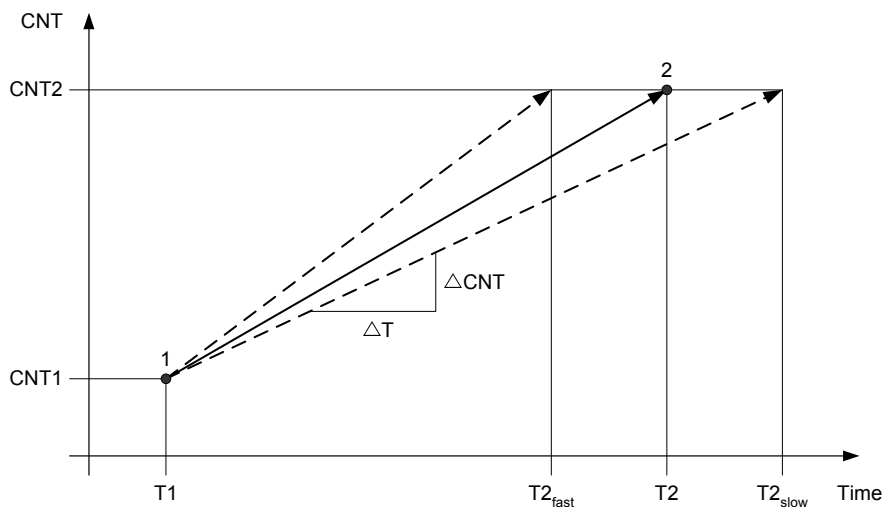


Figure 71: Ongoing correction of time failure

3.17 JTAG and ETM debug interface

3.17.1 JTAG interface

The netX debug interface is based on the Joint Test Action Group (JTAG) IEEE Standard 1149.1 and supports debugging tools compliant with this standard. It provides two different modes of operation: **ARM debug mode** and **boundary scan mode**. By default, the netX JTAG interface operates in ARM debug mode passing all JTAG signals to the integrated ARM CPU. For information on how to activate the boundary scan mode, see section *Boundary scan mode* on page 154.

Two different reset signals are involved when using the netX JTAG interface:

- **nSRST** Open collector output from the ICE to the target system reset. This is also an input to the ICE so that a reset initiated on the target can be reported to the debugger.
- **nTRST** Open collector output from the ICE to the reset signal on the netX JTAG port.

For a diagram which shows how to connect the standard JTAG connector or a Hilscher “mini-JTAG” connector, see reference [2].

3.17.1.1 Boundary scan mode

In addition to the (default) ARM debug mode, the netX JTAG interface also supports a boundary scan mode. This mode allows the user to run tests on the netX with the appropriate (third-party-) tools. To activate the boundary scan mode, refer to the following table indicating the required state of certain netX signals:

Signal	netX 51 pin	netX 52 pin	State
BSCAN_TRST	G5	E3	high

3.17.2 ETM - Embedded trace macrocell

The ETM is a real-time trace module capable of instruction and data tracing. The ETM is an integral part of the ARM microcontroller and works with special debug tools like the Hitex Tool chain for ARM. The ETM comprises the following main components:

Trace port Output signals that help you understand the operation of the processor.

Triggering and filtering facilities Extensible specifications enable you to control tracing by specifying the exact set of triggering and filtering resources required for a particular application. Resources include address comparators and data comparators, counters and sequencers.

The netX contains the ETM9 Rev 2a (ETM Architecture ETMv1.3) in medium configuration.

Note: For netX 51, ETM signals are shared signals and can be mapped to MMIO signals. If ETM is used, no other signals can be used at the same time on the MMIO pin.

Note: netX 52 can be used with JTAG only.

A connector and a signal table are documented in reference [2].

3.18 Vectored interrupt controller

The Vectored Interrupt Controller (VIC) supports 32 interrupt sources, 16 can be vectored. Priority and type of interrupt, IRQ or Fast IRQ, are configurable. All interrupts can be masked.

Some of the interrupts represent the result of a logical OR of up to 32 single interrupts of a function block. Hence, the ISR may have to check further registers to determine the actual source of an interrupt (e.g. resolving the GPIO interrupt to the GPIO input that caused it).

The interrupt sources are listed in Table 35 on page 108.

The VIC provides a software interface to the interrupt system. An ARM system has two interrupt levels:

1. Fast Interrupt Request (FIQ) for fast, low latency interrupt handling
2. Interrupt Request (IRQ) for more general interrupts.

Generally, only one single FIQ source is used at a time in a system, to provide a true low-latency interrupt. This has the following advantages:

1. The ISR can be executed directly without having to determine the interrupt source.
2. Interrupt latency is reduced. The banked registers available for FIQ interrupts can be used more efficiently because saving the context is not required.

There are 32 interrupt lines. The VIC uses one bit position for each different interrupt source. The software can control each request line to generate software interrupts.

There are 16 vectored interrupts which can only generate an IRQ interrupt. The vectored and non-vectored IRQ interrupts provide an address for an ISR. Reading from the vector interrupt address register, VICVectAddr, provides the ISR address and updates the interrupt priority hardware that masks out the current and any lower priority interrupt requests. Writing to the VICVectAddr register indicates to the interrupt priority hardware that the current interrupt is serviced allowing lower priority interrupts to become active.

The FIQ interrupt has the highest priority followed by interrupt vector 0 to interrupt vector 15. Non-vectored IRQ interrupts have the lowest priority. A programmed interrupt request allows you to generate an interrupt under software control. This register is typically used to downgrade an FIQ interrupt to an IRQ interrupt.

The block diagram on the following page shows an overview of the VIC.

Note: The ARM gives FIQ priority over IRQ. The VIC can trigger an FIQ and an IRQ at the same time.

Since the VIC is compatible to ARMPrimeCell VIC (PL190), consult the appropriate ARM documentation.

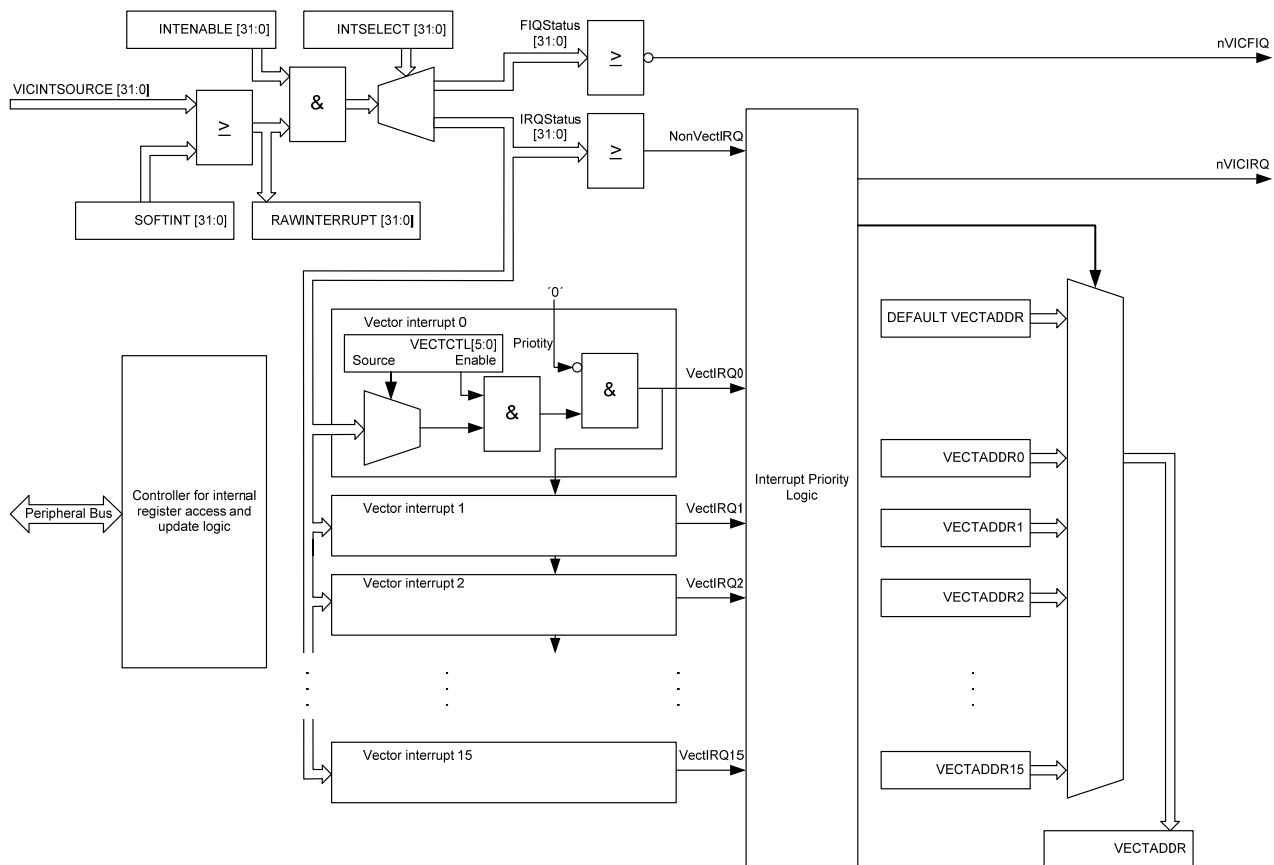


Figure 72: Block diagram of the vectored interrupt controller

3.18.1 Interrupt generation

Interrupt request generation

For generation of FIQStatus[31:0] and IRQStatus[31:0], the interrupt requests from the peripherals are received and combined with the software interrupt requests. Then any undesired interrupt requests are masked out and the results are either routed to FIQStatus[31:0] or IRQStatus[31:0] (see block diagram).

Non-vectored FIQ interrupt (nVICFIQ) generation

By combining FIQStatus[31:0] (see block diagram) the non-vectored FIQ (nVICFIQ) which is connected to the ARM CPU, is generated.

Non-vectored IRQ interrupt generation

By combining IRQStatus[31:0] (see block diagram) the non-vectored IRQ is generated. This signal is used as input of the interrupt priority logic.

Vectored interrupt generation

There are 16 vectored interrupt blocks which generate 16 vectored interrupt signals (VectIRQ0-15). The vectored interrupt blocks receive the IRQStatus[31:0] (interrupt requests) and set the VectIRQx if the following conditions are met:

- the selected interrupt is active
- the selected interrupt currently is the highest requesting interrupt
- the selected interrupt is enabled in the vector control register (VICIntCntl[0-15])

Software interrupts

The software can control the source interrupt lines to generate software interrupts. These interrupts are generated before interrupt masking in the same way as external source interrupts. Software interrupts are cleared by writing to the software interrupt clear register, VICSoftIntClear (see Software interrupt clear register, VICSoftIntClear). Normally, this occurs at the end of the ISR.

3.18.2 Interrupt priority logic

The interrupt priority block prioritizes the following requests:

- Non-vectored interrupt requests
- vectored interrupt requests

The highest-priority request generates an IRQ interrupt if the interrupt is currently not serviced. The FIQ interrupt has the highest priority (outside the interrupt priority logic) followed by interrupt vector 0 to interrupt vector 15. Non-vectored IRQ interrupts have the lowest priority.

3.18.3 Interrupt flow sequence

Vectored interrupt flow sequence:

The following procedure shows the sequence for the vectored interrupt flow:

- An interrupt occurs.
- The ARM processor jumps to the IRQ or to the FIQ interrupt vector.
- If the interrupt is an IRQ, read the VICVectAddr register and jumps to the ISR using an LDR PC instruction. Reading the VICVectorAddr register updates the hardware priority register of the interrupt controller.
- Stack the workspace so that IRQ interrupts can be re-enabled.
- Enable the IRQ interrupts so that a higher priority can be serviced.
- Execute the ISR.
- Clear the requesting interrupt in the peripheral or write to the VICSoftIntClear register if the request was generated by a software interrupt.
- Disable the interrupts and restore the workspace.
- Write to the VICVectAddr register to clear the respective interrupt in the internal interrupt priority hardware.
- Return from the interrupt to re-enable the interrupts.

Simple interrupt flow:

The following procedure shows how you can use the interrupt controller without using vectored interrupts or the interrupt priority hardware. For example, you can use it for debugging.

- An interrupt occurs.
- Jump to IRQ or FIQ interrupt vector.
- Jump to the interrupt handler.
- Interrogate the VICIRQ Status register to determine which source generated the interrupt.
- Compute a number of instructions to prioritize the interrupts if there are multiple active interrupt sources.
- Jump to and execute the correct ISR.

- Clear the interrupt. If the request was generated by a software interrupt, write to register VICSoftIntClear. Check status register VICIRQ to make sure that no other interrupt is active. If there is an active request, go to step 4 (Interrogate ...).
- Return from the interrupt.

Note: If the simple flow is used, you need not read or write to register VICVectorAddr.

3.19 DMA controller

The netX 51/52 is equipped with a DMA controller that provides the following features:

- Compatible with ARM DMAC software and register
- 1 AHBL (32-bit) master port for DMA transfer and list operations
- 1 AHBL (32-bit) slave port for programming interface
- 4 DMA channels with separated linked lists
- 4 Dword (32-bit) FIFO per channel
- Linked list operation support on each channel
- Incrementing or non-incrementing addressing for source and destination (support FIFO read and write).
- Software programmable DMA channel priority strategy. Hardware priority (0 highest, 3 lowest) or priority lists (last served channel gets new lowest priority).
- Programmable burst size
- Memory-to-memory, memory-to-peripheral, peripheral-to-memory and peripheral-to-peripheral DMA transfers.
- DMAC or peripheral flow control. Support peripheral DMA flow control signals (request, last burst).
- Error and finish interrupt generation
- Interrupt masking, clear interrupt
- 32-, 16- and 8-bit support for source and destination in all combinations.
- Each DMA channel has a specific hardware priority.
DMA channel 0 has the highest, channel 3 the lowest priority.
- Programmable interrupt capabilities

3.19.1 Functional description

The following section provides a detailed description of the DMA controller and its features.

The 3-channel DMA controller supports the following transactions in netX 51/52:

- Peripheral-to-memory transfer
- Memory-to-peripheral transfer
- Peripheral-to-peripheral transfer
- Memory-to-memory transfer

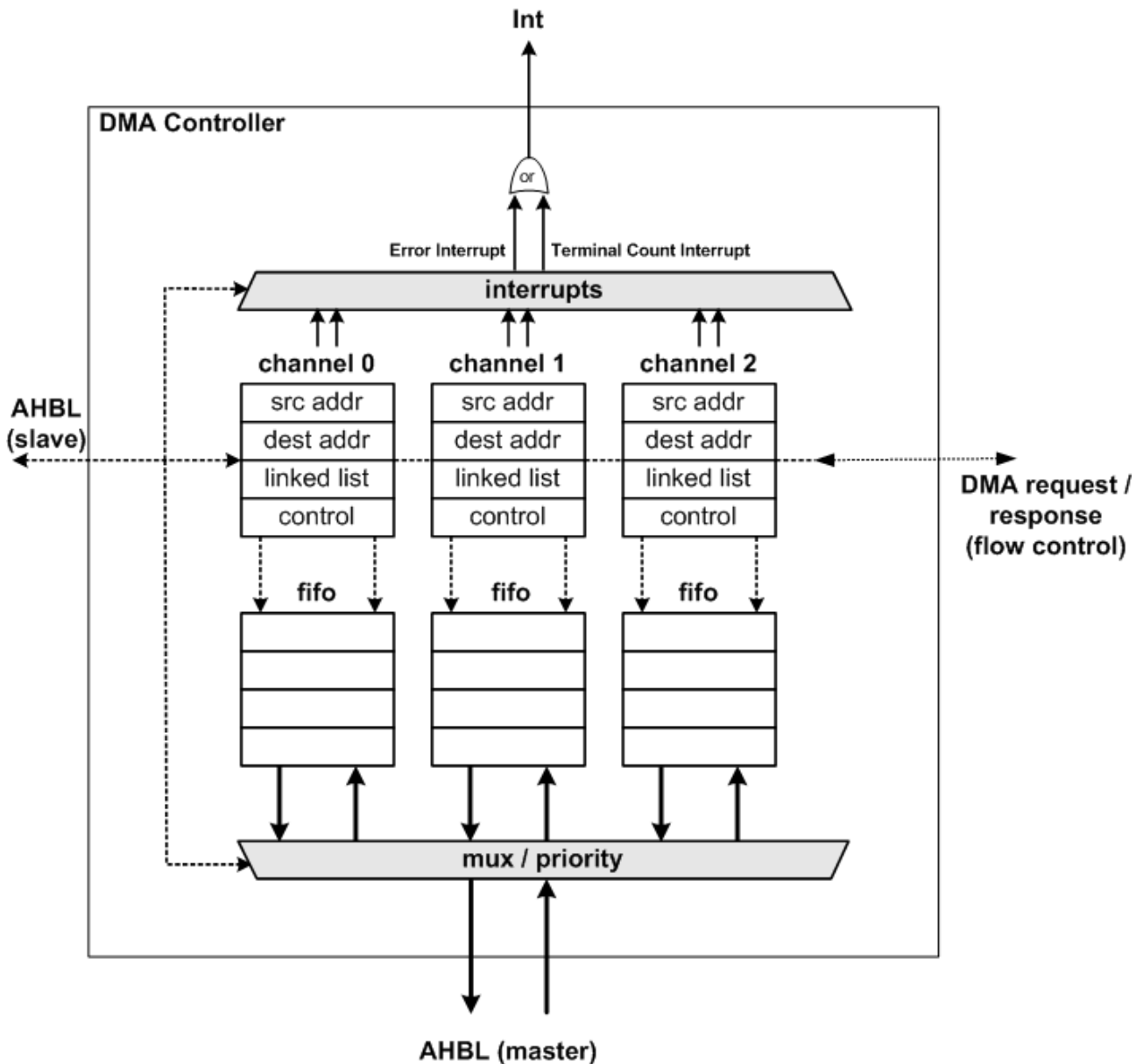


Figure 73: DMA controller

Each channel supports a unidirectional up to 32-bit DMA transfer for a single source and destination address. A bidirectional transfer therefore requires one stream for transmit and one for receive. The source and destination address can be a memory region or a peripheral device of the netX 51/52.

The default bus width is 32-bit. Source and destination of transfers can have different widths, and can be the same width or narrower than the physical bus width. The DMA controller packs or unpacks data according to the programmed parameters.

The DMA controller supports little endian addressing only. Internally, the DMAC treats all data as a stream of bytes instead of 16-bit or 32-bit quantities.

Note: To avoid byte swapping of data, always address peripheral interfaces in 32-bit mode.

Source width	Destination width	Source transfer	Source data	Destination transfer	Destination data
8	8	1/[7:0] 2/[15:8] 3/[23:16] 4/[31:24]	21 43 65 87	1/[7:0] 2/[15:8] 3/[23:16] 4/[31:24]	21212121 43434343 65656565 87878787
8	16	1/[7:0] 2/[15:8] 3/[23:16] 4/[31:24]	21 43 65 87	1/[15:0] 2/[31:16]	43214321 87658765
8	32	1/[7:0] 2/[15:8] 3/[23:16] 4/[31:24]	21 43 65 87	1/[31:0]	87654321
16	8	1/[7:0] 1/[15:8] 2/[23:16] 2/[31:24]	21 43 65 87	1/[7:0] 2/[15:8] 3/[23:16] 4/[31:24]	21212121 43434343 65656565 87878787
16	16	1/[7:0] 1/[15:8] 2/[23:16] 2/[31:24]	21 43 65 87	1/[15:0] 2/[31:16]	43214321 87658765
16	32	1/[7:0] 1/[15:8] 2/[23:16] 2/[31:24]	21 43 65 87	1/[31:0]	87654321
32	8	1/[7:0] 1/[15:8] 1/[23:16] 1/[31:24]	21 43 65 87	1/[7:0] 2/[15:8] 3/[23:16] 4/[31:24]	21212121 43434343 65656565 87878787
32	16	1/[7:0] 1/[15:8] 1/[23:16] 1/[31:24]	21 43 65 87	1/[15:0] 2/[31:16]	43214321 87658765
32	32	1/[7:0] 1/[15:8] 1/[23:16] 1/[31:24]	21 43 65 87	1/[31:0]	87654321

Table 38: DMA controllers data packing or unpacking depending on the programmed mode

The DMA channel priority is fixed. DMA channel 0 has the highest, channel 3 the lowest priority. If the DMA controller is transferring data for the lower priority channel and afterwards the higher priority channel becomes active: the DMA controller completes the number of transfers (delegated to the master interface by the lower priority channel) before it switches over to transfer data for the higher priority channel

The netX 51/52 has 4 AHB masters in total:

- 1 ARM966
- 1 HIF
- 1 DMA controller
- 1 XC unit

Due to the netX 51/52 bus matrix, all masters can operate in parallel, unless shared resources are used. Should these masters come in conflict by accessing the same resources (e.g. external memory) the bus matrix solves this conflict by a fix priority for each master. The priority sequence (highest to lowest) is as follows:

1. HIF
2. XC unit
3. ARM966
4. DMA controller

To reduce latency and to improve the DMA transfer performance, observe the following recommendations:

- Reduce conflicts a priori by separating software of the system processors (ARM966 and XC) running in different memory areas.
- If feasible, use separate memory areas for data storage and linked list information.
- All memory transactions should be 32-bit wide to improve bus efficiency.

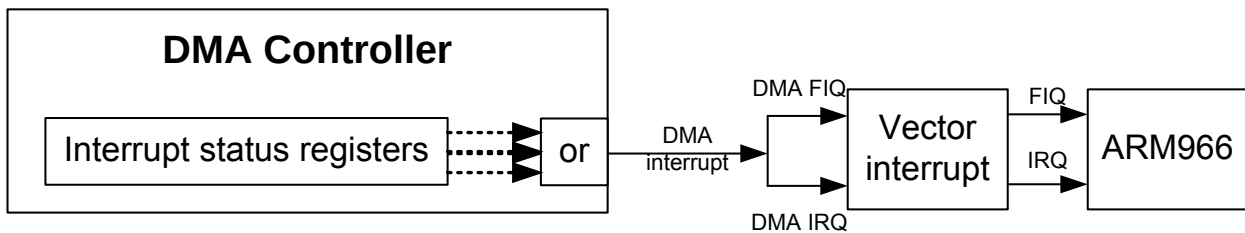


Figure 74: Internal structure of the DMA controller interrupts

The DMAC controller generates a combined interrupt output as an OR function of the individual interrupt requests.

The vector interrupt controller (VIC) has an OR function of all peripherals and allows masking the DMA interrupt for the fast interrupt request (FIQ) and the general interrupt request (IRQ) of the ARM CPU of each interrupt source. For more information, see register description of DMA controller and vector interrupt controller.

3.20 Multiplex matrix

To provide more flexibility in the use of the peripheral resources, the netX 51/52 is equipped with a multiplex I/O matrix. This matrix connects the netX with the “outside world” via a total of 40/48 MMIO pins (multiplex matrix input/output). On the chip side, a total of 98 signals are connected to the matrix, which can be routed to any of the MMIOs by setting the configuration registers of the MMIOs accordingly.

The following figure shows an overview of the multiplex matrix:

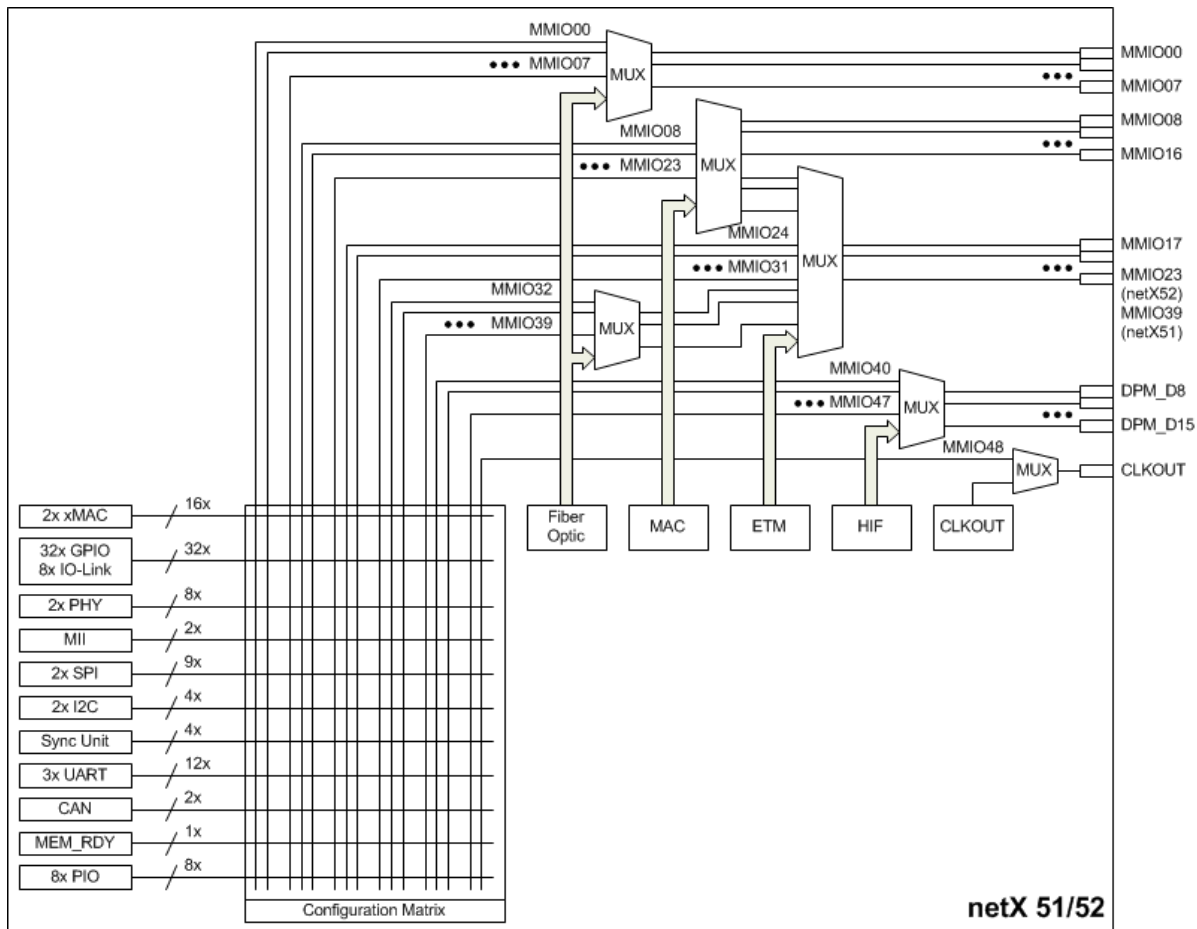


Figure 75: Multiplex matrix

By default (after reset or power-on), MMIO0-31 are mapped to GPIO0-31 and MMIOs32-35 are mapped to UART0 (to allow the serial boot mode option to be used with UART0). For a complete list of all MMIO signals, see chapter *Multiplex matrix signals netX 51* on page 310 or in *Multiplex matrix signals netX 52* on page 342.

Each MMIO pin has a corresponding configuration register which allows mapping any of the 98 internal signals to this pin. Moreover, input and output signals can be inverted independently. This also allows mapping an internal signal to more than one MMIO pin. If the internal signal is an output signal, this signal is simply replicated on the mapped MMIO outputs. If the internal signal is an input signal, the level of this signal is the result of a logical OR of all MMIOs that are mapped to this signal.

As some interfaces have or may have certain timing requirements the multiplex matrix cannot meet, some of the MMIO pins are directly shared with those interfaces and allow a direct signal routing. This applies to the ETM interface which always uses MMIO pins 17–39 when used, as well as the Ethernet PHYs when in Fiber optic mode (using MMIO 32–35 (channel 0) and 36–39 (channel 1) and some XMAC signals (TX, ECLK and FBCLK). However, for the XMAC signals this is optional (in common applications, they are routed via the multiplex matrix).

3.21 IO-Link controller

netX 51 and netX 52 have an integrated IO-Link controller:

- IO-Link V1.1
- 8 channels

The xPIC CPU is used for IO-Link. A HAL interface is provided to use the IO-Link controller, see reference [6].

3.21.1 Introduction

IO-Link is a new communication standard interface for sensors and actuators. IO-Link allows an inexpensive point-to-point connection between sensor/actuator and the I/O assembly for the "last meter to the process". IO-Link allows diagnosis and parameter information to be exchanged between sensor/actuator and the automation system in addition to the measurement signal through the specified communication mechanism, maximizing the performance of state-of-the-art intelligent sensors and actuators.

The "IO-Link working group" under the umbrella of PROFIBUS International (PI) has initiated the specification for a "fieldbus-independent" communication interface for intelligent sensors and actuators in industrial automation. This document is based on version 0.96 of the IO-Link specification which is still stable regarding hardware requirements.

An IO-Link Interface always uses a point-to-point topology and supports two or three physical wire interface (incl. power). It supports the following baud rates:

Port	baud rates (fCLK)	bit time (Tbit)
Com1	4800 Baud	208.33 us
Com2	38400 Baud	26.04 us
Com3	230400 Baud	4.34 us

Table 39: IO-Link Baud rates

IO-Link was designed as an open interface and can be integrated in all current communication systems in automation.

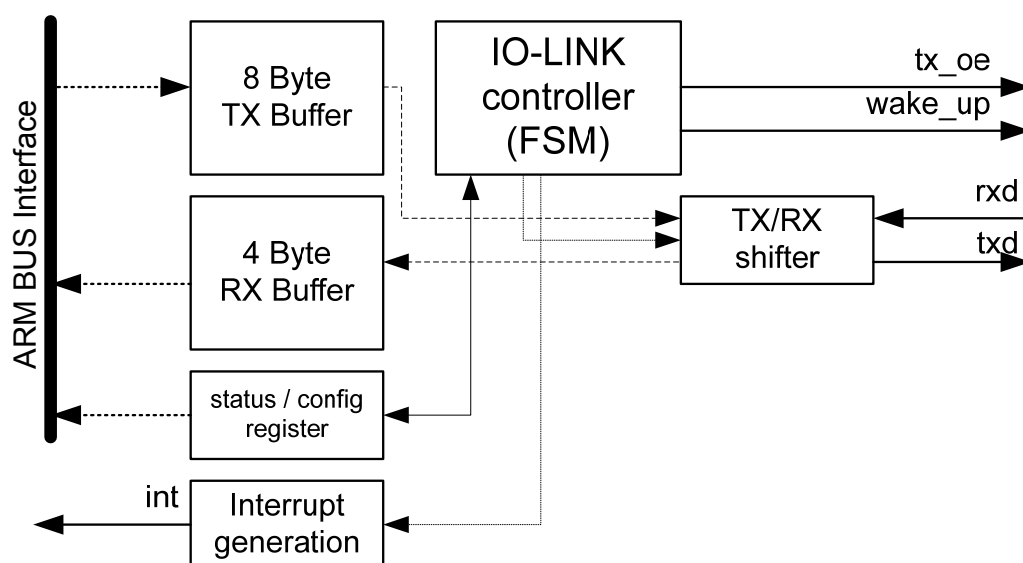


Figure 76: IO-Link: Block diagram of a single port

3.21.2 Typical application

The netX 51/52 supports up to eight IO-Link ports.

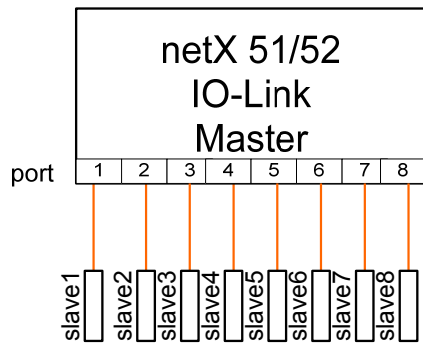


Figure 77: netX 51/52 with 8 IO-Link ports

Highlights of IO-Link:

- Dynamical update of sensors parameters by an SPS
- Service and online exchange of sensors
- Uniform wiring of variable sensors / actuators
- Uniform tools for parameterization

Industrial Ethernet / PROFINET

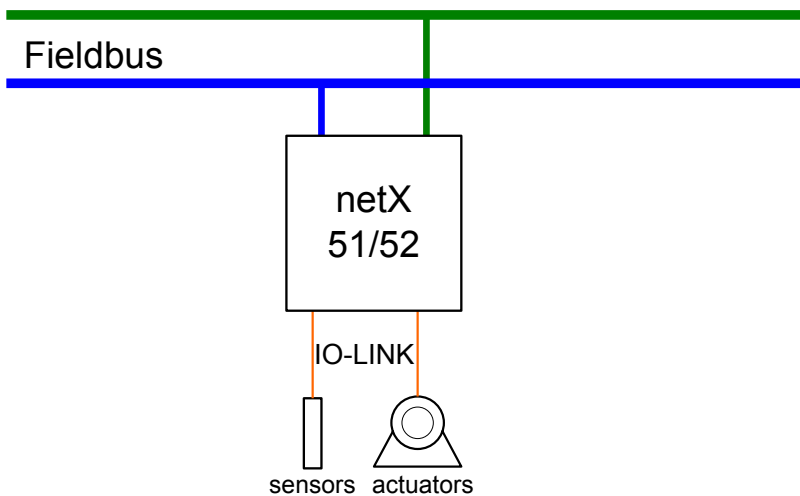


Figure 78: netX 51/52 IO-Link master interfacing

Features:

- 8 IO-Link ports (master)
- TX-buffer size programmable to up to 8 byte
- RX-buffer size programmable to up to 4 byte
- Programmable interrupt capabilities (frame finished, tx finished, rx finished, wake up finished)
- Loop frame transfer mode for continuous data transmission
- Single frame transfer mode for debug and analysis
- Automatic wake up generation

3.22 CAN controller

netX 51/52 has an integrated CAN controller which is designed to be compatible with SJA 1000. PeliCAN mode is supported, Basic mode is not.

Alternatively:

A HAL interface is provided to use the CAN controller, see reference [4].

3.22.1 Features

- SJA1000 compatibility with PeliCAN mode
- Extended receive buffer (64-byte FIFO)
- CAN 2.0B protocol compatibility
- Supports 11-bit and 29-bit identifier
- Bit rates up to 1 Mbits/s
- PeliCAN mode:
 - Error counters with read/write access
 - Programmable error warning limit
 - Last error code register
 - Error interrupt for each CAN-bus error
 - Arbitration lost interrupt with detailed bit position – Single-shot transmission (no re-transmission)
 - Listen only mode (no acknowledge, no active error flags)
 - Hot plugging support (software driven bit rate detection)
 - Acceptance filter extension (4-byte code, 4-byte mask)
 - Reception of message to be sent (self reception request)
- CAN signals programmable to MMIO pin
- BasicCAN of SJA1000 is not supported

3.23 UART

The three UARTs are 16550-compliant with 16 bytes transmit and receive FIFOs.

They can be configured to support speeds of up to 3.125 MBaud. The interface supports configurations of:

- five, six, seven or eight data bit transfers
- one or two stop bits
- even, odd or no parity
- IrDA SIR encoding and decoding

The request-to-send (RTS) and clear-to-send (CTS) modem control signals also are available with the interface for hardware flow control. Special features like stick parity and adjustable FIFO trigger level are implemented. Since UART0 is commonly used as a diagnostic port, we do not recommend using it for other purposes, especially when using loadable Hilscher firmware.

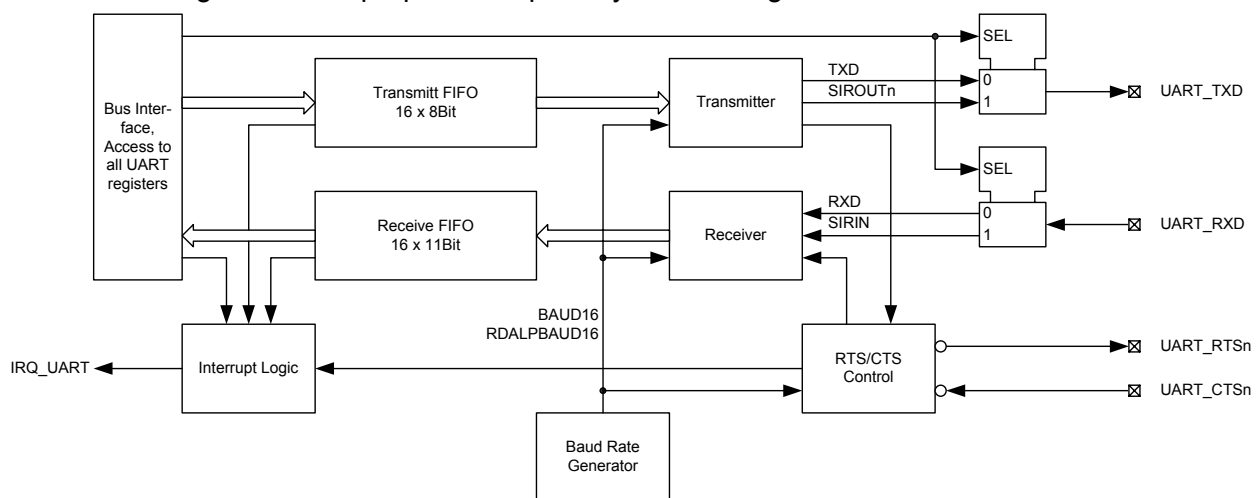


Figure 79: Block diagram of the UART

The ARM CPU reads and writes data and control/status information via the peripheral bus interface. The UART module can generate four individually-maskable interrupts which are combined to a single interrupt so that the output is activated if any of the individual interrupts are activated and unmasked.

If a framing, parity or break error occurs during reception, the corresponding error bit is set and stored in the FIFO. If an overrun condition occurs, the overrun register bit will be set immediately and FIFO data will be prevented from being overwritten.

Baud rate generator

The baud rate generator contains free-running counters which generate the internal Baud16 or IrLP-Baud16 signal. Baud16 or IrLPBaud16 provide timing information for UART transmit and receive control. Baud16 is a stream of pulses with a width of 10 ns and a frequency of sixteen times the baud rate.

Transmit FIFO

The transmit FIFO is an 8-bit wide, 16-bit deep, first-in, first-out memory buffer. CPU data written across the bus interface is stored in the FIFO until read out by the transmit logic. The transmit FIFO can be disabled to act as a one-byte holding register.

Receive FIFO

The receive FIFO is an 11-bit wide, 16-bit deep, first-in, first-out memory buffer. Received data and corresponding error bits are stored in the receive FIFO by the receive logic until read out by the CPU across the bus interface. The FIFO can be disabled to act as a one-byte holding register.

Transmitter

The transmit logic performs parallel-to-serial conversion on the data read from the transmit FIFO. Control logic outputs the serial bit stream which begins with a start bit, data bits, least significant bit (LSB) first, followed by a parity bit, and stop bits according to the configuration programmed in the control registers.

Receiver

The receive logic performs serial-to-parallel conversion on the received bit stream after the detection of a valid start pulse. Parity, frame error checking and line break detection are also performed, and the data with associated parity, framing and break error bits is written to the receive FIFO.

Interrupt logic

Four individual maskable active HIGH interrupts are generated in the UART module and combined to one interrupt output. This output is generated as an OR function of the individual interrupt requests.

The single combined interrupt is used with the system interrupt controller that provides another level of masking on a peripheral basis. This allows you to use modular device drivers which always know where to find the interrupt source control register bits.

IrDA SIR Endec

The transmitter and receiver block contain an IrDA SIR protocol Endec. The SIR protocol Endec can be enabled for serial communication via signals nSIROUT and SIRIN to an infrared transducer instead of using the signals TXD and RXD. The SIR protocol Endec can both receive and transmit, but it is half-duplex only, so it cannot receive while transmitting, or vice versa.

The SIR transmit encoder modulates the Non Return-to-Zero (NRZ) transmit bit stream. The IrDA SIR physical layer specifies the use of a Return to Zero, Inverted (RZI) modulation scheme which represents the logic 0 as an infrared light pulse. The modulated output pulse stream is transmitted to an external output driver and an infrared LED.

In normal mode the transmitted pulse width is specified as three times the period of the internal x16 clock (Baud16), that is, $3 / 16$ of a bit period.

Low-power mode of the transmitted infrared pulse is set to 3 times the period of the internal generated IrLPBaud16 signal. The frequency of IrLPBaud16 signal is set by writing the appropriate divisor value to UARTILPR.

The active low encoder output is normally LOW for the marking state (no light pulse). The encoder outputs a high pulse to generate an infrared light pulse representing a logic 0 or a spacing state.

The SIR receive decoder demodulates the return-to-zero bit stream from the infrared detector and outputs the received NRZ serial bit stream to the internal logic. The decoder input is normally HIGH (marking state) in the idle state (the transmit encoder output has the opposite polarity to the decoder input).

A start bit is detected when the decoder input is LOW. Regardless of being in normal or low-power mode, a start bit is deemed valid if the decoder is still LOW, one period of IrLPBaud16 after the LOW was first detected.

UART communication

Data received or transmitted is stored in two 16-byte FIFOs, the receive FIFO has 3 extra bits per character for status information.

For transmission, data is written into the transmit FIFO. This causes a data frame to start transmission with the parameters indicated in UARTLCR. Data transmission continues until there is no data left in the transmit FIFO. The BUSY signal goes HIGH as soon as data is written to the transmit FIFO (that is, the FIFO is non-empty) and remains set to HIGH while data is being transmitted. BUSY is negated only when the transmit FIFO is empty and the last character has been transmitted from the shift register including the stop bits. BUSY can be set to HIGH even though the UART module may no longer be enabled.

When the receiver is idle (RXD continuously 1, in the marking state) and a LOW is detected on the data input (a start bit has been received), the receive counter, with the clock enabled by Baud16, begins running and data is sampled on the 8th cycle of that counter (half way through a bit period).

The start bit is valid if RXD is still LOW on the 8th cycle of Baud16, otherwise a false start bit will be detected and ignored.

If the start bit was valid, successive data bits are sampled on every 16th cycle of Baud16 (that is, one bit period later) according to the programmed length of the data characters. If parity mode is enabled, the parity bit will be checked.

Finally, if RXD is HIGH, a valid stop bit will be confirmed, if not, a framing error has occurred. If a full word has been received, the data will be stored in the receive FIFO, with any error bits associated to that word.

Error bits

The three error bits are stored in bits 10:8 of the receive FIFO, and are associated to a particular character. There is an additional error which indicates an overrun error, but it is not associated with a particular character in the receive FIFO. The overrun error is set when the FIFO is full and the next character has been completely received in the shift register. The data in the shift register is overwritten, but it is not written into the FIFO.

FIFO bits 7:0: received data

FIFO bit 8: framing error

FIFO bit 9: parity error

FIFO bit 10: break error

Disabling the FIFOs

Moreover, the FIFOs can be disabled. In that case, the transmit and receive sides of the UART module have 1-byte holding registers (the bottom entry of the FIFOs). The overrun bit will be set when a word is received and the previous word has not yet been read.

Note: To use the UART 1 and UART 2 signals, route them to the MMIOs via the Multiplex Matrix (see chapter 3.20). UART 0 is also routed to MMIOs, but there is a default routing on power up (see chapter 5.4, Multiplex matrix signals).

3.24 USB

3.24.1 Overview

USB provides user-friendly solution for communication between a PC and several devices. It also supports hot-plugging, i.e. the user may connect and disconnect devices to the PC at run-time.

The USB interface only provides the USB device function which can be used as a simple UART-like communication interface to a USB host.

3.24.1.1 Features

- USB 1.1 / USB 2.0 full-speed support (12 MBit operation)
- USB protocol completely realized in hardware
- UART-like communication channel for user data (i.e. USB CDC ACM profile allows usage of standard OS drivers)
- Simple FIFO interface for receive and transmit data
- Stream, packet and transaction-oriented transmission possible
- USB bulk transfer class to make use of USB error checking and handling
- Hardware-controlled switching of internal pull-up resistor for device's presence detection
- DMA coupling for UART channel available

3.24.1.2 Typical applications

The USB interface provides a UART-like communication channel and, thus, offers a very easy way to exchange data between a USB host and the netX. Implementing a more sophisticated protocol in the application software is also possible.

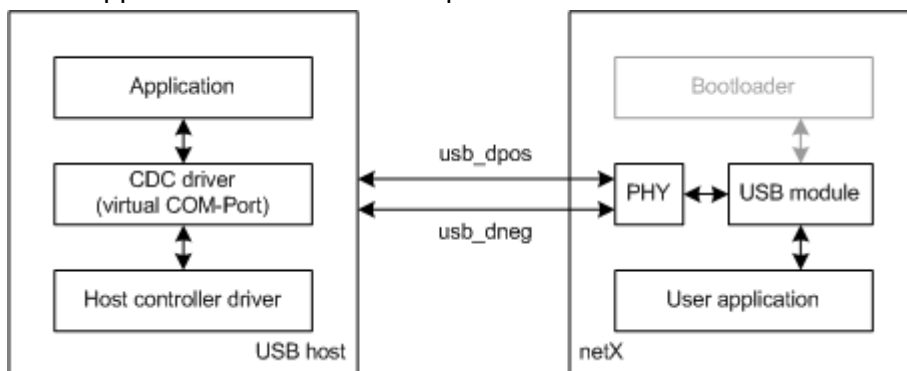


Figure 80: Typical application scenario with external connection to a USB host

The USB interface is disabled at application startup by default, i.e. it must be initialized by the user application. One exception is when the bootloader of the netX is configured to start in the “serial boot mode” (offering a “management console” to upload firmware, etc.). In this case the bootloader will initialize and use the USB interface. Once the user application has taken over control, it can directly use the USB interface.

3.24.2 Functional description

The USB interface offers USB 2.0 full-speed operation. The USB protocol, completely realized in hardware, offers a simple interface between USB and system CPU. The USB interface implements the USB CDC ACM profile, so standard OS drivers for the UART communication channel can be used.

3.24.2.1 Basic communication concept

FIFOs are used to exchange data between USB host and system CPU. The interface normally offers two FIFOs for data exchange. FIFO 1 sends data to the USB host, FIFO 2 holds data sent from the host. If CDC drivers on the host are not used to communicate with the interface, you can also use FIFO 3 to transmit short information to the USB host via an interrupt USB endpoint.

The interface offers two modes for each FIFO:

- Mode 1: Stream transmission
- Mode 2: Packet and transaction-oriented transmission

All modes are available with and without zero-length packet (ZLP) generation.

Note FIFO controller and concept have been completely revised since netX 10. Therefore, the complete interface to the ARM has changed and is *not* software-compatible.

The easiest approach is to use the stream transmission mode which is the default for the UART communication channel. In this mode all data that is placed into the `uart_tx` FIFO will be sent to the USB host immediately (when the host requests data from the USB interface). The same mechanism is used for the `uart_rx` FIFO. All data sent by the host will immediately be available on the netX.

In some situations the user may need packet or transaction-oriented transmission. Switching each FIFO individually to the desired mode is possible. For a detailed description of the modes, see the chapters below.

Internally, a handshaking mechanism for the FIFOs is needed. This is due to the nature of the USB protocol which transfers data in packets and may request retransmissions or discard incorrect data - if necessary. The FIFO controller performs the handshaking internally. Some modes require a simple acknowledgement mechanism initiated by the system CPU. Details are described along with the corresponding modes. For mode selection of the FIFOs, see reference [1].

3.24.2.1.1 FIFO concept

USB uses a packet-oriented transmission mechanism at the lower levels of the protocol stack. So, data exchange is normally block-oriented. To make user access easier, the USB interface translates this to a FIFO concept. FIFOs are usually easier to handle because advanced buffering need not to be implemented while packets are on the wire.

Since internally used FIFOs (i.e. for control endpoint and JTAG channel) are 64 bytes deep they can hold only one max.-sized packet at once. This is enough because they are handled by the hardware immediately.

The two UART FIFOs can hold up to 352 bytes each. This allows storing more than 5 max.-sized packets for both directions independently. Thus, high bandwidth transfers can be achieved on the UART channel even if the system CPU is temporarily busy with other tasks.

The FIFO controller has been designed to minimize interaction with the system CPU. The most trivial mode, the stream mode, does not need any interaction with the system CPU at all (apart from putting data into or reading out of the FIFOs).

In the packet and transaction modes the system CPU only needs very little handshaking with the FIFO controller. For more details, read the following chapters.

3.24.2.1.2 Stream transmission

FIFOs in stream mode offer the user a semi-continuous stream of data in transmitting and receiving direction. This is a very convenient mode when packet borders are not relevant for the exchanged data. This is most likely comparable with a classic UART.

Receive operation

The USB interface places received data in the corresponding FIFO (e.g. for the UART channel: 'usb_dev_uart_rx_data'). The data is immediately available to the system CPU after correct packet reception. Erroneous or incomplete packets will be dropped and are not visible to the system CPU. There is no need for any handshaking with the FIFO controller.

Transmit operation

Transmitting is very simple: The system CPU puts the data to be transmitted into the corresponding FIFO (e.g. for the UART channel: 'usb_dev_uart_tx_data'). Data will be transmitted automatically whenever requested by the USB host. There is no need for any handshaking with the FIFO controller.

ZLP behavior

The stream mode is available as a ZLP¹ and non-ZLP mode:

- The modes behave identical for receive FIFOs because the reception of a ZLP is irrelevant for the data stream. The reception of a ZLP on the UART channel is indicated by an interrupt flag ('uart_rx_zlp_received') so that the software can adapt its behavior if necessary (see section *FIFO assignment and default configuration* on page 174 on changing the default configuration of the UART channel).
- There is a difference in behavior for transmit FIFOs. For some high level protocols or drivers it is essential to use transmissions with ZLPs (this applies especially for the CDC drivers of most operating systems). The USB protocol has a concept of transactions. A transaction is complete whenever a packet of less than the max. size has been received or - if the last packet was max.-sized - if a ZLP has been received. The stream mode with ZLPs enabled tries to emulate this behavior on the stream in the following way: If the last packet sent was 64 bytes (i.e. max. packet size) and the FIFO is empty on the next host's IN request, a ZLP will be sent to terminate the transaction. If new data is placed in the FIFO before the host requests the next IN transfer, the current available data will be sent instead of a ZLP.

Besides the fill level interrupts (empty and full), the 'packet_sent' and 'packet_received' interrupts are also generated for stream FIFOs. They can be used as an indication for data reception.

3.24.2.1.3 Packet-oriented transmission

The FIFO controller also offers a packet mode which allows controlling packet borders of the data being sent. The packet mode also supports determining packet borders within the data received from the USB host.

Receive operation

If a receiving FIFO is configured for packet mode, the FIFO controller only allows one packet at once in the FIFO. All subsequent OUT transfers of the USB host will be NAKed, i.e. the USB host

¹ ZLP stands for "zero length packet". For more information on how and why they are used, see chapter 3.24.2.1.4 (transaction-oriented transmission).

will keep retrying to send the packet. After reading out the data from the FIFO the system CPU needs to read the corresponding length register (e.g. 'usb_dev_fifo_ctrl_uart_ep_rx_len' for the UART receive FIFO). Thereafter, the next packet sent by the host will be accepted.

It is also possible to read the length register prior to reading out the data. This allows higher bandwidths because the UART FIFOs can hold more than one max.-sized packet at once. In this situation the USB host is able to place the data of a new packet while the system CPU is still fetching data of the previous packet. There are no disadvantages in reading the length register first, but keep in mind that doing so can result in triggering the next 'packet_received' IRQ before the reading out of the data of the previous packet is finished.

Packets containing data will trigger the 'packet_received' IRQ. Zero-length-packets will be ignored in packet mode, i.e. they are not visible within the FIFO. The 'packet_received' IRQ will not be triggered either. Instead, the 'uart_rx_zlp_received' IRQ will be triggered for the UART receive FIFO.

Transmit operation

FIFOs used for transmitting data from the netX to the USB host do not have a specific packet mode because this is a sub-case of the transaction mode. FIFOs configured to packet mode will automatically behave as configured for transaction mode. To achieve transmission of single packets the programmed length must be less than or equal to 64 bytes (max. packet size). For more information on programming and ZLP behaviour, see the next chapter.

3.24.2.1.4 Transaction-oriented transmission

To make data exchange of undetermined size possible, higher level USB protocols use the concept of transactions. A transaction may have more bytes than the current max. packet size. Therefore it may consist of more than one data packet on the wire. A transaction is finished if one of the following conditions is met:

- The requested (by the host) number of bytes has been transmitted (received or sent). In this case zero-length-packets don't need to be used.
- A packet of less than the max. packet size has been received. If packets of max. size have been transmitted prior to this packet, they all belong to the transaction. If this was the only packet, the transaction consisted of this one packet only.
- A zero-length packet has been received after one or more max. packets have been received. All prior max.-sized packets belong to the transaction.

Figure 81 shows common cases of the mentioned cases. This excludes the case of pre-determined transfer size.



Figure 81: USB transaction examples

The transaction mode allows reception and transmission of this kind with and without ZLP generation.

Receive operation

If a receiving FIFO is configured to transaction mode, every received packet will trigger the 'packet_received' IRQ. Packets are accepted as long as space is available. If the last packet has been received (i.e. its size was less than the max. packet size or a ZLP has been received), the FIFO will stop accepting packets (of a new transaction) until the length register (e.g.

'usb_dev_fifo_ctrl_uart_ep_rx_len' for the UART channel) has been read. The last packet of the transaction will trigger the 'packet_received' and 'transaction_received' IRQs. The length register may be read while the transaction is still in progress, but this is no requirement. The length register contains information on the current / finished transaction: The packet length represents the size of the last packet received. The transaction length stays zero until the transaction is finished. The bits 'transaction_max_reached' and 'transaction_continued' come into play when the global 'maximum_transaction_len' has been set. In this case a transaction that is bigger than the configured amount of data will be virtually split into more than one transaction. The 'transaction_max_reached' bit will be set then. The next virtual transaction will have the 'transaction_continued' bit set.

Note: If the 'maximum_transaction_len' is not set, but a transaction with more than 8191 bytes (i.e. max. size of the transaction length counter) will be received, the counter will overflow. The reception of the transaction will continue, but the counter is not correct. If such situations are expected and it is essential that the counter is correct, the 'maximum_transaction_len' must be configured and the software must handle split transactions to determine the real transaction size.

Note: The transaction mode in receiving direction always uses ZLPs (or packets with less than the max. packet size) to determine transaction boundaries. Therefore it is not possible to use the transaction mode if the host will never emit a ZLP where normally necessary. The packet mode should be used instead. If stopping the reception after each packet is not desired, the transaction mode can still be used: 'transaction_received' IRQs will only be emitted on packets of less than the max. packet size (with possibly overflowing transaction length counter), but the 'packet_received' IRQ can still be used as in the "pure" packet mode.

Transmit operation

Sending data in transaction mode (and also packet mode because they are identical) is straight forward: At first, the user must program the transaction for the FIFO using the length register (e.g. 'usb_dev_fifo_ctrl_uart_ep_tx_len' for the UART transmit FIFO). There, the current transaction size in bytes is programmed along with ZLP behavior. The data can then be filled into the FIFO by writing the bytes to the data register (e.g. 'usb_dev_uart_tx_data' for the UART transmit FIFO).

When the transmission is finished, the behavior depends on the setting of the 'transaction_no_zlp' bit. If the bit is set, no ZLPs will be sent at all. This can be used for protocols that use transactions of pre-determined size and thus do not need ZLPs. If the bit is not set and the last packet of the transaction has maximum size, the FIFO controller will instruct the USB core to send a ZLP for the next IN request. The 'transaction_sent' IRQ will not be generated until the ZLP has been sent. Any further IN requests from the host will then be NAKed. NAKs will directly be sent (without a previous ZLP) if the last packet is smaller than maximum size.

Note: A new transaction must not be programmed before the previous transaction is finished. Use the corresponding FIFO status register ('transaction_active' bit) to determine when the programmed transaction has finished or use the corresponding 'transaction_sent' IRQ.

Note: If transactions of more than 8191 bytes need to be transmitted: Program the first transaction with the first 8128 bytes (i.e. 127 max.-sized packets), but the 'transaction_no_zlp' bit set (i.e. the host will receive NAKs after the last packet has been sent → transaction not yet finished). Afterwards program a transaction with the remaining bytes and the 'transaction_no_zlp' bit as required (normally unset → ZLP will be generated if necessary).

Under certain circumstances the user may need to abort an active transmission. This is possible: The FIFO controller supports aborts by programming a transaction length of '0'. However, this should be used carefully: Signalization of an aborted transaction to the host on the application level could be required. The 'transaction_no_zlp' bit will also be respected by taking the size of the last packet into account. If a packet is currently in transit, the abort will be performed after the packet has been sent. Be sure to check the 'transaction_active' bit in the status register after aborting a transaction.

Note: The 'transaction_sent' IRQ will also be generated, but only if the abort has been delayed because of an active packet transfer. This should help avoid continuous polling of the 'transaction_active' bit.

3.24.2.1.5 FIFO assignment and default configuration

The FIFO assignment is fixed within the descriptors of the USB interface. Consequently, the function and direction are fixed for each FIFO. The table below summarizes properties of the FIFOs found in the USB interface.

FIFO	EP	Function	EP Type	FIFO size	FIFO direction & mode
0	0	Control endpoint	Control	64 bytes	host → netX, stream with ZLP
1	0	Control endpoint	Control	64 bytes	netX → host, stream with ZLP
2	1	Interrupt transmit data	Interrupt	64 bytes	netX → host, transaction with ZLP
3	2	UART receive data	Bulk	352 bytes	host → netX, stream with ZLP
4	3	UART transmit data	Bulk	352 bytes	netX → host, stream with ZLP
5	4	JTAG receive data	Bulk	64 bytes	host → netX, stream
6	5	JTAG transmit data	Bulk	64 bytes	netX → host, stream

Since the control endpoint is handled internally in all possible use cases, the mode is also fixed and not user-configurable.

Interrupt endpoint and UART FIFOs are used to implement the USB CDC ACM profile (see next section for details). This assumes the use of standard drivers of operating systems. These endpoints make use of zero-length packets in their default configuration. When reconfiguring these FIFOs, it is strongly recommended to use modes that also use zero-length packet transmission to ensure proper operation. If different drivers are used (e.g. with a protocol that uses predetermined transfer sizes), the user may choose to use non-ZLP modes.

The JTAG FIFOs are normally handled internally by the debug hardware and cannot be used for user applications. These FIFOs must stay in stream mode without ZLP generation. They may be controlled by other applications if they have been configured to JTAG bypass mode by the host (see USB to JTAG module documentation for details). In this case it is possible to choose different modes for the FIFOs.

3.24.2.2 USB CDC ACM profile

For easier access on the host side, the USB interface implements the "Communication Device Class" using the "Abstract Control Model". Common operating systems offer standardized drivers for such devices. Usually a virtual COM-port will be created. Access from user software running on the host is similar to using the older standard serial ports when these drivers are used.

The USB interface implements the needed basic CDC operations, e.g. "set line coding" and "get line coding", directly in hardware. Thus, no user interaction on the netX is required. Moreover, all endpoints corresponding to the CDC are bundled within one interface. The vendor-specific JTAG

channel is bundled in another interface. This approach should allow handling those interfaces independently and at the same time from different software modules.

3.24.3 USB timing

3.24.3.1 Timeouts within the USB core

The USB core implements a timeout for receiving ACK tokens from the USB host in addition to data sent via the interface. This is essential to detect lost packets on the wire and as a consequence mark packets for retransmission.

The USB spec gives the host 622 ns in full-speed mode to respond to data sent from the USB interface to the host. The transmission of the ACK token takes approx. 1500 ns on the wire (sync pattern + token + end-of-packet). The max. time from the end of the last transmitted data packet to the complete reception of the ACK token is approx. 2122 ns.

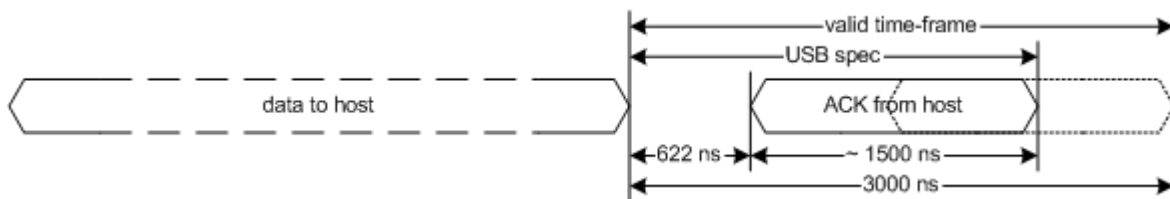


Figure 82: ACK-token receive timeout

To be a bit more tolerant and to compensate pad and sampling delays, the USB interface will trigger a timeout error after 3000 ns. The last packet sent will then be marked for retransmission.

The same applies for the reception of the DATAx token after the USB host has sent an OUT token because the USB spec defines the same 622 ns reaction time: After sending the OUT token the host has 622 ns time to start transmitting the DATAx packet (i.e. sync pattern + token of data packet). This will also take approx. 1500 ns like the transmission of the ACK token above. Thus, the same 3000 ns window is implemented in the USB core.

3.24.3.2 Pull-up resistor switching

The USB 2.0 specification requires a variable pull-up resistor for devices which cannot meet the specification of $1.5\text{ k}\Omega \pm 5\%$. As this is not possible on the netX SoC, the USB interface implements the switched variant.

The spec requires a change from the low resistance phase ($900\ \Omega$ to $1575\ \Omega$) to a high resistance phase ($1425\ \Omega$ to $3090\ \Omega$) when a J-K-transition is detected. Switching must occur within 0.5 bit-times. The switching from a high resistance phase back to a low resistance phase should occur if an SE0 is detected on the bus for more than 0.5 bit-times (see Figure 83) or, alternatively, if a J-state is detected for more than 7 bit-times (see Figure 84). The USB interface implements a combination of the two alternatives.

The USB interface is clocked by a 100 MHz system-clock. A half USB bit-time in full-speed mode lasts 41 ns which is approx. four system clock ticks.

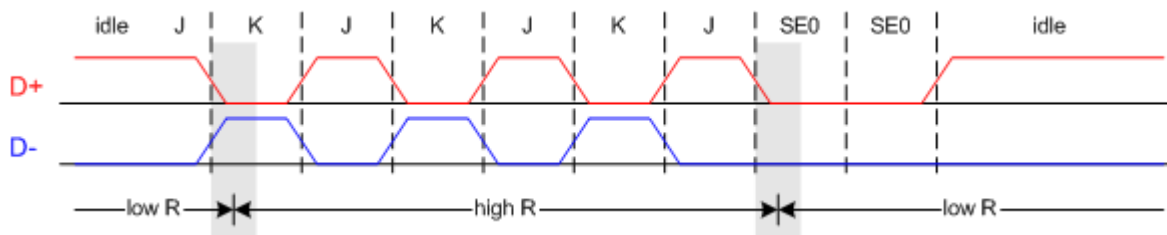


Figure 83: Resistor switching back when SE0 is detected

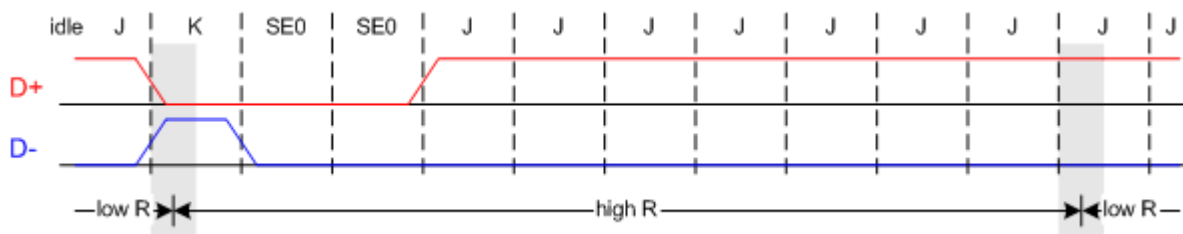


Figure 84: Resistor switching back after 7 bit-times J-state

The shaded areas in the figures above represent the area where the switching must occur. Detection is performed on the unfiltered USB data lines. This results in only two system-clocks delay for the sampling flip-flops and one additional system-clock delay for edge detection. The result is a three system-clock delay (i.e. 30 ns for 100 MHz system clock) for the detection that fulfils the USB specification.

3.25 I2C interface

3.25.1 Overview

The I2C interface is a simple 2-wire interface providing a clock and a data line. Transfers are serial, 8-bit oriented and bidirectional between master and slaves. Each device connected to the I2C interface is addressable by a unique address.

3.25.1.1 Features

The netX 51/52 I2C unit has full master and slave function. It provides SCL clock rates from 50 kHz up to 3.4 MHz.

For high efficient data exchange, the module includes a standard ARM DMA interface together with a 16-byte master data FIFO and a 16-byte slave data FIFO. It provides interrupts for the most important events like slave selection, FIFO requests and errors, I2C bus collision detection and end-of-transfer.

To keep the main processor load low, a state machine is implemented which can run complex I2C sequences like acknowledge polling or long data transfers. I2C features like 7 and 10-bit slave addressing and multi master arbitration are supported by this module.

Note: This unit is not compatible with the netX 100/netX 500 I2C unit!

3.25.1.2 Typical applications

There are many I2C devices on the market today. Typical are low bandwidth devices as:

- EEPROMs
- display controllers
- card readers
- various types of sensors
- microcontrollers
- various types of ICs with I2C configuration channel

3.25.2 Functional description

For detailed I2C standard description, see Philips I2C-Bus specification (version 2.1, 01.2001). The following section gives a brief overview.

I2C devices are either master only, slave only or master-and-slave devices. The netX 51/52 I2C unit is a master-and-slave device.

3.25.2.1 Block diagram

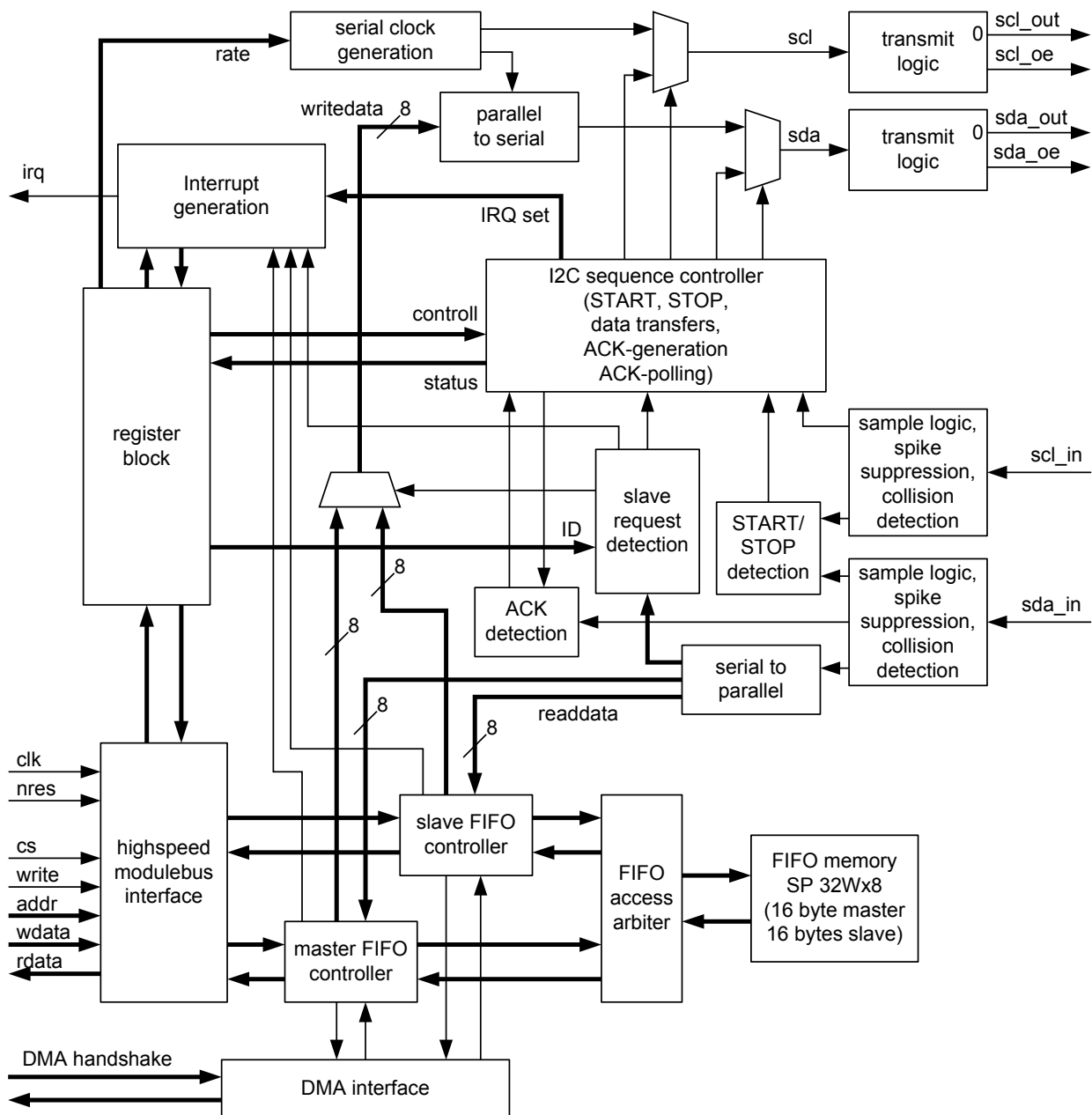


Figure 85: I2C module block diagram

3.25.2.2 I2C signals

The I2C interface defines only two signals:

- SCL: Serial clock
- SDA: Serial data

To avoid signal driving conflicts, both signals are never actively driven to high level. High level is realized by (internal) pad pull-up resistors.

3.25.2.3 I2C signal conditions

A transfer is initiated by a master with a start condition (START) or a repeated start condition (rSTART). The start condition is represented by a falling edge of SDA while SCL is high.

At the end of a transfer, the master must set a stop condition (STOP). The stop condition is represented by a rising edge of SDA while SCL is high.

During data transfers, SDA is valid and must not change, while SCL is high.

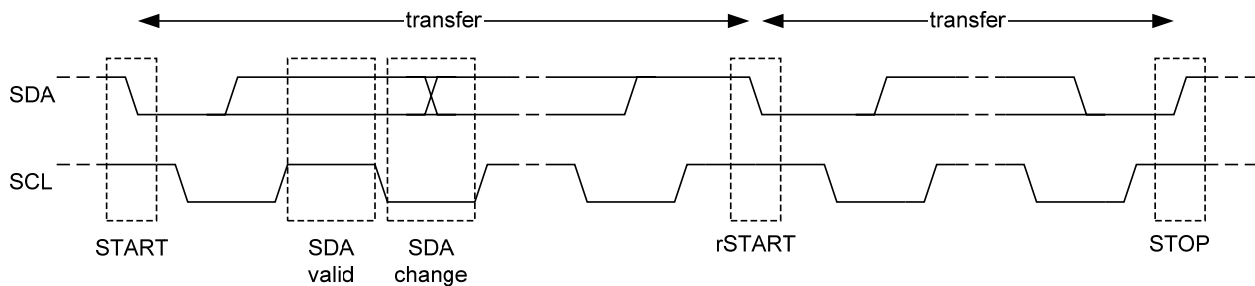


Figure 86: I2C signal condition states

3.25.2.4 I2C transfers

An I2C transfer always transmits eight bits of data (MSB first) followed by an active low acknowledge bit generated by the receiving device.

The first transferred byte after (r)START is always generated by a master. It contains a 7-bit slave address and an nWrite/Read bit which indicates the transfer direction (0: write transfer master to slave, 1: read transfer master from slave).

Every I2C slave device has its own slave ID. If any ID matches the address generated by the master during the first byte after (r)START, the respective slave will acknowledge the first byte.

For detailed I2C address range specification, see Philips I2C-bus specification.

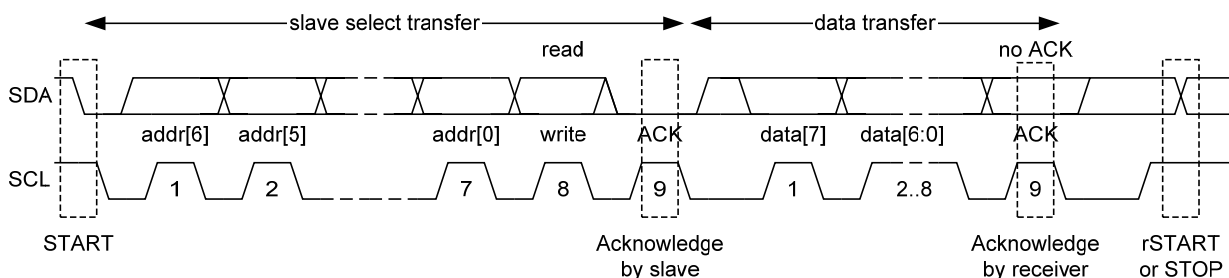


Figure 87: I2C 1 byte transfer

The low active acknowledge bit is always generated by the receiving device (during write transfers by the slave, during read transfers by the master) after each transferred byte.

The diagram illustrates the timing of an I2C write operation. It shows a sequence of events on a bus:

- S**: START condition (white box).
- slave address**: Address sent from master to slave (white box).
- R/nW**: Read/Write bit, set to 0 for write (white box).
- A**: Acknowledge from slave (hatched box).
- data**: Data sent from master to slave (white box).
- A**: Acknowledge from slave (hatched box).
- data**: Data sent from master to slave (white box).
- A**: Acknowledge from slave (hatched box).
- data**: Data sent from master to slave (white box).
- A/Ā**: Acknowledge from slave (hatched box).
- P/rS**: STOP condition or Repeated START (white box).

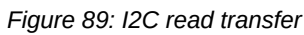
Annotations below the diagram:

- 0 (write)**: Points to the R/nW bit.
- transferred data**: Points to the data segments.
- (repeated until no ACK or rSTART or STOP)**: Indicates the loop for data transfer.

Legend:

- White box: from master to slave
- Hatched box: from slave to master
- A = acknowledge
- Ā = not acknowledge
- S = START
- rS = repeated START
- P = STOP

After the last byte of a read transfer, the master (as receiver) must not generate an acknowledge to mark the end of the transfer to the slave. Otherwise the slave will continue sending data and produce a bus error.



Note: After an acknowledged (r)START with set read bit, at least one byte must be transferred from slave to master.

The diagram illustrates the I2C protocol for writing and reading data. It shows two sequences: a write sequence and a read sequence.

Write Sequence: S (START) → slave address → R/nW (0) → data (8 bytes) → A (acknowledge) → rS (repeated START).

Read Sequence: S (START) → slave address → R/nW (1) → data (8 bytes) → A (acknowledge) → P/rS (STOP).

Legend:

- White box: from master to slave
- Hatched box: from slave to master
- A = acknowledge
- $\bar{A}$ = not acknowledge
- S = START
- rS = repeated START
- P = STOP

Figure 90: I2C transfer with direction change

3.25.2.6 I2C 10-bit addressing

The Philips I2C specification describes an extended 10-bit addressing mode that is entered by a 2-byte start sequence. The first byte after (r)START contains a 7-bit address matching the reserved pattern “11110XX”. “XX” are the two MSB bits of the requested 10-bit address.

Note: For detailed I2C address ranges and pattern, see Philips I2C specification.

If a 10-bit slave device is on the I2C-bus with the appropriate MSB address bits, the slave will acknowledge the first byte. The master will then transfer the second start byte containing the lower eight address bits.

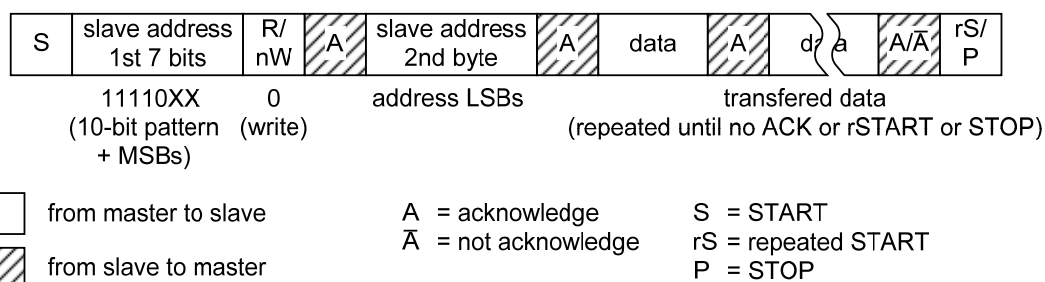


Figure 91: 10-bit addressing

Since the second byte of a 10-bit start sequence is always transferred from master to slave, the read bit in the first transferred byte must always be 0.

A read transfer from a 10-bit addressed slave is initiated by a write start sequence followed by an rSTART and the first start byte containing the 10-bit address pattern, the slave address MSBs and the read bit set to 1. The second start byte will not be transferred again.

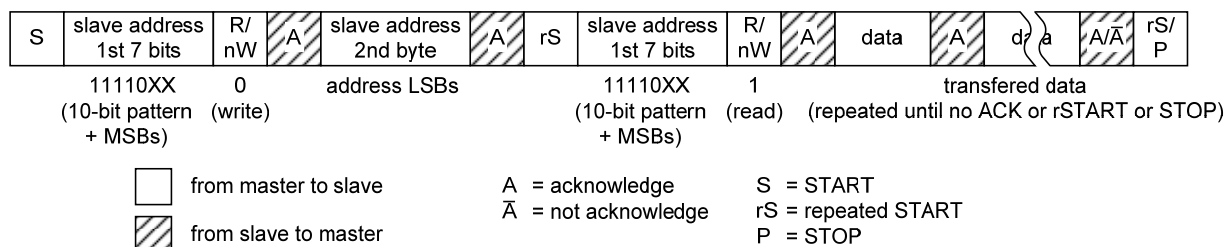


Figure 92: I2C initializing a read transfer from a 10-bit address slave

Various types of transfers can be combined as described in the Philips I2C specification. E. g. write data may be inserted before restarting in Figure 92.

3.25.2.7 I2C general call

The I2C general call function is provided by the reserved 7-bit address pattern “0000000” transferred after START initiating a write transfer (read-bit set to 0).

Note: For detailed I2C address ranges and pattern, see Philips I2C specification.

3.26 QSPI/SPI interface

Apart from the I2C, the SPI is the most common serial interface for peripherals and memory components.

SPI (Serial Peripheral Interface) is a full-duplex interface defined by Motorola. Transfers are serial, typically 8-bit oriented and bidirectional between master and slaves. Slave devices are selected by a chip select signal.

Beside the full-duplex interface (Standard SPI), netX 51/52 offers a Single SPI, Dual SPI and Quad SPI interface which is half-duplex.

netX 51/52 offers the following modes for SPI0 and SPI1:

Mode	SPI0	SPI1
Standard SPI	full-duplex, FIFO Master only	full-duplex, FIFO Master or slave
Quad SPI	half-duplex, FIFO Master only	modes not supported
Dual SPI		
Single SPI		
XiP (Quad SPI)	Execution of programs Master only	mode not supported

Table 40: netX 51/52 – SPI0 and SPI1

3.26.1 QSPI interface (SPI0)

The QSPI module provides a standard Motorola-based SPI (Serial Peripheral Interface) with one receive and one transmit data line, Dual SPI (2-bit data lines) and “Serial Quad I/O” (SQI, i. e. Quad SPI; 4-bit data lines). All modes support master functionality only, slave functionality is not available.

Note: The SPI interface (SPI1) provides the SPI slave functionality.

Standard SPI: Standard SPI function and programming are compatible with netX 50 SPI master, but netX 51/52 no longer supports netX 100-software compatibility. The netX 51/52 provides several new features like dummy cycles or half-duplex modes (necessary for Dual and Quad SPI operation) for standard SPI.

Quad SPI (SQI) with eXecute-in-Place: For Quad SPI devices, an additional high speed eXecute-in-Place (XiP, SQIROM) mode is implemented. In this mode, data from a QSPI device is readable similar to a linear external memory (e. g. a parallel Flash device) instead of being read from a single data register with a FIFO. Reading a QSPI device becomes fully transparent then: The QSPI device is mapped into a linear 16 Mbyte memory area and can be treated like standard Flash. It can thus be used for direct code execution without hard runtime requirements. Copying the data into a RAM before code execution is not necessary. Depending on the code style, SQIROM provides similar performance or can even outperform parallel Flash devices.

3.26.1.1 QSPI features

- Standard SPI full-duplex master mode:
 - Programmable data word size from 4 to 16 bit (default: byte), MSB transferred first
 - 16-word deep FIFOs for transmit and receive data
 - Registers are compatible with netX 50 SPI module
- Single SPI, Dual SPI and Quad SPI (1, 2 and 4-bit) half-duplex master modes:
 - Byte-oriented, LSB or MSB first, programmable 32-bit endianness
 - 64 byte deep FIFO for receive or transmit data (1-bit FIFO depths: 16 words)
- High speed eXecute-in-Place (XiP/SQIROM) mode for Quad SPI Flash devices:
 - Mapped as 24-bit (16 Mbyte) linear address range for system masters
 - Widely configurable to support many different devices up to 128 Mbit (16 Mbyte)
 - Complete QSPI read sequences including instruction and chip-select generation without CPU interaction
 - Optimized QSPI sequence generation: Continuous streams for reads (read bursts) with incrementing addresses, 4-byte read ahead buffer
 - Linear code data rate: 320 Mbit/s @ 80 MHz (16-bit parallel FLASH @ 70 ns: 220 Mbit/s)
- Supported SPI modes:
 - All modes (clock polarity high or low and clock phase 0 or 1) in non-XiP mode
 - Mode 0 and 3 in eXecute-in-Place (XiP/SQIROM)
- Serial clock rates up to:
 - 50 MHz in non-XiP mode
 - 133 MHz in eXecute-in-Place (XiP/SQIROM) mode
- IRQ-based FIFO interaction possible
- DMA interface for receive and transmit data to minimize system CPU load
- Input signal over-sampling and filtering for hazard suppression
- Static or dynamic chip-select controlling in standard SPI mode

Note: The netX 50/100/500 SPI module differs from the netX 51/52 QSPI module which has many new features and improved functions.

3.26.1.2 Typical applications

Today's market offers many SPI devices. Typical are low bandwidth devices such as:

- Serial Flashes
- MMC cards
- various types of sensors
- microcontroller interfaces

The following figures show different possibilities of external SPI interconnection to SPI slave devices.

Note: Up to 3 chip-select signals are available for the QSPI module. However, only chip-select signal 0 is always available by default. Further chip-select signals are available as multiplex matrix I/Os (MMIOs).

Note: On netX 51/52 all chip-select signals are also controllable as PIOs from within the QSPI module. This limitation does not touch chip-select generation with 'sqi_cr1'.

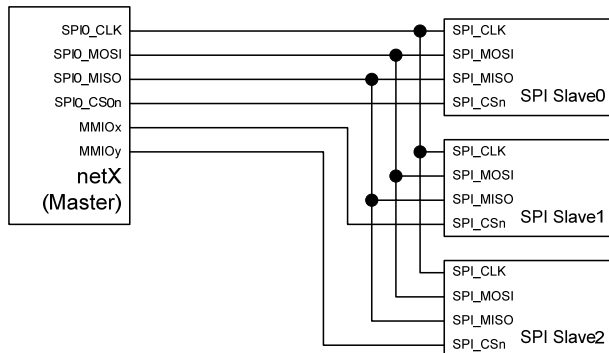


Figure 93: Standard external SPI interconnection

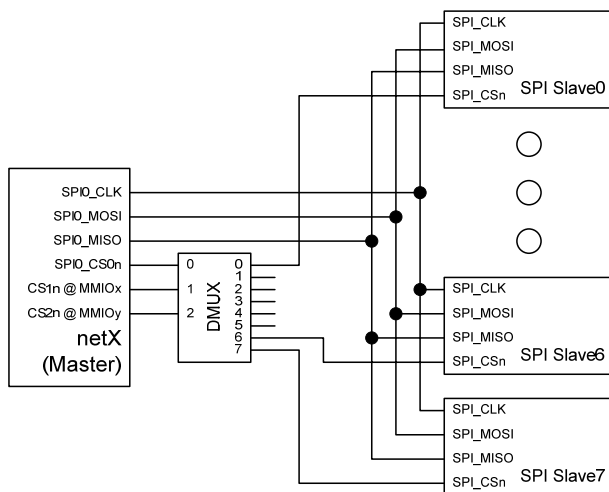


Figure 94: Extended external SPI interconnection with DMUX

Figure 95 shows an interconnection to a high-bandwidth QSPI flash in 4-bit mode.

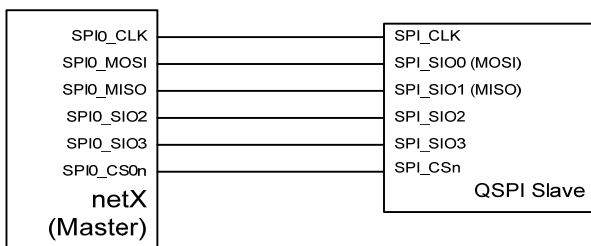


Figure 95: Quad SPI and XiP/SQIROM interconnection

In 2-bit mode SPI_SIO2 and SPI_SIO3 are left unconnected. In this case, XiP mode cannot be used.

3.26.1.3 Module organization

The QSPI module consists of two submodules which separate the standard QSPI peripheral operation mode (QSPI peripheral submodule) from the SQIROM/XiP operation mode (SQIROM submodule).

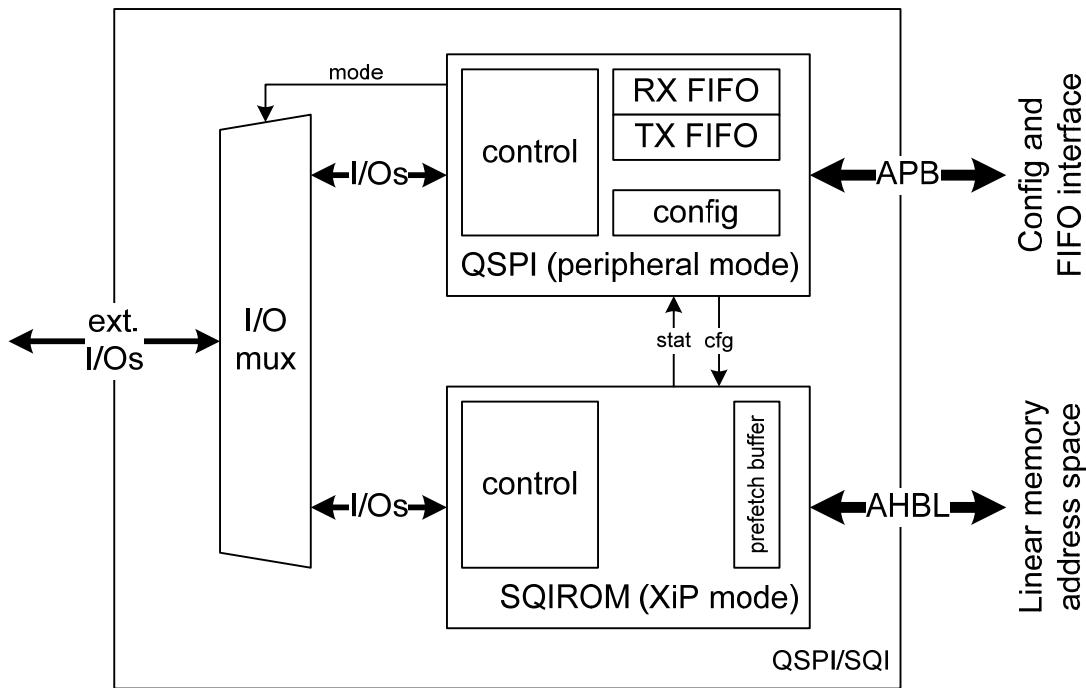


Figure 96: Module organization: QSPI peripheral mode and SQIROM/XiP mode

Figure 96 shows that the I/Os of each submodule are multiplexed. For this reason peripheral QSPI mode (FIFO-controlled) and SQIROM/XiP mode cannot be used at the same time. This is no limitation because it is not recommended to connect additional SPI components in parallel to a QSPI flash device (in XiP mode, e.g. for code execution) due to performance reasons.

The standard QSPI peripheral mode allows connecting standard SPI devices (1-bit I/O mode; standard Motorola SPI) as well as devices that support 2-bit and 4-bit I/O modes (Dual and Quad SPI). It is possible to read and write to such devices using a FIFO-based interface.

The SQIROM/XiP mode can be used to access QSPI flash devices in a high-performance 4-bit mode. This mode supports only read operations that are transparently mapped to a linear address space (e.g. code execution). Write operations to flash devices can be performed in the standard QSPI peripheral mode.

Note: To use the SQIROM/XiP mode the following sequence has to be observed: A vendor-defined command sequence has to be sent to the SQIROM/XiP-capable flash device in standard QSPI peripheral mode, before the QSPI module can be switched to the high-performance SQIROM/XiP mode.

3.26.1.4 Functional description

All QSPI_XIP module functions are based on standard Motorola SPI which is introduced in the next section. Dual and Quad SPI are enhanced versions of Motorola SPI for higher data rates. Related module properties are provided in section *Dual and Quad I/O interface* on page 189. Quad SPI eXecute-in-Place (XiP/SQIROM) function and device requirements can be found in section *XiP (eXecute-in-Place)/SQIROM mode* on page 192.

3.26.1.4.1 Standard Motorola SPI mode

This section gives a brief overview over Motorola SPI (Serial Peripheral Interface). SPI is a full-duplex 4-wire interface. Transfers are serial, typically 8-bit-oriented and bidirectional between master and slaves. Slave devices are selected by a chip-select signal.

SPI devices are either master or slave devices. The QSPI module provides master functionality only.

The SPI interface consists of four signals:

- Clock signal (SPI_CLK)
- Low active chip-select signal (SPI_CSn)
- Master Out (transmit), Slave In (receive) data signal (SPI_MOSI)
- Master In (receive), Slave Out (transmit) data signal (SPI_MISO)

SPI transfer format

SPI data transfer is activated by a low active chip-select signal. Transfers are full-duplex bidirectional always serialized MSB first. SPI signal generation, sampling states and timing can be selected by programmable Serial Clock Polarity *SPO* and Serial Clock Phase *SPH*.

SPI clock polarity (SPO) is the idle state of SPI clock signal SPI_CLK when chip-select is inactive.

SPI clock phase (SPH) controls SPI data generation and sample timing. If SPH 0 is selected, data on SPI_MOSI and SPI_MISO will be sampled on the first clock edge of SPI_CLK. If SPH 1 is selected, the signal will be sampled on the second edge. Data is always generated one edge earlier than sampling.

Clock polarity and clock phase often are combined to the SPI mode: SPO is interpreted as bit 1 and SPH is interpreted as bit 0 of the mode (e.g. SPO = 0, SPH = 1 is mode 1). The QSPI module supports the generation of all 4 modes, but the SQIROM mode supports modes 0 and 3 only.

Chip-select generation

Typically, slave devices require chip-select signal staying low for a whole transfer. This is reached by static chip-select controlling: Chip-select can be activated and deactivated by software like a programmable IO. However, some devices require a chip-select signal that becomes inactive between each transferred word. This is reached by dynamic chip-select generation: The chip-select signal will be driven inactive automatically for at least half a serial clock period between each word.

Note: Chip-select generation can be controlled by bit *fss_static* of register *sqi_cr1*.

SPI frame format with SPO=0 and SPH=0 (mode 0)

Clock idle state is low and data is sampled on the first (rising) edge of any SPI_CLK period. MSB data is generated when chip-select is activated. To generate the MSB data bit of the next word, chip-select toggling would be required after each data word. Since this is not suitable for many slaves, the MSB of a new data word is generated by the last (falling) edge of the preceding word instead.

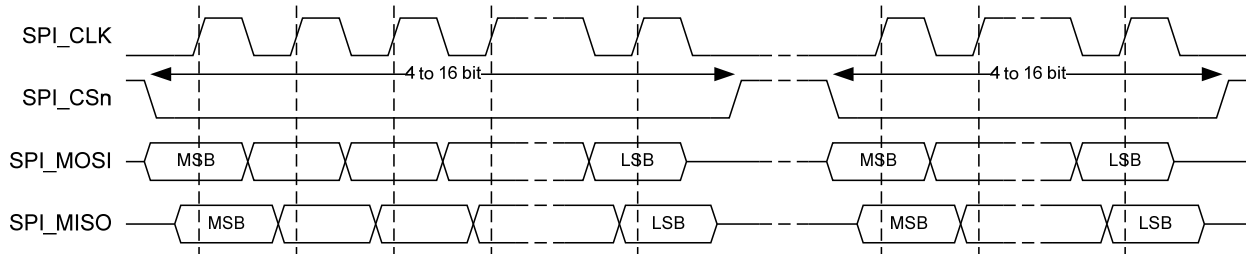


Figure 97: SPI SPO=0 and SPH=0 transfer (mode 0)

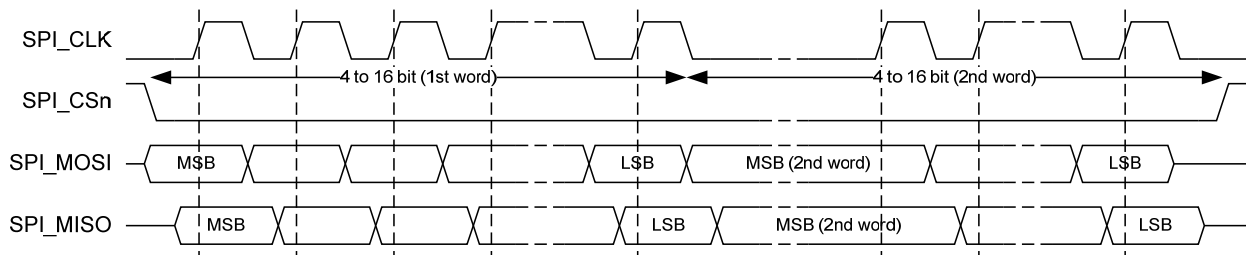


Figure 98: SPI SPO=0 and SPH=0 transfer (mode 0) without chip-select toggling

SPI frame format with SPO=0 and SPH=1 (mode 1)

Clock idle state is low and data is sampled on the second (falling) edge of any QSPI_SCK period. MSB data is generated on the first (rising) QSPI_SCK edge. Slave devices do not need chip-select toggling at transfer start to generate MSB data bit.

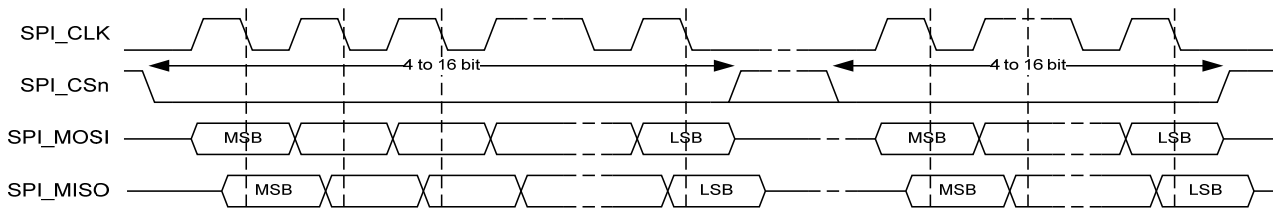


Figure 99: SPI SPO=0 and SPH=1 transfer (mode 1)

SPI frame format with SPO=1 and SPH=0 (mode 2)

Clock idle state is high and data is sampled on the first (falling) edge of any SPI_CLK period. MSB data is generated when chip-select is activated. To generate the MSB data bit of the next word, chip-select toggling would be required after each data word. Since this is not suitable for many slaves, the MSB of a new data word is generated by the last (rising) edge of the preceding word instead.

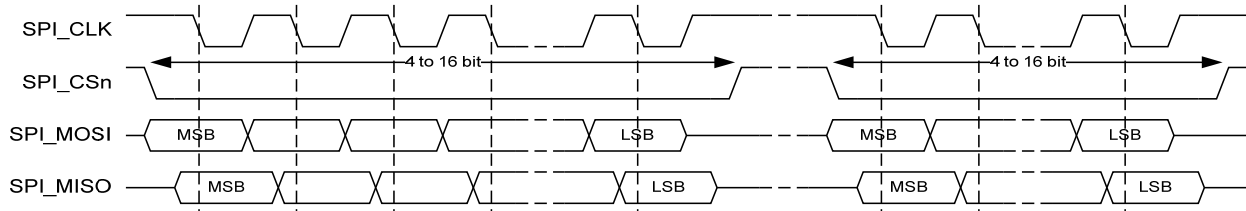


Figure 100: SPI SPO=1 and SPH=0 transfer (mode 2)

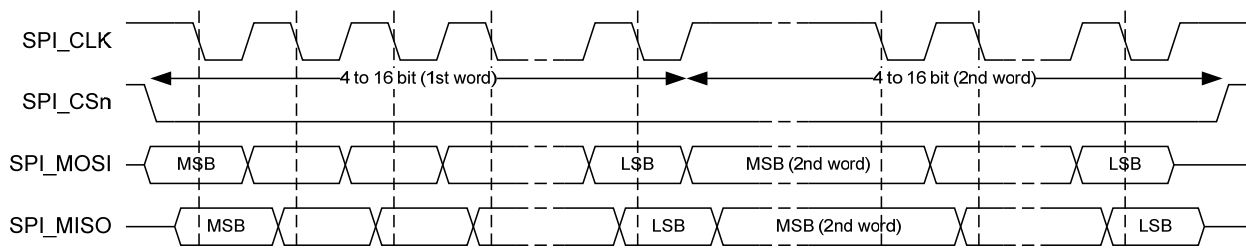


Figure 101: SPI SPO=1 and SPH=0 transfer (mode 2) without chip-select toggling

SPI frame format with SPO=1 and SPH=1 (mode 3)

Clock idle state is high and data is sampled on the second (rising) edge of any QSPI_SCK period. MSB data is generated on the first (falling) SPI_CLK edge. Slave devices do not need chip-select toggling at start of transfer to generate MSB data bit.

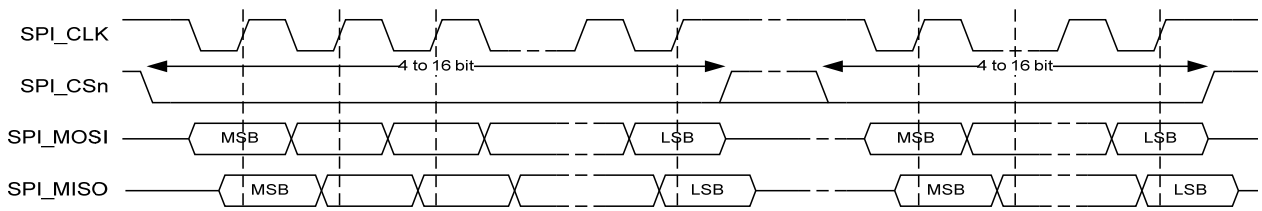


Figure 102: SPI SPO=1 and SPH=1 transfer (mode3)

3.26.1.4.2 Dual and Quad I/O interface – QSPI mode

Dual and Quad SPI are enhanced SPI modes providing higher data rates by using 2 or 4 data lines for each data direction instead of just one (standard SPI). Contrary to standard SPI, Dual and Quad SPI work in half-duplex mode, i.e. data can either be written to a slave or be read from a slave. A simultaneous data exchange in both directions is not possible. The difference between Dual SPI and Quad SPI is that Dual SPI is physically identical with standard SPI (i.e. requires only four wires) while QSPI requires two additional data lines. Dual SPI needs 4 serial clock periods to transfer one byte while QSPI only needs 2 periods.

Dual and Quad SPI signal timing is identical with the standard SPI mode. All four SPI modes (clock polarity 0 or 1 and clock phase 0 or 1) are available.

Note: The following text describes only the Quad SPI mode. For Dual SPI consider that transferring one byte will take 4 serial clock periods instead of 2 and that only 2 data lines are required.

QSPI signals

In QSPI mode all data signals can be used for transmit and receive. The SPI names MOSI and MISO are thus no longer used. The signals are simply named Serial IO 0 to 3 (SIO0...3) instead. For backward compatibility to standard SPI, the SIO0 function is located on SPI_MOSI and the SIO1 function is located on SPI_MISO. In total, the Quad SPI uses six signals, the Dual SPI only the 4 standard SPI signals:

- Clock signal (SPI_CLK)
- Low active chip-select signal (SPI_CSn)
- Serial I/O
 - Quad SPI: Serial IO (SIO) 0 to 3 (SPI_MOSI, SPI_MISO, SPI_SIO2, SPI_SIO3)
 - Dual SPI: Serial IO (SIO) 0 and 1 (SPI_MOSI, SPI_MISO)

QSPI transfers

QSPI is typically used to access serial memories (Flash devices). Data exchange with such devices is transfer-oriented. The chip-select signal is used to separate transfers from each other: During a transfer it must remain stable and active. Between transfers it must become inactive.

QSPI transfers typically require a transfer header always generated by the QSPI master. The header contains at least a command word, but often more information, e. g. an access address. Data will be transferred after the header.

It is quite common for flash devices to transfer some dummy cycles between header and data. Dummy cycles do not transfer any data, but clocking is needed. This operation is also supported in a special mode in order not to fill the transmit/receive FIFOs with unneeded/unused data.

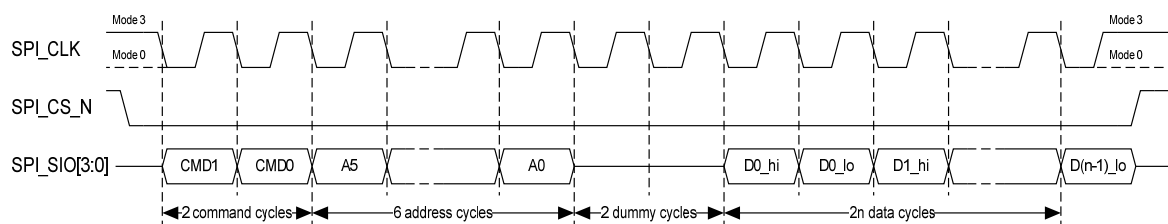


Figure 103: QSPI mode 0 and 3 transfer example

The QSPI module can perform three different types of transfers:

- Dummy transfer
 - No FIFO interaction
 - Data lines can be controlled (driven high, driven low or high impedance)
 - Purpose: Device-required dummy cycles between header and data or to write constant data (e.g. all bytes 0xFF or 0x00)
- QSPI module is transmitter – half-duplex transmit mode
 - Data from TX-FIFO will be sent
 - Purpose: Send header or write data
- QSPI module is receiver – half-duplex receive mode
 - Data is stored in RX FIFO
 - Purpose: Read data from QSPI devices

The QSPI mode does not support the automatic chip-select generation ('fss_static' bit is ignored). A full transfer sequence can thus be combined between falling and rising chip-select by several sub-transfers of different transfer types. The status bit 'busy' or the IRQ 'trans_end' can detect the end of each sub-transfer.

The transfer size can be programmed for each sub-transfer up to 524288 bytes or dummy cycles. Each sub-transfer must be started by a software trigger ('start_transfer' bit). To start immediately, bit 'start_transfer' can be programmed together with the transfer size.

Transfers requiring FIFO interaction (i. e. non-dummy transfers) will be paused in case of imminent FIFO errors (i.e. RX-FIFO full in receive mode or TX FIFO empty in transmit mode). They will be continued as soon as the related FIFO is capable of receiving or transmitting again without producing an error (i.e. a full RX-FIFO must be read or new data must be stored in an empty TX-FIFO).

Note: QSPI transfers use a transfer mechanism that differs from the old transfer scheme (which is still the default for standard SPI operation): In QSPI modes and standard SPI half-duplex modes transfers are not started immediately when data is stored in the TX FIFO. The old scheme is not possible because QSPI mode uses half-duplex transfers. Therefore, the number of bytes to be transferred has to be programmed. However, standard SPI mode can be programmed to use the QSPI transfer scheme by setting bit 'spi_trans_ctrl' of register 'sqi_cr1'.

QSPI data is typically serialized MSB (Most Significant Bit) first. This module additionally supports LSB (Least Significant Bit) first serialization.

Note: To select MSB and LSB first data serialization, use bit 'ms_bit_first' of register 'sqi_tcr'. MSB first is commonly used and selected by default.

Note: Standard SPI does not support LSB first data serialization. The configuration mode bit 'ms_bit_first' will be ignored even in standard SPI half-duplex modes.

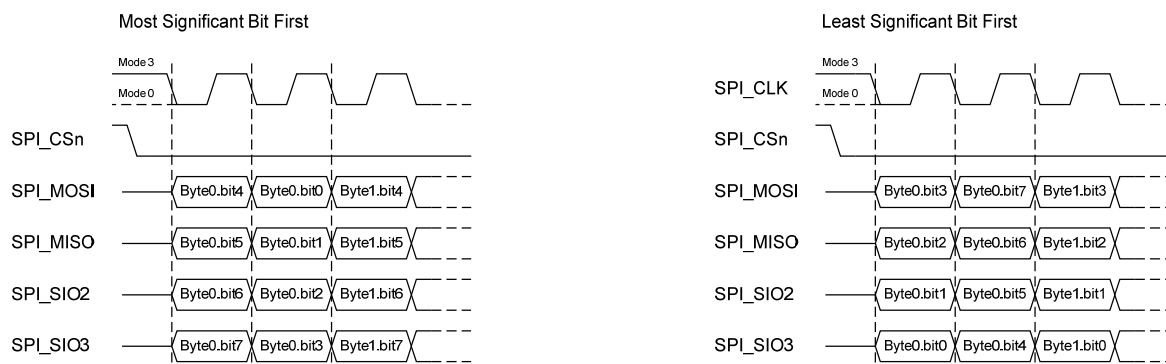


Figure 104: Transfer bit order: MSB or LSB

Additionally, QSPI mode allows you to select the byte order within a 32-bit word:

- little-endian (least significant byte first) or
- big-endian (most significant byte first)

To switch between little and big-endian byte order, use bit 'ms_byte_first' of register 'sqi_tcr'. Little endian byte order is selected by default.

Note: Byte and bit order can be configured separately and have nothing to do with each other, i.e. selecting big-endian byte order will also respect the bit order selected by bit 'ms_bit_first'.

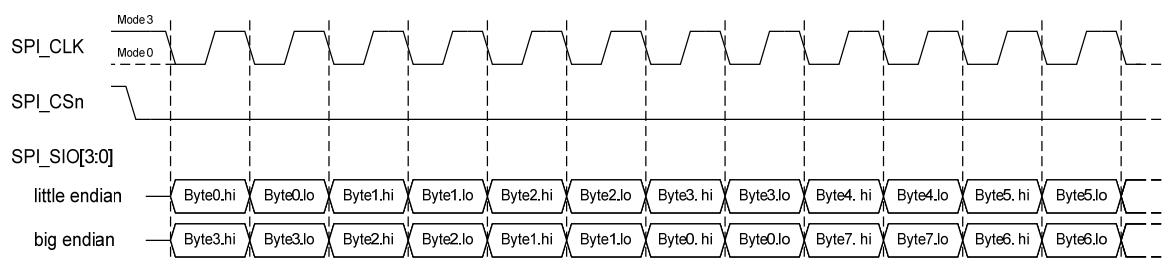


Figure 105: Transfer byte order: Little or big endianness, MSB first

For max. performance the QSPI FIFO is organized in 32-bit words. This means first read data is available when at least 4 bytes have been received. If less than 4 bytes are transferred, data is available as soon as the programmed number of bytes has been received.

Due to the 32-bit word organization of the FIFO in Dual and Quad SPI modes, the fill level also represents the number of 32-bit words stored in the corresponding FIFO. If you transfer data that is not a multiple of 4 bytes, the last word contains only the remaining bytes, but also counts as a 32-bit word in the FIFO.

Since QSPI modes transfer data in a half-duplex manner, the FIFO normally used for the opposite direction (i.e. RX FIFO when transmitting data) will be held in reset. Consequently, the corresponding fill level stays zero.

The following example will clarify the relation between the data seen on the wire and the organization in the FIFO: 6 bytes will be transferred (either received or transmitted). Most significant bit first transmission is assumed. Figure 106 shows the data on the wire.

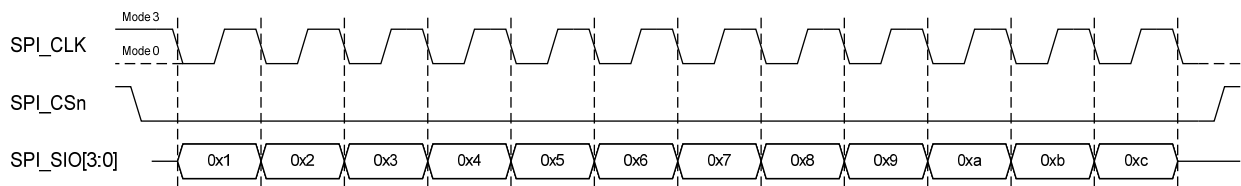


Figure 106: Data on wire for data placement example of a 6 byte transfer, MSB first

Table 41 shows the data received in the FIFO (or the data transmitted by writing to the FIFO). The table shows the data order within 32-bit words for little endian (least significant byte first) and big endian (most significant byte first) transmission.

Word in FIFO	Least significant byte first (little endian)				Most significant byte first (big endian)			
	Byte 3	Byte 2	Byte 1	Byte 0	Byte 3	Byte 2	Byte 1	Byte 0
0	0x78	0x56	0x34	0x12	0x12	0x34	0x56	0x78
1	-	-	0xbc	0x9a	-	-	0x9a	0xbc

Table 41: Data placement example of a 6-byte transfer, MSB first

Assuming the module has been configured and the data transfer has already been set up, the example data could be transmitted as follows:

In little endian mode: `s_ptSqiArea->ulSqi_dr = 0x78563412;`
`s_ptSqiArea->ulSqi_dr = 0x0000bc9a;`

In big endian mode: `s_ptSqiArea->ulSqi_dr = 0x12345678;`
`s_ptSqiArea->ulSqi_dr = 0x00009abc;`

Note: In big endian mode the remaining data of the last 32-bit word is also aligned correctly as in the little endian mode, i.e. upper bytes are not used when not transferring a full 32-bit word at the end.

For a description of module operation and programming details, see reference [1].

3.26.1.4.3 XiP (eXecute-in-Place)/SQIROM mode

The SQIROM mode is designed for eXecute-in-Place (XiP) operation on QSPI flash devices. Read access to a QSPI device is fully transparent to the user in this mode, i.e. the data of the QSPI device is mapped into a linear system memory area. The SQIROM address area can thus be used for the direct code execution without hard runtime requirements. Depending on the code style, the SQIROM provides similar performance or can even outperform parallel Flash devices.

For netX 51/52 SQIROM XiP address area has 16 Mbytes and is located:

- SQIROM XiP start address: `0x0C000000`
- SQIROM XiP end address: `0x0CFFFFFF`

From the point of view of the software the XiP part of the SQIROM module is a standard system slave which behaves like a read-only memory. Master modules (e. g. CPUs or DMA controllers) can read data from the external QSPI flash device by load operations. The whole related serial transfer to the QSPI device including the generation of the read command, the address and the chip-select signal will be performed by the SQIROM module without any further CPU interaction.

Note: Only SPI modes 0 (clock polarity and phase, both 0) and 3 (clock polarity and phase, both 1) are supported for SQIROM usage.

Note: The use of the linear address range for programming a serial Flash is not supported. A Flash device must be programmed by a normal SPI or QSPI peripheral function.

Standard SPI and QSPI transfer generation (section *Standard Motorola SPI mode* on page 186 and *Dual and Quad I/O interface* on page 189) is not available when SQIROM mode is enabled. SQIROM function is available only for devices connected to chip-select signal 0.

SQIROM access generation

The first read access to the SQIROM XiP address area will be transformed into a QSPI transfer including a header consisting of a 4-bit (Quad-IO) read command.

For the following access a new header is generated only if data is not read with incrementing addresses. Address-incrementing read access will continue the external transfer without overhead of a new header if there is no access to SQIROM, the last serial transfer will be stretched by deactivating the external clock, but not by deactivating the chip-select signal.

Note: The SQIROM access header depends on the 4-bit read command, the address format and the timing requirements of the used QSPI device. These parameters must be taken from the data sheet of the used QSPI device. All configurations beside SPI mode selection can be performed in register 'sqi_sqirom_cfg'.

Chip-select will be deactivated only if new addressing is required. The min. chip-select idle time between deactivating and activating the chip-select signal can be configured between 1 and 4 serial clock periods.

Serial clock rates can be configured by a programmable 8-bit divider of a 400 MHz clock.

The min. divider is 3. The following serial clock rates are available:

400 MHz/3 = 133 MHz, 400 MHz/4 = 100 MHz, 80 MHz, ... 1.55 MHz.

SQIROM device requirements

There are many devices on the market which work together with this module (e. g. SST 26 series, Winbond W25 series, Macronix MX25 series). Yet, there is no real QSPI device standard and devices vary in detail. The following list shows the requirements for using a QSPI device as SQIROM XiP device:

- The device must provide a 4-bit Quad-IO-read command which
 - provides linear continuous burst
 - consists of a command/mode byte followed by 5 to 8 address nibbles (Figure 107) or of 5 to 8 address nibbles followed by a command/mode byte (Figure 108). The address nibbles must be ordered most significant nibble first.
 - is generated completely in 4-bit / Quad-IO mode and does not fall back to any other mode when the chip-select signal becomes inactive
 - can be repeated immediately after the chip-select signal has become inactive and active again without requiring another command in between
 - requires max. 7 (0 to 7) dummy cycles for turnaround / before data cycles start
- SPI mode 0 or 3 must be supported
- Serial clock freezing must be allowed
- The min. chip-select idle time must not exceed 4 serial clock periods.

To support a wide range of available QSPI devices, two header (Quad-IO read command) formats are supported: Typically, the first byte of a transfer is a device-specific command byte which is typically followed by 5 address nibbles. Some devices require a special mode byte after the address nibbles instead of a command byte before the address. This module supports both header types controlled by bit 'addr_before_cmd'.

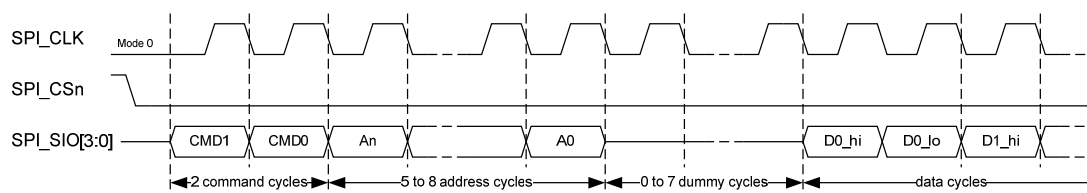


Figure 107: SQIROM header starting with command byte (bit 'addr_before_cmd' not set)

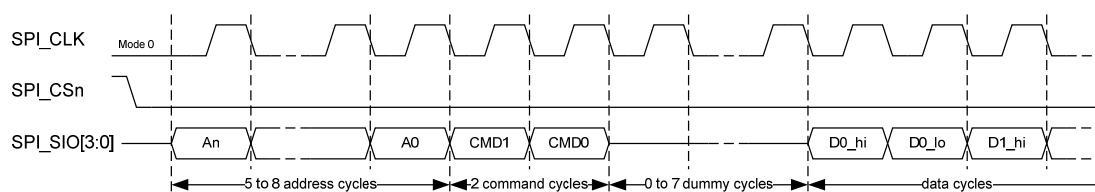


Figure 108: SQIROM header starting with address bytes (bit 'addr_before_cmd' set)

SQIROM data organization

For XiP, the data organization inside the QSPI Flash device is byte-organized. From the user's point of view, it is similar to a little-endian-organized 8-bit parallel Flash.

Transferring one byte requires 2 serial clocks. The higher nibble (bits 4 to 7) is transferred during the first cycle, the lower nibble (bits 0 to 3) during the second cycle. SIO0 always carries the least significant bit of a nibble, SIO3 the most significant bit.

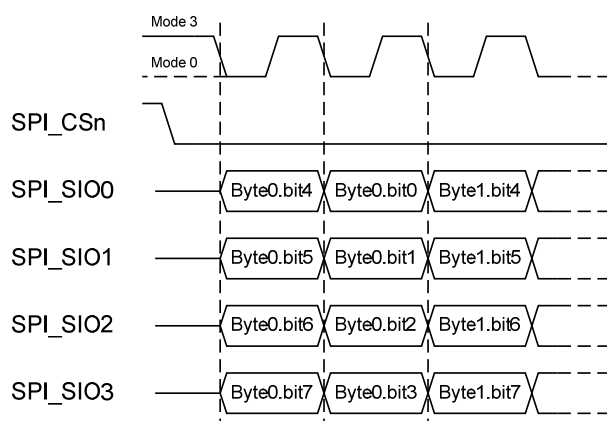


Figure 109: SQIROM data organization

Note: In 1-bit SPI mode SPI_SIO0 is SPI_MOSI (Master Out Slave In) and SPI_SIO1 is SPI_MISO (Master In Slave Out).

Example

The bytes stored in a QSPI flash device, starting at address 0x1000 with incrementing addresses, are:

0x21, 0x43, 0x65, 0x87, ...

Reading out a 32-bit data word using XiP from address 0x1000 will deliver on HRDATA of the internal AHB interface:

0x87654321

SQIROM access error detection

SQIROM-related error cases are flagged in module status register 'sqi_sr' and can be set to an IRQ event. The SQIROM module part is able to detect 3 different error types:

- Access-while-disabled-error: This error is flagged if a system master tries to access the SQIROM address area although the SQIROM function is disabled. In this case the master will receive invalid data, but it will not be stalled.
- Write access-error: This error will be flagged if a system master tries to write data to the read-only SQIROM address area. In this case the write data will be ignored.
- Timeout-error: The receive part of the SQIROM logic uses a feedback of the external serial clock. If no feedback can be detected, the timeout error will be generated. In this case the master will receive invalid data, but it will not be stalled. Possible reasons for the missing clock feedback: The serial clock IO is clamped to fix level externally or wrong programming of the global IO configuration.

3.26.2 SPI interface (SPI1)

3.26.2.1 Features

- Full master and slave functionality.
- SPI_CLK clock rates up to:
 - 50 MHz in master mode
 - 33 MHz in slave mode
- All four Motorola SPI modes supported in master and slave mode:
 - Clock polarity high or low
 - Clock phase 0 or 1
- Flexible data frame size from 4 to 16-bit data words
- 16-word deep FIFO for transmit data
- 16-word deep FIFO for receive data
- IRQ generation FIFO interaction
- DMA interface for receive and transmit data to minimize system CPU load
- Input signal oversampling and filtering for hazard suppression
- Extended chip select controlling:
 - Three individual external chip selects (may be demultiplexed externally to 8 signals)
 - Static or dynamic chip select handling configurable
- netX 100-compatibility mode and register set

Note: While the signals of SPI0 are directly available on dedicated pins (except chip select 2), the signals of SPI 1 and the CS2 of SPI 0 must be routed to MMIOs, using the multiplex matrix (see chapter 3.20) if SPI1 or CS2 of SPI0 is to be used.

3.26.2.2 Functional description

3.26.2.2.1 Block diagram

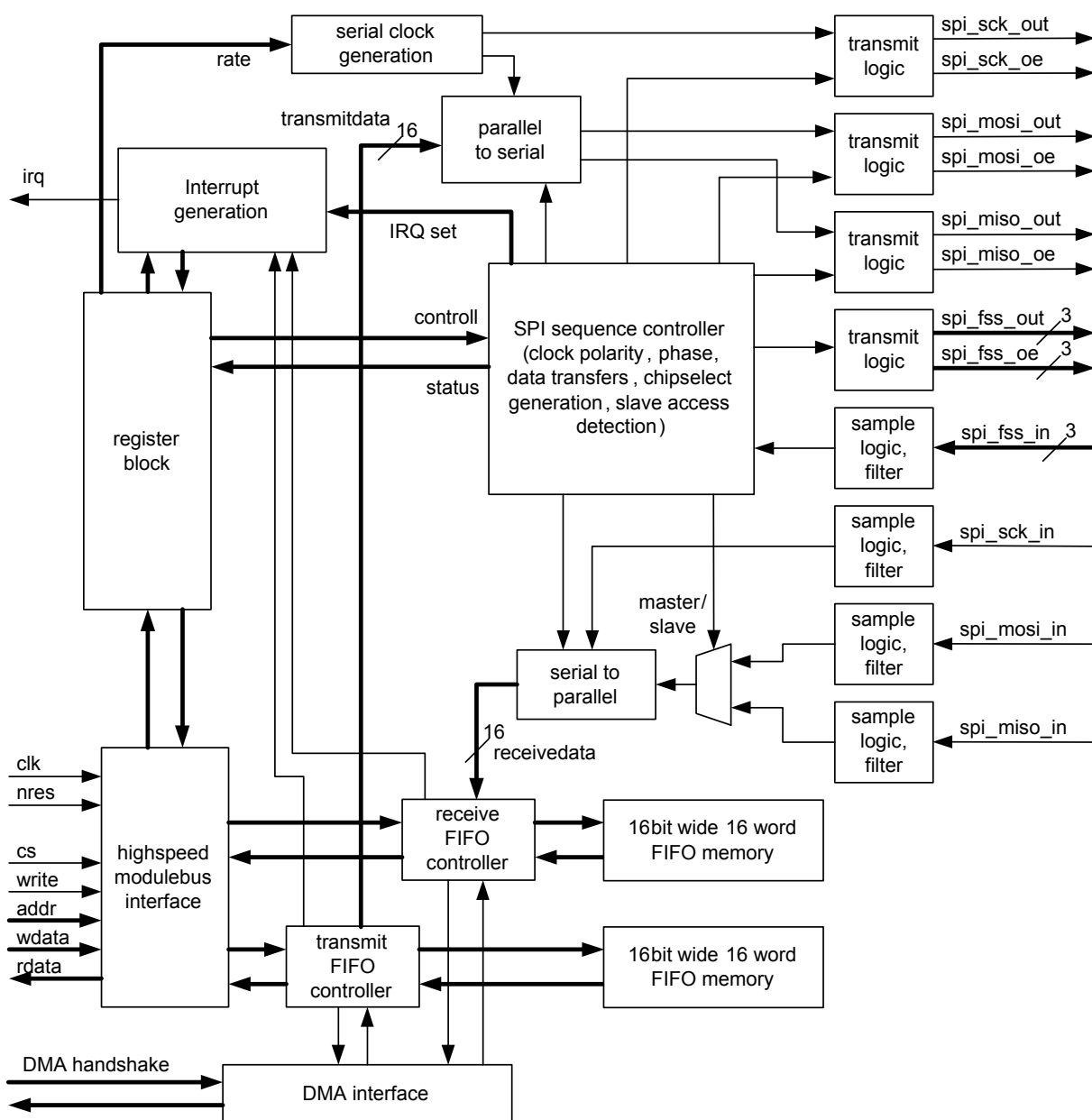


Figure 110: Block diagram of SPI module

3.26.2.2.2 SPI signals

The Motorola SPI interface defines four signals:

- Clock signal (SPI_CLK)
- Master transmit, slave receive data signal (SPI_MOSI)
- Master receive, slave transmit data signal (SPI_MISO)
- Low active chip select signal (SPI_CSn)

Data transfers are full duplex bidirectional. Data is always serialized MSB first.

3.26.2.3 SPI transfer format

SPI data transfer is activated by low active chip select signal. SPI signal generation and sampling states and timing can be selected by programmable SPO and SPH.

Note: For more flexibility in master mode, chip select can be generated dynamically by the SPI module state machine or can be static, generated by register `spi_cr1`, bit `fss_static`.

SPI clock polarity (SPO)

SPI clock polarity is the idle state of SPI clock signal SPI_CLK when no data is transferred.

SPI clock phase (SPH)

SPI clock phase controls SPI data generation and sample timing. If SPH 0 is selected, data on SPI_MOSI and SPI_MISO will be sampled on the first clock edge of SPI_CLK, if SPH 1 is selected the signal will be sampled on the second edge. Data is generated one edge before.

SPI frame format with SPO=0 and SPH=0

Clock idle state is low and data is sampled on the first (rising) edge of any SPI_CLK period. MSB data is generated when chip select is activated. Slave devices need chip select toggling at start of transfer to generate MSB data bit.

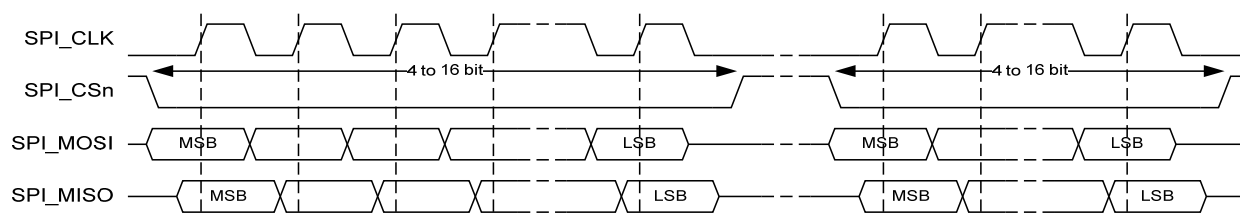


Figure 111: SPI SPO=0 and SPH=0 transfer (mode 0)

SPI frame format with SPO=0 and SPH=1

Clock idle state is low and data is sampled on the second (falling) edge of any SPI_CLK period. MSB data is generated on the first (rising) SPI_CLK edge. Slave devices do not need chip select toggling at start of transfer to generate MSB data bit.

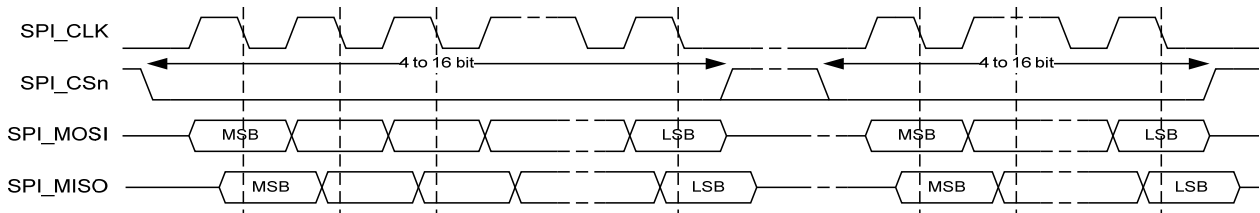


Figure 112: SPI SPO=0 and SPH=1 transfer (mode 1)

SPI frame format with SPO=1 and SPH=0

Clock idle state is high and data is sampled on the first (falling) edge of any SPI_CLK period. MSB data is generated when chip select is activated. Slave devices need chip select toggling at start of transfer to generate MSB data bit.

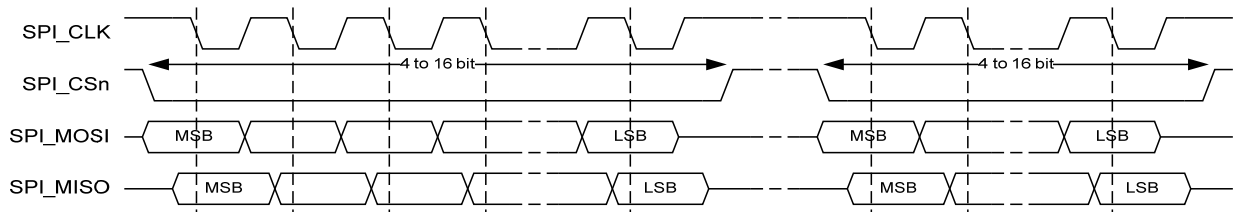


Figure 113: SPI SPO=1 and SPH=0 transfer (mode 2)

SPI Frame Format with SPO=1 and SPH=1

Clock idle state is high and data is sampled on the second (rising) edge of any SPI_CLK period. MSB data is generated on the first (falling) SPI_CLK edge. Slave devices do not need chip select toggling at start of transfer to generate MSB data bit.

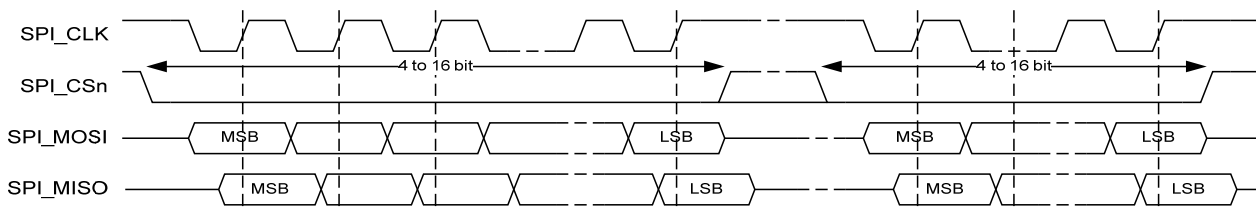


Figure 114: SPI SPO=1 and SPH=1 transfer (mode 3)

3.26.2.4 Typical applications

There are many SPI devices on the market today. Typical are low bandwidth devices as:

- Serial FLASHs
- MMC cards
- Various types of sensors
- Microcontroller Interface

The following figures show different possibilities of external SPI interconnection to SPI slave and master devices.

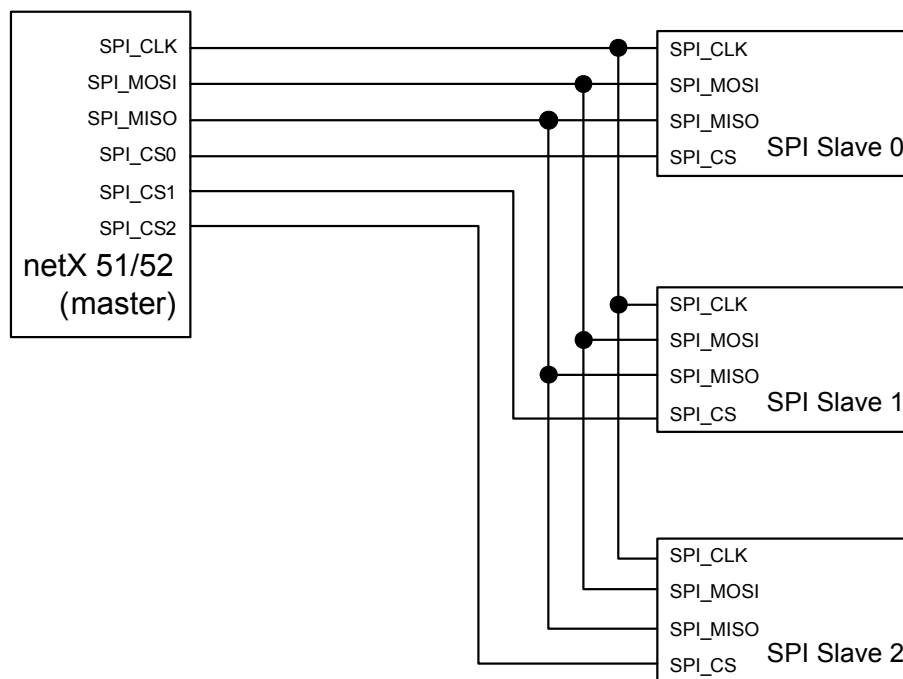


Figure 115: Standard external SPI interconnection

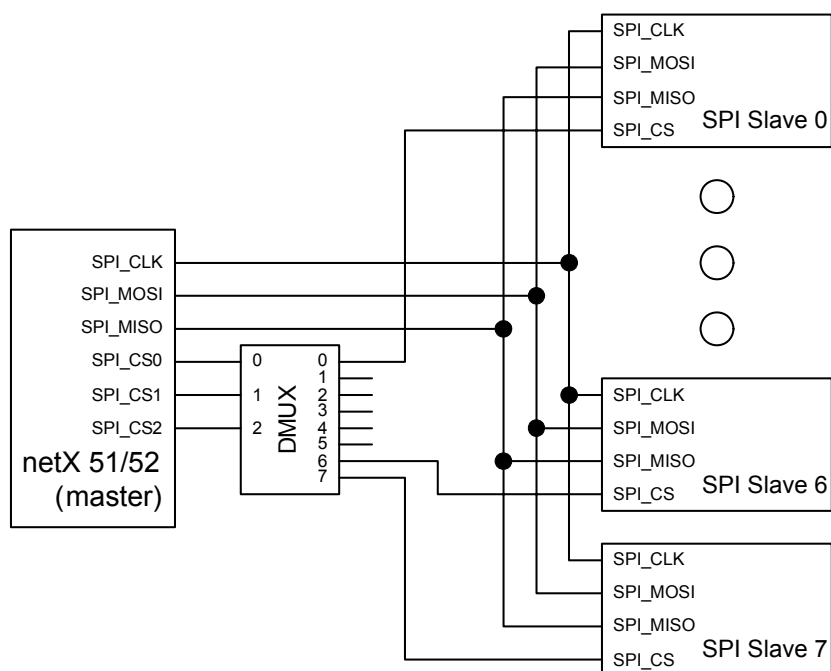


Figure 116: Extended external SPI interconnection with DMUX

3.27 MII interface

Functions: The MII Interface offers two modes:

- to connect an external PHY
- to connect another MII (to communicate with another host system)

The signals are:

Signal	
MII_TXER	Ethernet MAC Tx Error
MII_COL	Ethernet MAC Collision
MII_CRS	Ethernet MAC Carrier Sense
MII_RXD0	Ethernet MAC Rx Data 0
MII_RXD1	Ethernet MAC Rx Data 1
MII_RXD2	Ethernet MAC Rx Data 2
MII_RXD3	Ethernet MAC Rx Data 3
MII_RXDV	Ethernet MAC Rx Data Valid
MII_TXD0	Ethernet MAC Tx Data 0
MII_TXD1	Ethernet MAC Tx Data 1
MII_TXD2	Ethernet MAC Tx Data 2
MII_TXD3	Ethernet MAC Tx Data 3
MII_TXEN	Ethernet MAC Tx Enable
MII_TXCLK	Ethernet MAC Tx Clock
MII_MDIO	Ethernet MAC Management Data I/O Line
MII_MDC	Ethernet MAC Management Data Clock Line
MII_RXER	Ethernet MAC Rx Error
MII_RXCLK	Ethernet MAC Rx Clock

A HAL interface is provided to use the MII interface, see reference [5].

Note: The xPIC CPU is used for MII.

3.27.1 MII controller (ETH unit)

The netX-internal MII controller aka *ETH unit* controls the MII pins of the netX that are not connected to any XC unit. So it can work as an additional legacy Ethernet port besides any real-time Ethernet or fieldbus protocol running on the XCs.

The MII pins are driven/sampled according to Ethernet specification [7]. This also includes waiting for a free carrier before transmitting a frame as well as aborting a transmission in case of a collision. MII controller also appends the Frame Check Sequence (FCS) to every transmitted frame and validates any incoming FCS.

The MII was designed to connect an Ethernet MAC with a PHY. Since the ETH unit applies to both sides of the MII connection, it has two main operating modes: *MAC mode* and *PHY mode*.

By default the ETH unit serves as the MAC and all signals are named conforming to this use case. The MAC mode can drive an external PHY in Half and Full Duplex operation with a bit rate of 10 or 100 Mbit/s.

The PHY mode is only supported for Full Duplex operation with a bit rate of 100 Mbit/s. Note that the signals are still named after the MAC scheme and must therefore be cross-connected with the pins of the MAC device.

The correct signal connections for both modes are described in Figure 117. For a more detailed description of the mapping of the MII signals to the netX pins, see section 3.27.2.

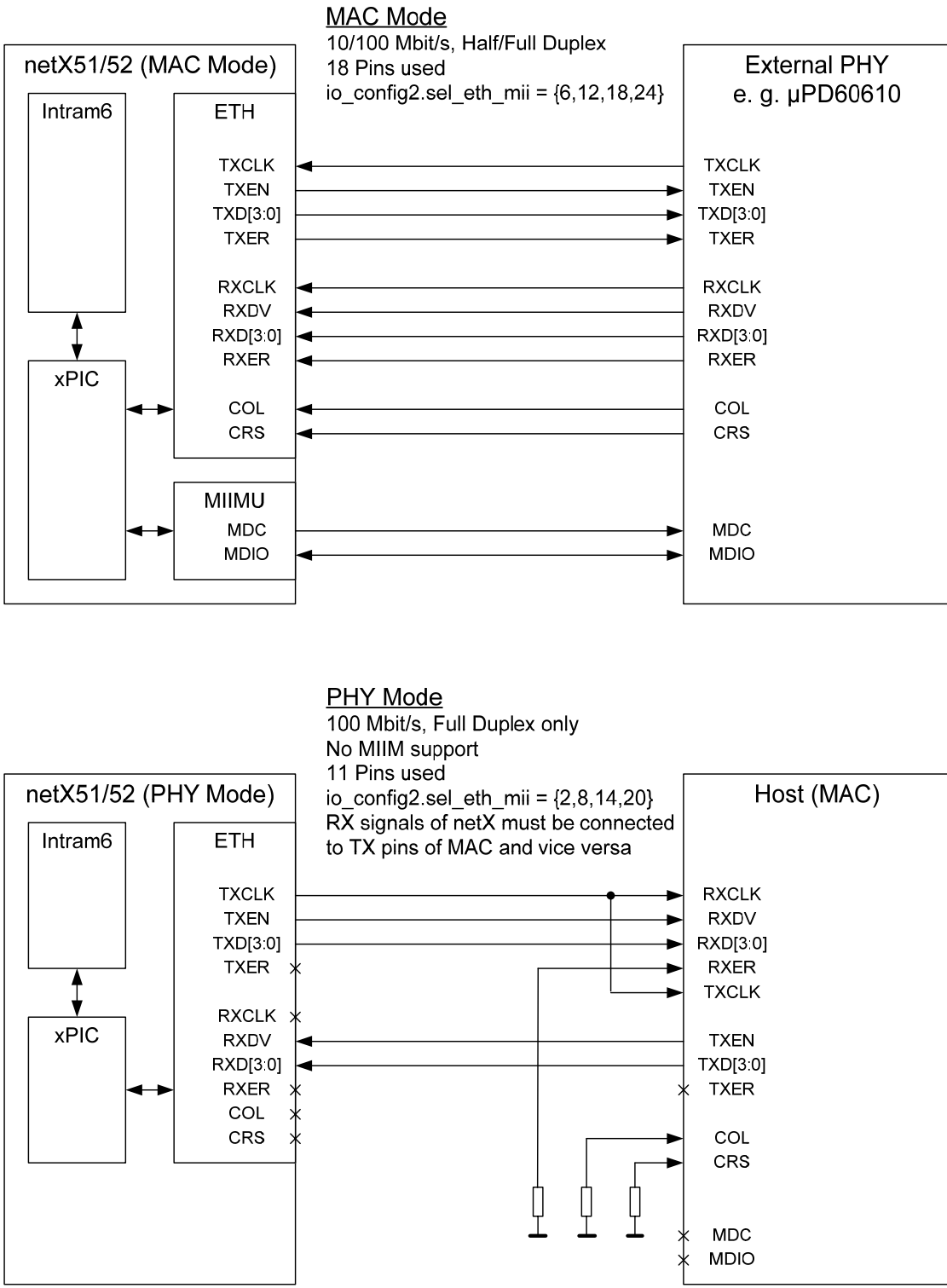


Figure 117: MII Connection in MAC and PHY mode

3.27.2 Pin assignment options

The MII data interface consists of 8 to 16 digital signals depending on the use case. The MII Feature Level (MFL) defines the signals to be used. Table 4 shows which signals are available for each MII Feature Level.

MFL	RXD0, RXD1	RXD2, RXD3	RXDV	RXCLK	RXER	TXD0, TXD1	TXD2, TXD3	TXEN	TXCLK	TXER	COL	CRS
off	-	-	-	-	-	-	-	-	-	-	-	-
0	X	-	X	-	X	X	-	X	X	-	-	-
1	X	X	X	-	-	X	X	X	X	-	-	-
2	X	X	X	X	-	X	X	X	X	-	-	-
3	X	X	X	X	X	X	X	X	X	-	-	-
4	X	X	X	X	X	X	X	X	X	-	X	X
5	X	X	X	X	X	X	X	X	X	X	X	X

Table 42: Available MII feature levels and signals

The MII signals can be routed to four different pin positions of the netX 51/52.

MII Feature Level and pin positions are selected by bit field *sel_eth_mii* of register *io_config2*. The possible value of *sel_eth_mii* ranges from 0 to 24.

Table 5 shows the MII pin positions resulting from the value of *sel_eth_mii*.

Signal	<i>sel_eth_mii</i> = 0	<i>sel_eth_mii</i> = 1 + MFL	<i>sel_eth_mii</i> = 7 + MFL	<i>sel_eth_mii</i> = 13 + MFL	<i>sel_eth_mii</i> = 19 + MFL
RXCLK	n/a	mmio8	hif_d28	hif_d28	hif_rdy
RXD0	n/a	mmio9	hif_d23	hif_dirq	hif_a3
RXD1	n/a	mmio10	hif_d27	hif_d13	hif_a4
RXD2	n/a	mmio11	hif_d30	hif_d14	hif_a5
RXD3	n/a	mmio12	hif_d31	hif_d15	hif_a6
RXDV	n/a	mmio13	hif_d18	hif_bhe3	hif_a7
RXER	n/a	mmio14	hif_d29	hif_d29	hif_bhe1
TXCLK	n/a	mmio15	hif_d20	hif_d20	hif_a13
TXD0	n/a	mmio16	hif_d21	hif_d21	hif_a8
TXD1	n/a	mmio17	hif_d22	hif_d22	hif_a9
TXD2	n/a	mmio18	hif_d23	hif_d23	hif_a10
TXD3	n/a	mmio19	hif_d24	hif_d24	hif_a11
TXEN	n/a	mmio20	hif_d25	hif_d25	hif_a12
TXER	n/a	mmio21	hif_d16	hif_d10	hif_a0
COL	n/a	mmio22	hif_ahi0	hif_d8	hif_a1
CRS	n/a	mmio23	hif_ahi1	hif_d9	hif_a2

Table 43: MII pin positions

3.28 GPIO

The netX 51/52 provides a total of 32 general purpose IOs. The basic function is to program each IO individually as input or output, inverted or non-inverted. There is one GPIO_CFGi register per IO pin. The outputs can be set individually or by a common register (GPIO_LINE).

The more advanced feature of the GPIO module is to combine a GPIO pin with 1 of 5 GPIO counters/timers or the global sys_time counter. In these modes each GPIO uses its own 32-bit TC (threshold-capture) register which allows many different PWM (TC is used as threshold) and capture functions (TC is used to store a counter value on external events):

- capture counter on edge or level,
- capture counter once or continuously,
- count external events (edges or duration of level),
- set PWM-threshold directly or at counter=0 (hazard free, requires 2nd TC register),
- use symmetric (triangle) or asymmetric PWM counter (sawtooth),
- DC-DC PWM: Generate PWM-signal depending on level of other GPIO-input,
- watchdog mode: Generate IRQ, if input does not change in programmed time,
- generate interrupt on external edge or level.

In netX 51/52 all GPIO registers are accessible via 2 channels on 2 address ranges. This allows the ARM and xPIC to work fully parallel and without the influence of the other processor on different GPIOs. The interrupt registers exist twice: INTLOGIC registers are connected to the ARM, INTLOGIC_MOTION registers to the xPIC.

Note: Due to pinout limitations of netX 51/52, GPIO pins are shared with IO-Link pins. Each IO-Link channel needs 4 pins, so IO-Link0 is shared with GPIO0...3, IO-Link1 is shared with GPIO4...7 etc. The desired behavior must be configured in register ASIC_CTRL-IO_CONFIG2.

Figure 118 shows the registers of the GPIO module.

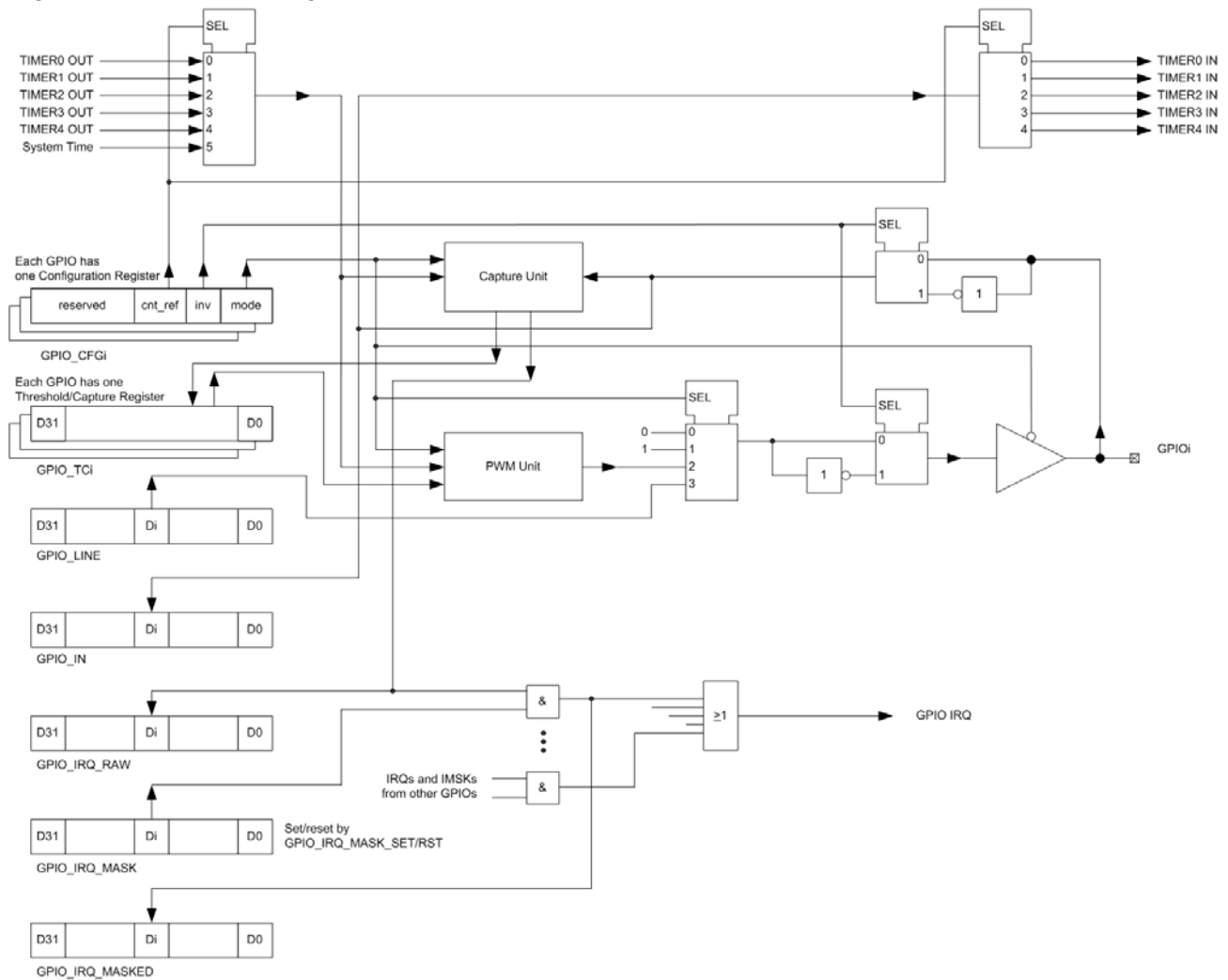


Figure 118: GPIO structure

3.28.1 Timer

The netX provides five 32-bit counters which can be configured to:

- count from zero to a max. value and backward (symmetric mode)
- count from zero to a max. value and set back to zero (asymmetric mode)
- single shot or count continuously
- generate an interrupt if it reaches zero
- count external events
- set back to zero by an external event
- capture the timer value by an external event
- generate a PWM signal by comparing the timer value with a threshold value

As external events, any GPIO can be assigned. This can be a rising or falling edge respectively a high or low level at the GPIO by setting the inverting bit at the GPIO configuration register.

The counter value can be read and overwritten any time.

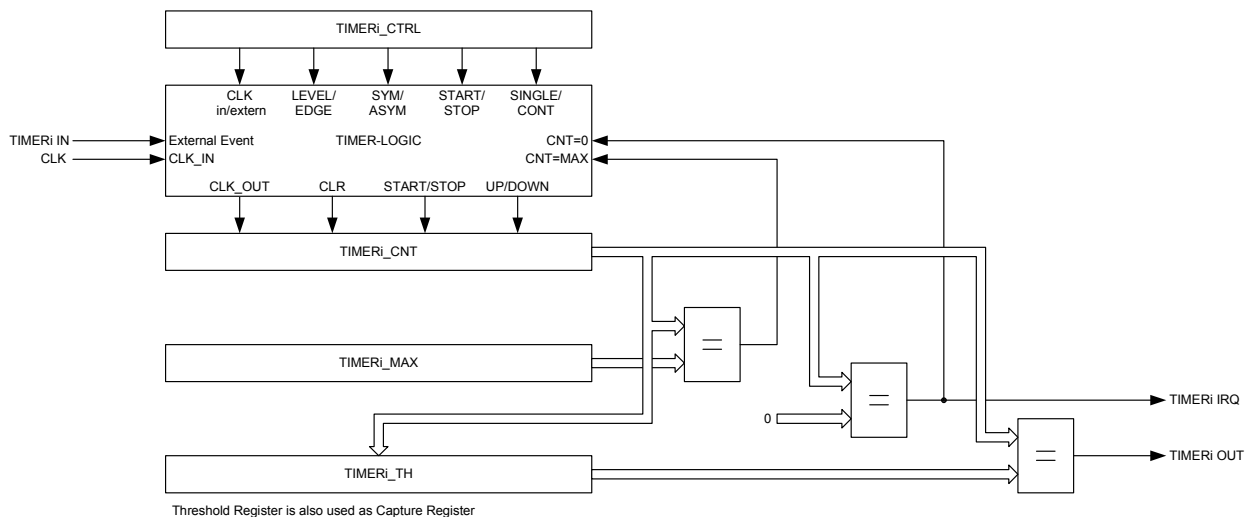


Figure 119: Timer function diagram

3.29 PIO

The netX 51/52 provides several programmable input / output lines. Each of the 62 PIO signals can be used as simple input or output without any additional features. PIO0-PIO7 are accessible via the multiplex matrix and usually drive fieldbus and RT Ethernet status LEDs. The remaining 54 PIOs (PIO32-PIO84) are shared with host interface pins. (PIOs 8 - 31 do not exist in the netX 50).

PIO 0 ... 7

The first 8 PIO signals are controlled by three registers: 'PIO_IN - PIO Input Register', 'PIO_OUT - PIO Output Register' and 'PIO_OE - PIO Output Enable Register'. In order to use PIOs 0 – 7, they need to be mapped to MMIO pins'.

PIO 32 ... 85

These programmable input/output pins are multiplexed with the host interface and are hence also referenced as “HIF-PIOs”. If the host interface is switched to 'I/O mode', all pins work as normal input/output lines. If the extension bus or DPM mode is selected, unused interface lines can be used as programmable input/output pins. It is possible to switch each pin between host interface operation and programmable input/output pin via the mode registers. The following list shows all netX control registers for programming. For a detailed description of all registers, see reference [1].

DPM_ARM_IF_CFG0 This register selects the different operating modes for the host interface. The interface may be switched to 'Disable', 'Extension Bus', 'Dual-Port Memory' or 'IO Mode' operation.

DPM_ARM_IO_DATA0

These data registers set the output data for all HIF-PIOs with enabled

DPM_ARM_IO_DATA1

output drivers or contain the sampled input data of all HIF-PIOs.

DPM_ARM_IO_DRV_EN0

The driver enable register controls the output drivers of each pin.

DPM_ARM_IO_DRV_EN1

If the DPM or extension bus function is enabled, the setting of the driver enable bit will be ignored. After power-on reset all drivers will be disabled.

DPMAS_IO_MODE0 The mode registers allow you to configure each pin to either I/O mode or

DPMAS_IO_MODE1 Host interface mode. If 'I/O Mode' is selected in register **DPM_ARM_IF_CFG0** (switches all pins to I/O mode), the settings of these mode registers have no effect. The mode of each pin can be changed anytime, also during operation.

The **IN_CONTROL[1:0]** flags control the sampling of the input pin signal. Four modes can be selected.

Mode '00' (reset condition) samples all input signal lines shortly after the rising edge of the power-on reset signal allowing configuration by external pull-up or pull-down resistors. The firmware can read the input data register after starting.

In mode '01' the input latches are always enabled and the data is stored in internal flip flops, clocked by the system clock.

In mode '1x' the input flip flops will be enabled only if the PIO77 pin has a high (mode 11) or low (mode 10) level. Data will be stored in the input flip flops controlled by the internally synchronized PIO77 pin.

Power-on reset sampling of PIO 32–85 pins

After releasing the power-on reset signal, the state of all 54 host interface pins is sampled and stored in two registers. This allows the selection of different device configurations by using pull-down resistors (pins have weak internal pull-ups) and implementing the corresponding configurations in the firmware. The following figure shows the power-on reset schematic of data sampling.

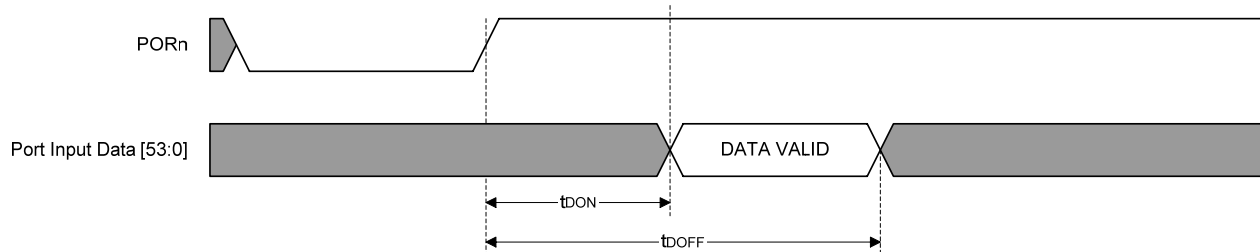


Figure 120: Power-on reset sampling of PIO 32–85 pins

Parameter	Description	Value	Dimension
t _{DON}	Max. time between activation of the power-on reset signal and valid (external) data supplied to the PIOs	11	ms
t _{DOFF}	Min. time between activation of the power-on reset signal and deactivation of (external) data supplied to the PIOs	11.5	ms

Note: The timing values and the figure above are only relevant when driving data signals from an external data source instead of using (the internal) pull-up or (external) pull-down resistors. In that case, users have to make sure that the host interface configuration that might be carried out by firmware or boot loader (DPM boot mode or extension bus boot mode) will not collide with the data source connected to the PIOs!

Input sampling of PIO 32–85 pins

All input pins are synchronized to the internal system clock. Besides continuous data sampling, it is also possible to enable/disable all input flip flops via the PIO77 pin and initiate data sampling. When this external signal is active, the input flip flops are enabled. The signal will be synchronized to the system clock as well. The following figure shows the input signal delay at the two different sampling modes.

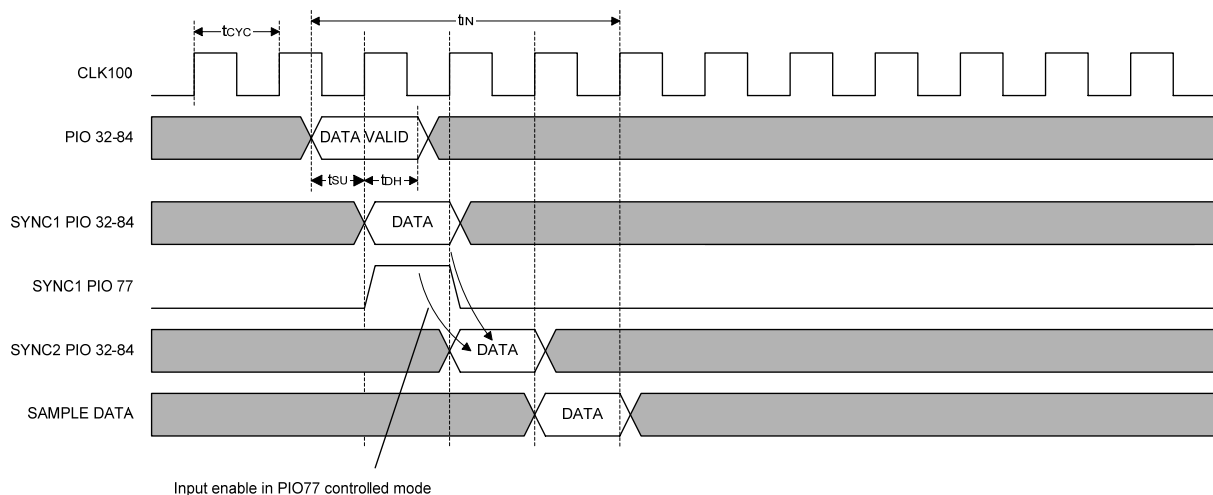


Figure 121: Input sampling of PIO 32–85 pins

Parameter	Description	Value	Dimension
t _{CYC}	Internal 100 MHz clock cycle	10	ns
t _{SU}	Input setup time before internal rising clock edge	< 1	CYC
t _{DH}	Hold time after internal rising clock edge	< 1	CYC
t _{IN}	Synchronization delay for input signals until valid internal sampling data	3...4	CYC

3.30 Ethernet interface

The netX 51/52 contains two Ethernet MACs with integrated PHYs. They support:

- 10Base-T / 100Base-TX
- 100Base-FX with external drivers
- Auto-Negotiation
- Auto-Crossover
- Auto-Polarity

They are fully compliant with IEEE 802.3 / 802.3u to run the protocols:

- PROFINET RT and IRT
- EtherNet/IP
- Open Modbus/TCP

Additional the Ethernet MAC includes special logic to support:

- Time synchronization based on IEEE 1588
- POWERLINK
- EtherCAT
- sercos third generation

For a schematic about how to connect one or two Ethernet interfaces, see reference [2].

Important: With applications that do not make use of the Ethernet interface, the PHY power must be connected (power has to be supplied and reference resistor has to be connected)! The schematic is shown in reference [2].

Real-time Ethernet

In addition to the standard Ethernet function, the netX Ethernet channels support all current real-time Ethernet protocols. The different protocols use special functions:

POWERLINK uses the HUB function to forward Ethernet telegrams. The protocol stack running on the ARM CPU helps avoid access conflicts on the Ethernet. To increase performance, the xPEC immediately answers a poll-request telegram with a poll-response telegram, without the interrupt latency that would occur if the ARM CPU were involved.

EtherCAT forwards the Ethernet data immediately from one port to another and takes out or fills in the local data. The xMACs are therefore directly connected with each other. Moreover, they have a special filter function to identify the local data within Ethernet frames. The xPEC is responsible for the transfer into and out of the ARM memory.

Current features:

- Slave implementation
- 6.5 KByte process data memory
- 8 SyncManager units
- 8 FMMUs (fieldbus memory management unit, support only byte-wise mapping)
- Mailbox functionality for CANopen over EtherCAT, Ethernet over EtherCAT
- Powerful AL controller integrated

sercos uses the same filters and the state machines which are defined by the sercos international e.V.

EtherNet/IP uses only the standard Ethernet functions of one channel. However, for motion control with synchronization down to a microsecond, the distributed time according to the international Standard IEEE 1588 is used. This means, to measure transmit and receive time by hardware at the MII interface between xMAC and PHY, using a special protocol, this information is exchanged between the network devices and the local system time is adjusted to a common network time.

The real-time communication of PROFINET uses a switch function and priority control according to IEEE 802.1Q. For this the xPECs owns local memory to manage the routing data and a state machine to control the telegram transfers. The transfer time through the internal switch is less than 3 μ s.

For motion control, PROFINET uses isochronous transmission. This requires synchronization of the local time by IEEE 1588 and transmission of the telegrams results according to a time table. This time control is another mode of the xPEC state machine. The netX is fully compliant to the PROFINET IRT specification.

These features allow device manufactures to use a unified hardware, activating the special real-time Ethernet functions by appropriate software, as required.

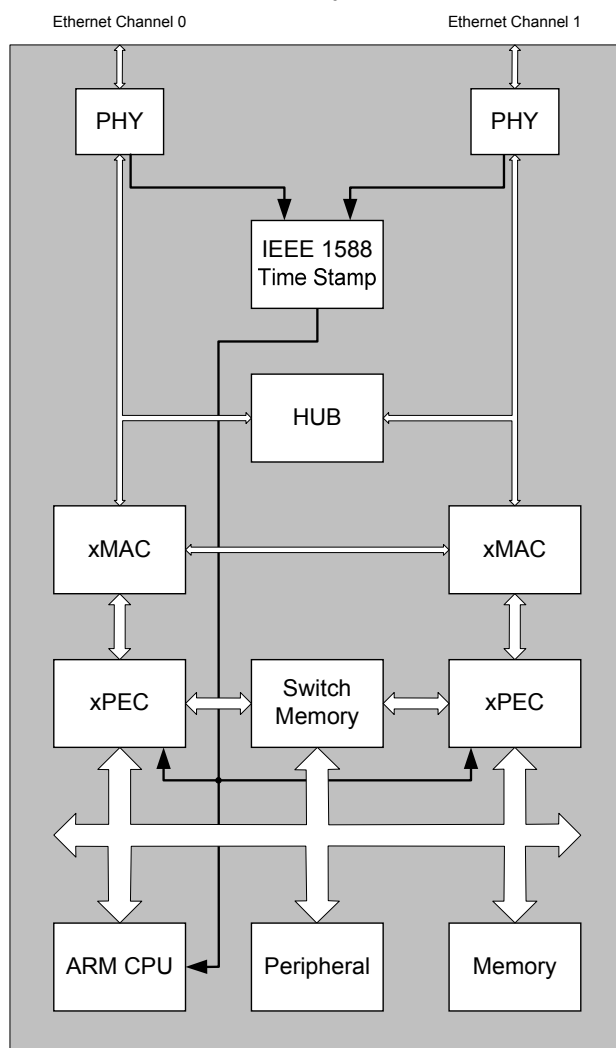


Figure 122: Block diagram of the special real-time Ethernet features

Note: For better understanding, the data switch is not shown.

3.31 Fieldbus interface

The XMAC/XPEC units of the netX can operate as fieldbus controllers (one XMAC and one XPEC per fieldbus channel), for virtually any existing and future fieldbus system, like:

- CANopen Slave
- CC-Link Slave
- DeviceNet Slave
- PROFIBUS DP Slave

Different systems can be combined.

For schematics about how to connect a fieldbus interface, see reference [2].

4 Electrical specification

4.1 Absolute max. ratings

Stresses beyond the following ratings may cause permanent damage to the device. Note that these stress ratings do not imply the functional operation of the device, and note that exposure to these ratings for extended periods of time may affect device reliability.

Symbol	Parameter	Conditions	Ratings	Unit
V _{DDC}	Power supply, core voltage		-0.5 to 2.0	V
V _{DDIO}	Power supply, IO voltage		-0.5 to 4.6	V
USB_V _{DDC}	USB power supply, core voltage		-0.5 to 2.0	V
USB_V _{DDIO}	USB power supply, IO voltage		-0.5 to 4.6	V
OSC_V _{DDC}	Oscillator power supply, core voltage		-0.5 to 2.0	V
PHY_V _{DDCART}	PHY power supply		-0.5 to 2.0	V
PHY_V _{SSACP}	PHY power supply		-0.5 to 2.0	V
PHY_V _{DDIOAC}	PHY power supply		-0.5 to 4.6	V
PHY_V _{DDIOAT}	PHY power supply		-0.5 to 4.6	V
PHY_V _{DDD}	PHY digital power supply		-0.5 to 2.0	V
V _I	Input voltage	$V_I < V_{DDIO} + 0.5 \text{ V}$	-0.5 to 4.6	V
V _O	Output voltage	$V_O < V_{DDIO} + 0.5 \text{ V}$	-0.5 to 4.6	V
I _O	Output current	6 mA type	±21	mA
		9 mA type	±29	mA
		USB type	±37	mA
T _A	Operating ambient temperature		-40 to +85	°C
T _j	Junction temperature		-40 to +125	°C
T _{stg}	Storage temperature		-65 to +150	°C

4.2 Power-up and down sequencing

All power supplies of the netX chip must ramp up or down within 100 ms time.

Important: To make sure that no I/O pins drive, the voltages must be applied in the following order: The core supply voltage must have reached at least approx. 0.8 V before the IO supply voltage is applied.

All voltages must have reached the $0.9 \cdot VDD$ condition not later than 100 ms after the first supply started to ramp up (time measured from $0.1 \cdot VDD$ condition). The following figure shows details of powering up and down.

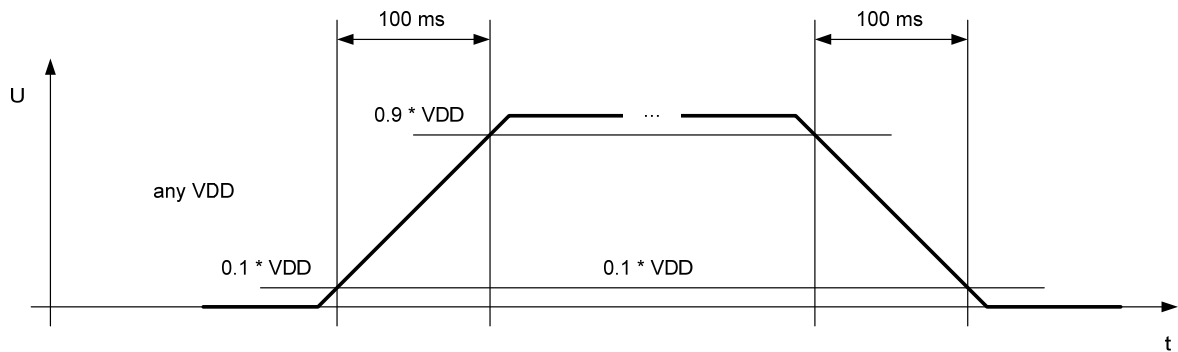


Figure 123: Power-up and down sequencing

Note: All power supply voltage pins have to be connected to the power supplies even if certain peripheral parts of the netX chip are not used!

4.3 Power consumption / Power dissipation

The typical power consumption of the netX 51 and netX 52 is approx. 1.5 Watts (integrated PHYs in use) or 0.8 Watts (PHYs turned off). The following tables give an overview of power consumption of single chip modules and some standard applications.

4.3.1 Power consumption of netX 51 and netX 52

netX power consumption at typical setups:

Symbol	Parameter	Typ.	Unit
P _{BOOT}	Serial boot mode, no external communication	0.5	W
P _{3ETH_SDRAM}	2 Port Full Duplex Ethernet and 1 MAC (with external PHY), SDRAM	2.15	W
P _{3ETH}	2 Port Full Duplex Ethernet and 1 MAC (with external PHY)	2.05	W
P _{2ETH}	2 Port Full Duplex Ethernet	1.5	W
P _{1ETH}	1 Port Full Duplex Ethernet	1	W
P _{FB_SPI}	Fieldbus 12 MBaud, one channel, SPI-FLASH	0.7	W

Power supply current overview:

Symbol	Parameter	Conditions	Typ.	Unit
I _{DD_3V3}	Complete power supply 3.3V	3.3 V	160	mA
I _{DD_1V5}	Complete power supply 1.5V	1.5 V	660	mA
I _{DDC}	Power supply current, netX core	1.5 V	420	mA
I _{DDIO}	Power supply current, netX IO	3.3 V	50	mA

4.4 AC/DC specifications

4.4.1 DC parameters

The following table describes the standard pad cells of the netX 51/52. The special pad cells for crystal, USB and PHYs are described in the following chapters.

Symbol	Parameter	Conditions		Min.	Typ.	Max.	Unit
V _{DDC}	Power supply voltage, core			1.425	1.5	1.65	V
V _{DDIO}	Power supply voltage, IO			3.0	3.3	3.6	V
V _N	Negative trigger voltage			0.6		1.8	V
V _P	Positive trigger voltage			1.2		2.4	V
V _H	Hysteresis voltage			0.3		1.5	V
V _{IL}	Input voltage, low			0		0.8	V
V _{IH}	Input voltage, high	3.3 V		2.0		V _{DDIO}	V
slew _{ri}	Input rise (normal input)			0.015			V/ns
slew _{fa}	Input fall time (normal input)			0.015			V/ns
slew _{ri}	Input rise (Schmitt input)			0.27			V/ms
slew _{fa}	Input fall time (Schmitt input)			0.27			V/ms
I _{OZ}	Off-state current	V _O = V _{DDIO} or GND				±10	µA
I _{OS}	Output short circuit current					-250 Note 1)	mA
I _{LI}	Input leakage current	V _I = V _{DDIO} or GND	Normal input		±0.1	±10 Note 2)	µA
		V _I = GND	pull-up 50 kΩ	-37	-103	-253	µA
		V _I = V _{DDIO}	pull-down 50 kΩ	26	73	175	µA
I _{OL}	Output current, low	V _{OL} = 0.4V	6 mA type	6.0			mA
			9 mA type	9.0			mA
I _{OH}	Output current, high	V _{OH} = 2.4V	6 mA type	-6.0			mA
			9 mA type	-9.0			mA
V _{OL}	Output voltage, low	I _{OL} = 0 mA				0.1	V
		I _{OL} = 6 mA (6 mA type)				0.4	V
		I _{OL} = 9 mA (9 mA type)				0.4	V
V _{OH}	Output voltage, high	I _{OH} = 0 mA		V _{DDIO} -0.1			V
		I _{OH} = 6 mA (6 mA type)		2.4			V
C _{IN}	Pin capacitance	Input buffer		2	4	6	pF
C _{IO}	Pin capacitance	Output and bidirectional buffer		2	4	6	pF
R _{PU}	Resistance of internal 50k pull-up resistor	V _{DDIO} = 3.3 ± 0.3V T _A = -40 to +85°C		14.2	31.9	80.7	kΩ
R _{PD}	Resistance of internal 50k pull-down resistor	V _{DDIO} = 3.3 ± 0.3V T _A = -40 to +85°C		20.6	44.9	116.4	kΩ

Notes

1. Output short circuit time no longer than one second and only one pin of the chip!

2. This value only applies to (also internally) unconnected standard pad cells. As most netX pins are equipped with internal pull-up or pull-down resistors, also refer to chapter 5.1 or 6.1 (Signal Definitions) to determine the actual leakage current of a specific pin!

Symbol definition:

Symbol	Description
V_{DDC}, V_{DDIO}	voltage range for normal logic operations that occur when $V_{SS} = 0 \text{ V}$
V_N	input level at which the output level is inverted when the input is changed from high-level to low-level side
V_P	input level at which the output level is inverted when the input is changed from low-level side to high-level side
V_H	differential between positive and negative trigger voltage
V_{IL} :	voltage which must be applied to the input pins to guarantee a logical low input level
V_{IH} :	voltage which must be applied to the input pins to guarantee a logical high input level
slew_{ri}	slew rate rise time
slew_{fa}	slew rate fall time
I_{OZ}	current that flows from the power supply pins when the rated power supply voltage is applied when a 3-state output has high impedance
I_{OS}	current that flows when the output pins are shorted (to GND pins) when output is at high level The output short circuit must not exceed more than one second and is only for one pin
I_{LI}	current that flows via an input pin when a voltage is applied to that pin
I_{OL}	current that flows to the output pins when the rated low-level output voltage is being applied
I_{OH}	current that flows from the output pins when the rated high-level output voltage is being applied
V_{OL}	output voltage at low level and when the output pin is open
V_{OH}	output voltage at high level and when the output pin is open

Output buffer output current (I_{OL} , I_{OH})

The cell base IC output current is defined at the conditions of $V_{OL} = 0.4 \text{ V}$ and $V_{OH} = 2.4 \text{ V}$ in a 3.3 V output buffer. Since current applications use different V_{OL} and V_{OH} values, the following coefficient serves to estimate the I_{OL} and I_{OH} characteristics according to the conditions of use.

$$V_{OL} = 0.4 \text{ to } 0.6 \text{ V}, V_{OH} = (V_{DD} - 0.4 \text{ V}) \text{ to } (V_{DD} - 0.6 \text{ V})$$

Since I_{OL} and I_{OH} change almost in proportion to the output voltage, use the following method for direction approximation:

$$I_{OL}' = I_{OL} \times V_{OL} / 0.4 \text{ (mA)}$$

$$I_{OH}' = I_{OH} \times (V_{DD} - V_{OH}) / 0.6 \text{ (mA)}$$

I_{OL} : I_{OL} specification when $V_{OL} = 0.4 \text{ V}$

V_{OL} : V_{OL} value being used

I_{OH} : I_{OH} specification when $V_{OH} = 2.4 \text{ V}$ (3.3 V output buffer)

V_{OH} : V_{OH} value being used

The MIN, TYP and MAX designators in the graph show the curve under the following conditions:

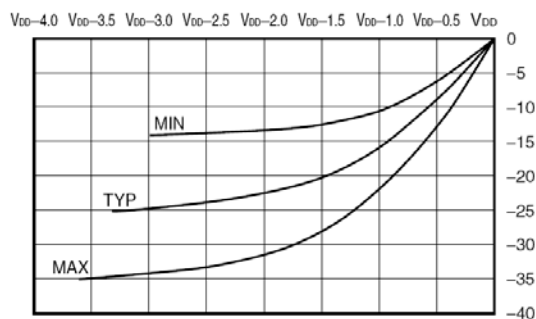
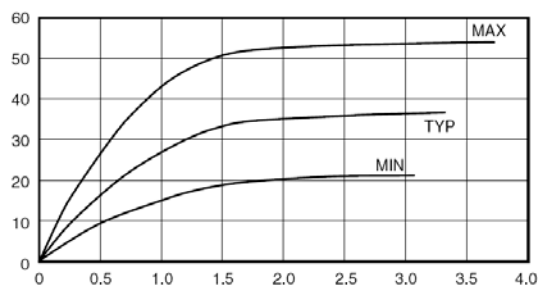
$$\text{Min.: } V_{DD} = 3.0 \text{ V (3.3 V output buffer), } T_J = 125^\circ\text{C}$$

Typ.: $V_{DD} = 3.3 \text{ V}$ (3.3 V output buffer), $T_J = 25^\circ\text{C}$

Max.: $V_{DD} = 3.6 \text{ V}$ (3.3 V output buffer), $T_J = -40^\circ\text{C}$

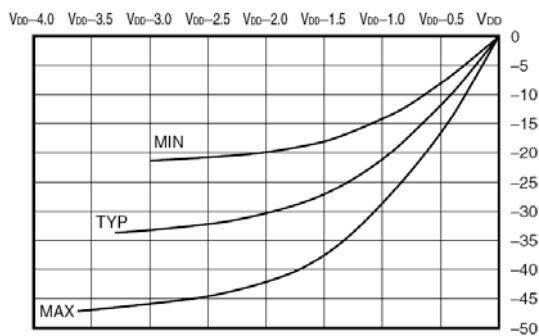
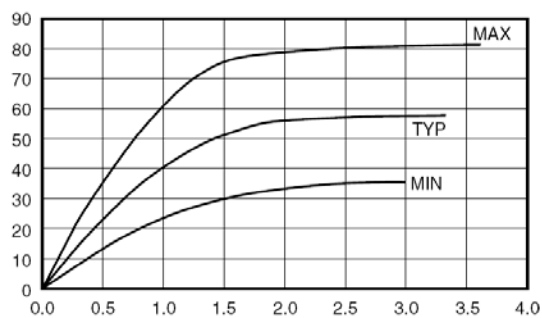
6 mA output driver

(typical block type: TDOPAC33NN06, TDOPAC33NL06)



9 mA output driver:

(typical block type: TDOPAC33NN09, TDOPAC33NL09)



4.4.2 System oscillator / PLL

The system oscillator circuit, along with the internal PLL, generates all internal clocks of the netX 51/52 chip. For clock generation, either a quartz crystal (oscillation mode: fundamental) may be used with the internal oscillator circuit or a quartz oscillator connected to the clock input pin.

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Unit
OSC_V _{DDC}	Oscillator power supply core		1.425	1.5	1.65	V
V _{IL}	Input voltage, low	Note 1)	0		0.8	V
V _{IH}	Input voltage, high	Note 1)	2.0		V _{DDIO} + 0.5	V
V _{OL}	Output voltage, low	Note 1)			0.4	V
V _{OH}	Output voltage, high	Note 1)	2.0	3.0	V _{DDIO}	V
f _{CLK}	System clock frequency			25		MHz
	System clock tolerance		-100		+100	ppm
	System clock duty cycle		40	50	60	%
	System jitter tolerance			20		ps [RMS]
T _{CYC}	System clock cycle time			40		ns
T _{high}	System clock high time		14		20	ns
T _{low}	System clock low time		14		20	ns
	System clock slew rate		0.5			V/ns
C _{IN}	Pin capacitance	input buffer	2.5	3.5	4.5	pF
C _{OUT}	Pin capacitance	output buffer	2.7	3.7	4.7	pF
OSC_I _{VDDC}	Oscillator current	1.5 V			6	mA

Note: These DC parameter values do not apply to the dynamical system of a crystal circuit. For crystals the circuit should be designed in such a way that the levels are kept within the limits of the absolute max. ratings (-0.5 V to V_{DDIO} + 0.5 V).

4.4.3 Power-on reset/reset input

The netX provides two reset inputs:

- The asynchronous reset input with a Schmitt-trigger input cell can be used for the mandatory power-on reset of the chip.
- The synchronous reset input (optional) can be used to perform a system reset on the powered up and running chip.

The picture and table below show the behavior of the reset system and the timing requirements for the reset input signals.

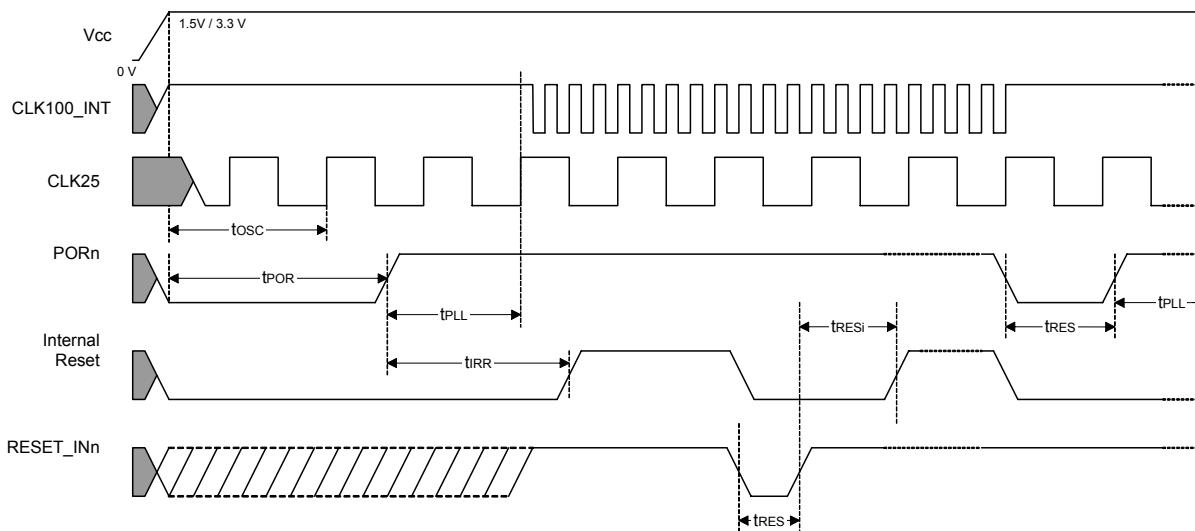


Figure 124: Reset input signal timing requirements

Parameter	Description	Value	Dimension
t _{OSC}	Startup time for the internal oscillator system (25 MHz)	< 10	ms 2)
t _{POR}	Min. duration of external power-on reset signal	> t _{osc}	2)
t _{PLL}	Time between deactivation of PORn and start of internal PLL	10.5	ms
t _{IRR}	Time between deactivation of PORn and release of internal reset	11.5	ms
t _{RES}	Min. reset pulse width for running chip	10	ns
t _{RESi}	Active time of internal reset after releasing RESET_INn	640	ns (64 clks)

Notes:

1. The use of RESET_INn is optional and shown in this diagram for reference only. It is not a required part of the startup reset sequence.
2. The oscillator startup time depends on the ambient temperature and crystal type. The given values are typical values evaluated with a certain crystal. They may differ from the values experienced with the user's application. The external reset signal source (reset generator) may release the PORn signal before the system oscillator has settled completely. In that case, however, the values for t_{PLL} and t_{IRR} may diverge from the values above since they derive from counter values and the counters are clocked by the system oscillator.

4.4.4 MMIOs

For relationship between different netX 51/52 MMIO signals, MMIO output timing is related to a virtual internal clock root and MMIO input timing to a virtual external clock. The MMIO relation can be calculated as described in the example below.

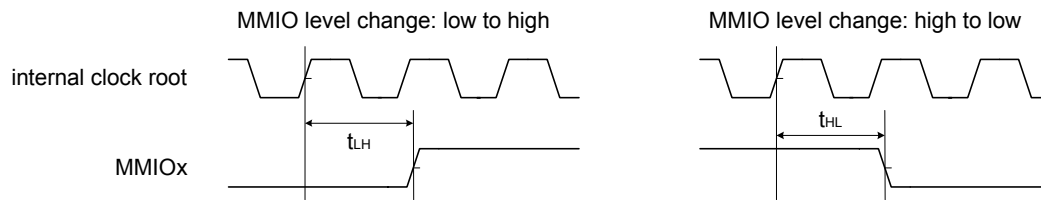


Figure 125: MMIO level change

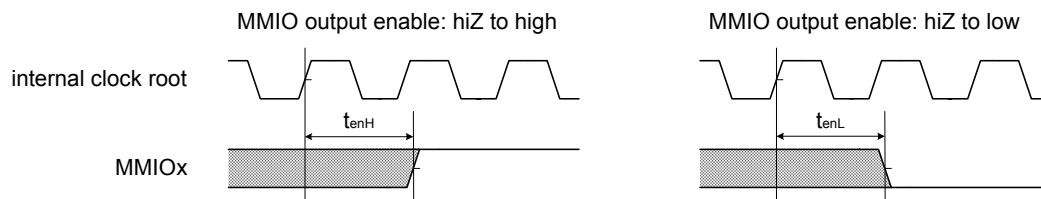


Figure 126: MMIO output enable

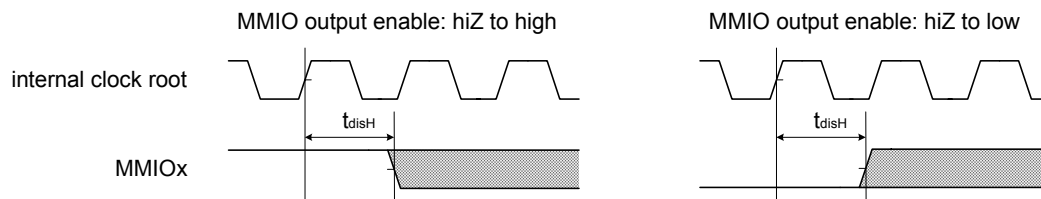


Figure 127: MMIO output disable

MMIO output timing relationship calculation example

In this example, MMIO12 and MMIO16 are used in relation to each other. Scope of interest is rising edge output timing. The timing table at the end of this chapter provides the following t_{LH} –values:

MMIO12: t_{LH_MMIO12} 10.6 ns

MMIO16: t_{LH_MMIO16} 12.5 ns

That means: If MMIO12 and MMIO16 change from low to high by netX 51/52 configuration register setting at the same moment (i.e. the same internal system clock cycle, e.g. by a single configuration register access), MMIO12 changes externally 1.9 ns earlier from low to high than MMIO16 ($t_{LH\_MMIO16} - t_{LH\_MMIO12}$) under max. timing operating conditions.

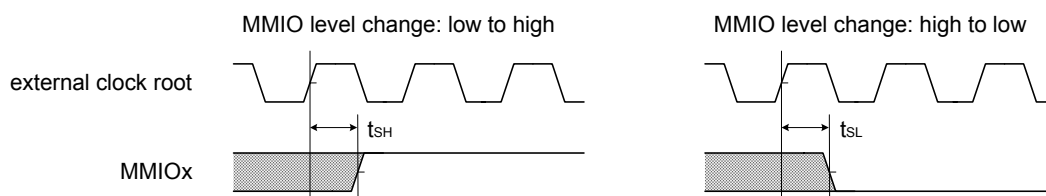


Figure 128: MMIO input sampling

MMIO input timing relationship calculation example

MMIO12 and MMIO16 are used in relation to each other. The high level input timing of MMIO12 and the low level input timing of MMIO16 are of interest. The timing table at the end of this chapter provides the following t_{SH} – and t_{SL} values:

- MMIO12: t_{SH_MMIO12} 1.9 ns
- MMIO16: t_{SL_MMIO16} 1.3 ns

That means: If MMIO12 is to be sampled high and MMIO16 is to be sampled low by the same netX-internal clock cycle, MMIO12 can change from low to high 1.9 ns after a virtual external clock and MMIO16 can change from high to low 1.3 ns after that virtual clock (max. timing operating conditions). Or in other words, the external signal on MMIO16 must change from high to low at least 0.6 ns before MMIO12 changes from low to high in order to sample both states using the same netX-internal clock edge.

MMIO output-input timing relationship calculation example

MMIO12 is used as output pin (e.g. active low request signal) and MMIO16 is used as input pin (e.g. active high acknowledge signal) including external logic. If the external access time between request and acknowledge t_{ac} (e.g. 8.7 ns) is known, the min. number of cycles for sampling valid active high acknowledge on MMIO16 must match the following expression:

$$10 \text{ ns} * n - 5 \text{ ns} - t_{HL\_MMIO12} + t_{SL\_MMIO16} > t_{ac}$$

$$10 \text{ ns} * n - 5 \text{ ns} - 8.9 \text{ ns} + 1.3 \text{ ns} > 8.7 \text{ ns}$$

$$n > 3$$

That means the value sampled from MMIO16 can be used netX-internally 3 system clocks after MMIO12 signal generation.

Parameter	Description	Signal	Min	Typ.	Max.	Unit
t_{LH}	Output low to high level change time	MMIO0 ... MMIO39	6		14	ns
t_{HL}	Output high to low level change time	MMIO0 ... MMIO39	5		12	ns
t_{enH}	Output enable hiZ to high level time	MMIO0 ... MMIO39	6		14	ns
t_{enL}	Output enable hiZ to low level time	MMIO0 ... MMIO39	5		9	ns
t_{disH}	Output disable high level to hiZ time	MMIO0 ... MMIO39	3 ⁽¹⁾		5 ⁽¹⁾	ns
t_{disL}	Output disable low level to hiZ time	MMIO0 ... MMIO39	3 ⁽¹⁾		8 ⁽¹⁾	ns
t_{SH}	High level input sampling time	MMIO0 ... MMIO39	0.7		2	ns
t_{SL}	Low level input sampling time	MMIO0 ... MMIO39	0.7		2	ns

Table 44: MMIO min./max. timing values

Note

1. Output driver disabling is externally propagated after t_{disH} (or t_{disL}). If no other device drives the corresponding MMIO line, a previously driven high level will persist for some time due to capacitive loads (depending on pull-up / pull-down resistors).

4.4.5 USB

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Unit
USB_V _{DDC}	USB power supply, core		1.425	1.5	1.65	V
USB_V _{DDIO}	USB power supply, IO		3.0	3.3	3.6	V
USB_I _{VDDC}	USB core current	1.5 V			5	mA
USB_I _{VDDIO}	USB IO current	3.3 V			5	mA
V _{IH}	Input-high voltage	single end	2.0		V _{DDIO} + 0.5	V
V _{IL}	Input-low voltage	single	0		0.8	V
V _{DI}	Differential input voltage	0.8 V < V _{IN} < 2.5 V	200			mV
V _{OH}	Output-high voltage	ext. 15 kΩ pull-up to V _{DDIO}	2.8		V _{DDIO}	V
V _{OL}	Output-low voltage	ext. 1.5 kΩ pull-down to GND	0		0.3	V
C _{PIN}	Pin capacitance		5		7	pF

4.4.6 PHY

The power supplies of the PHYs must always be connected even if they are not used.

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Unit
PHY_V _{DDCART}	PHY power supply		1.425	1.5	1.65	V
PHY_V _{DDCAP}	PHY central analog power supply		1.425	1.5	1.65	V
PHY_V _{DDIOAC}	PHY central analog power supply		3.0	3.3	3.6	V
PHY_V _{DDIOAT}	PHY analog test power supply		3.0	3.3	3.6	V
P _{10BT}	for both PHYs	1.5 V		120	140	mW
		3.3 V		420	500	mW
P _{100BT}	for both PHYs	1.5 V		230	270	mW
		3.3 V		260	300	mW
P _{100FX}	for both PHYs	1.5 V		100	120	mW
		3.3 V		5	6	mW
PHY_R _{EXTRES}	Reference resistor	Must always be connected, 12.4 kΩ / 1 %				
PHY_I _{VDDC}	PHY core current, includes I _{VDDCART} and I _{VDDCAP}	1.5 V	2	150	180	mA
PHY_I _{VDDIO}	PHY IO current, includes I _{VDDIOAC} and I _{VDDIOAT}	3.3 V	1	130	150	mA

Table 45: Max. power and current values apply at max. voltage conditions

100BASE-TX:

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Unit
V _{100OUTH}	TX output, high level differential signal, TXP/TXN		0.95		1.05	V
V _{100OUTL}	TX output, low level differential signal, TXP/TXN		-0.95		-1.05	V
V _{100OUTM}	TX output, mid. level differential signal, TXP/TXN		-0.05		+0.05	V
V _{OVS}	TX output, overshoot differential signal, TXP/TXN		0		5	%
V _{100INTHON}	RX input, turn-on threshold level differential signal, RXP/RXN				1.0	V
V _{100INTHOFF}	RX input turn-off threshold level differential signal, RXP/RXN		0.20			V
t _r	Rise time, TXP/TXN		3		5	ns
t _f	Fall time, TXP/TXN		3		5	ns
	Duty cycle distortion, TXP/TXN		0		0.5	ns(pp)
	Transmit jitter, TXP/TXN		0		1.4	ns(pp)

These specs are compliant with ANSI/IEEE802.3 Std.

10BASE-T:

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Unit
V _{10OUT}	TX output amplitude differential signal, TXP/TXN		2.2		2.8	V
V _{10INTH}	RX input threshold level differential signal, RXP/RXN		0.30		0.585	V

These specs comply with ANSI/IEEE802.3 Std.

4.4.7 SDRAM

This section provides netX 51/52 SDRAM-related timing characteristics. All timing diagrams are valid for both netX 51/52 SDRAM interfaces: HIF-eMI and MEM-eMI.

Note: In the following diagrams SDRAM signals RAS, CAS, WE are combined as “Command”.

Initialization

Figure 129 shows the SDRAM power-up initialization performed by the netX SDRAM controller.

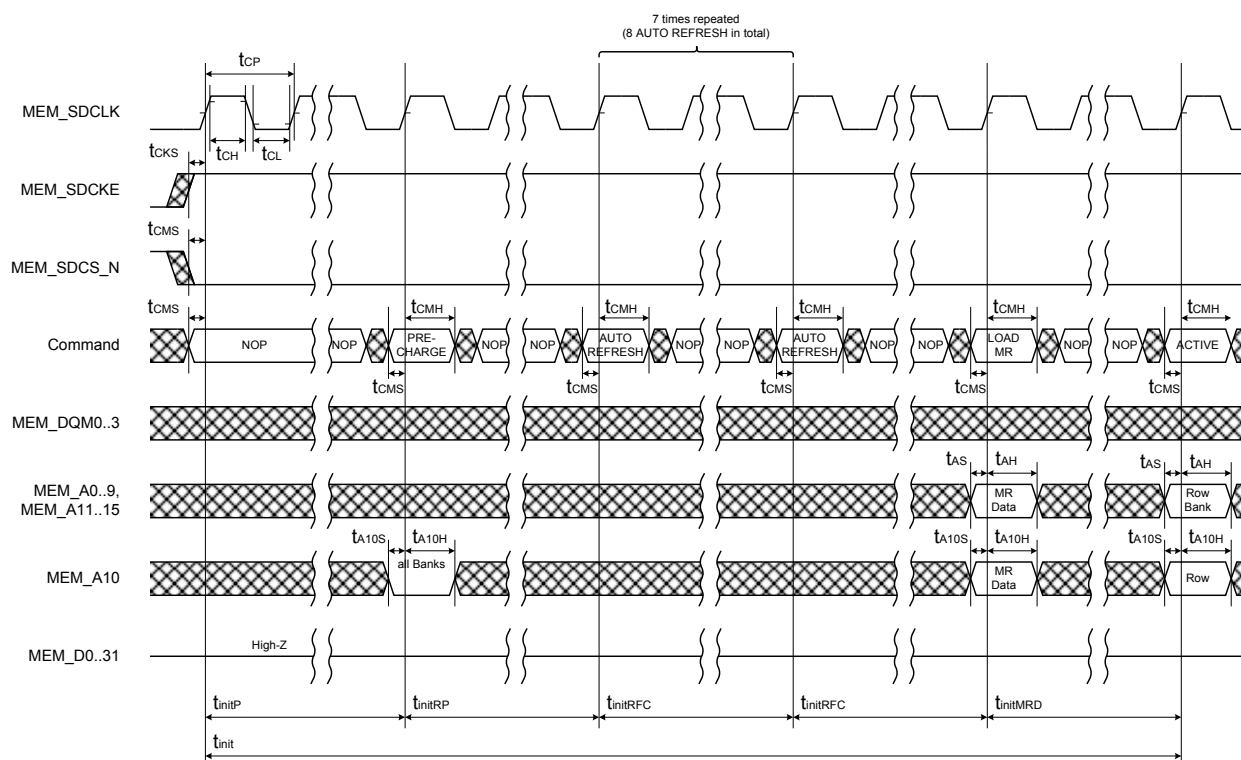


Figure 129: SDRAM power-up and mode register initialization

Auto refresh cycles

Figure 130 shows the SDRAM Auto refresh cycles performed by the netX SDRAM controller.

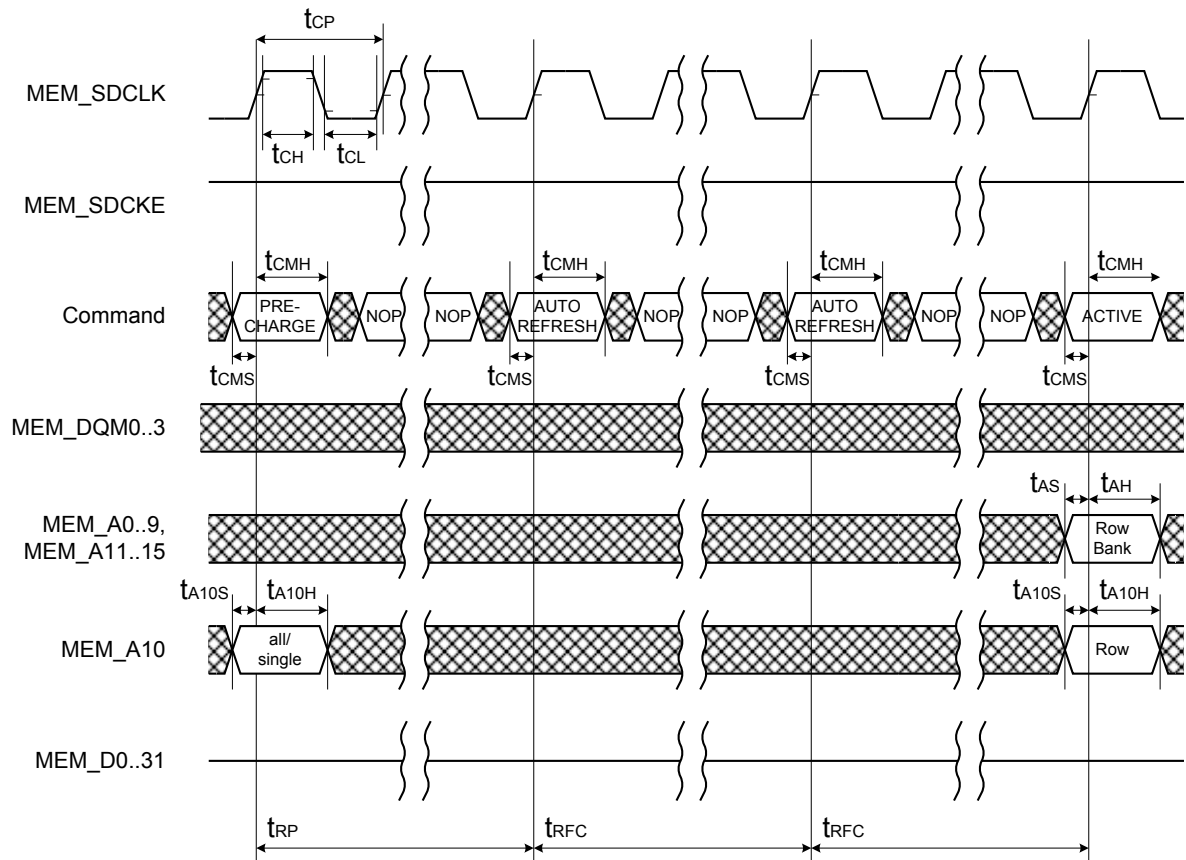


Figure 130: SDRAM auto refresh generation

Self refresh mode, entry and exit

For SDRAM power save without data loss, netX SDRAM controller can set SDRAM devices to self refresh mode. After the self refresh mode entry, the SDRAM controller switches off MEM_SDCLK until the software requests exit from the self refresh mode.

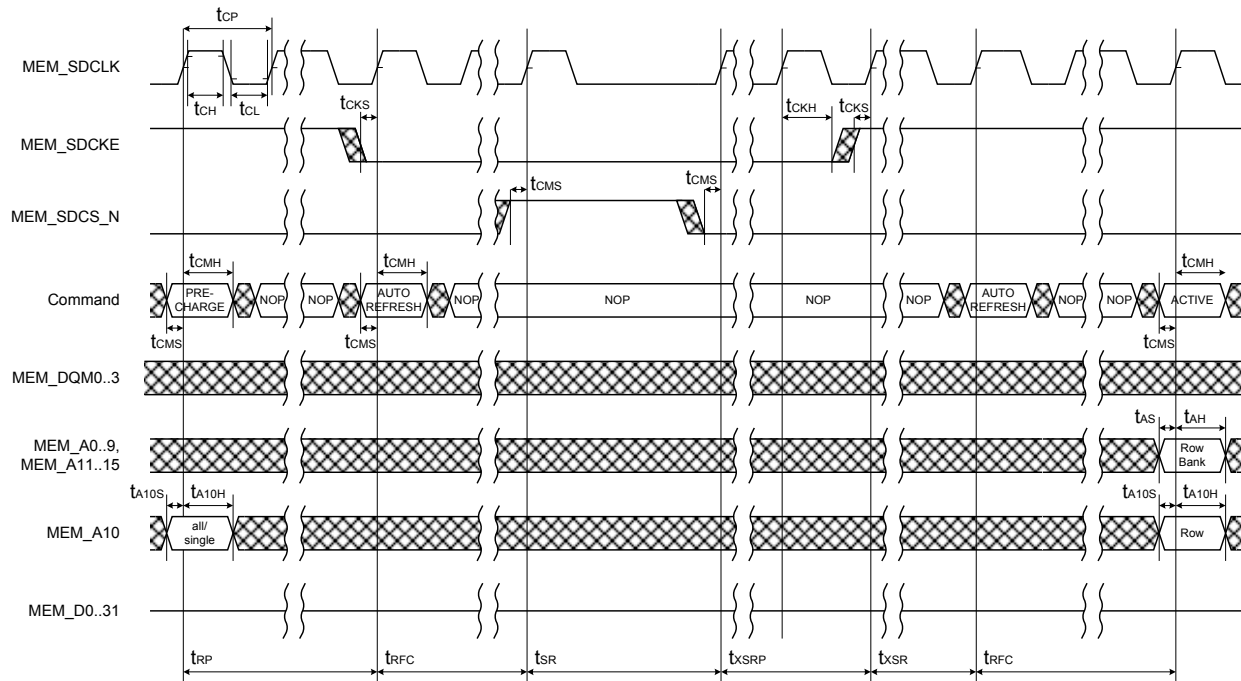


Figure 131: SDRAM self refresh mode entry and exit

Write access

Figure 132 shows a single bank SDRAM write cycle. MEM_SDCKE is always high and thus not shown.

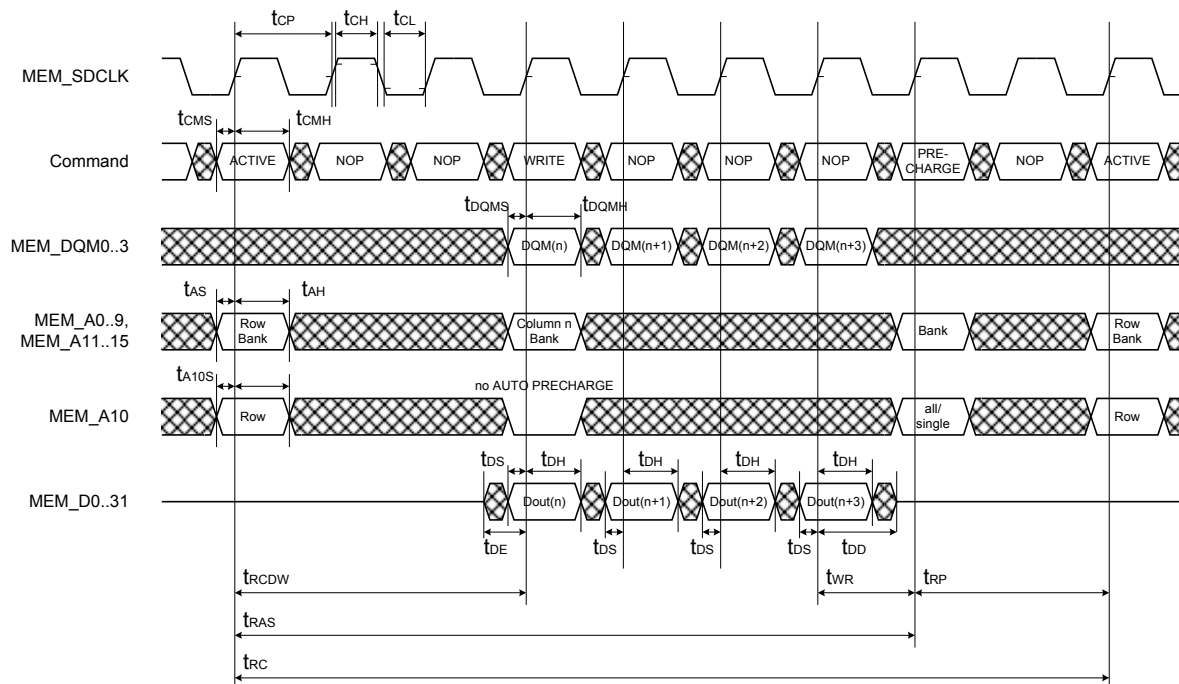


Figure 132: SDRAM write timing (MEM_SDCKE is always high)

Read access

Figure 133 shows a single bank SDRAM read cycle. MEM_SDCKE is always high and thus not shown.

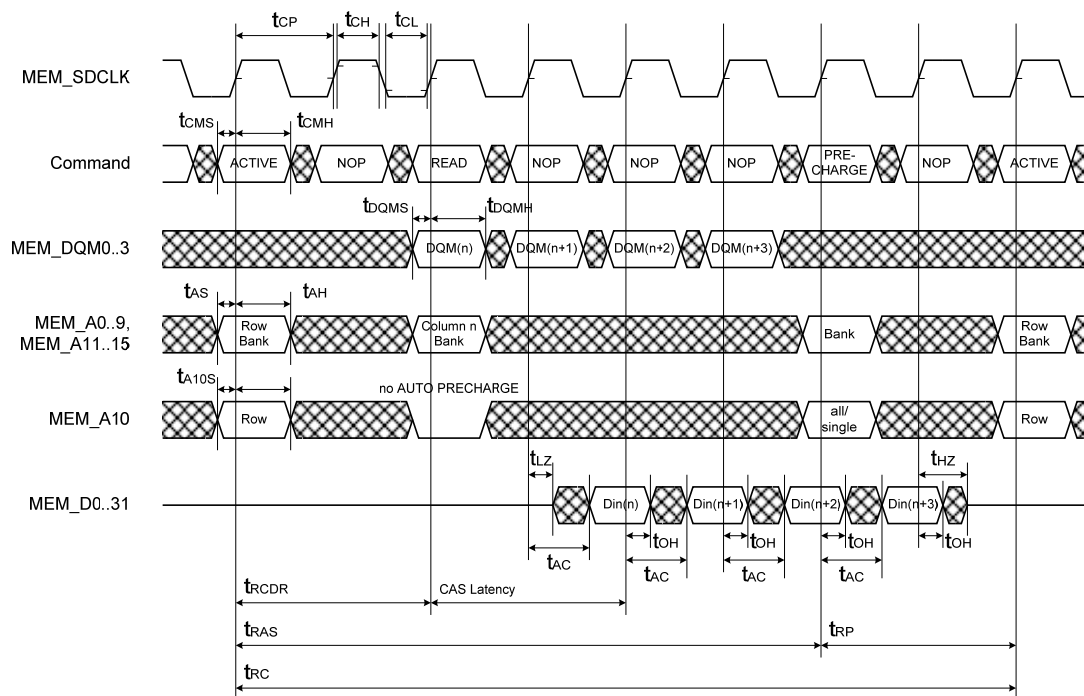


Figure 133: SDRAM read timing (MEM_SDCKE is always high)

Timing characteristics configurable by register *sdram_timing_ctrl* (for MEM-eMI and HIF-eMI):

Symbol	Parameter	Min	Typ	Max	Unit
t_{RAS}	ACTIVE to PRECHARGE time	3..10			t_{CP}
t_{RC}	ACTIVE to ACTIVE (same bank) time	4..13 ⁽¹⁾			t_{CP}
t_{RCDR}	ACTIVE to READ (same bank) time	1..3 ⁽²⁾			t_{CP}
t_{RCDW}	ACTIVE to WRITE (same bank) time	2..4 ⁽²⁾			t_{CP}
t_{REFI}	Average periodic REFRESH interval (Refresh period divided by rows to refresh within refresh period)	3.9 ⁽³⁾	7.8 ⁽³⁾ 15.6 ⁽³⁾	31.2 ⁽³⁾	us
t_{RFC}	AUTO REFRESH period	4..19			t_{CP}
t_{RP}	PRECHARGE command period	1..3			t_{CP}
t_{RRD}	ACTIVE to ACTIVE different bank time	1..3 ⁽⁴⁾			t_{CP}
t_{WR}	WRITE recovery time	1..3			t_{CP}
t_{SR}	Self Refresh period	1			t_{CP}
t_{XSRP}	Self Refresh Clock active to exit period exit	4			t_{CP}
t_{XSR}	Self Refresh exit to command period	4..19 ⁽⁵⁾			t_{CP}

Notes

1. Minimum t_{RC} is $t_{RAS} + t_{RP}$.
2. ACTIVE to WRITE (same bank) time is ACTIVE to READ (same bank) time plus 1 additional cycle from memory controller internal arbitration.
3. Expecting 100 MHz system clock.
4. netX 50 SDRAM controller uses t_{RCDR} for t_{RP} .
5. Programmed t_{RFC} settings will be used for t_{XSR} .

SDRAM initialization and power-up timing characteristics (for MEM-eMI and HIF-eMI):

Symbol	Parameter	Min	Typ	Max	Unit
t_{init}	Whole SDRAM initialization time till first ACTIVE	20317		20317	t_{CP}
t_{initP}	SDRAM Power Up time (NOP till first PRECHARGE)	20050		20050	t_{CP}
t_{initRP}	Initialization PRECHARGE command period	16		16	t_{CP}
$t_{initRFC}$	Initialization AUTO REFRESH period	23		32	t_{CP}
$t_{initMRD}$	Load MR to first command period	4 ⁽¹⁾			t_{CP}

Note: (1) If more $t_{initMRD}$ time is required, software can wait until flag *sdram_ready* is set in register *sdram_general_ctrl* before the first SDRAM memory access.

4.4.7.1 IO timing parameters for SDRAM of MEM-eMI

All timings are related to the following SDRAM clock phase settings (to be set by programming register *sdram_timing_ctrl* of area *ext_sdram_ctrl*):

ext_sdram_ctrl_sdram_general_ctrl.mem_sdclk_phase: 2

ext_sdram_ctrl_sdram_general_ctrl.data_sample_phase: 1

Electrical characteristics of netX 51 MEM-eMI SDRAM part (C_L MEM_D0...31: 15 pF, all other signals 10 pF):

Symbol	Parameter	Min	Typ	Max	Unit
t_{CP}	SDRAM clock (MEM_SDCLK) cycle time	10.0 ⁽¹⁾	10.0 ⁽¹⁾	⁽¹⁾	ns
t_{CH}	SDRAM clock (MEM_SDCLK) high level width	3.5		4.7	ns
t_{CL}	SDRAM clock (MEM_SDCLK) low level width	3.8		4.8	ns
t_{CKS}	Clock enable (CKE, MEM_SDCKE) setup time	4.0 ⁽²⁾			ns
t_{CKH}	Clock enable (CKE, MEM_SDCKE) hold time	2.3 ⁽²⁾			ns
t_{CMS}	Command setup time CS_N MEM_SDCKS_N WE_N MEM_SDWE_N RAS_N MEM_SDRAS_N CAS_N MEM_SDCAS_N	4.2 3.6 3.8 3.5			ns
t_{CMH}	Command hold time CS_N MEM_SDCKS_N WE_N MEM_SDWE_N RAS_N MEM_SDRAS_N CAS_N MEM_SDCAS_N	2.2 2.6 2.5 2.6			ns
t_{AS}	Address (without A10) setup time	2.7			ns
t_{AH}	Address (without A10) hold time	2.5			ns
t_{A10S}	A10 setup time	3.3			ns
t_{A10H}	A10 hold time	2.8			ns
t_{DQMS}	Data Qualifier Mask (DQM*) setup time	3.2			ns
t_{DQMH}	Data Qualifier Mask (DQM*) hold time	2.7			ns
t_{DS}	netX Data-out setup time	3.0			ns
t_{DH}	netX Data-out hold time	2.7			ns
t_{DE}	netX Data-out enable (Low Z) time (write)			4.2	ns
t_{DD}	netX Data-out disable (High Z) time (write)			4.3	ns
t_{AC}	SDRAM Access time (read)			6.0	ns
t_{OH}	SDRAM Data-out hold time (read)	0.0			ns
t_{LZ}	SDRAM Data-out enable (Low Z) time (read)	0.0			ns
t_{HZ}	SDRAM Data-out disable (High Z) time (read)			12.9	ns

Notes

1. MEM_SDCLK always runs at the same frequency as netX-internal system clock.
2. MEM_SDCKE is only used for SDRAM power-down (Self refresh mode or SDRAM disabled).

Important: Connecting SDRAM and SRAM/FLASH is not netX 50-compatible and explicitly not allowed for netX 51/52 (and netX 50). In that case SDRAM timing cannot be guaranteed because the IO load will exceed specified max. values).

4.4.7.2 IO timing parameters for SDRAM of HIF-eMI

All timings are related to the following SDRAM clock phase settings (to be set by programming register *sdram_timing_ctrl* of area *hif_sdram_ctrl*):

hif_sdram_ctrl_sdram_general_ctrl.mem_sdclk_phase: 3

hif_sdram_ctrl_sdram_general_ctrl.data_sample_phase: 0

Electrical characteristics of netX 51/52 HIF-eMI SDRAM part (C_L 30 pF):

Symbol	Parameter	Min	Typ	Max	Unit
t_{CP}	SDRAM clock (HIF_SDCLK) cycle time	10.0 ⁽¹⁾	10.0 ⁽¹⁾	⁽¹⁾	ns
t_{CH}	SDRAM clock (HIF_SDCLK) high level width	3.5		4.7	ns
t_{CL}	SDRAM clock (HIF_SDCLK) low level width	3.8		4.8	ns
t_{CKS}	Clock enable (CKE, HIF_RDY) setup time	3.1 ⁽²⁾			ns
t_{CKH}	Clock enable (CKE, HIF_RDY) hold time	3.0 ⁽²⁾			ns
t_{CMS}	Command setup time CS_N HIF_CSN WE_N HIF_WRN RAS_N HIF_AHI0 CAS_N HIF_AHI1	3.2 3.7 3.3 3.0			ns
t_{CMH}	Command hold time CS_N HIF_CSN WE_N HIF_WRN RAS_N HIF_AHI0 CAS_N HIF_AHI1	2.8 2.5 2.8 3.0			ns
t_{AS}	Address (without A10) setup time ⁽³⁾	2.9			ns
t_{AH}	Address (without A10) hold time ⁽³⁾	2.4			ns
t_{A10S}	A10 setup time	3.7			ns
t_{A10H}	A10 hold time	2.5			ns
t_{DQMS}	Data Qualifier Mask setup time ⁽³⁾	3.1			ns
t_{DQMH}	Data Qualifier Mask hold time ⁽³⁾	2.5			ns
t_{DS}	netX Data-out setup time ⁽³⁾	2.7			ns
t_{DH}	netX Data-out hold time ⁽³⁾	1.3			ns
t_{DE}	netX Data-out enable (Low Z) time (write) ⁽³⁾			6.2	ns
t_{DD}	netX Data-out disable (High Z) time (write) ⁽³⁾			4.2	ns
t_{AC}	SDRAM Access time (read) ⁽³⁾			6.0	ns
t_{OH}	SDRAM Data-out hold time (read) ⁽³⁾	0.0			ns
t_{LZ}	SDRAM Data-out enable (Low Z) time (read) ⁽³⁾	0.0			ns
t_{HZ}	SDRAM Data-out disable (High Z) time (read) ⁽³⁾			11.9	ns

Notes

1. HIF_SDCLK always runs at the same frequency as netX-internal system clock.
2. CKE (on HIF_RDY) is only used for SDRAM power-down (self refresh mode or SDRAM disabled).
3. See section 3.10 for HIF-IO setup, HIF-eMI signal mapping for DQMs, address and data lines (HIF_IO_CTRL). Read section 3.10 carefully before using and designing HIF-eMI.

4.4.8 SRAM / FLASH

This section provides netX 51/52 SRAM (e.g. NVRAM, Flash) related timing characteristics.

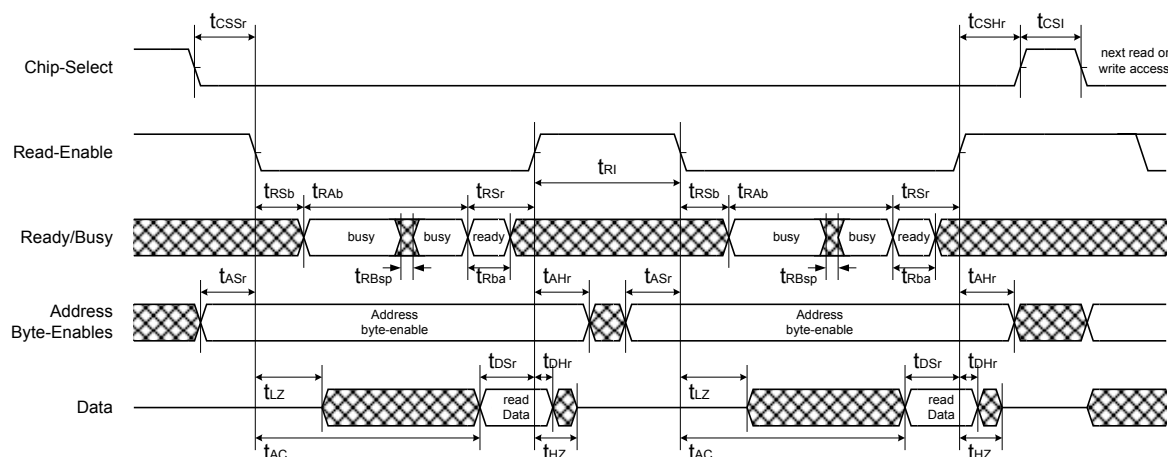


Figure 134: Read, pre and post-pauses enabled for sequential reads (write enable always high)

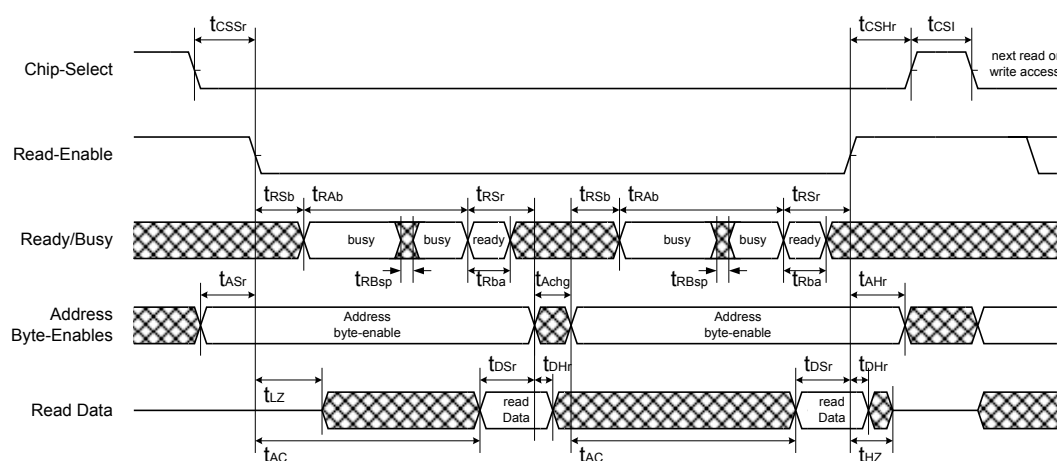


Figure 135: Read, pre and post-pauses disabled for sequential reads (write enable always high)

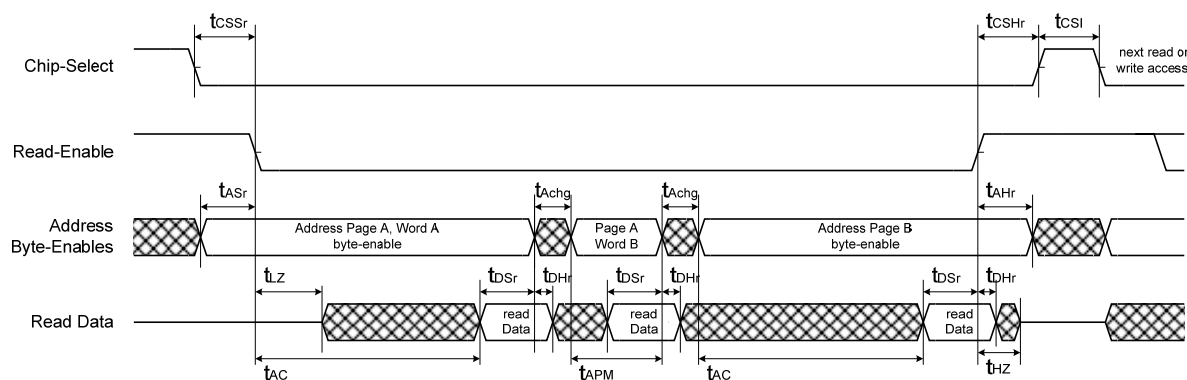


Figure 136: APM read burst, pre and post-pauses disabled between sequential reads

Note: APM read burst access cycles can also be extended by ready signal. Timing relations are the same as shown in Figure 135.

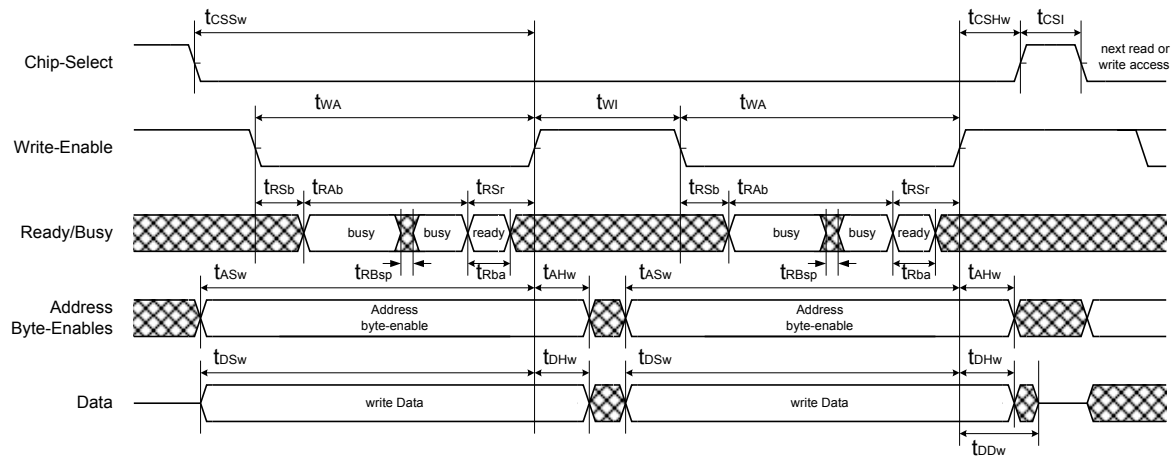


Figure 137: Write (read enable always high)

Note: Byte enable signals have address character. The address timing parameters cover the byte enable signals.

Timings of netX 51/52 SRAM interface are widely configurable by software. Detailed timings depend on the following programmable parameters that can be set for each chip-select area individually (exception: Ready filter time) by software.

Symbol	Parameter	Min	Typ	Max	Unit
t_{SYS}	netX system clock cycle time	10.0 ⁽¹⁾	10.0 ⁽¹⁾		ns
t_{pr}	programmable pre-pause wait phase (10 ns steps)	0.0	$n \cdot t_{SYS}$	$3 \cdot t_{SYS}$	ns
t_{po}	programmable post-pause wait phase (10 ns steps)	0.0	$n \cdot t_{SYS}$	$3 \cdot t_{SYS}$	ns
t_{pw}	programmable wait state phase (10 ns steps)	0.0	$n \cdot t_{SYS}$	$63 \cdot t_{SYS}$	ns
$t_{paw}^{(2)}$	programmable APM wait state phase (10 ns steps)	0.0	$n \cdot t_{SYS}$	$15 \cdot t_{SYS}$	ns
t_{rf}	programmable ready filter time	0.0	$n \cdot t_{SYS}$	$3 \cdot t_{SYS}$	ns

Notes

- Default is 100 MHz frequency as netX-internal system clock, but this can be decreased to save power.
- APM bursts are only supported for chip-select 0.

4.4.8.1 IO timing parameters for SRAM of external memory interface

All electrical characteristics of netX 51/52 SRAM MEM-eMI part are related to C_L : 15 pF.

For read access, device access times must meet the following conditions:

Symbol	Parameter	Min	Typ	Max	Unit
t_{AC}	Data access time			$t_{pw} + t_{SYS} - 5.3$	ns
t_{APM}	APM data access time			$t_{paw} + t_{SYS} - 6.6$	ns

Note: Access times can be extended by using the ready signal.

Symbol	Parameter	Min	Typ	Max	Unit
$t_{CSSr}^{(1)}$	Chip-select setup time (read)	$t_{pr}-1.5$			ns
$t_{CSHr}^{(2)}$	Chip-select hold time (read)	$t_{po}-1.6$			ns
t_{CSSw}	Chip-select setup time (write)	$t_{pr}+t_{pw}+t_{sys}-0.9$			ns
t_{CSHw}	Chip-select hold time (write) post-pause 0 post-pause > 0	$t_{sys}-1.6$ $t_{po}-1.6$			ns
t_{CSI}	Chip-select idle time	$t_{sys}-1.9$			ns
$t_{ASr}^{(1)}$	Address setup time (read)	$t_{pr}-2.6$			ns
$t_{AHR}^{(2)}$	Address hold time (read)	$t_{po}-2.2$			ns
t_{ASw}	Address setup time (write)	$t_{pr}+t_{pw}+t_{sys}-2.0$			ns
t_{AHw}	Address hold time (write) post-pause 0 post-pause > 0	$t_{sys}-2.2$ $t_{po}-2.2$			ns
t_{Achg}	Address change time (read burst)			2.9	ns
t_{DSr}	Data setup time (read)	5.9			ns
$t_{DHR}^{(2)}$	Data hold time (read)	-3.0			ns
t_{LZ}	Data low-Z time (read)	0.0			ns
t_{HZ}	Data high-Z time (read) pre-pause 0 pre-pause > 0			$t_{sys}-2.0$ $t_{pr}-2.0$	ns
t_{DSw}	Data setup time (write)	$t_{pr}+t_{pw}+t_{sys}-1.8$			ns
t_{DHW}	Data hold time (write) post-pause 0 post-pause > 0	$t_{sys}-2.0$ $t_{po}-2.0$			ns
t_{DDw}	Data disable time (write) post-pause 0 post-pause > 0			$t_{sys}-3.0$ $t_{po}-3.0$	ns
t_{RI}	Read enable idle time	$t_{pr}+t_{po}-2.0$			ns
t_{WI}	Write enable Idle time post- and pre-pause 0 post- or pre-pause > 0	$t_{sys}-1.9$ $t_{pr}+t_{po}-1.9$			ns
t_{WA}	Write enable active time	$t_{pw}+t_{sys}-0.9$			ns

Notes

1. For pre-pause setting 0 (t_{pr} is 0) chip-select and address-setup time could become negative at read access start. That means they could change while read enable has already become low in the first 2.7 ns of an access. This can be avoided by setting pre-pause to 1 or more. However, this will increase total access time by 10 ns.
2. For post-pause setting 0 (t_{po} is 0) chip-select and address hold time could become negative at read access end. That means they could already change while read enable still is low in the last 2.2 ns (t_{AHR}) of an access. This will not cause read data problems as read data is already sampled at least 3.0 ns (t_{DHR}) before read access end. To guarantee stable chip-select and address signals at read access end, post-pause can be set to 1 or more. However, this will increase total access time by 10 ns.

The following table provides timing parameters for optional ready signal. Wait state pause (t_{pw}) must be set at least to 1 when ready is used.

Note: MEM-eMI ready signal is part of netX 51/52 multiplex matrix signals and can be mapped to 1 of 49 MMIOs. The following table covers timings of all MMIOs.

Symbol	Parameter	Min	Typ	Max	Unit
t_{RSb}	Ready setup busy time	$t_{pw}-0.5 \cdot t_{sys} -6.9$			
t_{RAb}	Ready active busy time			$1024 \cdot t_{sys}^{(1)}$	ns
t_{RSr}	Ready setup ready time	$t_{rf}+1.5 \cdot t_{sys} -7.4$			
t_{RAr}	Ready active ready time	$t_{rf}+t_{sys} +2.0$			
t_{Rbsp}	Ready busy spike time			$t_{rf}-1.0$	

Note 1: Ready timeout. Timeout can be disabled by software. However, netX will be stalled if ready signal is permanently set to busy state.

4.4.8.2 IO timing parameters for SRAM of host interface memory interface

All electrical characteristics of netX 51 SRAM MEM-eMI part are related to C_L : 30 pF.

For read access, device access times must match the following conditions:

Symbol	Parameter	Min	Typ	Max	Unit
t_{AC}	Data Access time			$t_{pw}+t_{sys}-7.0$	ns
t_{APM}	APM Data Access time			$t_{paw}+t_{sys}-8.7$	ns

Note: Access times can be extended by ready signal usage.

Symbol	Parameter	Min	Typ	Max	Unit
$t_{CSSr}^{(1)}$	Chip-Select Setup time (read)	$t_{pr}-2.1$			ns
$t_{CSHr}^{(2)}$	Chip-Select Hold time (read)	$t_{po}-1.7$			ns
t_{CSSw}	Chip-Select Setup time (write)	$t_{pr}+t_{pw}+t_{sys}-1.4$			ns
t_{CSHw}	Chip-Select Hold time (write) post-pause 0 post-pause > 0	$t_{sys}-1.5$ $t_{po}-1.5$			ns
t_{CSI}	Chip-Select Idle time	$t_{sys}-3.4$			ns
$t_{ASr}^{(1)}$	Address Setup time (read)	$t_{pr}-3.3$			ns
$t_{AHR}^{(2)}$	Address Hold time (read)	$t_{po}-3.0$			ns
t_{ASw}	Address Setup time (write)	$t_{pr}+t_{pw}+t_{sys}-2.5$			ns
t_{AHw}	Address Hold time (write) post-pause 0 post-pause > 0	$t_{sys}-2.8$ $t_{po}-2.8$			ns
t_{Achg}	Address Change time (read burst)			3.6	ns
t_{DSr}	Data Setup time (read)	8.2			ns
$t_{DHR}^{(2)}$	Data Hold time (read)	-4.9			ns
t_{LZ}	Data Low-Z time (read)	0.0			ns
t_{HZ}	Data High-Z time (read) pre-pause 0 pre-pause > 0			$t_{sys}-3.1$ $t_{pr}-3.1$	ns
t_{DSw}	Data Setup time (write)	$t_{pr}+t_{pw}+t_{sys}-2.7$			ns

Symbol	Parameter	Min	Typ	Max	Unit
t_{DHW}	Data Hold time (write) post-pause 0 post-pause > 0	$t_{sys}-2.8$ $t_{po}-2.8$			ns
t_{DDW}	Data-Disable time (write) post-pause 0 post-pause > 0			$t_{sys}-3.8$ $t_{po}-3.8$	ns
t_{RI}	Read-Enable Idle time	$t_{pr}+t_{po}-2.7$			ns
t_{WI}	Write-Enable Idle time post- and pre-pause 0 post- or pre-pause > 0	$t_{sys}-2.7$ $t_{pr}+t_{po}-2.7$			ns
t_{WA}	Write-Enable Active time	$t_{pw}+t_{sys}-0.5$			ns

Notes

1. For pre-pause setting 0 (t_{pr} is 0) chip-select and address setup time could become negative at read access start. That means they could change while read enable has already become low in the first 2.7 ns of an access. This can be avoided by setting pre-pause to 1 or more. However, this will increase the total access time by 10 ns.
2. For post-pause setting 0 (t_{po} is 0) chip-select and address hold time could become negative at read access end. That means they could already change while read enable still is low in the last 3.0 ns (t_{AHr}) of an access. However, that will not cause read data problems as read data is already sampled at least 4.9 ns (t_{DHW}) before read access end. To guarantee stable chip-select and address signals at read access end post-pause can be set to 1 or more. However, this will increase total access time by 10 ns.

The following table provides timing parameters for optional ready signal. Wait state-pause (t_{pw}) must be set at least to 1 when ready is used.

Symbol	Parameter	Min	Typ	Max	Unit
t_{RSb}	Ready Setup Busy time	$t_{pw}-0.5*t_{sys}-6.8$			
t_{RAb}	Ready Active Busy time			$1024 t_{sys}^{(1)}$	ns
t_{RSr}	Ready Setup Ready time	$t_{rr}+1.5*t_{sys}-8.0$			
t_{RAr}	Ready Active Ready time	$t_{rr}+t_{sys}+0.6$			
t_{Rbsp}	Ready Busy Spike time			$t_{rr}-0.6$	

Note: (1) Ready timeout. Timeout can be disabled by software. However, netX will be stalled if ready signal is permanently set to busy state.

4.4.9 Parallel dual-port memory timing

The difference between SRAM modes and Intel or Motorola modes is that SRAM modes have 2 dedicated enable signals for read and write (DPM_RDn and DPM_WRn) while Intel or Motorola modes have only 1 direction signal (located on DPM_RDn). The direction modes require byte strobe signals which become active when all other signals are already valid and stable. All other signals (with the exception of read-data and ready) must remain stable until the byte strobes become inactive at access end. For SRAM modes byte enables have address character and must be valid before the activation of read enable or the deactivation of write enable.

In multiplexed mode an additional signal DPM_ALE (Address-Latch, Address-Enable) flags the state of multiplexed address/data bus. Address part multiplexed on data lines must be valid around the change from address to data phase. Only the data phase will be extended by setting DPM_RDY to wait state. Then address phase will never be extended.

Note: The ready signal is optional only. If used, it will always become active for read access. For write it will be set only if the internal part of the DPM interface is busy.

Note: Polarity and IO mapping of signal DPM_ALE is programmable by the bits *aen_sel* and *aen_pol* of register *dpm_if_cfg*.

4.4.9.1 SRAM modes with separated read and write enable signals

This section is dedicated to the following modes:

- INTEL_8BIT_SRAM
- INTEL_16BIT_SRAM
- INTEL_32BIT_SRAM
- TIOMAP_16BIT_NON_MULTIPLEXED

Note: Byte enable signals (DPM_BE signals) have address character for SRAM modes. They are treated like address lines here. Byte enables can be selected by register *dpm_if_cfg.be_sel*.

4.4.9.1.1 SRAM mode read access

DPM read accesses can be read or chip-select signal controlled. Moreover, netX 51/52 DPM supports read bursts without the necessity of read or chip-select signal toggling between each access.

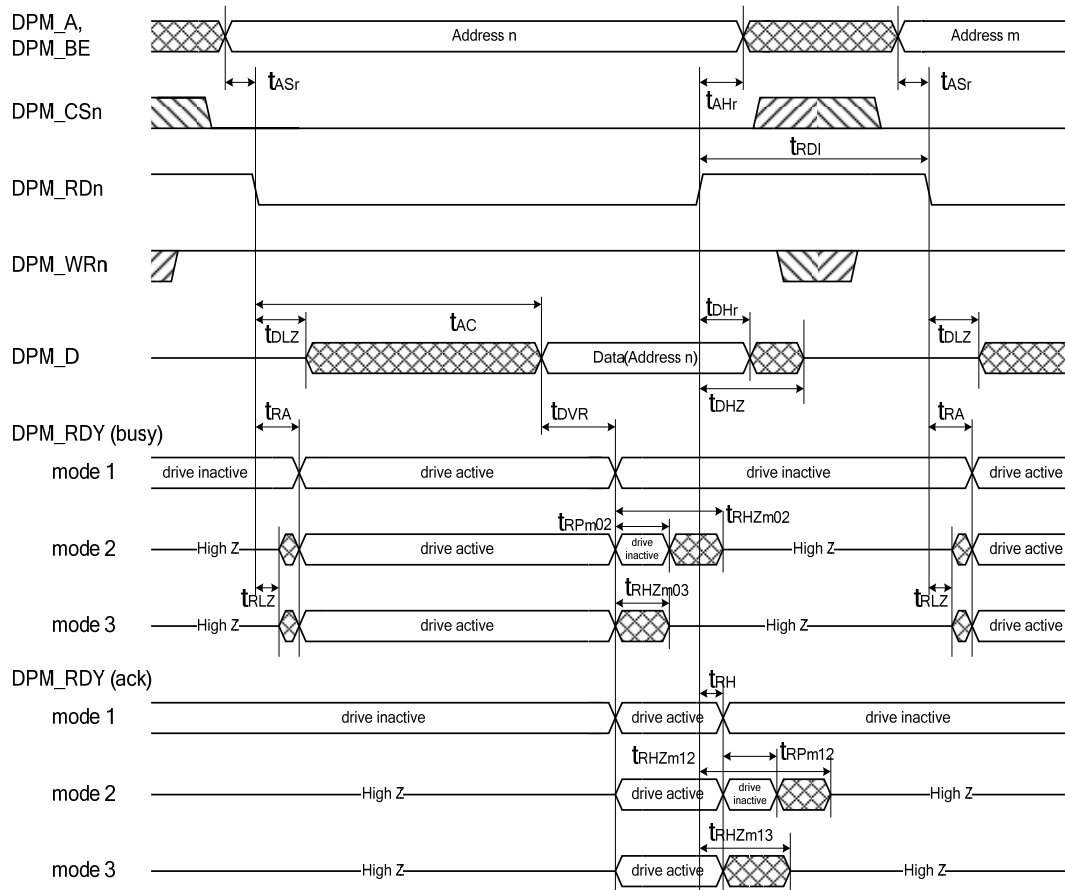


Figure 138: Detailed timing of DPM SRAM mode read access DPM_RDn controlled

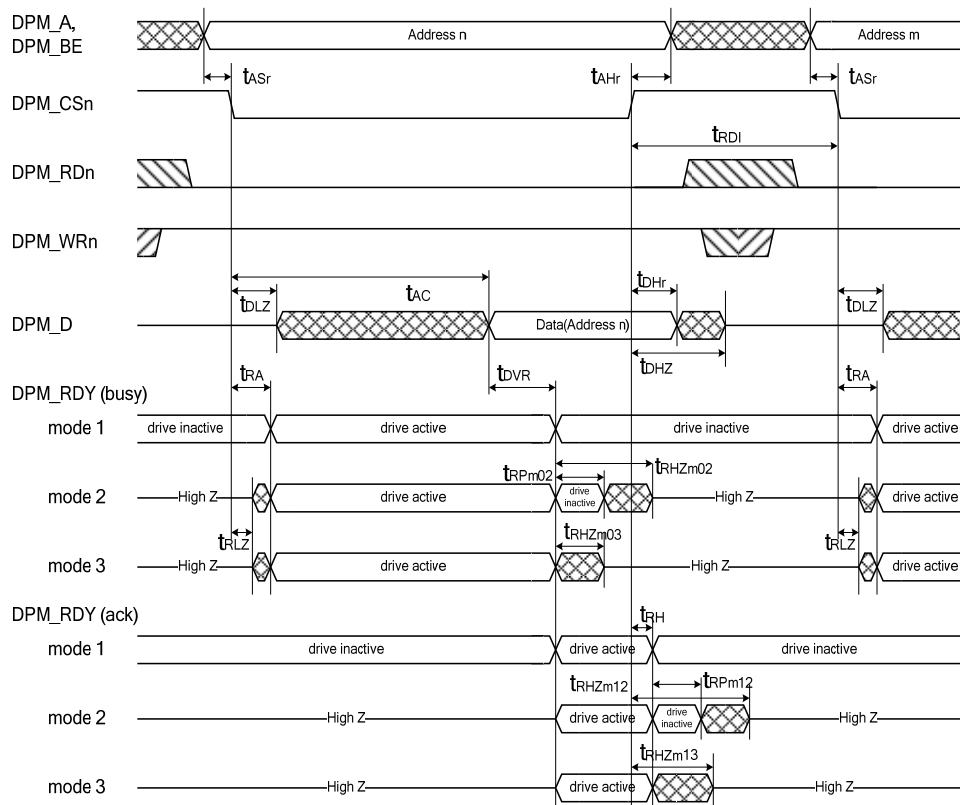


Figure 139: Detailed timing of DPM SRAM mode read access DPM_CSn controlled.

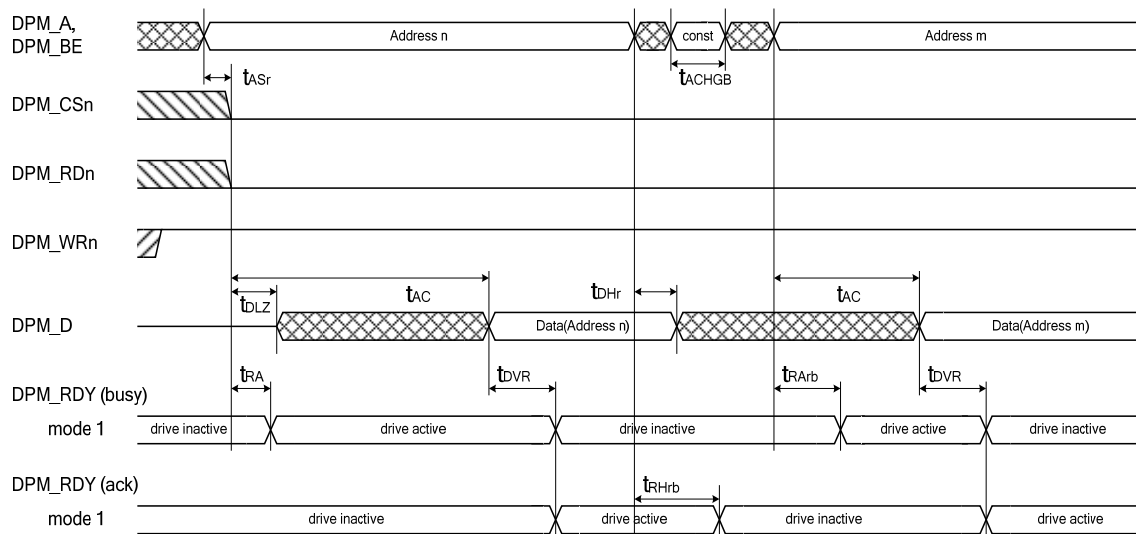


Figure 140: Detailed timing of DPM read burst access (DPM_A or DPM_DQM change without DPM_CSn or DPM_RDn toggling)

Note: Read burst support must be enabled by programming register *dpm_timing_cfg*.

Note: Since undriven idle states of ready drive modes 2 and 3 are too slow for read bursts, they are omitted in Figure 140.

Time t_{ACHGB} is the maximum time of stable addresses, but invalid addresses occur on the address bus when burst addresses change.

4.4.9.1.2 SRAM mode write access

DPM write cycles always end with a positive edge of active low DPM write signal (DPM_WRn).

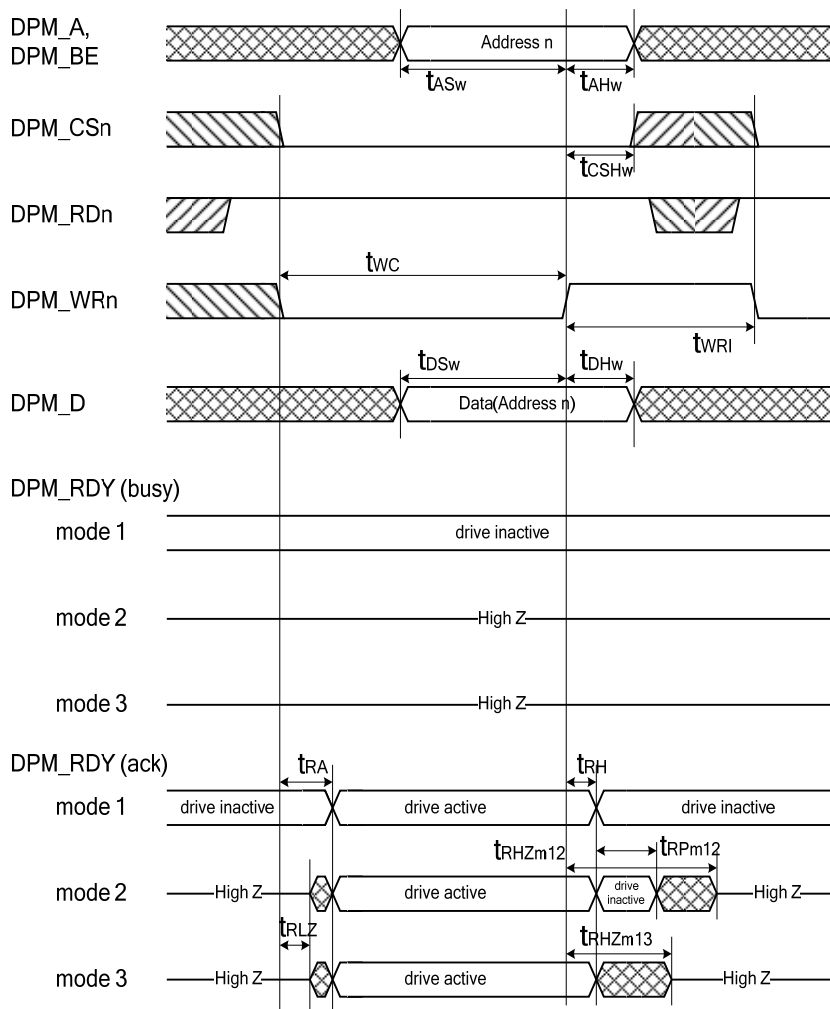


Figure 141: Detailed timing of DPM SRAM mode write access when internal netX DPM side is idle (no insertion of wait cycles)

After the end of an external write access (positive edge of signal DPM_WRn) the internal access to the netX address area will start. If a new external DPM write access is initiated before the internal write access is finished, external wait cycles will be inserted by setting the signal DPM_RDY.

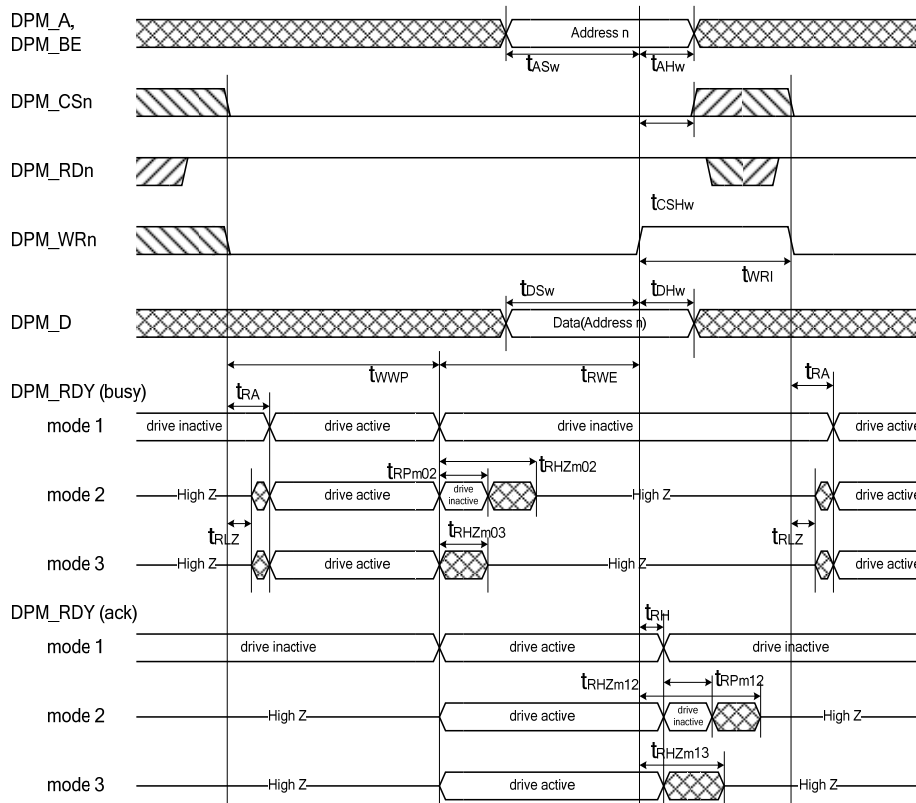


Figure 142: Detailed timing of DPM SRAM mode write access when internal netX DPM side busy (insertion of wait cycles)

4.4.9.2 Multiplexed SRAM modes with separated read and write enable signals

This section is dedicated to the following modes:

- INTEL_8BIT_MULTIPLEXED
- INTEL_16BIT_MUL_2BE_BYTE_ADDR (SRAM-like with 2 byte enables)
- INTEL_16BIT_MUL_2BE_WORD_ADDR (SRAM-like with 2 BEs and shifted address)
- INTEL_16BIT_MUL_NO_BES (no byte enables, for netX 50-compatibility)
- INTEL_32BIT_MUL_4BE_BYTE_ADDR (SRAM-like with 4 byte enables)
- INTEL_32BIT_MUL_4BE_DWORD_ADDR (SRAM-like with 4 BEs and shifted address)
- TIOMAP_16BIT_MULTIPLEXED

Note: Mode naming comes from the netX 50 documentation, but “INTEL” modes are rather SRAM-like modes there. The difference between SRAM modes and Intel modes is that SRAM modes have 2 dedicated enable signals for read and write (DPM_RDn and DPM_WRn) while Intel modes have only 1 direction signal (located on DPM_RDn). The Intel modes require byte strobe signals at least for write. They must become active *after* direction signal DPM_RDn is valid and stable. DPM_RDn must remain stable until the byte strobes become inactive at the end of access. For SRAM modes byte enables have address character and must be valid *before* activating read enable or deactivating write enable.

Note: For multiplexed Intel-like modes with byte strobes, see section 4.4.9.4.

Note: Byte enable signals (DPM_BE signals) have address character for multiplexed SRAM modes, but they are always available on dedicated signals (i.e. they are not part of the multiplexed address on the A/D-bus). They are treated similar to the high address lines here. Byte enables can be selected by register *dpm_if_cfg.be_sel*.

4.4.9.2.1 Multiplexed SRAM mode read access

In multiplexed mode read bursts are not supported. The read enable signal must become inactive between subsequent access operations.

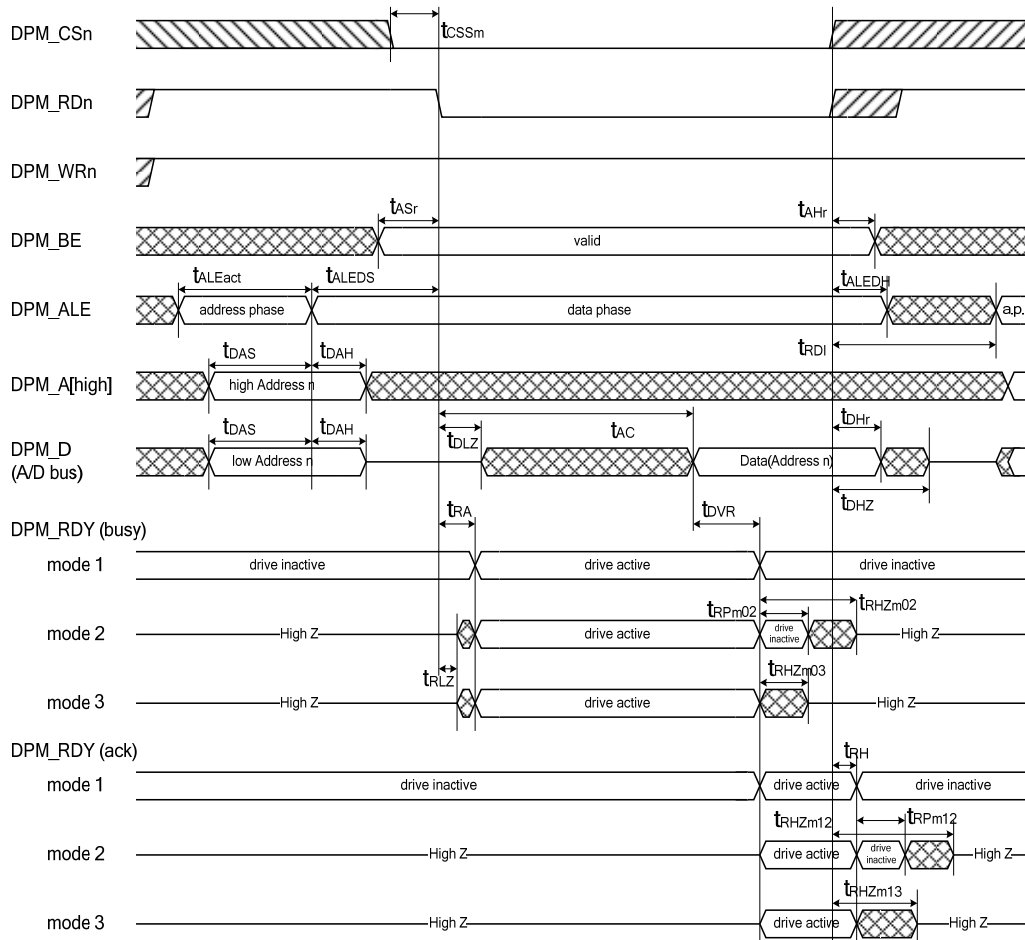


Figure 143: Detailed timing of multiplexed SRAM mode read access

The address phase could also overlap with the read active phase, i.e. DPM_ALE could remain in address phase state while DPM_RD_n is already set to active. However, the data phase never starts before DPM_ALE changes to data state, i.e. ready and read data will never be driven while DPM_ALE is in the address state. However, the host must make sure that the addresses are not driven too long for read access. Otherwise netX will drive read data against addresses driven by the host.

Important: If the address phase overlaps with the read active phase, the address hold time t_{DAH} must be shorter than the read data enable time t_{DLZ} to avoid damage to netX or host.

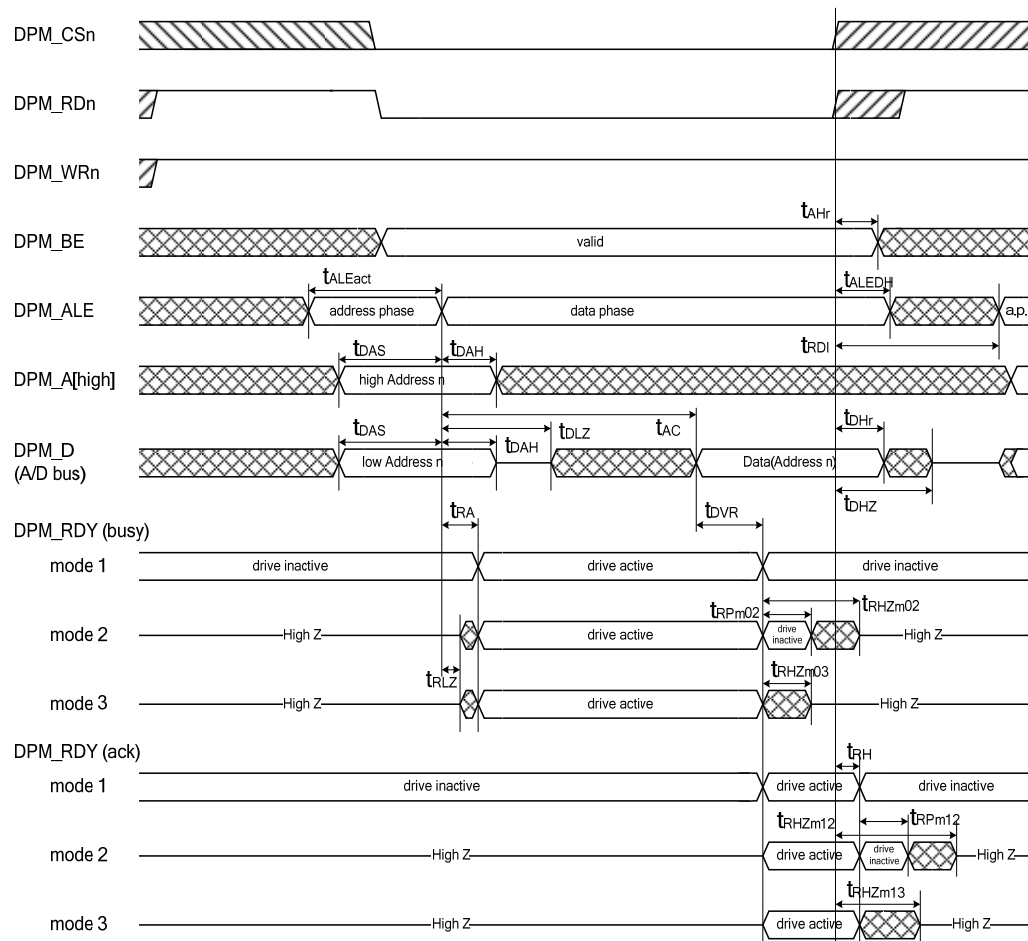


Figure 144: Detailed timing of multiplexed SRAM mode read access with address phase overlapping into read-active phase

4.4.9.2.2 Multiplexed SRAM mode write access

DPM write cycles always end with a positive edge of an active low DPM write signal (DPM_WRn).

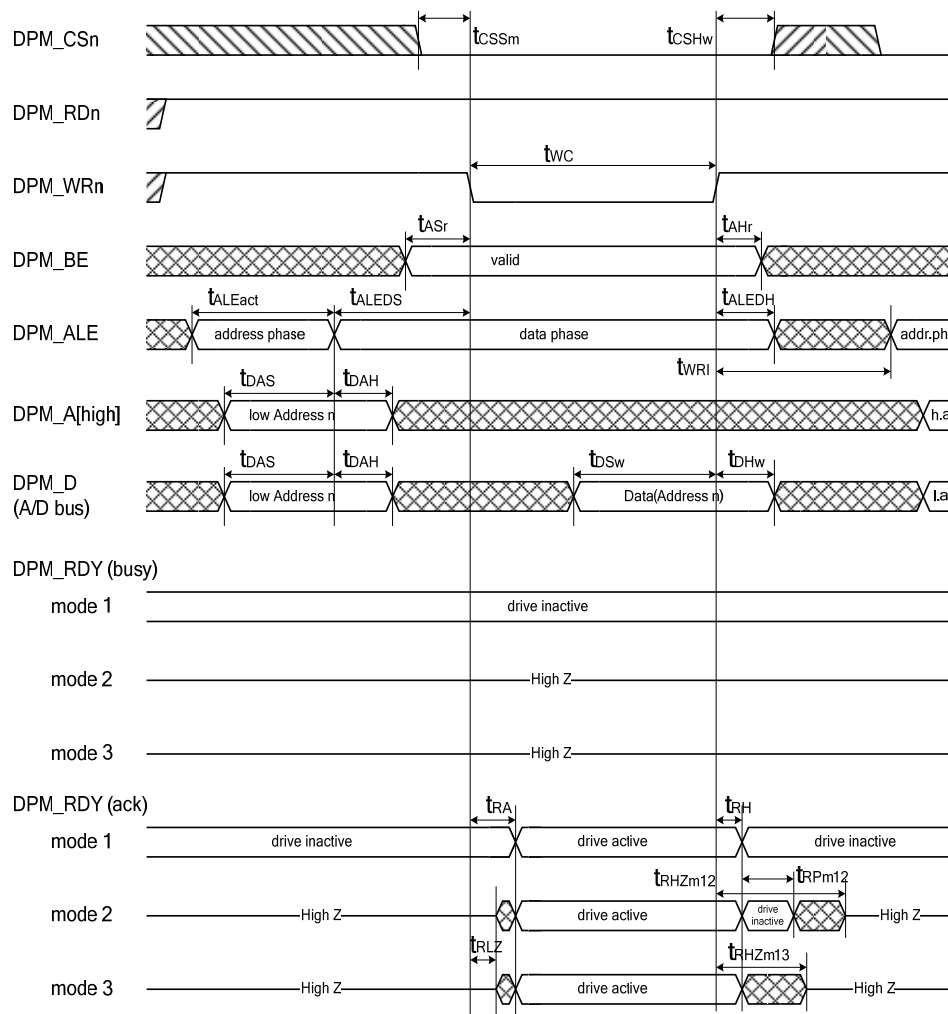


Figure 145: Detailed timing of multiplexed SRAM mode write access when internal netX DPM side is idle (no insertion of wait cycles)

The internal access to the netX address area will start after the end of an external write access (positive edge of DPM_WRn signal). If a new external DPM write access is initiated before the internal write access is finished, external wait cycles will be inserted by setting signal DPM_RDY during the data phase.

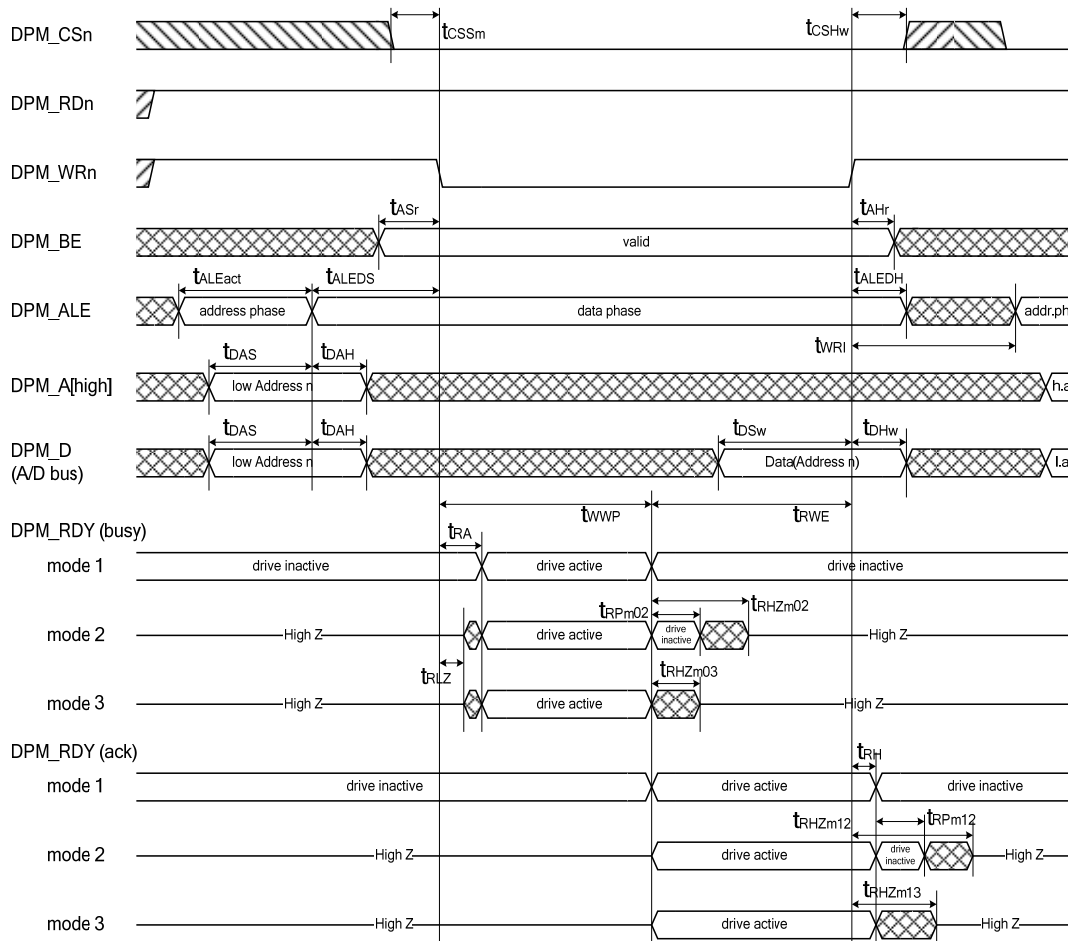


Figure 146: Detailed timing of multiplexed SRAM mode write access when internal netX DPM side busy (insertion of wait cycles)

The address phase could also overlap with the write-active phase, i.e. DPM_ALE could remain in the address phase state while DPM_WR_n is already set to active. However, the data phase never starts before DPM_ALE changes to data state, i.e. ready will never be driven while DPM_ALE is in the address state.

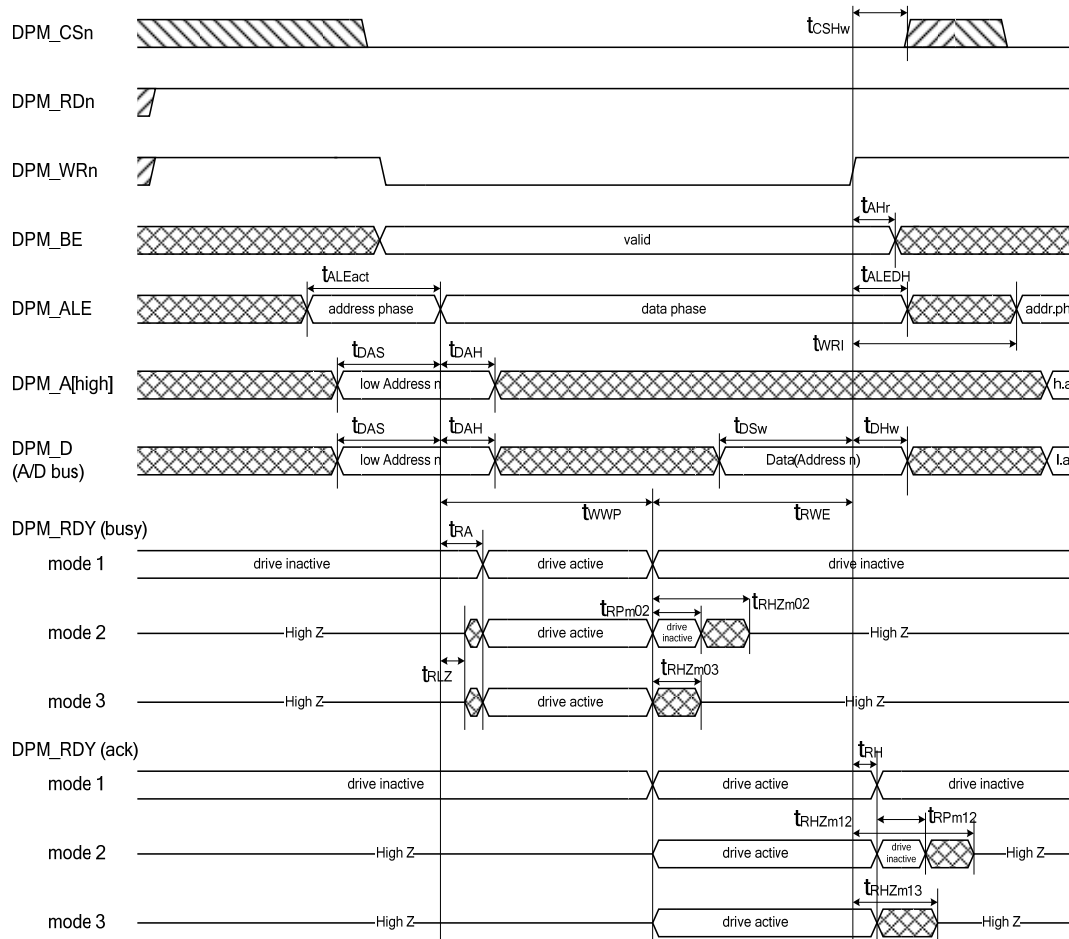


Figure 147: Detailed timing of multiplexed SRAM mode write access with address phase overlapping into write-active phase

4.4.9.3 Intel-like modes with data direction byte strobe signals (byte writes)

This section is dedicated to the following modes:

- INTEL_16BIT_BYTE_WRITE (read is not using byte enables)
- INTEL_32BIT_BYTE_WRITE (read is not using byte enables)

Byte enable signals (DPM_BE signals) have strobe character for Intel-like modes i.e. are byte strobes here. They must become active after the direction signal DPM_RD_n is already valid and stable. To avoid an illegal data direction change, DPM_RD_n must not change before the byte strobes become inactive at access end.

Important: Intel-like microprocessor interfaces often provide separated byte enable (or byte strobe) signals for read and write (i.e. byte write enable lines and byte read enable lines). This is not supported by netX DPM. It is recommended to connect byte write enable lines to netX DPM byte enables and to disable byte enables for read by programming bit *be_rd_dis* of register *dpm_if_cfg* to 1 (4.4.9.3.2). Byte write access will be possible while read access will always deliver data of the full programmed data bus width (by bits *dpm_cfg0x0.mode*). Observe the RDL and WBC rules (3.11.8.2).

Note: Byte strobe signals (on DPM_BE signals) can be selected using register *dpm_if_cfg.be_sel*.

Note: The address enable signal could also be used in non-multiplexed Intel modes, but this is unlikely. When used it must behave like an additional chip-select. Both signals, chip-select and address enable, must be active during an access.

4.4.9.3.1 Intel-like mode read access using byte enables

A read access using byte enables is started when byte enables (byte enable controlled) or chip-select (chip-select controlled) signals become active at access start. Address and data direction (RDn) signals must be stable then.

Note: Access detection will wait for stable byte enables of at least 1 netX system clock cycle at access start (t_{BEchg}). This could be extended using the programmable t_{OSA} timing parameter. Further changes of the byte enable signals will be ignored during the access until they all become inactive. The access is terminated as soon as all byte enables have returned to inactive state.

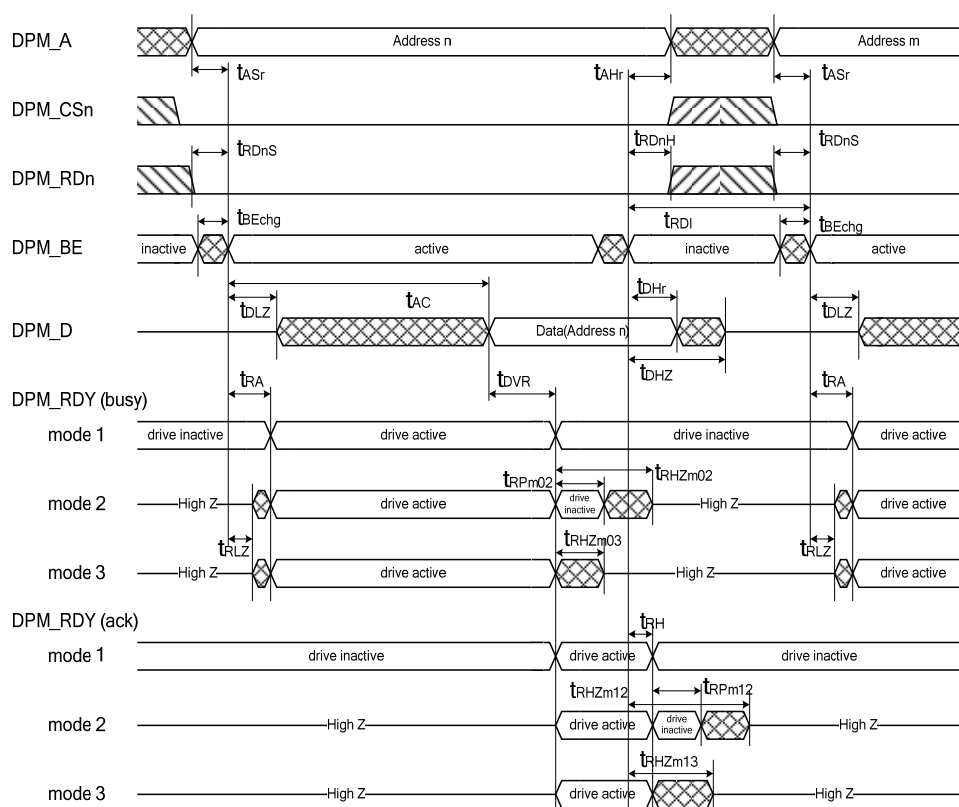


Figure 148: Detailed timing of an Intel-like byte enable controlled read access

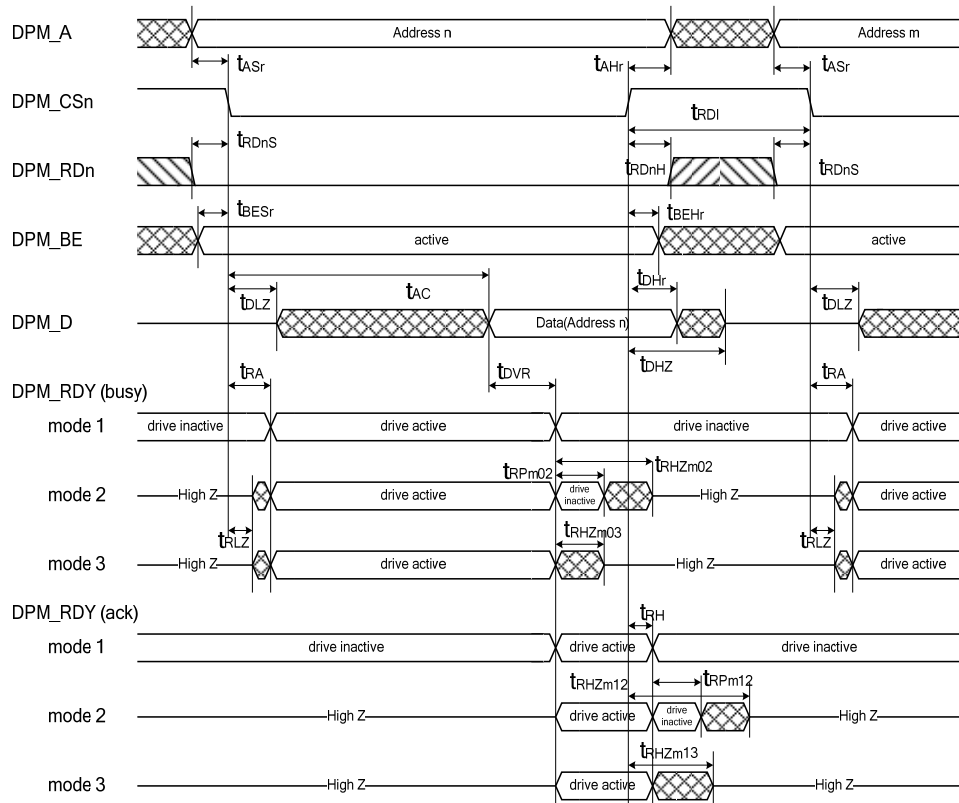


Figure 149: Detailed timing of an Intel-like chip-select controlled read access with read byte enables

4.4.9.3.2 Intel-like mode read access not using byte enables

Read byte enables can be ignored by programming bit *be_rd_dis* of register *dpm_if_cfg* to 1 (e.g. to support interconnection with separated byte enables for read and write). A read access starts if RDn and chip-select signal becomes low at access start. Address lines must be stable then.

Note: Figure 150 shows Intel-like read access controlled by chip-select or RDn which can be used instead of Intel-like read access with byte enables. However, at least one of those two signals must become inactive between access operations. The idle time t_{RDI} is the time when at least one signal is inactive.

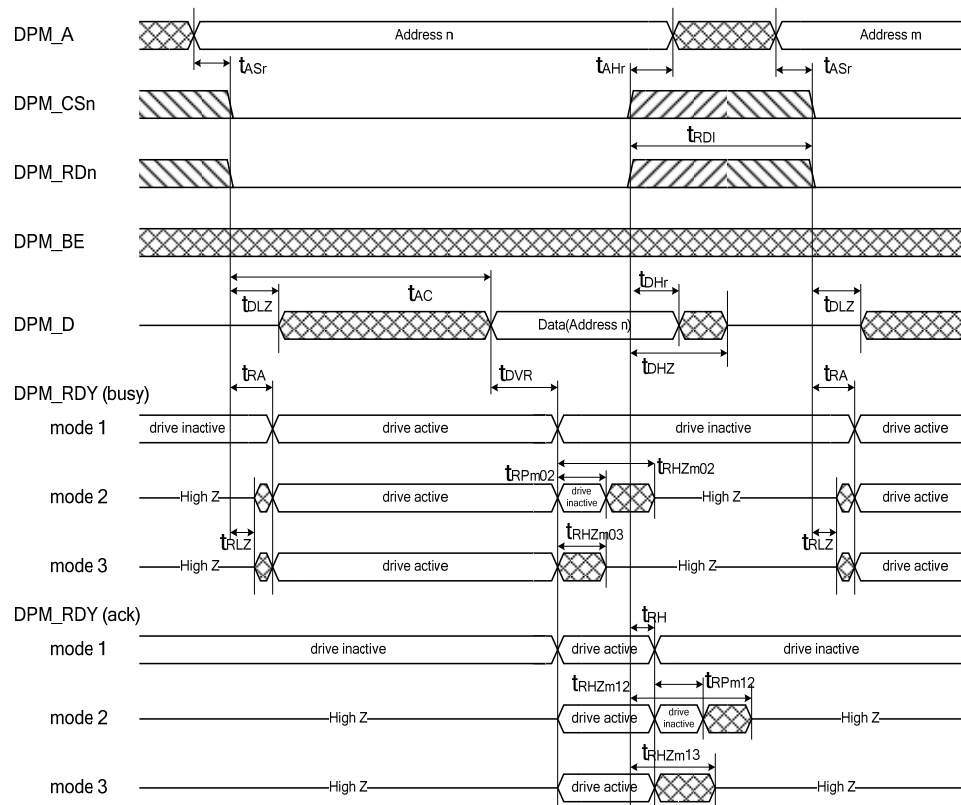


Figure 150: Detailed timing of an Intel-like read access not using read byte enables

4.4.9.3.3 Intel-like mode write access

- A write access is performed when byte enables become inactive at access end. Address, chip-select, data and RDn signals must be stable then.

Note: Byte enables are always required for write in Intel mode. Disabling them by programming bit *be_wr_dis* of register *dpm_if_cfg* to 1 is not allowed.

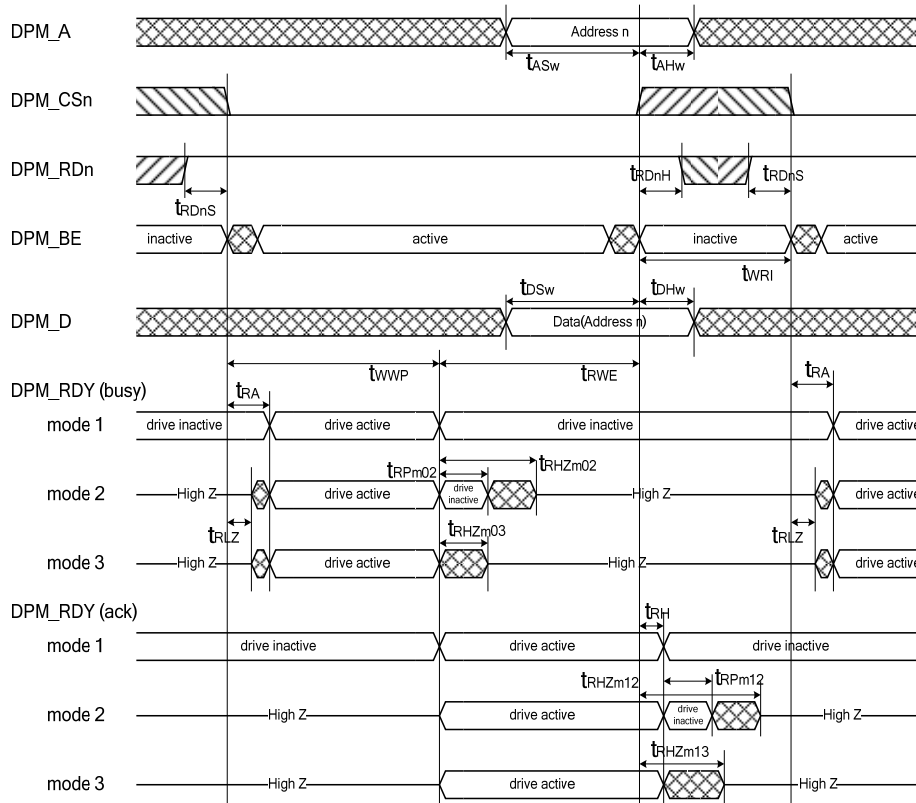


Figure 151: Detailed timing of an Intel-like write access with write byte enables

Note: For write, wait phase occurs only if DPM is busy internally. Acknowledge is always generated when DPM_RDY is in acknowledge mode (ack). This is similar to normal SRAM mode (Figure 141 and Figure 142).

4.4.9.4 Intel-like multiplexed modes with data direction byte strobe signals

This section is dedicated to the following modes:

- INTEL_16BIT_MUL_BYTE_ADDR
- INTEL_16BIT_MUL_BYTE_WRITE
- INTEL_32BIT_MUL_BYTE_ADDR
- INTEL_32BIT_MUL_DWORD_ADDR

Note: The multiplexed 8-bit Intel mode (INTEL_8BIT_MULTIPLEXED) and the netX 50-compatible multiplexed 16-bit Intel mode without byte enables (INTEL_16BIT_MUL_NO_BES) are rather multiplexed SRAM modes. To avoid confusion by mode naming, see the beginning of section 4.4.9.2.

Byte enable signals (DPM_BE signals) have strobe character for multiplexed Intel-like modes i.e. are byte strobes here. They must become active after the direction signal DPM_RDn is already

valid and stable. To avoid an illegal data direction change, DPM_RDn must not change before the byte strobes become inactive at access end.

Note: Byte strobe signals (on DPM_BE signals) are always available on dedicated signals for multiplexed Intel modes (i.e. are not part of the multiplexed address on the A/D-bus). They can be selected by register *dpm_if_cfg.be_sel*.

Note: The multiplexed 8-bit Intel mode (INTEL_8BIT_MULTIPLEXED) and the netX 50-compatible multiplexed 16-bit Intel mode without byte enables (INTEL_16BIT_MUL_NO_BES) are rather multiplexed SRAM modes and covered by section 4.4.9.2.

4.4.9.4.1 Intel-like multiplexed mode read access using byte enable

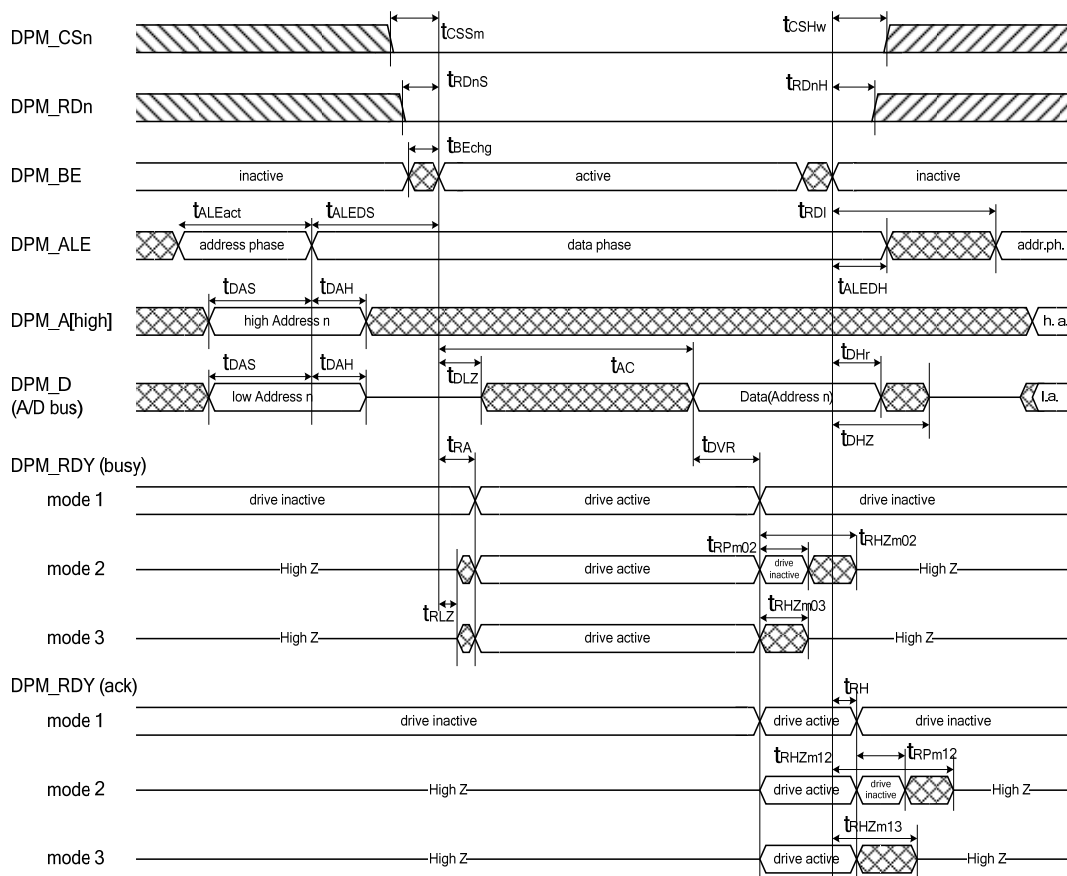


Figure 152: Detailed timing of an Intel-like multiplexed mode read access

The address phase could also overlap with the read-active phase, i.e. DPM_ALE could remain in address phase state while DPM_RDn and byte enables are already set to active. However, the data phase never starts before DPM_ALE changes to data state, i.e. ready and read data will never be driven while DPM_ALE is in address state. The host must make sure that the addresses are not driven too long for read access. Otherwise netX will drive read data against addresses driven by the host.

Important: If the address phase overlaps with the read-active phase, the address hold time t_{DAH} must be shorter than the read-data-enable time t_{DLZ} to prevent damage to netX or host.

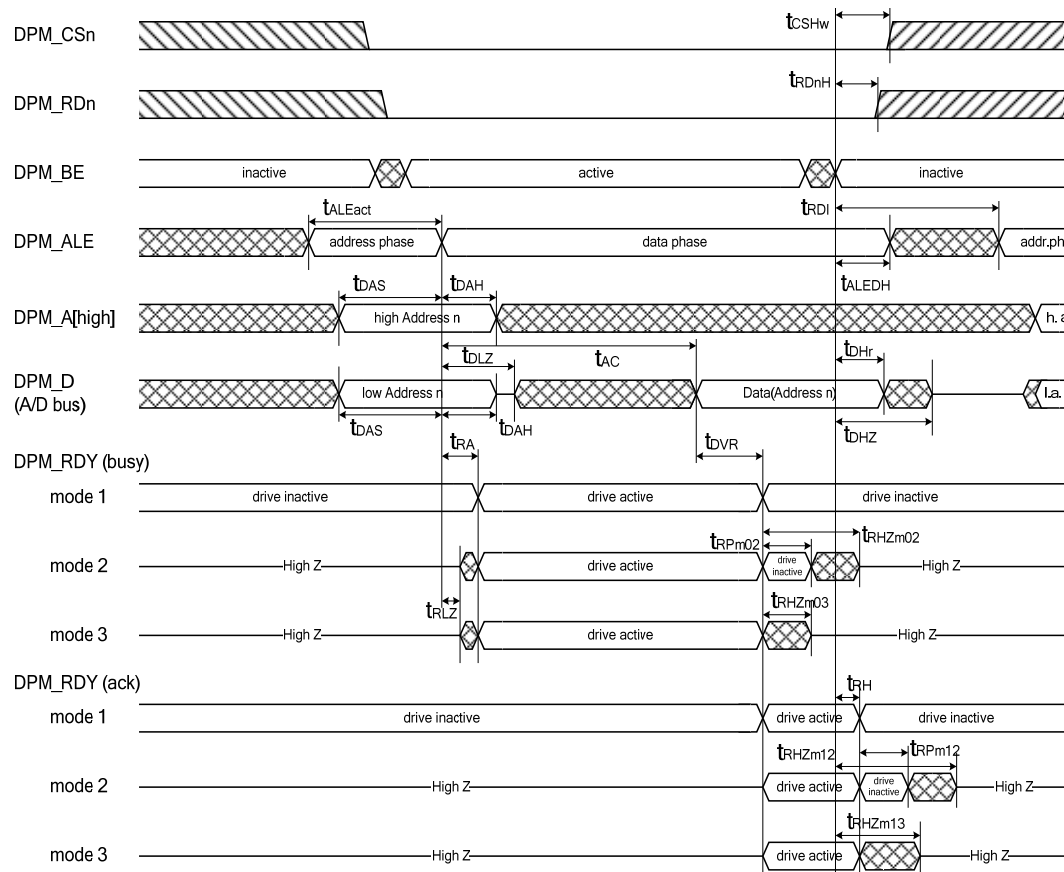
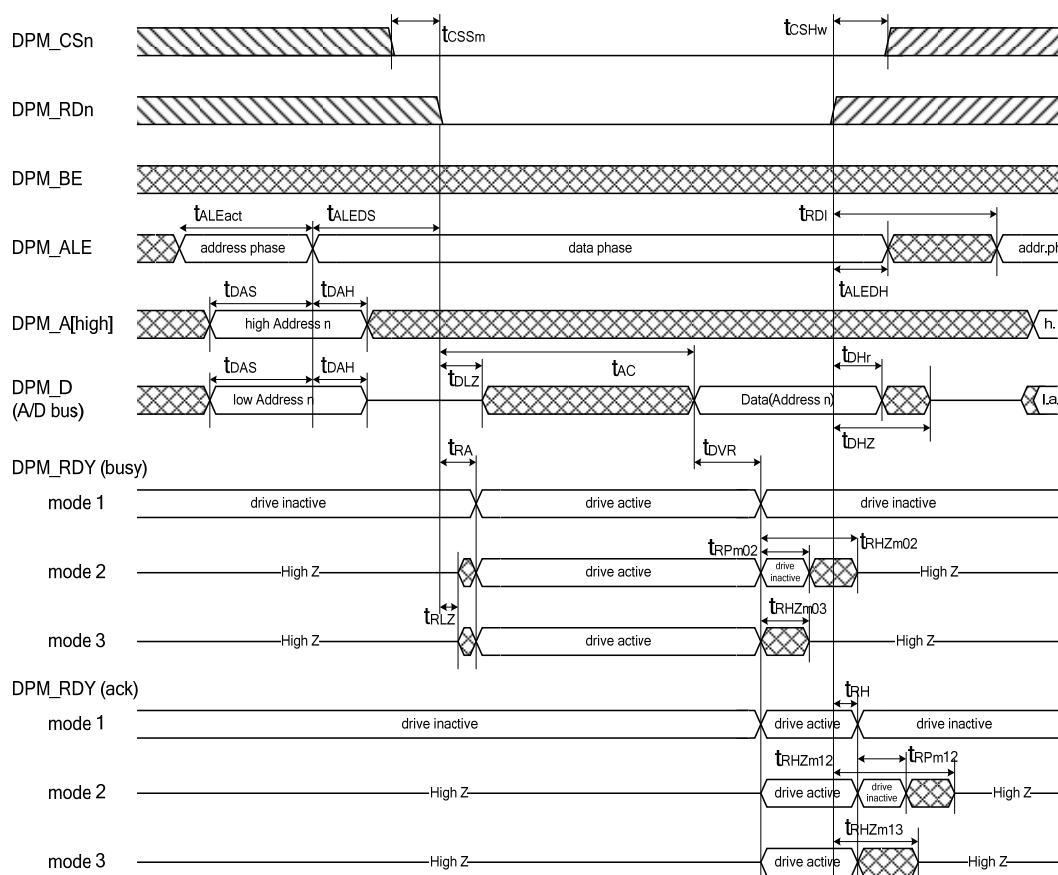


Figure 153: Detailed timing of an Intel-like multiplexed mode read access with address phase overlapping with read-active phase

Read byte enables can be ignored by programming bit *be_rd_dis* of register *dpm_if_cfg* to 1 (e.g. to support interconnection with separated byte enables for read and write). A read access is started then when RDn and chip-select signal become low at access start. Address lines must be stable then.

Note: Figure 154 shows Intel-like read access controlled by chip-select or RDn which can be used instead of Intel-like read access with byte enables. However, at least one of those two signals must become inactive between access operations. The idle time t_{RDI} is the time when at least one signal is inactive.



Note: If byte enables are ignored, the address phase could also overlap with the read-active phase. This is similar to Figure 153.

4.4.9.4.3 Intel-like multiplexed mode write access

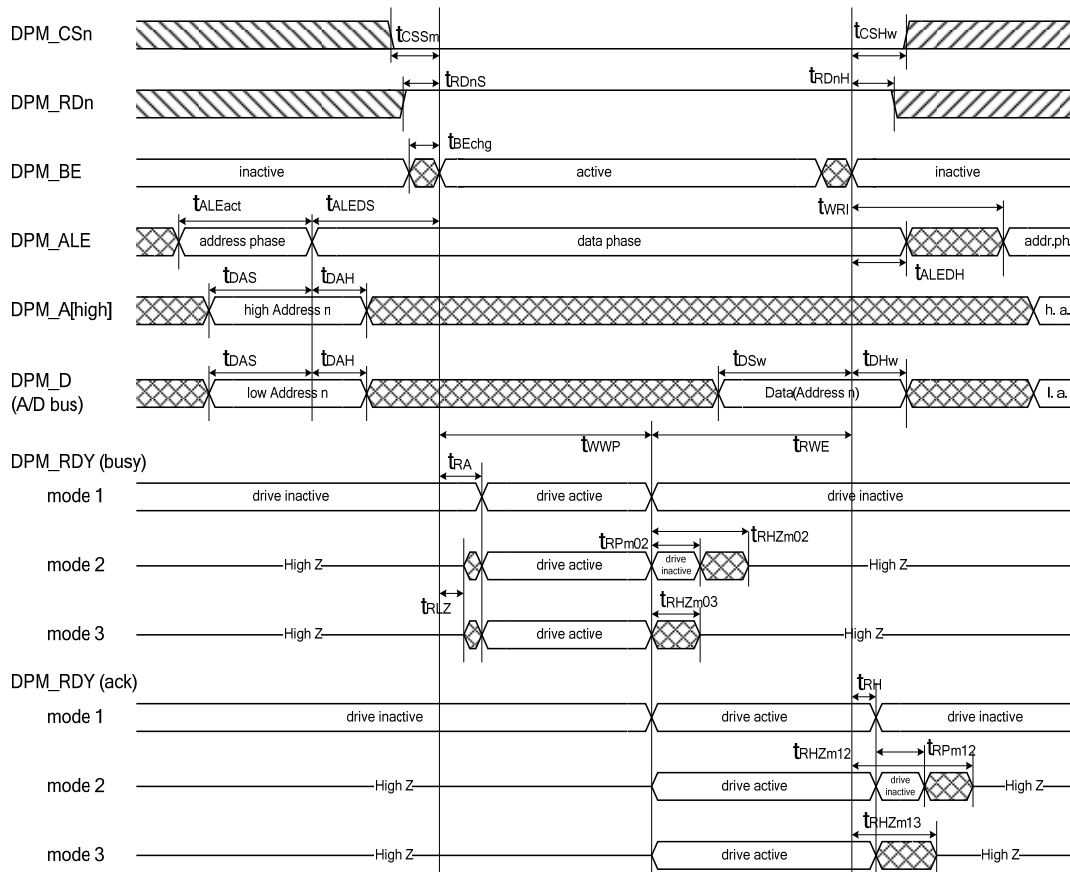


Figure 155: Detailed timing of an Intel-like multiplexed mode write access when internal netX DPM side busy (insertion of wait cycles)

Note: For write, wait phase occurs only if DPM is busy internally. Acknowledge is always generated when DPM_RDY is in acknowledge mode (ack). This is similar to normal SRAM mode (Figure 145 and Figure 146).

Note: Also for write address phase could overlap with write-active phase. This is similar to Figure 153.

4.4.9.5 Motorola-like modes with data direction byte strobe signals

This section is dedicated to the following modes:

- MOTOROLA_16BIT
- MOTOROLA_16BIT_68000
- MOTOROLA_32BIT
- MOTOROLA_8BIT_6800

Byte enable signals (DPM_BE signals) have strobe character for Motorola-like modes i.e. are byte strobes here. They must become active after the direction signal DPM_RDn is already valid and stable. To avoid an illegal data direction change, DPM_RDn must not change before the byte strobes become inactive at access end.

Note: Byte strobe signals (on DPM_BE signals) can be selected by register *dpm_if_cfg.be_sel*.

Note: The address enable signal is not mandatory. It could be used e.g. for Motorola 68000.

Note: Motorola 6800 is 8-bit with byte enable.

4.4.9.5.1 Motorola-like mode read access

A read access starts when byte enables (byte enable controlled) or chip-select (chip-select controlled) signals become active at access start. Address and data direction (RDn) signals must be stable then.

Note: Byte enables are always required for read in Motorola mode. Disabling them by programming bit *be_rd_dis* of register *dpm_if_cfg* to 1 is not allowed.

Note: Access detection will wait for stable byte enables of at least 1 netX system clock cycle at access start (t_{BEchg}). This could be extended using the programmable t_{OSA} timing parameter. Further changes of the byte enable signals will be ignored during access until they all become inactive. When all byte enables return to inactive state, access is terminated.

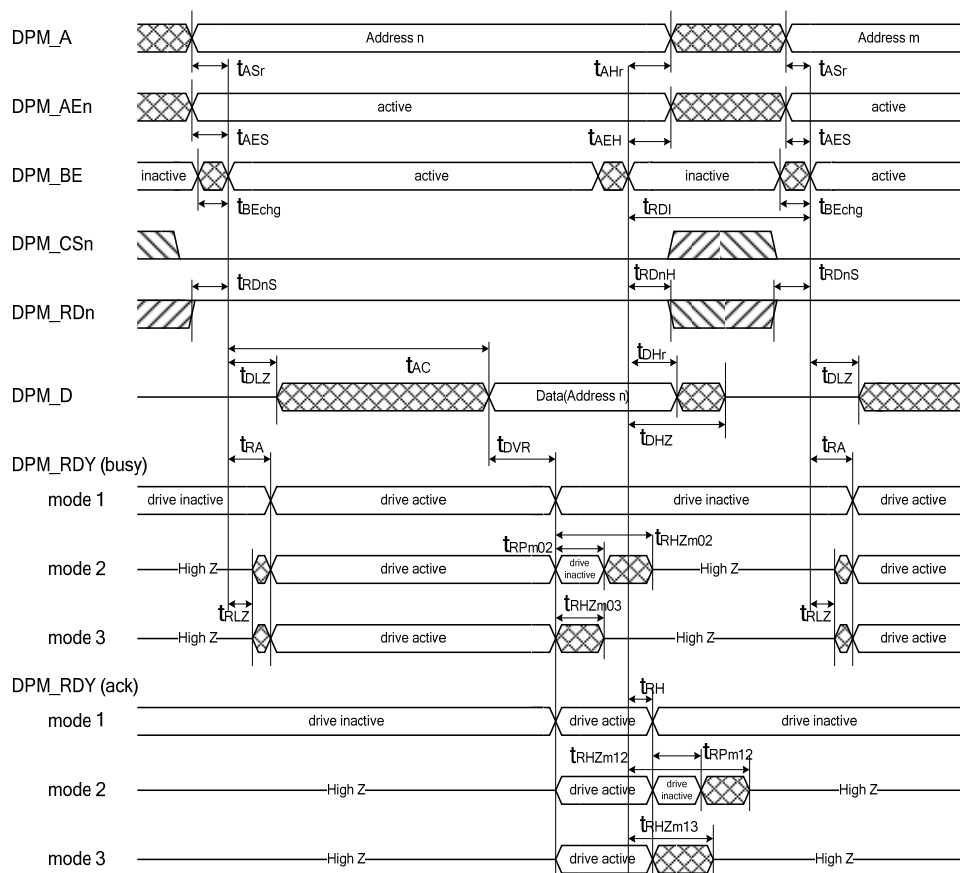


Figure 156: Detailed timing of a Motorola-like byte enable controlled read access

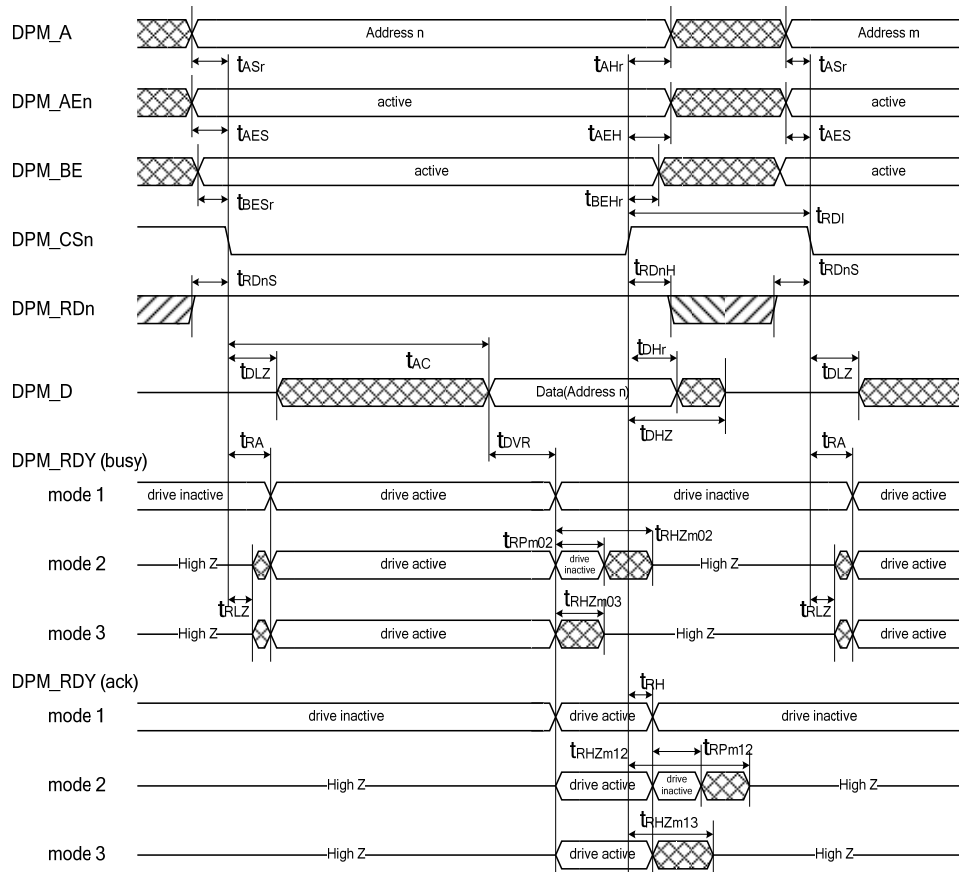


Figure 157: Detailed timing of a Motorola-like chip-select controlled read access

4.4.9.5.2 Motorola-like mode write access

- A write access is performed when byte enables become inactive at access end. Address, chip-select, data and RDn signals must be stable then.

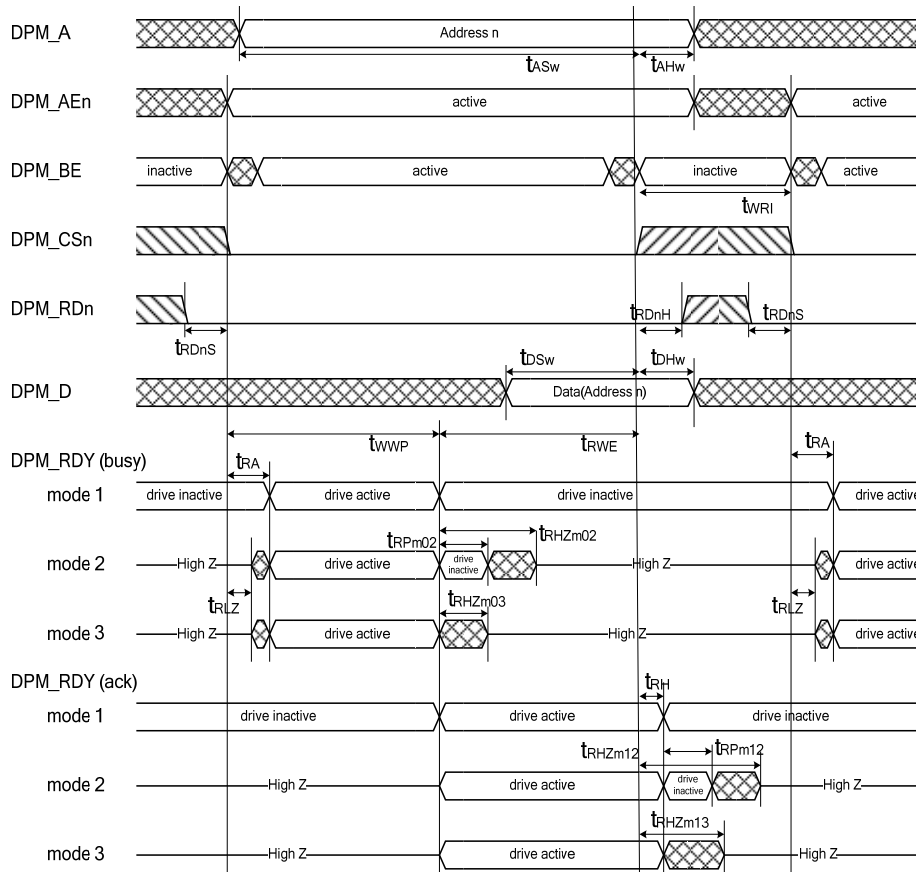


Figure 158: Detailed timing of a Motorola-like write access with write byte enables

Note: For write, wait phase occurs only if DPM is busy internally. Acknowledge is always generated when DPM_RDY is in acknowledge mode (ack). This is similar to normal SRAM mode (Figure 141 and Figure 142).

Note: Write byte enables could be ignored by programming bit *be_wr_dis* of register *dpm_if_cfg* to 1 for Motorola mode (e.g. to support interconnection with separated byte enables for read and write). Write data is stored at the positive edge of RDn in this case. All other DPM input signals must be valid and stable then.

4.4.9.6 Motorola-like multiplexed modes with data direction byte strobe signals

This section is dedicated to the following modes:

- MOTOROLA_8BIT_MULTIPLEXED
- MOTOROLA_16BIT_MUL_BYTE_ADDR
- MOTOROLA_16BIT_MUL_WORD_ADDR
- MOTOROLA_32BIT_MUL_BYTE_ADDR
- MOTOROLA_32BIT_MUL_DWORD_ADDR

Byte enable signals (DPM_BE signals) have strobe character for multiplexed Motorola-like modes i.e. are byte strobes here. They must become active after the direction signal DPM_RDn is already valid and stable. To avoid an illegal data direction change, DPM_RDn must not change before the byte strobes become inactive at access end.

Note: Byte strobe signals (on DPM_BE signals) are always available on dedicated signals for multiplexed Motorola modes (i.e. are not part of the multiplexed address on the A/D-bus). They can be selected by register *dpm_if_cfg.be_sel*.

4.4.9.6.1 Motorola-like multiplexed mode read access

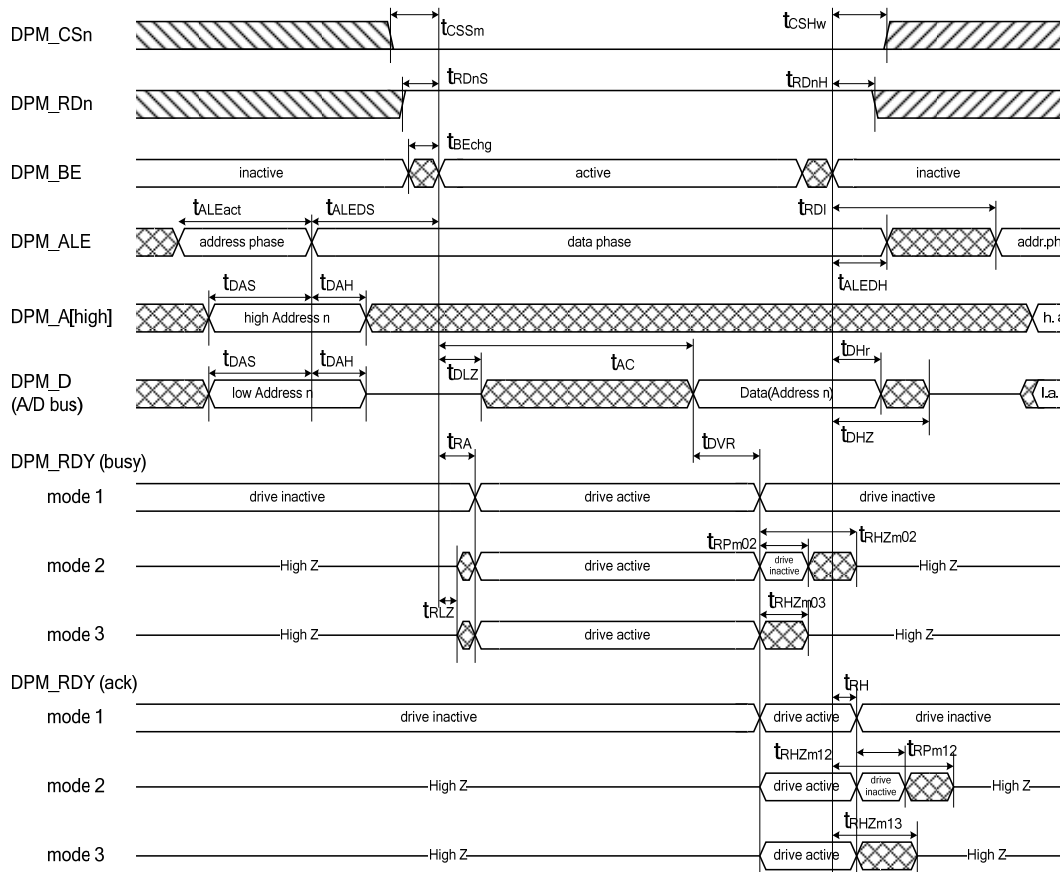


Figure 159: Detailed timing of a Motorola-like multiplexed mode read access

The address phase could also overlap into read-active phase, i.e. DPM_ALE could remain in address phase state while DPM_RD[n] and byte enables are already set to active. However, the data phase never starts before DPM_ALE changes to data state, i.e. ready and read data will never be driven while DPM_ALE is in the address state. The host must make sure that the addresses are not driven too long for read access. Otherwise netX will drive read data against addresses driven by host.

Important: If the address phase overlaps with the read-active phase address, the hold time t_{DAH} must be shorter than the read data enable time t_{DLZ} to prevent damage to netX or host.

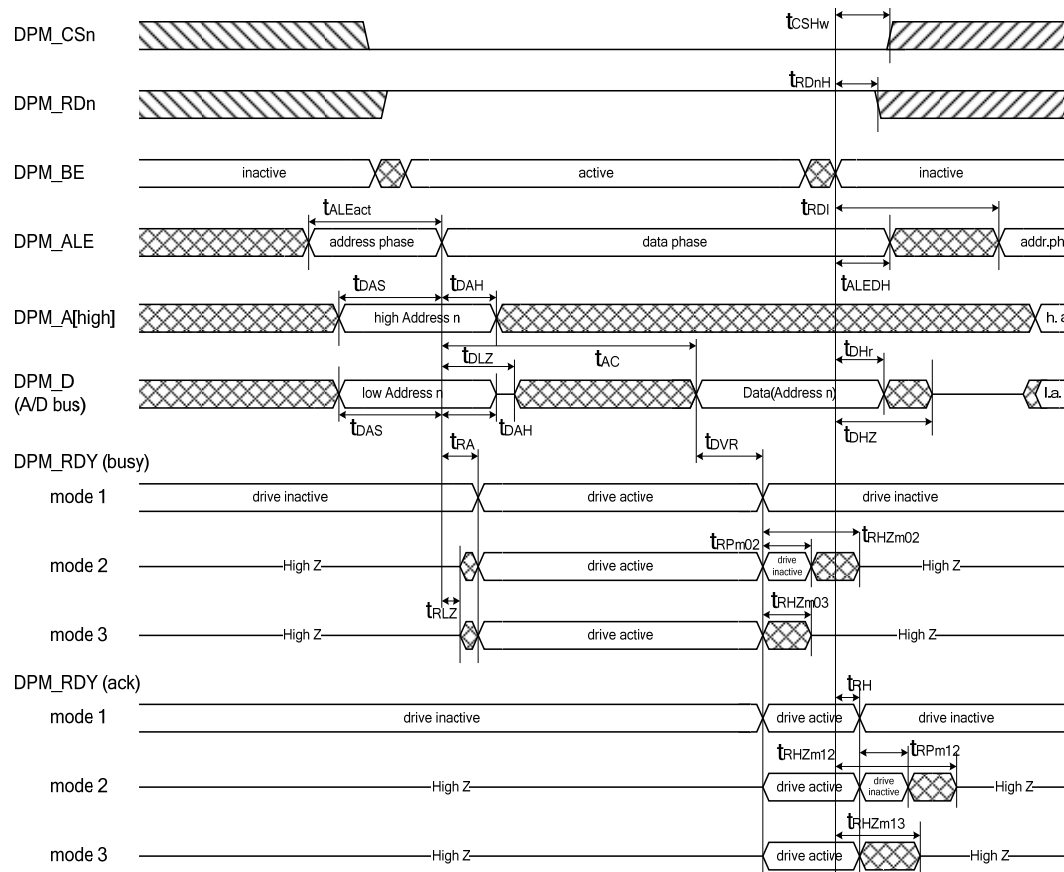


Figure 160: Detailed timing of a Motorola-like multiplexed mode read access with address phase overlapping with read-active phase

4.4.9.6.2 Motorola-like multiplexed mode write access

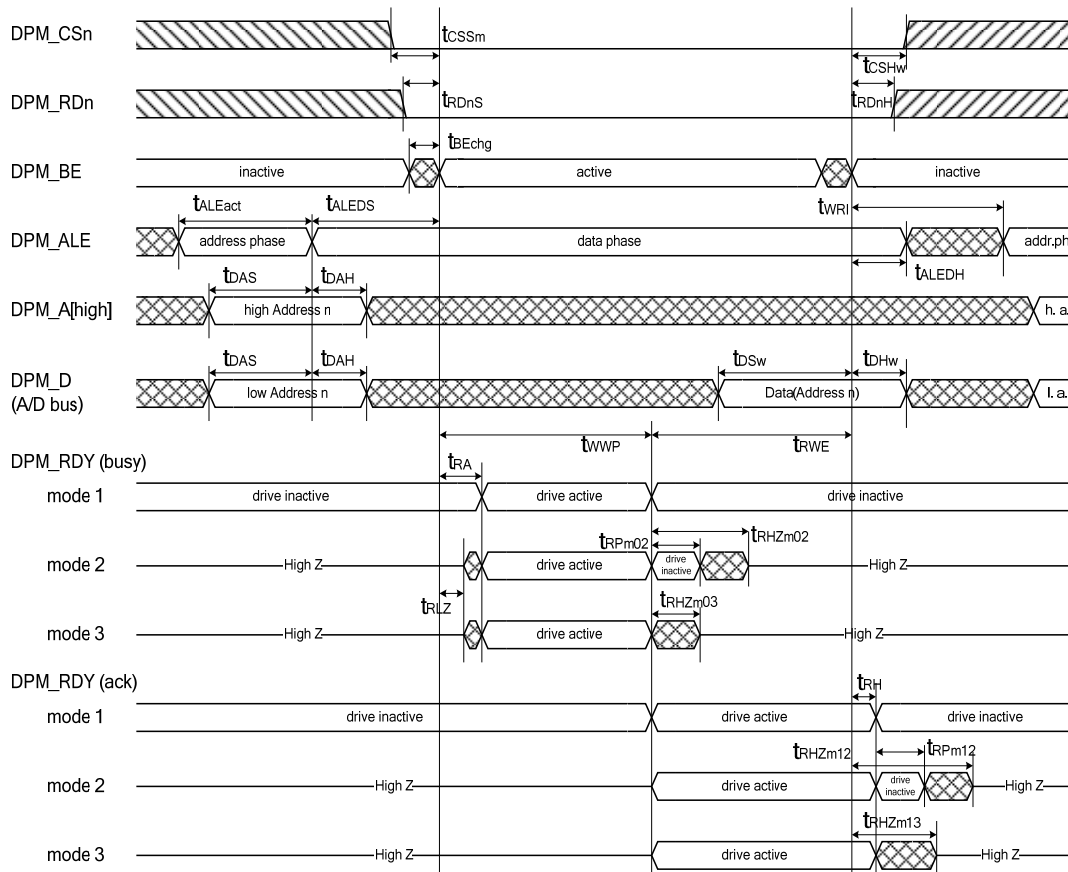


Figure 161: Detailed timing of an Intel-like multiplexed mode write access when internal netX DPM side busy (insertion of wait cycles).

Note: For write, wait phase occurs only if DPM is busy internally. Acknowledge is always generated when DPM_RDY is in acknowledge mode (ack). This is similar to normal SRAM mode (Figure 145 and Figure 146).

Note: Also for write address phase could overlap with the write-active phase. This is similar to Figure 160.

4.4.9.7 Parallel DPM timing parameters

Configurable timing parameters IO timings are derived from

Symbol	Parameter	Min	Typ	Max	Unit
$t_{\text{sys}}^{1)}$	Main system clock period	10.0			ns
$t_{\text{OSA}}^{2)}$	Programmable DPM address setup time parameter	0.0		$3t_{\text{sys}}$	ns
$t_{\text{RDS}}^{2)}$	Programmable DPM read data setup time parameter	0.0		$7t_{\text{sys}}$	ns
$t_{\text{f}}^{2)}$	Delay time of programmable input signal filter				ns
	filtering disabled	0.0		0.0	
	filtering enabled	t_{sys}		t_{sys}	

Notes

1. t_{sys} is a netX-internal system clock period (by default 10.0 ns). The system clock period affects DPM IO timing since sample and output register stages run on this clock. The system clock speed could be decreased (e.g. for power-down) by configuration registers in the netX *ASIC_CTRL* address area
2. Configurable by register *dpm_timing_cfg*.

Standard timing parameters

The following standard timing parameters can be applied for SRAM mode and access to netX address areas without wait states (i.e. INTRAM, handshake cells, DPM configuration, and status window 0). They are valid for normal netX operation (i.e. a netX running with 100 MHz system clock which is not stepped down). For all other cases the more complex general timing parameters (provided below) must be applied.

Symbol	Parameter	Min	Max	Unit
t_{ASr}	Address setup time (read cycle)	2.9 $2.9 - t_{OSA}^{1)2)}$		ns
t_{Ahr}	Address hold time (read cycle)	0.0		ns
t_{ASw}	Address setup time (write cycle)	12.8 $t_r + 12.8^{1)}$		ns
t_{Ahw}	Address hold time (write cycle)	2.9		ns
t_{AES}	Address enable setup time	0		ns
t_{AEH}	Address enable hold time	0		ns
t_{DSw}	Data setup time (write cycle)	12.8 $t_r + 12.8^{1)}$		ns
t_{Dhr}	Data hold time (read cycle)	2.1		ns
t_{Dhw}	Data hold time (write cycle)	2.8		ns
t_{DLZ}	Data enable low-Z time (read cycle)	5.7 $t_{OSA} + t_r + 5.7^{1)}$		ns
t_{DHz}	Data high-Z time (read cycle)		4.9	ns
t_{AC}	Access time (read cycle)		55.0 $55.0 + t_{OSA} + t_r^{1)}$	ns
t_{RDI} t_{WRI}	Read inter access idle time Write inter access idle time	12.5 $t_r + 12.5^{1)}$		ns
t_{WC}	Write cycle time	12.5 $t_r + 12.5^{1)}$		ns
t_{CSHw}	Chip-select hold time (write cycle)	0.0		ns
t_{WWP}	Write access wait phase	0.0		ns
t_{RWE}	Ready to write cycle end time	0.0		ns

Notes

1. This time is influenced by a DPM timing setting that can be programmed by software (3.11.5.7 and first table of this section).
2. The read address setup time can be decreased by the programmable timing parameter t_{OSA} (0..3 system clock cycles. The setup time becomes negative, e.g. $t_{OSA}=1$ leads to $t_{AS}=-9.1$ ns. That means a stable address is required 9.1 ns after chip-select and read enable have become active. That avoids problems caused by different signal runtimes if host devices simultaneously activate read enable, chip-select and address signals.

General timing parameters

Symbol	Parameter	Min	Max	Unit
t_{ASr}	Address setup time (read cycle)	2.9 $2.9 - t_{OSA}^{1)}$		ns
t_{Ahr}	Address hold time (read cycle)	0.0		ns
t_{ASw}	Address setup time (write cycle)	12.8 $t_{SYS} + t_r + 2.8$		ns
t_{Ahw}	Address hold time (write cycle)	2.9		ns
t_{AES}	Address enable Setup time	0		ns
t_{AEH}	Address enable hold time	0		ns
t_{BES}	Byte enable setup time	0		ns
t_{BEH}	Byte enable hold time	0		ns
t_{BEchg}	Byte enable change time		$t_{SYS} + t_{OSA}$	ns
t_{RDnS}	Data direction (RDn) setup time			ns
t_{RDnH}	Data direction (RDn) hold time			ns
t_{DSw}	Data setup time (write cycle)	12.8 $t_{SYS} + t_r + 2.8$		ns
t_{Dhr}	Data hold time (read cycle)	2.1		ns
t_{Dhw}	Data hold time (write cycle)	2.8		ns
t_{DLZ}	Data enable low-Z time (read cycle)	5.7 $t_{SYS} + t_{OSA} + t_r - 4.3$	$2t_{SYS} + t_{OSA} - 3.1$	ns
t_{DHz}	Data high-Z time (read cycle)		4.9	ns
t_{AC}	Access time (read cycle) Read-data latched Read data not latched not shared shared	33.7 ²⁾ $2t_{SYS} + t_{OSA} + t_r + 13.7^{2)}$ 53.2 ³⁾ $(4+w)t_{SYS} + t_{OSA} + t_r + 13.2^{4)}$ $(3+w_m)t_{SYS} + t_{OSA} + t_r + 13.2^{5)}$	$2048t_{SYS}^{6)}$ $2048t_{SYS}^{6)}$	ns
t_{RDI}	Read inter access idle time	12.5		ns
t_{WRI}	Write inter access idle time	$t_{SYS} + t_r + 2.5^{7)}$		ns
t_{WC}	Write cycle time	12.5 $t_{SYS} + t_r + 2.5$		ns
t_{CSHw}	Chip-select hold time (write cycle)	0.0		ns
t_{WWP}	Write access wait phase	0.0	$2048t_{SYS}^{6)}$	ns
t_{RWE}	Ready to write cycle end time	0.0		ns

Notes

1. The read address setup time can be decreased by the programmable timing parameter t_{OSA} (0..3 system clock cycles). The setup time becomes negative, e.g. $t_{OSA} = 1$ leads to $t_{AS} = -9.1$ ns. That means a stable address is required 9.1 ns after chip-select and read enable have become active. That avoids problems caused by different signal runtimes if host devices simultaneously activate read enable, chip-select and address signals.
2. This timing is valid when requested read data is already latched inside the DPM module (3.11.8.2.1) or when the read access targets the read ahead address area and the read access matches read ahead address

3. Standard read access to netX address area with 0 wait states (e.g. INTRAM, handshake cells). It has no effect whether a 0 wait state AHB slave is shared with other system masters or not. This timing is valid for first read after 4-byte address boundary changed or for read ahead mismatch.
4. Standard read access to netX address area with w wait states when netX AHB slave target is not shared with another netX system master. If access targets netX address area with w wait states, access time will be extended by 1 system clock for each wait state (netX wait state areas, system memory INTRAM and handshake cells run with 0 wait states.)
5. Standard read access to netX address area with w wait states when netX AHB slave target is shared with another netX system master. Example: DPM access to INTLOGIC area (e.g. to xC Pointer FIFO) when netX ARM CPU also performs access to INTLOGIC area (e.g. SPI controller). Max. wait states is $w_m = w_o + w_{DPM}$ where w_o are max. internal wait states of other master and w_{DPM} are max. internal wait states of DPM access. (netX wait state areas:, system memory INTRAM and handshake cells run with 0 wait states.)
6. Ready timeout if access permanently targets busy netX address area.
7. If read burst support is not enabled.

Multiplexed mode timing parameters

Note: There is no CS dependency for address-phase in multiplexed modes. Address sampling (also for high address lines, but not for SRAM mode byte enables) is performed around the change from address to data phase of the DPM_ALE signal also while chip-select is inactive.

Symbol	Parameter	Min	Type	Max	Unit
t_{ALEact}	Address phase active time	12.7 $t_{sys} + t_r + 2.7$			ns
t_{ALEDS}	ALE data phase setup time	0.0 ¹⁾			ns
t_{ALEDH}	ALE data phase hold time	0.0			ns
t_{CSSm}	Chip-select setup time for multiplexed modes	0.0			
t_{DAS}	Data multiplexed address setup time	12.6 $t_{sys} + t_r + 2.6$			ns
t_{DAH}	Data multiplexed address hold time	3.0			ns

Note

1. Address phase could even overlap with the active read or write enable phase. For read, netX will never drive data lines while ALE is active even if read enable is already activated by host.

Ready signal timing parameters

Symbol	Parameter	Min	Type	Max	Unit
t_{DVR}	Data valid to ready time (read cycle)	$t_{RDS} - 2.2^{1)}$			ns
t_{RA}	Ready active time			5.7	ns
t_{RLZ}	Ready enable low-Z time	2.8			ns
t_{RHZ}	Ready high-Z time	²⁾		²⁾	
t_{RH}	Ready hold time	2.6			ns

Notes

1. Data valid to ready time can be increased by programmable timing parameter t_{RDS} (0..7 system clock cycles, see reference [1]). For $t_{RDS} = 0$, DPM_RDY will be set 0.7 ns before read data is valid. For a correct t_{RDS} setting, see host device requirements and data sheet.
2. Ready high-Z times differ depending on the programmed ready mode. For information, see the following tables.

Special ready timing for ready signal mode 0 and drive mode 2

Symbol	Parameter	Min	Type	Max	Unit
t_{RPm02}	Ready pulse width	8.0 $t_{sys} + t_r - 2.0$			ns
t_{RHZm02}	Ready high-Z time			10.4 $t_{sys} + t_r + 0.4$	ns

Ready timing for ready signal mode 0 and drive mode 3

Symbol	Parameter	Min	Type	Max	Unit
t_{RHZm03}	Ready high-Z time			1.3	ns

Ready timing for ready signal mode 1 and drive mode 2

Symbol	Parameter	Min	Type	Max	Unit
t_{RPm12}	Ready pulse width	5.3 $t_{sys} + t_r - 4.7$			ns
t_{RHZm12}	Ready high-Z time			19.4 $2t_{sys} + t_r - 0.6$	ns

Ready timing for ready signal mode 1 and drive mode 3

Symbol	Parameter	Min	Type	Max	Unit
t_{RHZm13}	Ready high-Z time			3.8	ns

4.4.10 Serial dual-port memory timing

This section provides detailed timing information for 1-bit and 4-bit serial DPM.

Important: In 4-bit mode the SIO lines are used bidirectionally for both, read and write data. This could lead to driving conflicts which may cause permanent damage to netX and host. To avoid driving conflicts the host must not drive the SIO lines while they are driven by the netX. For details refer to the serial quad DPM protocol, section 3.12.4.1.

Note: For serial DPM pinning, see 3.12.2 and 3.12.4. For 1-bit DPM protocol, see 3.12.3.

The serial dual-port memory provides the programmable timing parameter *sdpm_miso_early*. This parameter controls the generation of serial output data. Timing parameter *sdpm_miso_early* is not active by default and serial dual-port memory behaves like standard SPI: Read data is changed on the clock edge following the sampling edge, i.e. it works on both serial clock edges. In this case, valid setup and hold times can always be reached by decreasing the serial clock rate. However, it is not possible to reach high clock rates then as there is only a half serial period margin for read data generation.

The timing parameter *sdpm_miso_early* must be set to reach fast serial data rates. Read data is generated on the sampling edge (i.e. half a period earlier). However, this could lead to critical hold timing: Due to the delay times of serial clock and data signals, the hold time will always be positive but can be very short.

For serial clock rates above 33 MHz, we recommend setting *sdpm_miso_early* to '1'. Default setting '0' should be used for lower clock rates. The 4-bit mode is limited to 33 MHz, thus, *sdpm_miso_early* should not be activated for the 4-bit mode.

Setting of <i>sdpm_miso_early</i>	Setup timing	Hold timing	Comment
0	can be relaxed by decreasing the serial clock rate.	can be relaxed by decreasing the serial clock rate.	Standard SPI timing, recommended for serial clock rates below 33 MHz.
1	can be relaxed by decreasing the serial clock rate.	cannot be relaxed.	Early read data generation, recommended for serial clock rates above 33 MHz.

Note: In 1-bit mode, the serial status is returned with the first header byte at transfer start. The first 3 bits of that are always 0.

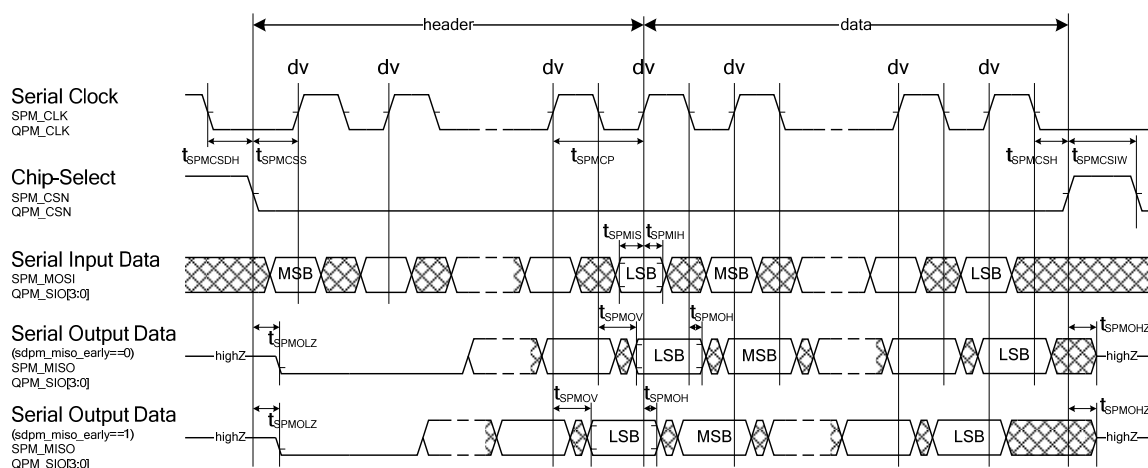


Figure 162: DPM SPI slave timing SPO=0 and SPH=0 transfer (data valid on positive edge of serial clock)

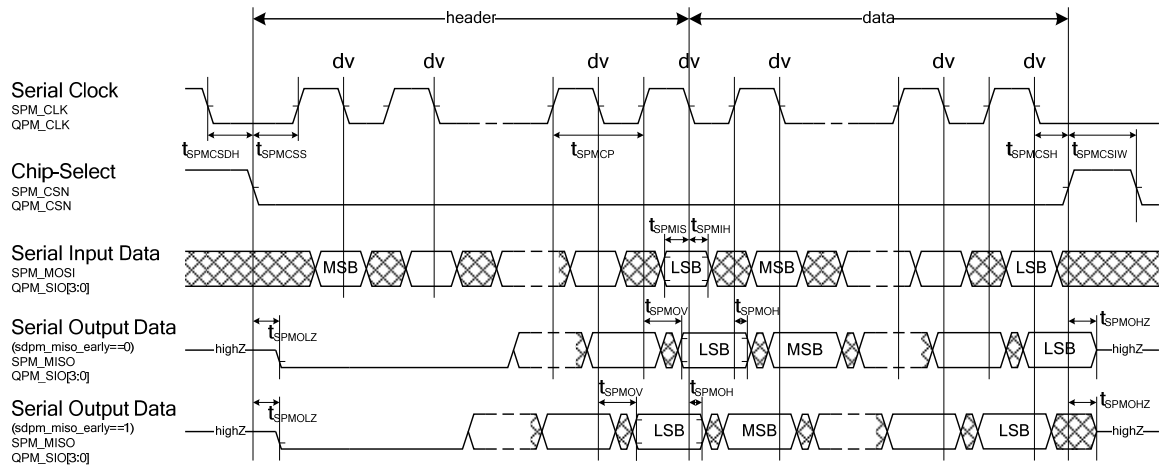


Figure 163: DPM SPI slave timing SPO=0 and SPH=1 transfer (data valid on negative edge of serial clock)

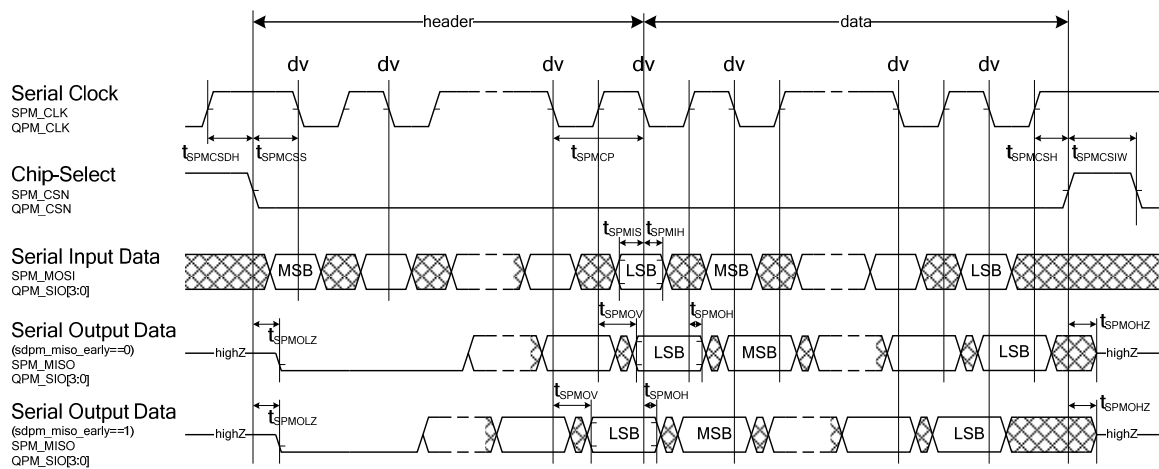


Figure 164: DPM SPI slave timing SPO=1 and SPH=0 transfer (data valid on positive edge of serial clock)

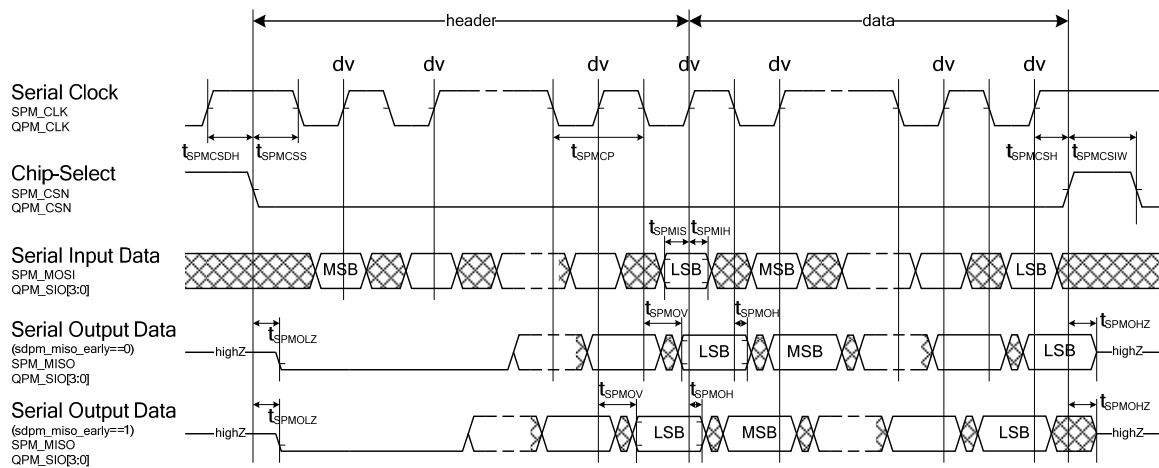


Figure 165: DPM SPI slave timing SPO=1 and SPH=1 transfer (data valid on negative edge of serial clock)

Serial DPM timing

Symbol	Timing parameter		Min	Type	Max	Unit
t _{SPMCP}	Serial clock period		1 bit	8.0 ¹⁾²⁾		ns
			4 bit	30.0 ¹⁾		
t _{SPMIS}	Serial input data setup time		1 bit	0.0		ns
			4 bit	1.4		
t _{SPMIH}	Serial input data hold time		1 bit	1.8		ns
			4 bit	2.0		
t _{SPMOV}	Serial output data valid time	sdpm_mosi_early programmed to 0	1 bit		8.0 ³⁾	ns
			4 bit		8.0 ³⁾	
		sdpm_mosi_early programmed to 1	1 bit		7.0 ³⁾	
			4 bit		8.0 ³⁾	
t _{SPMOH}	Serial output data hold time		2.5 ⁴⁾⁵⁾			ns
t _{SPMOLZ}	Chip select to serial output data low-Z time		2.0			ns
t _{SPMOHZ}	Chip select to serial output data high-Z time				3.0	ns
t _{SPMCSS}	Chip select to serial clock setup time		3.0			ns
t _{SPMCSDH}	Chip select from deselected serial clock edge hold time		3.0			ns
t _{SPMCSH}	Serial clock to chip deselect hold time		3.0			ns
t _{SPMCSIW}	Chip select idle width		7.0			ns

Notes

1. Serial clock duty cycle is 30% (due to the fact that the *sdpm_miso_early* parameter must be set by software for fast serial clock rates and the serial logic is working only on rising or falling clock edges then, but not on both).
2. Minimum serial clock period is limited by internal logic, not by serial data setup or hold times. If using serial clock rates above 66 MHz, observe the netX-internal access times (e.g. non INTRAM access, see section *Serial 1-bit DPM protocol* on page 133).
3. The serial output data valid timing refers to the output data generating clock edge. This can be controlled by the programmable timing parameter *sdpm_miso_early*. By default (*sdpm_miso_early* = '0') the timing refers to the serial clock edge before the data is sampled (i.e. half a serial clock period before it is sampled). For high serial clock rates with early output data generation (*sdpm_miso_early* set to '1') the timing refers to the serial clock edge when the preceding data bit is sampled (a full serial clock period before it is sampled).
4. The serial output data hold timing refers to the output data generating clock edge. This can be controlled by the programmable timing parameter *sdpm_miso_early*. By default (*sdpm_miso_early* = '0') the timing refers to the serial clock edge after the data is sampled (i.e. half a serial clock period after it is sampled). For high serial clock rates with early output data generation (*sdpm_miso_early* set to '1') the timing refers to the serial clock edge when the current data bit is sampled (sampling and generation take place on the same clock edge).
5. For all modes (1-bit, 4-bit, does not depend on programmable *sdpm_miso_early* parameter).

4.4.11 QSPI timing

The QSPI module I/O timing depends on pad delays and the internal module structure. All timings are related to an internal system clock running at 100 MHz.

The following figures show the timing for standard SPI mode.

Regarding the transfer direction, timing parameters are identical for Dual and Quad SPI mode: In receive mode timing for SPI_MISO can be applied to all I/O lines. In transmit mode timing for SPI_MOSI can be applied to all I/O lines.

In SQIROM mode timing from SPI Mode 0 (SPO=0, SPH=0) can be applied, but the max. SPI CLK frequency is increased to 133 MHz.

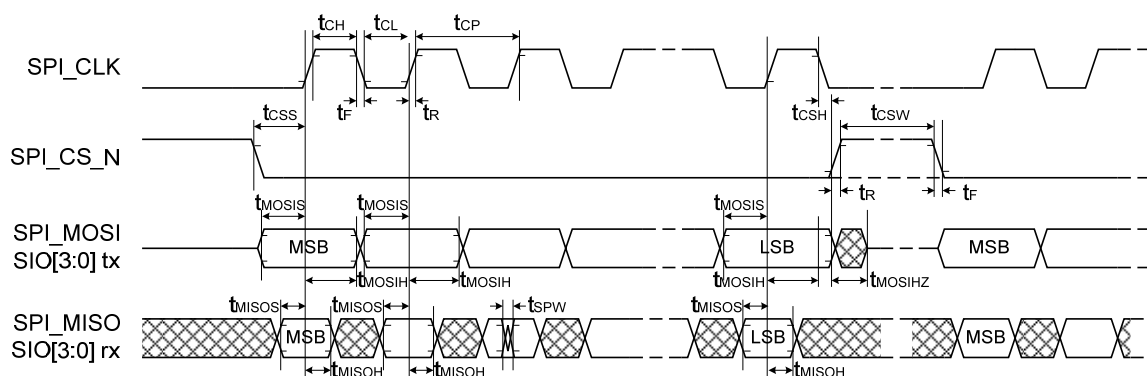


Figure 166: SPI timing SPO=0 and SPH=0 transfer (data generation on falling edge, sampling on rising edge of SPI CLK)

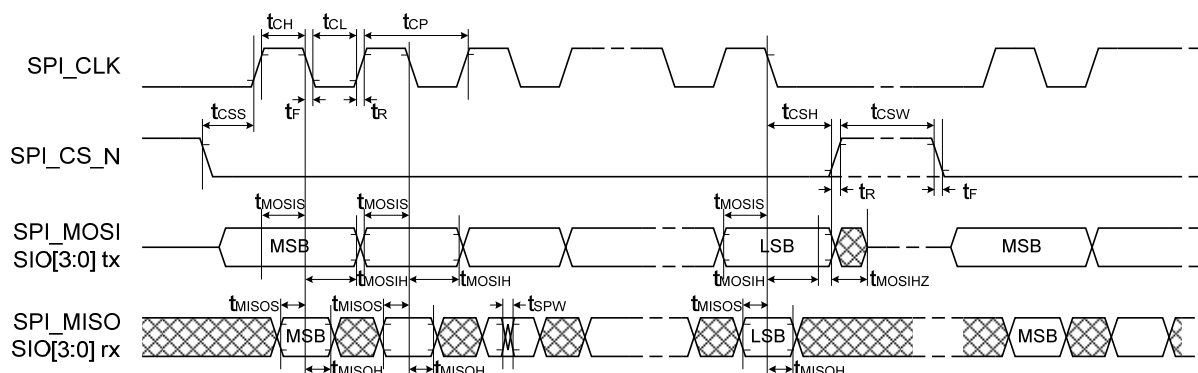


Figure 167: SPI timing SPO=0 and SPH=1 transfer (data generation on rising edge, sampling on falling edge of SPI_CLK)

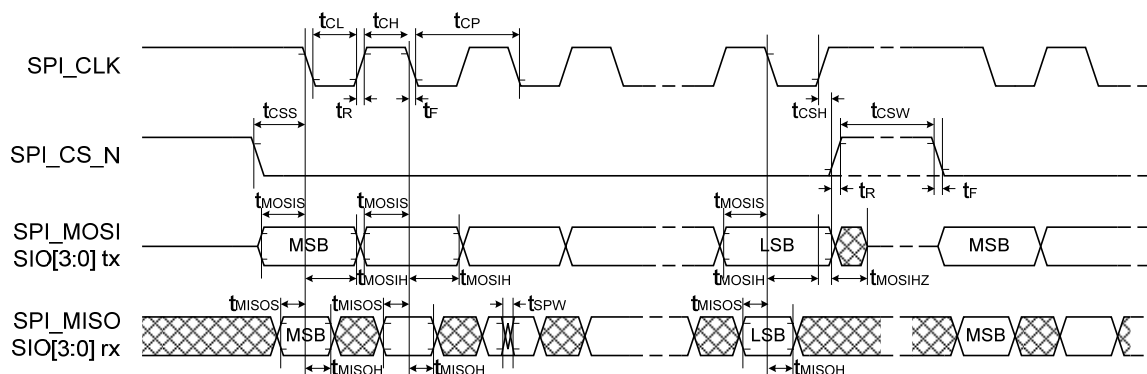


Figure 168: SPI timing SPO=1 and SPH=0 transfer (data generation on rising edge, sampling on falling edge of SPI CLK)

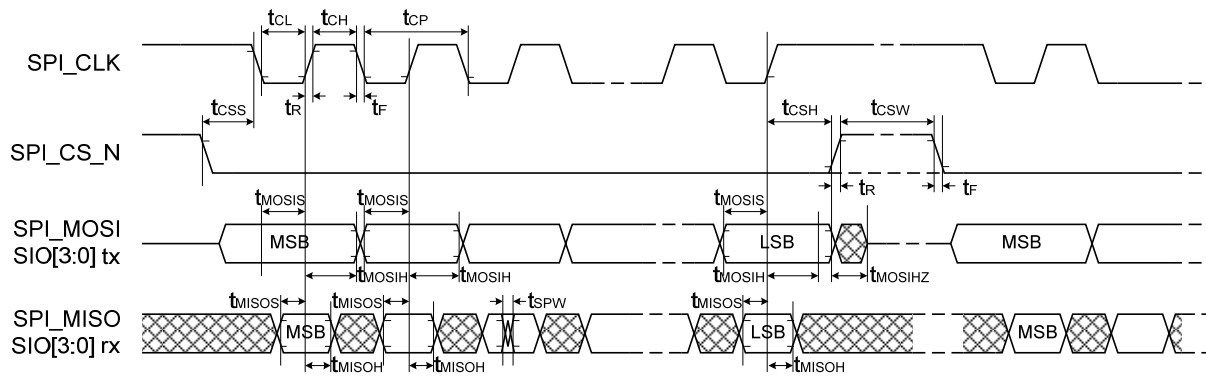


Figure 169: SPI timing SPO=1 and SPH=1 transfer (data generation on falling edge, sampling on rising edge of SPI_CLK)

The following values are valid for netX 51/52 when the module is in normal peripheral mode (not in SQIROM/ XiP mode, i.e. the *enable* bit of register *sqi_sqirom_cfg* is not set). They refer to worst case operating conditions: V_{DD} : 3.0 ... 3.6 V, T_j : -40 ... +125°C, C_L : 20 pF.

Symbol	Parameter	Min	Typ	Max	Unit
t_{CP}	SPI_CLK period ⁽¹⁾	20.0	10*4096/N	40960	ns
f_{CP}	SPI_CLK frequency ⁽¹⁾	0.025		50.0	MHz
t_{CH}	SPI_CLK high phase	$0.5 \cdot t_{CP} - 3.0$			ns
t_{CL}	SPI_CLK low phase	$0.5 \cdot t_{CP} - 3.0$			ns
t_R	Signal rise time ⁽²⁾	0.4		3.4	ns
t_F	Signal fall time ⁽²⁾	0.3		2.0	ns
t_{CSS}	SPI_CS_N to first SPI_CLK edge setup time	$0.5 \cdot t_{CP} - 3.0$			ns
t_{CSH}	last SPI_CLK edge to SPI_CS_N inactive time ⁽³⁾ - SPH 0 modes - SPH 1 modes	7.0 $0.5 \cdot t_{CP} + 7.0$			ns
t_{CSW}	SPI_CS_N minimum high pulse width ⁽³⁾	$0.5 \cdot t_{CP} - 3.0$			ns
t_{MOSIS}	SPI_MOSI (SPI_SIO3:0] transmit) to SPI_CLK setup time	$0.5 \cdot t_{CP} - 3.4$			ns
t_{MOSIH}	SPI_MOSI (SPI_SIO[3:0] transmit) hold time	$0.5 \cdot t_{CP} - 3.0$			ns
t_{MOSIHZ}	SPI_MOSI (SPI_SIO[3:0] transmit) high-Z time			3.5	ns
t_{MISOS}	SPI_MISO (SPI_SIO[3:0] receive) to SPI_CLK setup time	2.5			ns
t_{MISOH}	SPI_MISO (SPI_SIO[3:0] receive) hold time - without input filtering - with input filtering ⁽⁴⁾	4.5 14.5			ns
t_{SPW}	Tolerated spike pulse width - without input filtering - with input filtering ⁽⁴⁾			0.0 9.0	ns

Notes:

1. N is programmed by register *spi_cr0/sqi_cr0*, bits *sck_muladd*. $N = 1..2048$
2. Signal rise and fall times differ considerably depending on the external capacitive load. For approximation, use the following formula:

Rise times: $t_r = 0.350 + 0.150 \cdot C_L$ [ns]; C_L : External capacitive load

Fall times: $t_f = 0.237 + 0.087 \cdot C_L$ [ns]; C_L : External capacitive load

3. Only for automatic chip-select generation (if bit *fss_static* is not set in register *spi_cr1/sqi_cr1*).
4. Input filtering can be enabled/disabled in register *spi_cr0/sqi_cr0* by bit *filter_in*.

The following values are valid for netX 51/52 when the module is in SQIROM/XiP mode (not in normal peripheral mode, i.e. the *enable* bit of register *sqi_sqirom_cfg* is set). They refer to worst case operating conditions: V_{DD} : 3.0 ... 3.6 V, T_j : -40 ... +125°C, C_L : 20 pF. Only mode 0 and 3 are supported in SQIROM/XiP mode.

Symbol	Parameter	Min	Typ	Max	Unit
t_{CP}	SPI_CLK period ⁽¹⁾	7.5	(N+3)*2.5	645	ns
f_{CP}	SPI_CLK frequency ⁽¹⁾	1.55		133.0	MHz
t_{CH}	SPI_CLK high phase	$0.5 \cdot t_{CP} - 3.0$			ns
t_{CL}	SPI_CLK low phase	$0.5 \cdot t_{CP} - 3.0$			ns
t_R	Signal rise time ⁽²⁾	0.4		3.4	ns
t_F	Signal fall time ⁽²⁾	0.3		2.0	ns
t_{CSSr}	SPI_CS_N active to first rising SPI_CLK edge setup time ⁽³⁾	$t_{CP} - 1.0$			ns
t_{CSHr}	last rising SPI_CLK edge to SPI_CS_N inactive time ⁽³⁾	$t_{CP} - 1.0$			ns
t_{CSW}	SPI_CS_N minimum high pulse width ⁽⁴⁾	$t_{CP} - 3.0$	$1 \cdot t_{CP} - 3.0$	$4 \cdot t_{CP} - 3.0$	ns
t_{MOSIS}	SPI_MOSI (SPI_SIO[3:0] transmit) to SPI_CLK setup time	$0.5 \cdot t_{CP} - 2.6$			ns
t_{MOSIH}	SPI_MOSI (SPI_SIO[3:0] transmit) hold time ⁽⁶⁾	$0.5 \cdot t_{CP} - 1.2$			ns
t_{MOSIHZ}	SPI_MOSI (SPI_SIO[3:0] transmit) high-Z time			3.5	ns
t_{MISOS}	SPI_MISO (SPI_SIO[3:0] receive) to SPI_CLK setup time	0.3			ns
t_{MISOH}	SPI_MISO (SPI_SIO[3:0] receive) hold time ⁽⁷⁾	2.1			ns
t_{SPW}	Tolerated spike pulse width ⁽⁵⁾			0.0	ns

Notes:

1. N is programmed by *sqi_sqirom_cfg* register *clk_div_val* bits. N = 0...255. This leads to a frequency up to 133 MHz.
2. Signal rise and fall times differ greatly depending on external capacitive load. For approximation, see note 2 of the peripheral mode IO timing above, page 270.
3. The chip-select timing of QSPI devices is typically related to the rising edge (as described). However, for mode 0 a falling edge is generated $0.5 \cdot t_{CP}$ before chip-select becomes inactive (at transfer end) and for mode 1 and a falling edge is generated $0.5 \cdot t_{CP}$ after chip select becomes active (at transfer start).
4. The min. chip-select idle time can be programmed by bit *t_csh* of register *sqi_sqirom_cfg*. Between 1 and 4 SPI clock cycles can be selected (1). The device will not be deselected if no XiP access is desired. The SPI_CLK line will be held inactive instead. To disable the chip-select for longer times (e.g. for power save) switch to peripheral mode. A device will always be selected at least until the first 4 byte of data have been received in quad-IO-read (i.e. at least for 14 serial clock periods with a running serial clock).
5. Input filtering is not available in SQIROM/XiP mode.
6. The t_{MOSIH} value $0.5 \cdot t_{CP} - 3.0$ ns of earlier versions of this document was too pessimistic and has been corrected in this version.
7. The t_{MISOH} value 1.2 ns of earlier versions of this document was too optimistic and has been corrected in this version. This should have no impact on existing designs as the SPI protocol guarantees a hold time of the half clock period which also exceeds the correct values.

4.4.12 SPI timing

4.4.12.1 Master mode

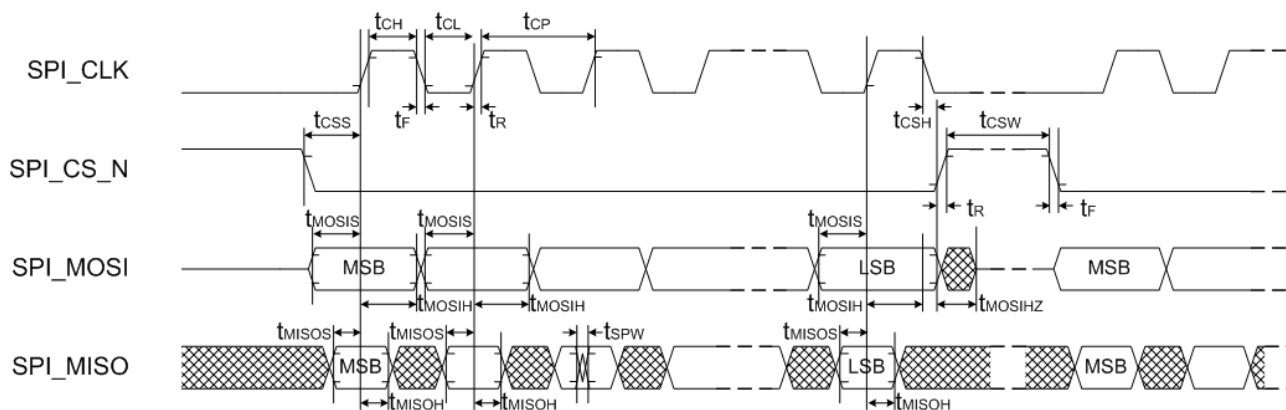


Figure 170: SPI master signal timing (SPO=0 and SPH=0)

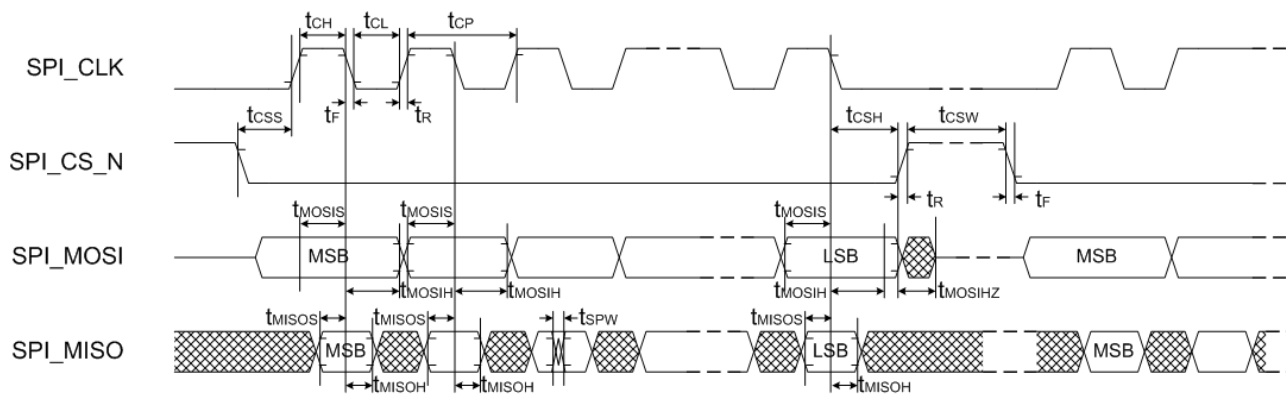


Figure 171: SPI master signal timing (SPO=0 and SPH=1)

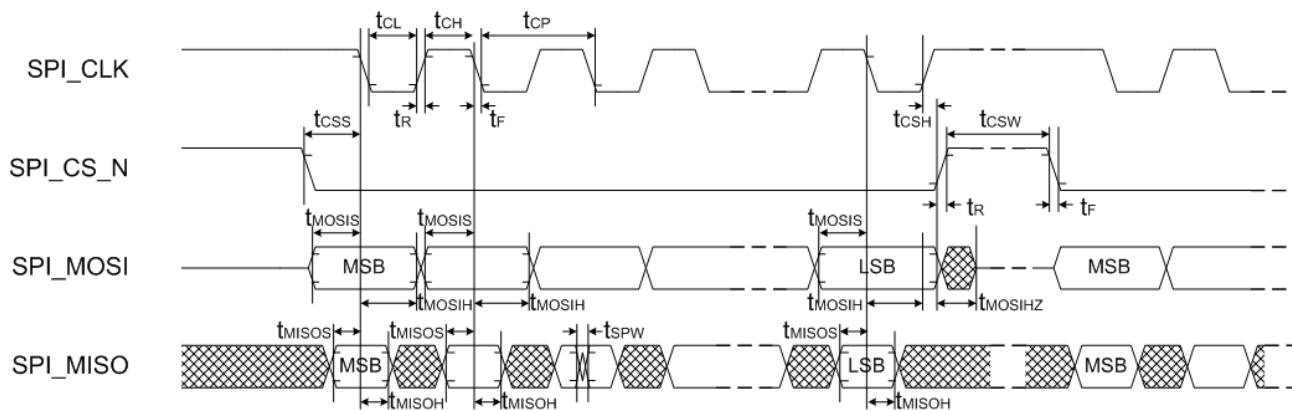


Figure 172: SPI master signal timing (SPO=1 and SPH=0)

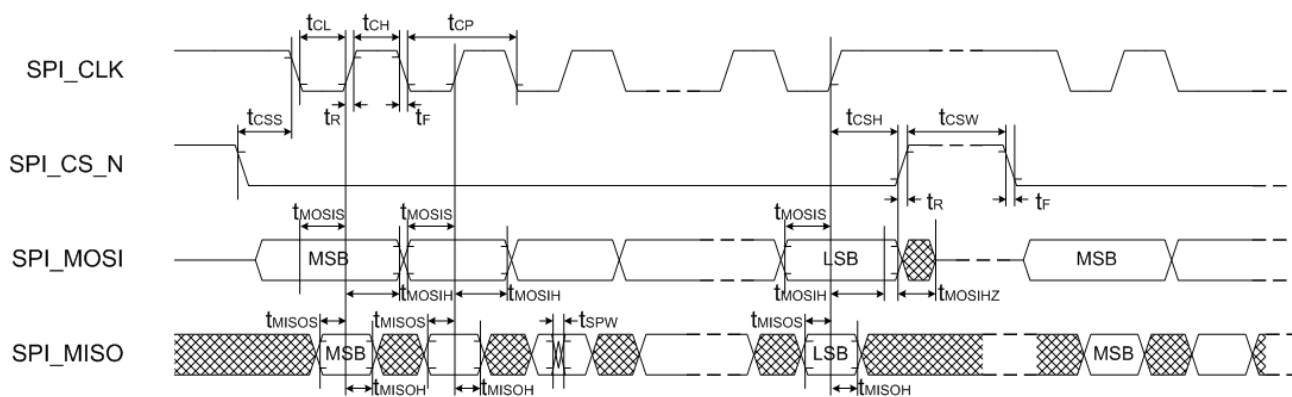


Figure 173: SPI master signal timing (SPO=1 and SPH=1)

SPI master timing for worst case operating conditions: V_{DD} : 3.0..3.6V, T_j : -40..+125°C, C_L : 20 pF. Values in brackets apply if input filtering is enabled.

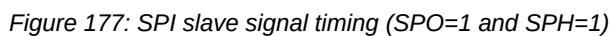
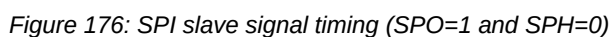
Symbol	Master mode parameter	Min	Typ	Max	Unit
t_{CP}	SPI_CLK period	20.0 ⁽¹⁾	40960/N ⁽¹⁾	40960 ⁽¹⁾	ns
t_{CH}	SPI_CLK high phase	$0.5 \cdot t_{CP} - 3.0$			ns
t_{CL}	SPI_CLK low phase	$0.5 \cdot t_{CP} - 3.0$			ns
t_R	Signal rise time	0.4 ⁽²⁾		3.4 ⁽²⁾	ns
t_F	Signal fall time	0.3 ⁽²⁾		2.0 ⁽²⁾	ns
t_{CSS}	SPI_CS_N to first SPI_CLK edge setup time	$0.5 \cdot t_{CP}$			ns
t_{CSH}	last SPI_CLK edge to SPI_CS_N inactive time - SPH 0 Modes - SPH 1 Modes	10.0 $0.5 \cdot t_{CP} + 10.0$		11.5 $0.5 \cdot t_{CP} + 11.5$	ns
t_{CSW}	SPI_CS_N minimum high pulse width	$0.5 \cdot t_{CP} - 3.0$ ^(3,4)			ns
t_{MOSIS}	SPI_MOSI to SPI_CLK setup time	$0.5 \cdot t_{CP} - 3.4$			ns
t_{MOSIH}	SPI_MOSI hold time	$0.5 \cdot t_{CP}$			ns
t_{MOSIHZ}	SPI_MOSI High-Z time			3.5	ns
t_{MISOS}	SPI_MISO to SPI_CLK setup time	2.5(12.5) ⁽⁵⁾			ns
t_{MISOH}	SPI_MISO hold time	6.5(16.5) ⁽⁵⁾			ns
t_{SPW}	Tolerated spike pulse width - with input filtering - without input filtering			9.0 0.0	ns

Notes

1. N is programmed by register *spi_cr0*, bits *sck_muladd*. $N = 1..2048$
2. Signal rise and fall times differ considerably depending on the external capacitive load. For approximation, use the following formula:
Rise times: $t_r = 0.350 + 0.150 \cdot C_L$ [ns]; C_L : External capacitive load
Fall times: $t_f = 0.237 + 0.087 \cdot C_L$ [ns]; C_L : External capacitive load
3. If bit *fss_static* is set in register *spi_cr1*, SPI_CS_N will not toggle between data words but half clock pause before next word MSB will be inserted anyway.
4. In SPH=1 modes SPI_CS_N does not become inactive during continuous transfers between LSB and next word MSB.
5. Input filtering can be enabled / disabled by bit *filter_in* in register *spi_cr0*.

[illegible][illegible]

© Hilscher, 2012–2017



SPI slave timing for worst case operating conditions: V_{DD} : 3.0..3.6 V, T_j : -40..+125°C, C_L : 20 pF. Values in brackets apply if input filtering is enabled.

Symbol	Slave mode parameter	Min	Typ	Max	Unit
t_{SCP}	SPI_CLK high phase	30.0(60.0) ^(1,4)			ns
t_{SCH}	SPI_CLK high phase	10.0(30.0)			ns
t_{SCL}	SPI_CLK low phase	10.0(30.0)			ns
t_R	Signal rise time			9.0(9.0)	ns
t_F	Signal fall time			9.0(9.0)	ns
t_{SMOSIS}	SPI_MOSI to SPI_CLK setup time	2.0(12.0)			ns
t_{SMOSIH}	SPI_MOSI hold time	12.0(22.0)			ns
t_{SMISOS}	SPI_CLK to SPI_MISO setup time			29.9(39.9) ⁽³⁾	ns
$t_{SMISOS1}$	SPI_CS_N to SPI_MISO MSB setup time			33.9(43.9)	ns
t_{SMISOH}	SPI_MISO hold time	17.3(27.3) ⁽³⁾			ns
$t_{SMISOLZ}$	SPI_CS_N to SPI_MISO MSB Low-Z time	17.5(27.5)			ns
$t_{SMISOHZ}$	SPI_CS_N to SPI_MISO LSB High-Z time	28.5(38.5)			ns
t_{SCSS}	SPI_CS_N to SPI_CLK setup time - SPH=0 modes - SPH=1 modes	$t_{SMISOS1}$ 0.5			ns
t_{SCSDH}	SPI_CS_N hold from deselected SPI_CLK edge time	40.0			ns
t_{SCSH}	SPI_CS_N hold time	10.5			ns
t_{SPW}	Tolerated spike pulse width - with input filtering - without input filtering			9.0 0.0	ns

Notes

1. Min. SPI_CLK period results from max. MISO data setup time (t_{SMISOS}) for valid read data.
2. Using SPH=0 modes SPI chip select falling edge is MSB data out trigger. Hence, SPI chip select must toggle between each transferred word. Using SPH=1 modes SPI chip select may remain active between transferred words.
3. For fast SPI clock rates early SPI_MISO signal generation can be enabled by bit *slave_sig_early* in register *spi_cr0*. SPI_MISO will then be generated one SPI_CLK edge before. For SPH=0 modes t_{SMISOS} and t_{SMISOH} will be related to SPI_CLK edges 1, 3, 5,... instead of 2, 4, 6,... (SPH=1 modes: SPI_CLK edges 2, 4, 6,... instead of 3, 5, 6,...).

4.4.13 I2C

The I2C Module IO timing depends on pad delays and internal module structure. All timings are related to an internal system clock running at 100 MHz.

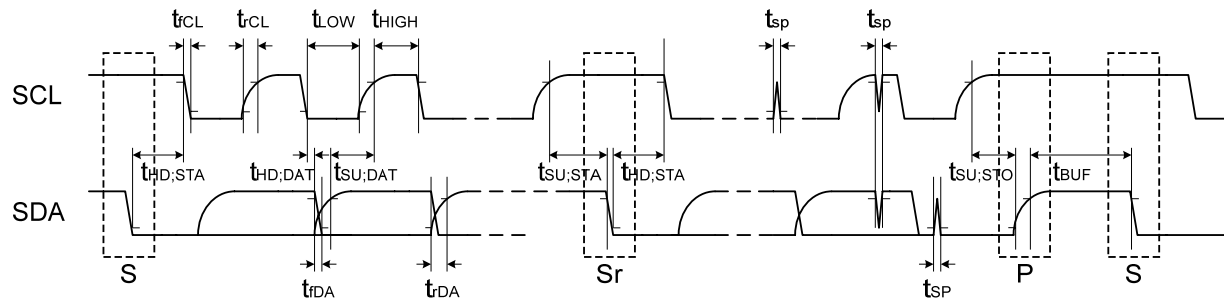


Figure 178: Timing on I2C bus

Symbol	Parameter	Min	Typ	Max	Unit
f_{SCL}	SCL clock frequency	50 ⁽¹⁾	-	3333 ⁽¹⁾	kHz
t_{SCL}	SCL clock period ($1/f_{SCL}$)	300 ⁽¹⁾	-	20000 ⁽¹⁾	ns
t_{fCL}	SCL fall time		3.1 ⁽²⁾		ns
t_{rCL}	SCL rise time	15.9	-	250 ⁽³⁾	ns
t_{fDA}	SDA fall time		3.1 ⁽²⁾		ns
t_{rDA}	SDA rise time	15.9	-	250 ⁽³⁾	ns
t_{LOW}	Low period of SCL clock	$t_{SCL}/2 - t_{fCL}$	-	- ⁽¹⁾	ns
t_{HIGH}	High period of SCL clock	$t_{SCL}/2 - t_{rCL}$	-	$t_{SCL}/2$	ns
$t_{HD;DAT}$	SDA hold time	0.0	-	0.0	ns
$t_{SU;DAT}$	SDA setup time	$t_{SCL}/2 - t_{rDA}$	-	-	ns
$t_{HD;STA}$	SCL hold time after (repeated) START condition	$t_{SCL}/2$	-	-	ns
$t_{SU;STA}$	SCL setup time before repeated START condition	$t_{SCL}/2 - t_{rCL}$	-	-	ns
$t_{SU;STO}$	SCL setup time before STOP condition	$t_{SCL}/2 - t_{rCL}$	-	-	ns
t_{BUF}	I2C bus idle time between STOP and START	$t_{SCL}/2 - t_{rDA}$	-	-	ns
t_{SP}	Pulse width of spikes suppressed by input filters	-	-	$t_{SCL}/32$	ns

Notes

1. SCL clock frequency may decrease (period may increase) if other devices hold SCL low. SCL frequency is set by mode-bits of i2c_mcr register.
2. Signal fall times are only typical. For approximation, use the following formula:
Fall times: $t_f = 0.197 + 0.058 * C_L$ [ns]; C_L : External capacitive load
3. Signal rise times depend on pull-up resistor and capacitive bus load. Specified rise times are for 500 Ohms pull-up resistor and 400 pF load. For I2C high-speed mode and high capacitive load external driver devices can be used to reach faster rise times.

4.4.14 UART

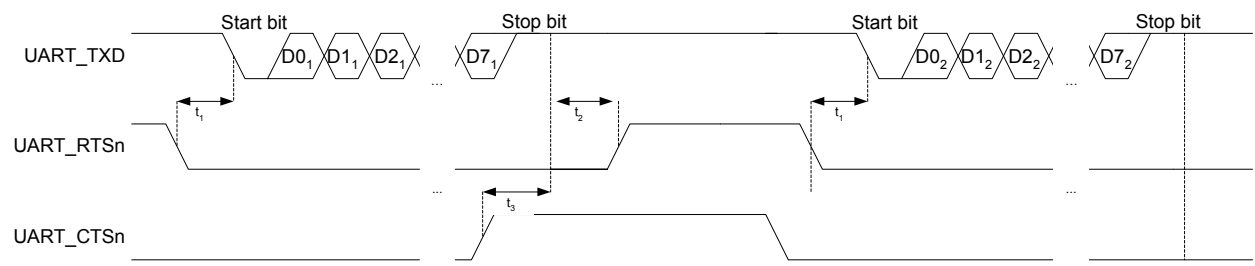


Figure 179: UART

Symbol	Description	Condition	Min	Max	Unit
t ₁	Programmable leading time	Bit Time	0	255	Bit Times
		System Time	0	2.55	µs
t ₂	Programmable trailing time	Bit Time	0	255	Bit Times
		System Time	0	2.55	µs
t ₃	Setup time before the end of stop bit		70		ns

Note: This example uses the following settings:

- 1 start bit
- 1 stop bit
- 8 data bits
- Mode = '1'

4.5 Failure rate (FIT)

Due to the comparatively low volume of netX controller production, failure rates for the complete chip are not available. However, as a guideline value, the known FIT values (as at December 2012) for the CB-12 process, the netX 51 and netX 52 are based on, can be used:

	Failure Rate (FIT), confidence level = 60%		
Junction temperature	Ea = 0.3eV	Ea = 0.5eV (Typ.)	Ea = 0.7eV
55 °C	22	5	1
70 °C	35	11	3
80 °C	47	18	6
90 °C	62	28	12
100 °C	80	42	22
110 °C	101	64	39
125 °C	143	112	86

Table 46: Failure rate (FIT)

5 netX 51 package and signal information

5.1 Pin table sorted by signals

Pin	Signal	Shared with								PAD type
G5	BSCAN_TRST									IDS
D1	CLKOUT					MMIO48				IOD6
C7	HIF_A0	DPM_A0	EXT_A0	DPM_BE0n	EXT_BE0n		SD_A0	MII3_TXER	PIO73	IOU9
B8	HIF_A1	DPM_A1	EXT_A1	DPM_BE2n	EXT_BE2n		SD_A1	MII3_COL	PIO70	IOU9
C8	HIF_A2	DPM_A2	EXT_A2				SD_A2	MII3_CRS	PIO69	IOU9
C10	HIF_A3	DPM_A3	EXT_A3				SD_A3	MII3_RXD0	PIO66	IOU9
A10	HIF_A4	DPM_A4	EXT_A4				SD_A4	MII3_RXD1	PIO65	IOU9
B9	HIF_A5	DPM_A5	EXT_A5				SD_A5	MII3_RXD2	PIO64	IOU9
C11	HIF_A6	DPM_A6	EXT_A6				SD_A6	MII3_RXD3	PIO61	IOU9
D11	HIF_A7	DPM_A7	EXT_A7				SD_A7	MII3_RXDV	PIO60	IOU9
C13	HIF_A8	DPM_A8	EXT_A8				SD_A8	MII3_TXD0	PIO57	IOU9
B12	HIF_A9	DPM_A9	EXT_A9				SD_A9	MII3_TXD1	PIO56	IOU9
C14	HIF_A10	DPM_A10	EXT_A10				SD_A10	MII3_TXD2	PIO53	IOU9
A17	HIF_A11	DPM_A11	EXT_A11				SD_A11	MII3_TXD3	PIO50	IOU9
B15	HIF_A12	DPM_A12	EXT_A12				SD_A12	MII3_TXEN	PIO49	IOU9
A16	HIF_A13	DPM_A13	EXT_A13				SD_DQM0n	MII3_TXCLK	PIO48	IOU9
B14	HIF_A14	DPM_A14	EXT_A14				SD_BA0		PIO54	IOU9
A15	HIF_A15	DPM_A15	EXT_A15	DPM_ALE32_netX 50			SD_BA1		PIO55	IOU9
E14	HIF_A16	DPM_A16				VDDC*	SD_RASn			IOU9/PWR
D14	HIF_A17	DPM_A17				VSS*	SD_CASn			IOU9/GND
A18	HIF_BHE1	DPM_BE1n	EXT_BE1n	DPM_BHEn	EXT_BHEn		SD_DQM1n	MII3_RXER	PIO43	IOU9
C15	HIF_BHE3	DPM_BE3n	EXT_BE3n	DPM_WRHn	EXT_CS3n	EXT_A24	SD_DQM3n	MII2_RXDV	PIO44	IOU9
B16	HIF_CS _n	DPM_CS _n	EXT_CS0n				SD_CS _n		PIO51	IOU9

Table 47: netX 51 – Signals B ... HIF (Pin table sorted by signals)

PAD type explanation, see section 7.1 on page 348.

VSS* = GND netX 50 mode;

VDDC* = 1.5V core power netX 50 mode.

Pin	Signal	Shared with								PAD type
A1	HIF_D0	DPM_D0	EXT_D0				SD_D0		PIO83	IOU9
B2	HIF_D1	DPM_D1	EXT_D1				SD_D1		PIO82	IOU9
C2	HIF_D2	DPM_D2	EXT_D2				SD_D2		PIO81	IOU9
C6	HIF_D3	DPM_D3	EXT_D3				SD_D3		PIO78	IOU9
A6	HIF_D4	DPM_D4	EXT_D4				SD_D4		PIO77	IOU9
A3	HIF_D5	DPM_D5	EXT_D5				SD_D5		PIO76	IOU9
A8	HIF_D6	DPM_D6	EXT_D6				SD_D6		PIO75	IOU9
B7	HIF_D7	DPM_D7	EXT_D7				SD_D7		PIO74	IOU9
J16	HIF_D8	DPM_D8	EXT_D16	DPM_SPI_MISO / SIO1	EXT_D8_netX50	MMIO40	SD_D16	MII2_COL	PIO32	IOU9
H15	HIF_D9	DPM_D9	EXT_D17	DPM_SPI_MOSI / SIO0		MMIO41	SD_D17	MII2_CRS	PIO34	IOU9
H16	HIF_D10	DPM_D10	EXT_D18	DPM_SPI_CS _n	EXT_D10_netX50	MMIO42	SD_D18	MII2_TXER	PIO33	IOU9
G16	HIF_D11	DPM_D11	EXT_D19	DPM_SPI_CLK	EXT_D11_netX50	MMIO43	SD_D19		PIO39	IOU9
G18	HIF_D12	DPM_D12	EXT_D26	DPM_SPI_DIRQ	EXT_D12_netX50	MMIO44	SD_D26		PIO38	IOU9
G15	HIF_D13	DPM_D13	EXT_D27	DPM_SPI_SIRQ	EXT_D13_netX50	MMIO45	SD_D27	MII2_RXD1	PIO37	IOU9
D18	HIF_D14	DPM_D14	EXT_D30	DPM_SPI_SIO2	EXT_D14_netX50	MMIO46	SD_D30	MII2_RXD2	PIO42	IOU9
C18	HIF_D15	DPM_D15	EXT_D31	DPM_SPI_SIO3	EXT_D15_netX50	MMIO47	SD_D31	MII2_RXD3	PIO41	IOU9
J18	HIF_D16	DPM_D16	EXT_D8			IRQ_P36	SD_D8	MII1_TXER	PIO36	IOU9
H17	HIF_D17	DPM_D17	EXT_D9	DPM_ALE16		IRQ_P35	SD_D9		PIO35	IOU9
H18	HIF_D18	DPM_D18	EXT_D10	DPM_SELA19	EXT_A24_netX50	IRQ_P40	SD_D10	MII1_RXDV	PIO40	IOU9
G17	HIF_D19	DPM_D19	EXT_D11				SD_D11		WDGACT	IOU9
B13	HIF_D20	DPM_D20	EXT_D20	DPM_A16	EXT_A16		SD_D20	MII12_TCLK	PIO58	IOU9
A14	HIF_D21	DPM_D21	EXT_D21	DPM_A17	EXT_A17		SD_D21	MII12_TXD0	PIO59	IOU9
A13	HIF_D22	DPM_D22	EXT_D22	DPM_A18	EXT_A18		SD_D22	MII12_TXD1	PIO62	IOU9
C12	HIF_D23	DPM_D23	EXT_D23	DPM_A19	EXT_A19		SD_D23	MII12_TXD2	PIO63	IOU9

Table 48: netX 51 – Signals HIF_D (Pin table sorted by signals)

Pin	Signal	Shared with								PAD type
B10	HIF_D24	DPM_D24	EXT_D24	DPM_SELA12	EXT_A20		SD_D24	MII12_TXD3	PIO67	IOU9
A11	HIF_D25	DPM_D25	EXT_D25	DPM_SELA13	EXT_A21		SD_D25	MII12_TXEN	PIO68	IOU9
A9	HIF_D26	DPM_D26	EXT_D12	DPM_SELA14	EXT_A22_netX50		SD_D12	MII1_RXD0	PIO71	IOU9
C9	HIF_D27	DPM_D27	EXT_D13	DPM_SELA15	EXT_A23_netX50	IRQ_P72	SD_D13	MII1_RXD1	PIO72	IOU9
B6	HIF_D28	DPM_D28	EXT_D28	DPM_SELA16	EXT_CS2n		SD_D28	MII12_RXCLK	PIO79	IOU9
A7	HIF_D29	DPM_D29	EXT_D29	DPM_SELA17	EXT_CS1n		SD_D29	MII12_RXER	PIO80	IOU9
A2	HIF_D30	DPM_D30	EXT_D14	DPM_SELA18	EXT_CS3n_netX50		SD_D14	MII1_RXD2	PIO84	IOU9
J17	HIF_D31	DPM_D31	EXT_D15	DPM_SELA11			SD_D15	MII1_RXD3	PIO85	IOU9
C17	HIF_DIRQ	DPM_DIRQ	EXT_CS1n_D32		EXT_INT	IRQ_P47		MII2_RXD0	PIO47	IOU9
B17	HIF_RDn	DPM_RDn	EXT_RDn	DPM_DATADIR			SD_DQM2n		PIO52	IOU9
B18	HIF_RDY	DPM_RDY	EXT_RDY				SD_CKE	MII3_RXCLK	PIO46	IOU9
B11	HIF_SDCLK	DPM_SIRQ					SD_CLK			IOD9
C16	HIF_WRn	DPM_WRn	EXT_WRn	DPM_WRLn			SD_WEn		PIO45	IOU9
L14	I2C_SCL									IOUS9 (5k pU)
L13	I2C_SDA									IOUS9 (5k pU)
K5	JT_TCLK									IUS
L6	JT_TDI									IUS
L5	JT_TDO									OZ6
K6	JT_TMS									IUS
J5	JT_TRSTn									IDS

Table 49: netX 51 – Signals HIF_D ... JT (Pin table sorted by signals)

Pin	Signal	Shared with								PAD type
G2	MEM_A0									OD6
G4	MEM_A1									OD6
F1	MEM_A2									OD6
F4	MEM_A3									OD6
F2	MEM_A4									OD6
G1	MEM_A5									OD6
H4	MEM_A6									OD6
G3	MEM_A7									OD6
J3	MEM_A8									OD6
K3	MEM_A9									OD6
H2	MEM_A10									OD6
J4	MEM_A11									OD6
H3	MEM_A12									OD6
K4	MEM_A13									OD6
E12	MEM_A14									OD6
D12	MEM_A15									OD6
D13	MEM_A16									OD6
E13	MEM_A17									OD6
P12	MEM_A18	QSPI_SIO2								OD6
R13	MEM_A19	QSPI_SIO3								OD6
K14	MEM_A20									OD6
K13	MEM_A21									OD6
J14	MEM_A22									OD6
J15	MEM_A23									OD6

Table 50: netX 51 – Signals MEM_A (Pin table sorted by signals)

Pin	Signal	Shared with								PAD type
N9	MEM_D0									IOD6
N10	MEM_D1									IOD6
R11	MEM_D2									IOD6
P11	MEM_D3									IOD6
N11	MEM_D4									IOD6
R12	MEM_D5									IOD6
R3	MEM_D6									IOD6
P3	MEM_D7									IOD6
R2	MEM_D8									IOD6
T2	MEM_D9									IOD6
R5	MEM_D10									IOD6
R6	MEM_D11									IOD6
P7	MEM_D12									IOD6
R7	MEM_D13									IOD6
N8	MEM_D14									IOD6
P8	MEM_D15									IOD6
E4	MEM_D16									IOD6
C3	MEM_D17									IOD6
E11	MEM_D18									IOD6
F11	MEM_D19									IOD6
D10	MEM_D20									IOD6
E10	MEM_D21									IOD6
F10	MEM_D22									IOD6
F9	MEM_D23									IOD6

Table 51: netX 51 – Signals MEM_D (Pin table sorted by signals)

Pin	Signal	Shared with								PAD type
D9	MEM_D24									IOD6
E9	MEM_D25									IOD6
E8	MEM_D26									IOD6
D7	MEM_D27									IOD6
D8	MEM_D28									IOD6
F8	MEM_D29									IOD6
B3	MEM_D30									IOD6
D3	MEM_D31									IOD6
P2	MEM_DQM0									OU6
N4	MEM_DQM1									OU6
F3	MEM_DQM2									OU6
E3	MEM_DQM3									OU6
M5	MEMDR_CASn									OU6
L3	MEMDR_CKE									OD6
M4	MEMDR_CLK									IOD6
L4	MEMDR_CSn									OU6
M3	MEMDR_RASn									OU6
N3	MEMDR_WEn									OU6
H14	MEMSR_CS0n									OU6
G14	MEMSR_CS1n									OU6
J12	MEMSR_CS2n									OU6
J13	MEMSR_OEn									OU6
K12	MEMSR_WEn									OU6

Table 52: netX 51 – Signals MEM_D ... MEMSR (Pin table sorted by signals)

Pin	Signal	Shared with								PAD type
H1	MMIO0				FO0_RD_A					IODS6
J1	MMIO1				FO0_TD_A					IODS6
J2	MMIO2				FO0_EN_A					IODS6
L2	MMIO3				FO1_RD_A					IODS6
K2	MMIO4				FO1_TD_A					IODS6
K1	MMIO5				FO1_EN_A					IODS6
L1	MMIO6				FO0_SD_A					IODS6
M2	MMIO7				FO1_SD_A					IODS6
M1	MMIO8									IODS6
N1	MMIO9									IODS6
N2	MMIO10									IODS6
P1	MMIO11									IODS6
R1	MMIO12									IODS6
U1	MMIO13									IODS6
T1	MMIO14									IODS6
V1	MMIO15									IODS6
U2	MMIO16									IODS6

Table 53: netX 51 – Signals MMIO00 ... MMIO16 (Pin table sorted by signals)

Pin	Signal	Shared with								PAD type
V2	MMIO17	ETM_TCLK								IODS6
V3	MMIO18	ETM_TSYNC								IODS6
T6	MMIO19	ETM_DREQ								IODS6
V6	MMIO20	ETM_DACK								IODS6
U6	MMIO21	ETM_PSTAT0								IODS6
T7	MMIO22	ETM_PSTAT1								IODS6
V7	MMIO23	ETM_PSTAT2								IODS6
U7	MMIO24	ETM_TPKT0								IODS6
R8	MMIO25	ETM_TPKT1								IODS6
V9	MMIO26	ETM_TPKT2								IODS6
U9	MMIO27	ETM_TPKT3								IODS6
T9	MMIO28	ETM_TPKT4								IODS6
U10	MMIO29	ETM_TPKT5								IODS6
T10	MMIO30	ETM_TPKT6								IODS6
U11	MMIO31	ETM_TPKT7								IODS6
V12	MMIO32	ETM_TPKT8	FO0_EN_B							IODS6
T11	MMIO33	ETM_TPKT9	FO0_RD_B							IODS6
U12	MMIO34	ETM_TPKT10	FO0_SD_B							IODS6
T12	MMIO35	ETM_TPKT11	FO0_TD_B							IODS6
V13	MMIO36	ETM_TPKT12	FO1_EN_B							IODS6
U13	MMIO37	ETM_TPKT13	FO1_RD_B							IODS6
T13	MMIO38	ETM_TPKT14	FO1_SD_B							IODS6
V14	MMIO39	ETM_TPKT15	FO1_TD_B							IODS6

Table 54: netX 51 – Signals MMIO17 ... MMIO39 (Pin table sorted by signals)

Pin	Signal	Shared with								PAD type
R10	OSC_VDDC									PWR
P9	OSC_VSS									GND
V10	OSC_XTI									XTAL
V11	OSC_XTO									XTAL
L15	PHY_ATP									ANA
K18	PHY_EXTRES									ANA
M15	PHY_VDDCAP									APWR
M16	PHY_VDDIOAC									APWR
R18	PHY_VDDIOAT									APWR
R17	PHY_VSSAT									AGND
M14	PHY_VSSCAP									AGND
P18	PHY0_RXN									PHY
P17	PHY0_RXP									PHY
N18	PHY0_TXN									PHY
N17	PHY0_TXP									PHY
T18	PHY0_VDDCART									APWR
P16	PHY0_VSSAR									AGND
N16	PHY0_VSSAT1									AGND
N15	PHY0_VSSAT2									AGND
L18	PHY1_RXN									PHY
L17	PHY1_RXP									PHY
M18	PHY1_TXN									PHY
M17	PHY1_TXP									PHY
L16	PHY1_VDDCART									APWR
K17	PHY1_VSSAR									AGND
K15	PHY1_VSSAT1									AGND
K16	PHY1_VSSAT2									AGND

Table 55: netX 51 – Signals OSC ... PHY (Pin table sorted by signals)

Pin	Signal	Shared with								PAD type
B1	PORn									IUS
E1	RDY									IOD6
D2	RSTINn									IUS
C1	RSTOUTn									OZ6
E2	RUN									IOD6
V15	SPI0_CLK	QSPI_CLK								IOD6
T14	SPI0_CS1n									IOU6
U15	SPI0_MISO	QSPI_SIO1								IOD6
V16	SPI0_MOSI	QSPI_SIO0								IOD6
U14	SPI0_CS0n	QSPI_CSn								IOU6
J6	TACT_TRST									IDS
A12	TEST									ID
H6	TMC1									I
H5	TMC2									I
V8	USB_DNEG									USB
U8	USB_DPOS									USB
N6	USB_VDDC									PWR
T8	USB_VDDIO									PWR

Table 56: netX 51 – Signals POR ... USB (Pin table sorted by signals)

Pin	Signal	Shared with	PAD type
E6, F6, F14 F15, F16, F17, F18, G7, G8, G10, G11, G12, G13, H7, J7, K7, M6, M8, M10, M11, M12, M13, N5, N7, N14, P5, P6, P15, R9, R16, T17, U18	VDDC	-	PWR
A4, B4, C4, D5, E15, E16, E17, E18, F12, F13, G6, G9, L7, L12, M7, M9, N12, R4, R14, T4, T15, U4, U16, V4	VDDIO	-	PWR
A5, B5, C5, D4, D6, D15, D16, D17, E5, E7, F5, F7, H8, H9, H10, H11, H12, H13, J8, J9, J10, J11, K8, K9, K10, K11, L8, L9, L10, L11, N13, P4, P10, P13, P14, R15, T3, T5, T16, U3, U5, U17, V5, V17, V18	VSS	-	GND

Table 57: netX 51 – Signals VDDC, VDDIO and VSS (Pin table sorted by signals)

5.2 Pin table sorted by pin number

Pins A1 ... A18

Pin	Signal	Shared with								PAD type
A1	HIF_D0	DPM_D0	EXT_D0				SD_D0		PIO83	IOU9
A2	HIF_D30	DPM_D30	EXT_D14	DPM_SELA18	EXT_CS3n_netX50		SD_D14	MII1_RXD2	PIO84	IOU9
A3	HIF_D5	DPM_D5	EXT_D5				SD_D5		PIO76	IOU9
A4	VDDIO									PWR
A5	VSS									GND
A6	HIF_D4	DPM_D4	EXT_D4				SD_D4		PIO77	IOU9
A7	HIF_D29	DPM_D29	EXT_D29	DPM_SELA17	EXT_CS1n		SD_D29	MII12_RXER	PIO80	IOU9
A8	HIF_D6	DPM_D6	EXT_D6				SD_D6		PIO75	IOU9
A9	HIF_D26	DPM_D26	EXT_D12	DPM_SELA14	EXT_A22_netX50		SD_D12	MII1_RXD0	PIO71	IOU9
A10	HIF_A4	DPM_A4	EXT_A4				SD_A4	MII3_RXD1	PIO65	IOU9
A11	HIF_D25	DPM_D25	EXT_D25	DPM_SELA13	EXT_A21		SD_D25	MII12_TXEN	PIO68	IOU9
A12	TEST									ID
A13	HIF_D22	DPM_D22	EXT_D22	DPM_A18	EXT_A18		SD_D22	MII12_TXD1	PIO62	IOU9
A14	HIF_D21	DPM_D21	EXT_D21	DPM_A17	EXT_A17		SD_D21	MII12_TXD0	PIO59	IOU9
A15	HIF_A15	DPM_A15	EXT_A15	DPM_ALE32_netX50			SD_BA1		PIO55	IOU9
A16	HIF_A13	DPM_A13	EXT_A13				SD_DQM0n	MII3_TXCLK	PIO48	IOU9
A17	HIF_A11	DPM_A11	EXT_A11				SD_A11	MII3_TXD3	PIO50	IOU9
A18	HIF_BHE1	DPM_BE1n	EXT_BE1n	DPM_BHEn	EXT_BHEn		SD_DQM1n	MII3_RXER	PIO43	IOU9

Table 58: netX 51 – Pin A1 ... A18 (Pin table sorted by pin number)

PAD Type explanation see section 7.1 on page 348.

Pins B1 ... B18

Pin	Signal	Shared with								PAD type
B1	PORn									IUS
B2	HIF_D1	DPM_D1	EXT_D1				SD_D1		PIO82	IOU9
B3	MEM_D30									IOD6
B4	VDDIO									PWR
B5	VSS									GND
B6	HIF_D28	DPM_D28	EXT_D28	DPM_SELA16	EXT_CS2n		SD_D28	MII12_RXCLK	PIO79	IOU9
B7	HIF_D7	DPM_D7	EXT_D7				SD_D7		PIO74	IOU9
B8	HIF_A1	DPM_A1	EXT_A1	DPM_BE2n	EXT_BE2n		SD_A1	MII3_COL	PIO70	IOU9
B9	HIF_A5	DPM_A5	EXT_A5				SD_A5	MII3_RXD2	PIO64	IOU9
B10	HIF_D24	DPM_D24	EXT_D24	DPM_SELA12	EXT_A20		SD_D24	MII12_TXD3	PIO67	IOU9
B11	HIF_SDCLK	DPM_SIRQ					SD_CLK			IOD9
B12	HIF_A9	DPM_A9	EXT_A9				SD_A9	MII3_TXD1	PIO56	IOU9
B13	HIF_D20	DPM_D20	EXT_D20	DPM_A16	EXT_A16		SD_D20	MII12_TCLK	PIO58	IOU9
B14	HIF_A14	DPM_A14	EXT_A14				SD_BA0		PIO54	IOU9
B15	HIF_A12	DPM_A12	EXT_A12				SD_A12	MII3_TXEN	PIO49	IOU9
B16	HIF_CSn	DPM_CSn	EXT_CS0n				SD_CSn		PIO51	IOU9
B17	HIF_RDn	DPM_RDn	EXT_RDn	DPM_DATADIR			SD_DQM2n		PIO52	IOU9
B18	HIF_RDY	DPM_RDY	EXT_RDY				SD_CKE	MII3_RXCLK	PIO46	IOU9

Table 59: netX 51 – Pin B1 ... B18 (Pin table sorted by pin number)

Pins C1 ... C18

Pin	Signal	Shared with								PAD type
C1	RSTOUTn									OZ6
C2	HIF_D2	DPM_D2	EXT_D2				SD_D2		PIO81	IOU9
C3	MEM_D17									IOD6
C4	VDDIO									PWR
C5	VSS									GND
C6	HIF_D3	DPM_D3	EXT_D3				SD_D3		PIO78	IOU9
C7	HIF_A0	DPM_A0	EXT_A0	DPM_BE0n	EXT_BE0n		SD_A0	MII3_TXER	PIO73	IOU9
C8	HIF_A2	DPM_A2	EXT_A2				SD_A2	MII3_CRS	PIO69	IOU9
C9	HIF_D27	DPM_D27	EXT_D13	DPM_SELA15	EXT_A23_netX50	IRQ_P72	SD_D13	MII1_RXD1	PIO72	IOU9
C10	HIF_A3	DPM_A3	EXT_A3				SD_A3	MII3_RXD0	PIO66	IOU9
C11	HIF_A6	DPM_A6	EXT_A6				SD_A11	MII3_RXD3	PIO61	IOU9
C12	HIF_D23	DPM_D23	EXT_D23	DPM_A19	EXT_A19		SD_D23	MII12_TXD2	PIO63	IOU9
C13	HIF_A8	DPM_A8	EXT_A8				SD_A8	MII3_TXD0	PIO57	IOU9
C14	HIF_A10	DPM_A10	EXT_A10				SD_A10	MII3_TXD2	PIO53	IOU9
C15	HIF_BHE3	DPM_BE3n	EXT_BE3n	DPM_WRHn	EXT_CS3n	EXT_A24	SD_DQM3n	MII2_RXDV	PIO44	IOU9
C16	HIF_WRn	DPM_WRn	EXT_WRn	DPM_WRLn			SD_WEn		PIO45	IOU9
C17	HIF_DIRQ	DPM_DIRQ	EXT_CS1n_D32		EXT_INT	IRQ_P47		MII2_RXD0	PIO47	IOU9
C18	HIF_D15	DPM_D15	EXT_D31	DPM_SPI_SIO3	EXT_D15_netX50	MMIO47	SD_D31	MII2_RXD3	PIO41	IOU9

Table 60: netX 51 – Pin C1 ... C18 (Pin table sorted by pin number)

Pins D1 ... D18

Pin	Signal	Shared with								PAD type
D1	CLKOUT					MMIO48				IOD6
D2	RSTINn									IUS
D3	MEM_D31									IOD6
D4	VSS									GND
D5	VDDIO									PWR
D6	VSS									GND
D7	MEM_D27									IOD6
D8	MEM_D28									IOD6
D9	MEM_D24									IOD6
D10	MEM_D20									IOD6
D11	HIF_A7	DPM_A7	EXT_A7				SD_A7	MII3_RXDV	PIO60	IOU9
D12	MEM_A15									OD6
D13	MEM_A16									OD6
D14	HIF_A17	DPM_A17				VSS*	SD_CASn			IOU9 / GND
D15	VSS									GND
D16	VSS									GND
D17	VSS									GND
D18	HIF_D14	DPM_D14	EXT_D30	DPM_SPI_SIO2	EXT_D14_netX50	MMIO46	SD_D30	MII2_RXD2	PIO42	IOU9

Table 61: netX 51 – Pin D1 ... D18 (Pin table sorted by pin number)

VSS* = GND netX 50 mode

Pins E1 ... E18

Pin	Signal	Shared with								PAD type
E1	RDY									IOD6
E2	RUN									IOD6
E3	MEM_DQM3									OU6
E4	MEM_D16									IOD6
E5	VSS									GND
E6	VDDC									PWR
E7	VSS									GND
E8	MEM_D26									IOD6
E9	MEM_D25									IOD6
E10	MEM_D21									IOD6
E11	MEM_D18									IOD6
E12	MEM_A14									OD6
E13	MEM_A17									OD6
E14	HIF_A16	DPM_A16				VDDC*	SD_RASn			IOU9 / PWR
E15	VDDIO									PWR
E16	VDDIO									PWR
E17	VDDIO									PWR
E18	VDDIO									PWR

Table 62: netX 51 – Pin E1 ... E18 (Pin table sorted by pin number)

VDDC* = 1.5V core power netX 50 mode

Pins F1 ... F18

Pin	Signal	Shared with								PAD type
F1	MEM_A2									OD6
F2	MEM_A4									OD6
F3	MEM_DQM2									OU6
F4	MEM_A3									OD6
F5	VSS									GND
F6	VDDC									PWR
F7	VSS									GND
F8	MEM_D29									IOD6
F9	MEM_D23									IOD6
F10	MEM_D22									IOD6
F11	MEM_D19									IOD6
F12	VDDIO									PWR
F13	VDDIO									PWR
F14	VDDC									PWR
F15	VDDC									PWR
F16	VDDC									PWR
F17	VDDC									PWR
F18	VDDC									PWR

Table 63: netX 51 – Pin F1 ... F18 (Pin table sorted by pin number)

Pins G1 ... G18

Pin	Signal	Shared with								PAD type
G1	MEM_A5									OD6
G2	MEM_A0									OD6
G3	MEM_A7									OD6
G4	MEM_A1									OD6
G5	BSCAN_TRST									IDS
G6	VDDIO									PWR
G7	VDDC									PWR
G8	VDDC									PWR
G9	VDDIO									PWR
G10	VDDC									PWR
G11	VDDC									PWR
G12	VDDC									PWR
G13	VDDC									PWR
G14	MEMSR_CS1n									OU6
G15	HIF_D13	DPM_D13	EXT_D27	DPM_SPI_SIRQ	EXT_D13_netX50	MMIO45	SD_D27	MII2_RXD1	PIO37	IOU9
G16	HIF_D11	DPM_D11	EXT_D19	DPM_SPI_CLK	EXT_D11_netX50	MMIO43	SD_D19		PIO39	IOU9
G17	HIF_D19	DPM_D19	EXT_D11				SD_D11		WDGACT	IOU9
G18	HIF_D12	DPM_D12	EXT_D26	DPM_SPI_DIRQ	EXT_D12_netX50	MMIO44	SD_D26		PIO38	IOU9

Table 64: netX 51 – Pin G1 ... G18 (Pin table sorted by pin number)

Pins H1 ... H18

Pin	Signal	Shared with								PAD type
H1	MMIO0				FO0_RD_A					IODS6
H2	MEM_A10									OD6
H3	MEM_A12									OD6
H4	MEM_A6									OD6
H5	TMC2									I
H6	TMC1									I
H7	VDDC									PWR
H8	VSS									GND
H9	VSS									GND
H10	VSS									GND
H11	VSS									GND
H12	VSS									GND
H13	VSS									GND
H14	MEMSR_CS0n									OU6
H15	HIF_D9	DPM_D9	EXT_D17	DPM_SPI_MOSI / SIO0		MMIO41	SD_D17	MII2_CRS	PIO34	IOU9
H16	HIF_D10	DPM_D10	EXT_D18	DPM_SPI_CSn	EXT_D10_netX50	MMIO42	SD_D18	MII2_TXER	PIO33	IOU9
H17	HIF_D17	DPM_D17	EXT_D9	DPM_ALE16		IRQ_P35	SD_D9		PIO35	IOU9
H18	HIF_D18	DPM_D18	EXT_D10	DPM_SELA19	EXT_A24_netX50	IRQ_P40	SD_D10	MII1_RXDV	PIO40	IOU9

Table 65: netX 51 – Pin H1 ... H18 (Pin table sorted by pin number)

Pins J1 ... J18

Pin	Signal	Shared with								PAD type
J1	MMIO1				FO0_TD_A					IODS6
J2	MMIO2				FO0_EN_A					IODS6
J3	MEM_A8									OD6
J4	MEM_A11									OD6
J5	JT_TRSTn									IDS
J6	TACT_TRST									IDS
J7	VDDC									PWR
J8	VSS									GND
J9	VSS									GND
J10	VSS									GND
J11	VSS									GND
J12	MEMSR_CS2n									OU6
J13	MEMSR_OEn									OU6
J14	MEM_A22									OD6
J15	MEM_A23									OD6
J16	HIF_D8	DPM_D8	EXT_D16	DPM_SPI_MISO / SIO1	EXT_D8_netX50	MMIO40	SD_D16	MII2_COL	PIO32	IOU9
J17	HIF_D31	DPM_D31	EXT_D15	DPM_SELA11			SD_D15	MII1_RXD3	PIO85	IOU9
J18	HIF_D16	DPM_D16	EXT_D8			IRQ_P36	SD_D8	MII1_TXER	PIO36	IOU9

Table 66: netX 51 – Pin J1 ... J18 (Pin table sorted by pin number)

Pins K1 ... K18

Pin	Signal	Shared with								PAD type
K1	MMIO5				FO1_EN_A					IODS6
K2	MMIO4				FO1_TD_A					IODS6
K3	MEM_A9									OD6
K4	MEM_A13									OD6
K5	JT_TCLK									IUS
K6	JT_TMS									IUS
K7	VDDC									PWR
K8	VSS									GND
K9	VSS									GND
K10	VSS									GND
K11	VSS									GND
K12	MEMSR_WEn									OU6
K13	MEM_A21									OD6
K14	MEM_A20									OD6
K15	PHY1_VSSAT1									AGND
K16	PHY1_VSSAT2									AGND
K17	PHY1_VSSAR									AGND
K18	PHY_EXTRES									ANA

Table 67: netX 51 – Pin K1 ... K18 (Pin table sorted by pin number)

Pins L1 ... L18

Pin	Signal	Shared with								PAD type
L1	MMIO6				FO0_SD_A					IODS6
L2	MMIO3				FO1_RD_A					IODS6
L3	MEMDR_CKE									OD6
L4	MEMDR_CS _n									OU6
L5	JT_TDO									OZ6
L6	JT_TDI									IUS
L7	VDDIO									PWR
L8	VSS									GND
L9	VSS									GND
L10	VSS									GND
L11	VSS									GND
L12	VDDIO									PWR
L13	I2C_SDA									IOUS9 (5k pU)
L14	I2C_SCL									IOUS9 (5k pU)
L15	PHY_ATP									ANA
L16	PHY1_VDDCART									APWR
L17	PHY1_RXP									PHY
L18	PHY1_RXN									PHY

Table 68: netX 51 – Pin L1 ... L18 (Pin table sorted by pin number)

Pins M1 ... M18

Pin	Signal	Shared with								PAD type
M1	MMIO8									IODS6
M2	MMIO7				FO1_SD_A					IODS6
M3	MEMDR_RASn									OU6
M4	MEMDR_CLK									IOD6
M5	MEMDR_CASn									OU6
M6	VDDC									PWR
M7	VDDIO									PWR
M8	VDDC									PWR
M9	VDDIO									PWR
M10	VDDC									PWR
M11	VDDC									PWR
M12	VDDC									PWR
M13	VDDC									PWR
M14	PHY_VSSCAP									AGND
M15	PHY_VDDCAP									APWR
M16	PHY_VDDIOAC									APWR
M17	PHY1_TXP									PHY
M18	PHY1_TXN									PHY

Table 69: netX 51 – Pin M1 ... M18 (Pin table sorted by pin number)

Pins N1 ... N18

Pin	Signal	Shared with								PAD type
N1	MMIO9									IODS6
N2	MMIO10									IODS6
N3	MEMDR_WEn									OU6
N4	MEM_DQM1									OU6
N5	VDDC									PWR
N6	USB_VDDC									PWR
N7	VDDC									PWR
N8	MEM_D14									IOD6
N9	MEM_D0									IOD6
N10	MEM_D1									IOD6
N11	MEM_D4									IOD6
N12	VDDIO									PWR
N13	VSS									GND
N14	VDDC									PWR
N15	PHY0_VSSAT2									AGND
N16	PHY0_VSSAT1									AGND
N17	PHY0_TXP									PHY
N18	PHY0_TXN									PHY

Table 70: netX 51 – Pin N1 ... N18 (Pin table sorted by pin number)

Pins P1 ... P18

Pin	Signal	Shared with								PAD type
P1	MMIO11									IODS6
P2	MEM_DQM0									OU6
P3	MEM_D7									IOD6
P4	VSS									GND
P5	VDDC									PWR
P6	VDDC									PWR
P7	MEM_D12									IOD6
P8	MEM_D15									IOD6
P9	OSC_VSS									GND
P10	VSS									GND
P11	MEM_D3									IOD6
P12	MEM_A18	QSPI_SIO2								OD6
P13	VSS									GND
P14	VSS									GND
P15	VDDC									PWR
P16	PHY0_VSSAR									AGND
P17	PHY0_RXP									PHY
P18	PHY0_RXN									PHY

Table 71: netX 51 – Pin P1 ... P18 (Pin table sorted by pin number)

Pins R1 ... R18

Pin	Signal	Shared with								PAD type
R1	MMIO12									IODS6
R2	MEM_D8									IOD6
R3	MEM_D6									IOD6
R4	VDDIO									PWR
R5	MEM_D10									IOD6
R6	MEM_D11									IOD6
R7	MEM_D13									IOD6
R8	MMIO25	ETM_TPKT1								IODS6
R9	VDDC									PWR
R10	OSC_VDDC									PWR
R11	MEM_D2									IOD6
R12	MEM_D5									IOD6
R13	MEM_A19	QSPI_SIO3								OD6
R14	VDDIO									PWR
R15	VSS									GND
R16	VDDC									PWR
R17	PHY_VSSAT									AGND
R18	PHY_VDDIOAT									APWR

Table 72: netX 51 – Pin R1 ... R18 (Pin table sorted by pin number)

Pins T1 ... T18

Pin	Signal	Shared with								PAD type
T1	MMIO14									IODS6
T2	MEM_D9									IOD6
T3	VSS									GND
T4	VDDIO									PWR
T5	VSS									GND
T6	MMIO19	ETM_DREQ								IODS6
T7	MMIO22	ETM_PSTAT1								IODS6
T8	USB_VDDIO									PWR
T9	MMIO28	ETM_TPKT4								IODS6
T10	MMIO30	ETM_TPKT6								IODS6
T11	MMIO33	ETM_TPKT9	FO0_RD_B							IODS6
T12	MMIO35	ETM_TPKT11	FO0_TD_B							IODS6
T13	MMIO38	ETM_TPKT14	FO1_SD_B							IODS6
T14	SPI0_CS1n									IOU6
T15	VDDIO									PWR
T16	VSS									GND
T17	VDDC									PWR
T18	PHY0_VDDCART									APWR

Table 73: netX 51 – Pin T1 ... T18 (Pin table sorted by pin number)

Pins U1 ... U18

Pin	Signal	Shared with								PAD type
U1	MMIO13									IODS6
U2	MMIO16									IODS6
U3	VSS									GND
U4	VDDIO									PWR
U5	VSS									GND
U6	MMIO21	ETM_PSTAT0								IODS6
U7	MMIO24	ETM_TPKT0								IODS6
U8	USB_DPOS									USB
U9	MMIO27	ETM_TPKT3								IODS6
U10	MMIO29	ETM_TPKT5								IODS6
U11	MMIO31	ETM_TPKT7								IODS6
U12	MMIO34	ETM_TPKT10	FO0_SD_B							IODS6
U13	MMIO37	ETM_TPKT13	FO1_RD_B							IODS6
U14	SPI0_CS0n	QSPI_CSn								IOU6
U15	SPI0_MISO	QSPI_SIO1								IOD6
U16	VDDIO									PWR
U17	VSS									GND
U18	VDDC									PWR

Table 74: netX 51 – Pin U1 ... U18 (Pin table sorted by pin number)

Pins V1 ... V18

Pin	Signal	Shared with								PAD type
V1	MMIO15									IODS6
V2	MMIO17	ETM_TCLK								IODS6
V3	MMIO18	ETM_TSYNC								IODS6
V4	VDDIO									PWR
V5	VSS									GND
V6	MMIO20	ETM_DACK								IODS6
V7	MMIO23	ETM_PSTAT2								IODS6
V8	USB_DNEG									USB
V9	MMIO26	ETM_TPKT2								IODS6
V10	OSC_XTI									XTAL
V11	OSC_XTO									XTAL
V12	MMIO32	ETM_TPKT8	FO0_EN_B							IODS6
V13	MMIO36	ETM_TPKT12	FO1_EN_B							IODS6
V14	MMIO39	ETM_TPKT15	FO1_TD_B							IODS6
V15	SPI0_CLK	QSPI_CLK								IOD6
V16	SPI0_MOSI	QSPI_SIO0								IOD6
V17	VSS									GND
V18	VSS									GND

Table 75: netX 51 – Pin V1 ... V18 (Pin table sorted by pin number)

5.3 Pin overview netX 51

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18		
A	HIF_D0	HIF_D30	HIF_D5	VDDIO	VSS	HIF_D4	HIF_D29	HIF_D6	HIF_D26	HIF_A4	HIF_D25	TEST	HIF_D22	HIF_D21	HIF_A15	HIF_A13	HIF_A11	HIF_BHE1	A	
B	PORn	HIF_D1	MEM_D30	VDDIO	VSS	HIF_D28	HIF_D7	HIF_A1	HIF_A5	HIF_D24	HIF_SDCLK	HIF_A9	HIF_D20	HIF_A14	HIF_A12	HIF_CSn	HIF_RDn	HIF_RDY	B	
C	RSTOUTn	HIF_D2	MEM_D17	VDDIO	VSS	HIF_D3	HIF_A0	HIF_A2	HIF_D27	HIF_A3	HIF_A6	HIF_D23	HIF_A8	HIF_A10	HIF_BHE3	HIF_WRn	HIF_DIRQ	HIF_D15	C	
D	CLKOUT	RSTINn	MEM_D31	VSS	VDDIO	VSS	MEM_D27	MEM_D28	MEM_D24	MEM_D20	HIF_A7	MEM_A15	MEM_A16	HIF_A17	VSS	VSS	VSS	HIF_D14	D	
E	RDY	RUN	MEM_DQM3	MEM_D16	VSS	VDDC	VSS	MEM_D26	MEM_D25	MEM_D21	MEM_D18	MEM_A14	MEM_A17	HIF_A16	VDDIO	VDDIO	VDDIO	VDDIO	E	
F	MEM_A2	MEM_A4	MEM_DQM2	MEM_A3	VSS	VDDC	VSS	MEM_D29	MEM_D23	MEM_D22	MEM_D19	VDDIO	VDDIO	VDDC	VDDC	VDDC	VDDC	VDDC	F	
G	MEM_A5	MEM_A0	MEM_A7	MEM_A1	BSCAN_TRST	VDDIO	VDDC	VDDC	VDDIO	VDDC	VDDC	VDDC	VDDC	VDDC	MEMSR_CS1n	HIF_D13	HIF_D11	HIF_D19	HIF_D12	G
H	MMIO0	MEM_A10	MEM_A12	MEM_A6	TMC2	TMC1	VDDC	VSS	VSS	VSS	VSS	VSS	VSS	MEMSR_CS0n	HIF_D9	HIF_D10	HIF_D17	HIF_D18	H	
J	MMIO1	MMIO2	MEM_A8	MEM_A11	JT_TRSTn	TACT_TRST	VDDC	VSS	VSS	VSS	VSS	MEMSR_CS2n	MEMSR_OEn	MEM_A22	MEM_A23	HIF_D8	HIF_D31	HIF_D16	J	
K	MMIO5	MMIO4	MEM_A9	MEM_A13	JT_TCLK	JT_TMS	VDDC	VSS	VSS	VSS	VSS	MEMSR_WEn	MEM_A21	MEM_A20	PHY1_VSSAT1	PHY1_VSSAT2	PHY1_VSSAR	PHY_EXTRES	K	
L	MMIO6	MMIO3	MEMDR_CKE	MEMDR_CSn	JT_TDO	JT_TDI	VDDIO	VSS	VSS	VSS	VSS	VDDIO	I2C_SDA	I2C_SCL	PHY_ATP	PHY1_VDDCART	PHY1_RXP	PHY1_RXN	L	
M	MMIO8	MMIO7	MEMDR_RASn	MEMDR_CLK	MEMDR_CASn	VDDC	VDDIO	VDDC	VDDIO	VDDC	VDDC	VDDC	VDDC	VDDC	PHY_VSSCAP	PHY_VDDCAP	PHY_VDDIOAC	PHY1_TXP	PHY1_TXN	M
N	MMIO9	MMIO10	MEMDR_WEn	MEM_DQM1	VDDC	USB_VDDC	VDDC	MEM_D14	MEM_D0	MEM_D1	MEM_D4	VDDIO	VSS	VDDC	PHY0_VSSAT2	PHY0_VSSAT1	PHY0_TXP	PHY0_TXN	N	
P	MMIO11	MEM_DQM0	MEM_D7	VSS	VDDC	VDDC	MEM_D12	MEM_D15	OSC_VSS	VSS	MEM_D3	MEM_A18	VSS	VSS	VDDC	PHY0_VSSAR	PHY0_RXP	PHY0_RXN	P	
R	MMIO12	MEM_D8	MEM_D6	VDDIO	MEM_D10	MEM_D11	MEM_D13	MMIO25	VDDC	OSC_VDDC	MEM_D2	MEM_D5	MEM_A19	VDDIO	VSS	VDDC	PHY_VSSAT	PHY_VDDIOAT	R	
T	MMIO14	MEM_D9	VSS	VDDIO	VSS	MMIO19	MMIO22	USB_VDDIO	MMIO28	MMIO30	MMIO33	MMIO35	MMIO38	SPI0_CS1n	VDDIO	VSS	VDDC	PHY0_VDDCART	T	
U	MMIO13	MMIO16	VSS	VDDIO	VSS	MMIO21	MMIO24	USB_DPOS	MMIO27	MMIO29	MMIO31	MMIO34	MMIO37	SPI0_CS0n	SPI0_MISO	VDDIO	VSS	VDDC	U	
V	MMIO15	MMIO17	MMIO18	VDDIO	VSS	MMIO20	MMIO23	USB_DNEG	MMIO26	OSC_XTI	OSC_XTO	MMIO32	MMIO36	MMIO39	SPI0_CLK	SPI0_MOSI	VSS	VSS	V	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18		

5.4 Multiplex matrix signals netX 51

Function	Direction	Description	Power-n MMIO mapping	Hilscher default MMIO mapping
GPIOs / IO-LINK				
GPIO00-GPIO31	IO	General Purpose IOs or IOLINK signal blocks	MMIO00-31	MMIO00-31
I²C				
I2C0_SCL	IO	I ² C Interface Clock Signal	-	-
I2C0_SDA	IO	I ² C Interface Data Signal	-	-
I2C1_SCL	IO	I ² C Interface Clock Signal	-	-
I2C1_SDA	IO	I ² C Interface Data Signal	-	-
MII (Media Independent Interface (Ethernet))				
MII_MDC	O	-	-	-
MII_MDIO	IO	-	-	-
MII0_COL	I	-	-	-
MII0_CRS	I	-	-	-
MII0_LED0	I	-	-	-
MII0_LED1	I	-	-	-
MII0_LED2	I	-	-	-
MII0_LED3	I	-	-	-
MII0_RXCLK	I	-	-	-
MII0_RXD0	I	-	-	-
MII0_RXD1	I	-	-	-
MII0_RXD2	I	-	-	-
MII0_RXD3	I	-	-	-
MII0_RXDV	I	-	-	-
MII0_RXER	I	-	-	-
MII0_TXCLK	I	-	-	-
MII0_TXD0	OZ	-	-	-
MII0_TXD1	OZ	-	-	-
MII0_TXD2	OZ	-	-	-
MII0_TXD3	OZ	-	-	-
MII0_TXEN	O	-	-	-
MII0_TXER	O	-	-	-
MII1_COL	I	-	-	-
MII1_CRS	I	-	-	-
MII1_LED0	I	-	-	-
MII1_LED1	I	-	-	-
MII1_LED2	I	-	-	-
MII1_LED3	I	-	-	-
MII1_RXCLK	I	-	-	-
MII1_RXD0	I	-	-	-
MII1_RXD1	I	-	-	-
MII1_RXD2	I	-	-	-
MII1_RXD3	I	-	-	-

Function	Direction	Description	Power-n MMIO mapping	Hilscher default MMIO mapping
MII1_RXDV	I	-	-	-
MII1_RXER	I	-	-	-
MII1_TXCLK	I	-	-	-
MII1_TXD0	OZ	-	-	-
MII1_TXD1	OZ	-	-	-
MII1_TXD2	OZ	-	-	-
MII1_TXD3	OZ	-	-	-
MII1_TXEN	O	-	-	-
MII1_TXER	O	-	-	-
PIOs				
PIO0 - 3	IO	Programmable IOs 0 - 3	-	MMIO28 - 31
PIO4 - 7	IO	Programmable IOs 4 - 7	-	
PHY LEDs				
PHY0_LED0 - 1	O	Status LEDs 0 - 1 of internal PHY0	-	MMIO12 - 13
PHY0_LED2 - 3	O	Status LEDs 2 - 3 of internal PHY0	-	-
PHY1_LED0 - 1	O	Status LEDs 0 - 1 of internal PHY1	-	MMIO14 - 15
PHY1_LED2 - 3	O	Status LEDs 2 - 3 of internal PHY1	-	-
SPI				
SPI0_CS2n	IO	Chip Select 2 signal of SPI0 interface	-	-
SPI0_SIO2	IO	Serial IO Data 2	-	-
SPI0_SIO3	IO	Serial IO Data 2	-	-
SPI1_CLK	IO	Clock signal of SPI1 interface	-	-
SPI1_CS0n	IO	Chip Select 0 signal of SPI1 interface	-	-
SPI1_CS1n	IO	Chip Select 1 signal of SPI1 interface	-	-
SPI1_CS2n	IO	Chip Select 2 signal of SPI1 interface	-	-
SPI1_MISO	IO	Master In / Slave Out signal of SPI1 interface	-	-
SPI1_MOSI	IO	Master Out / Slave In signal of SPI1 interface	-	-
UARTs				
UART0_CTSn	I	UART0 Clear To Send	MMIO32	MMIO32
UART0_RTSn	OZ	UART0 Request To Send	MMIO33	MMIO33
UART0_RXD	I	UART0 Receive Data	MMIO34	MMIO34
UART0_TXD	OZ	UART0 Transmit Data	MMIO35	MMIO35
UART1_CTSn	I	UART1 Clear To Send	MMIO36	MMIO18
UART1_RTSn	OZ	UART1 Request To Send	MMIO37	MMIO19
UART1_RXD	I	UART1 Receive Data	MMIO38	MMIO16
UART1_TXD	OZ	UART1 Transmit Data	MMIO39	MMIO17
UART2_CTSn	I	UART2 Clear To Send	-	MMIO22
UART2_RTSn	OZ	UART2 Request To Send	-	MMIO23
UART2_RXD	I	UART2 Receive Data	-	MMIO20
UART2_TXD	OZ	UART2 Transmit Data	-	MMIO21
XMAC0 (Fieldbus 0)				
XM0_IO0	IO	XMAC0 Programmable IO 0	-	MMIO03
XM0_IO1	IO	XMAC0 Programmable IO 1	-	MMIO02
XM0_IO2 - 5	IO	XMAC0 Programmable IO 2 - 5	-	-

Function	Direction	Description	Power-n MMIO mapping	Hilscher default MMIO mapping
XM0_RX	I	XMAC0 Receive Data	-	MMIO00
XM0_TX_OE	O	XMAC0 Transmit Enable	-	-
XM0_TX_OUT	OZ	XMAC0 Transmit Data	-	MMIO01
XMAC1 (Fieldbus 1)				
XM1_IO0	IO	XMAC1 Programmable IO 0	-	MMIO07
XM1_IO1	IO	XMAC1 Programmable IO 1	-	MMIO05
XM1_IO2 - 5	IO	XMAC1 Programmable IO 2 - 5	-	-
XM1_RX	I	XMAC1 Receive Data	-	MMIO06
XM1_TX_OE	O	XMAC1 Transmit Enable	-	-
XM1_TX_OUT	OZ	XMAC1 Transmit Data	-	MMIO04
CAN				
CAN_RX	I	CAN Receive Data	-	-
CAN_TXD	O	CAN Transmit Data	-	-
RTE Sync Signals				
XC_SAMPLE0	I	-	-	MMIO08
XC_SAMPLE1	I	-	-	MMIO09
XC_TRIGGER0	OZ	-	-	MMIO10
XC_TRIGGER1	OZ	-	-	MMIO11
MEM				
MEM_RDY	I	MEM IF ready/busy input	-	-

Table 76: Multiplex matrix signals (netX 51)

5.5 netX 51 package

The netX 51 comes in a 324-pin PBGA package.

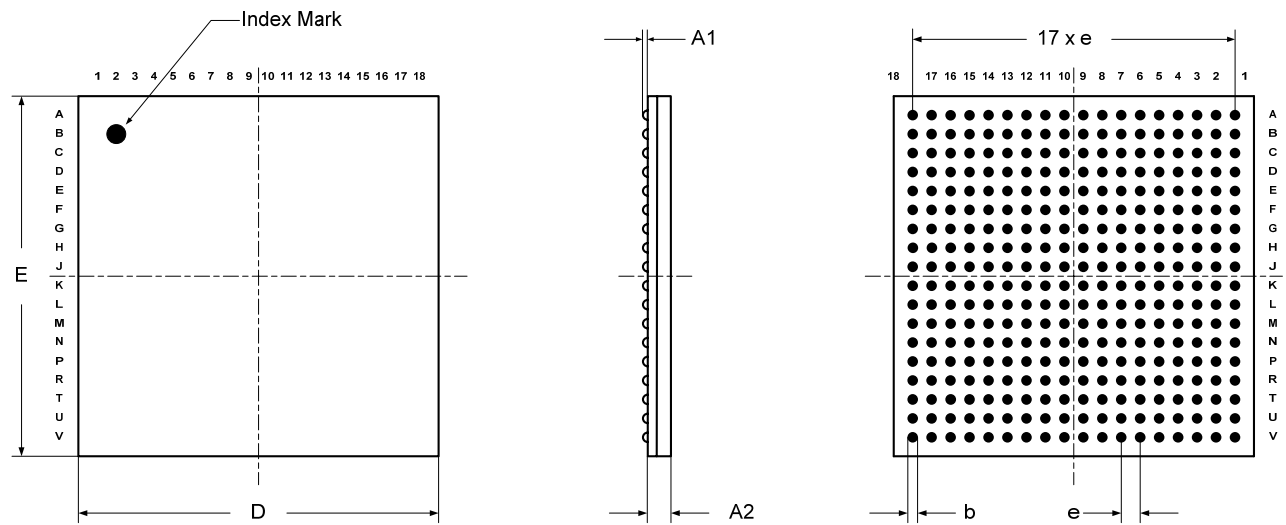


Figure 180: Mechanical dimensions of netX 51

Symbol	Min.	Typ.	Max.
A1	0.40 mm	0.50 mm	0.60 mm
A2		1.33 mm	
b	0.50 mm	0.60 mm	0.70 mm
E	18.90 mm	19.00 mm	19.10 mm
e		1.00 mm	
D	18.90 mm	19.00 mm	19.10 mm

5.6 Ordering information

Order number	Item	Number of pieces
2231.001	NETX 51 STEP B	1 piece
2231.100	NETX 51 B (BOX)	15 pcs
2231.200	NETX 51 B (TRAY)	70 pcs
2231.560	NETX 51 B (PACKAGE560)	560 pcs

5.7 Thermal package specification

Absolute max. junction temperature: $T_{j\_max} = 125^{\circ}\text{C}$

Absolute min. junction temperature: $T_{j\_min} = -40^{\circ}\text{C}$

Recommended operating ambient temperature:

Operating ambient temperature: $-40^{\circ}\text{C} \dots +85^{\circ}\text{C}$ @ typ. 1.5 Wt (with heat sink $R_{th} \leq 6.5 \text{ K/W}$, ICK S 18x18x6.5 SA, Fischer Elektronik)

Operating ambient temperature: $-40^{\circ}\text{C} \dots +70^{\circ}\text{C}$ @ typ. 1.5 W (without heat sink)

Thermal characterization (based on JEDEC-2S2P PCB)

Symbol	Parameter	Air flow [m/s]				Unit
		0	0.2	1	2	
θ_{ja}	Thermal resistance, junction-to-ambient	15.9	14.3	12.5	11.7	$^{\circ}\text{C/W}$
ψ_{jt}	Thermal parameter, junction to the top center of the package surface	0.08	0.11	0.17	0.20	$^{\circ}\text{C/W}$
ψ_{ta}	Thermal parameter, the top center of the package surface to ambient	15.8	14.2	12.3	11.5	$^{\circ}\text{C/W}$
θ_{jc}	Thermal resistance, junction-to-case	4.5				$^{\circ}\text{C/W}$

Case 1 - Without external heat sink	Case 2 - With external heat sink
Heat flow path exists not only on package top	Package surface dominates heat flow path via heat sink
$T_t = T_a + \psi_{ta} \times P_{netX}$	$T_j = T_a + (\theta_{jc} + R_{th}) \times P_{netX}$
$T_j \approx T_t$	

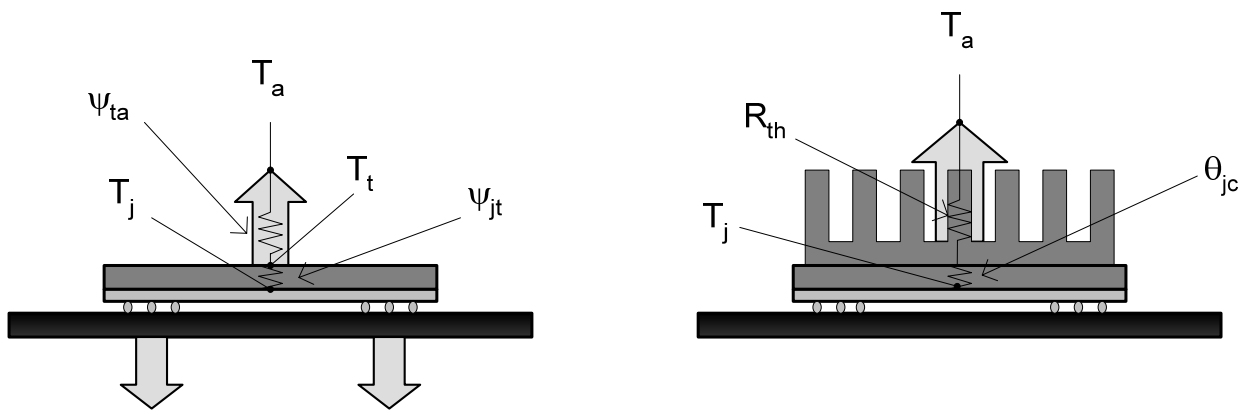


Figure 181: Symbols of thermal characterization

5.8 Soldering conditions

The following soldering parameters are recommended for infrared reflow soldering. The netX package is suitable for a Pb-free soldering process.

5.8.1 Infrared reflow soldering characterization

Symbol	Parameter	Value
T_{PSTmax}	Max. temperature, surface temperature of package	260 °C
t_1	Preheating time at 160 to 180 °C	60 ... 120 s
t_2	Max. time of temperature higher than 220 °C	≤ 60 s
t_3	Max. time at max. temperature	≤ 10 s
	Max. chlorine content of rosin flux	0.2 %

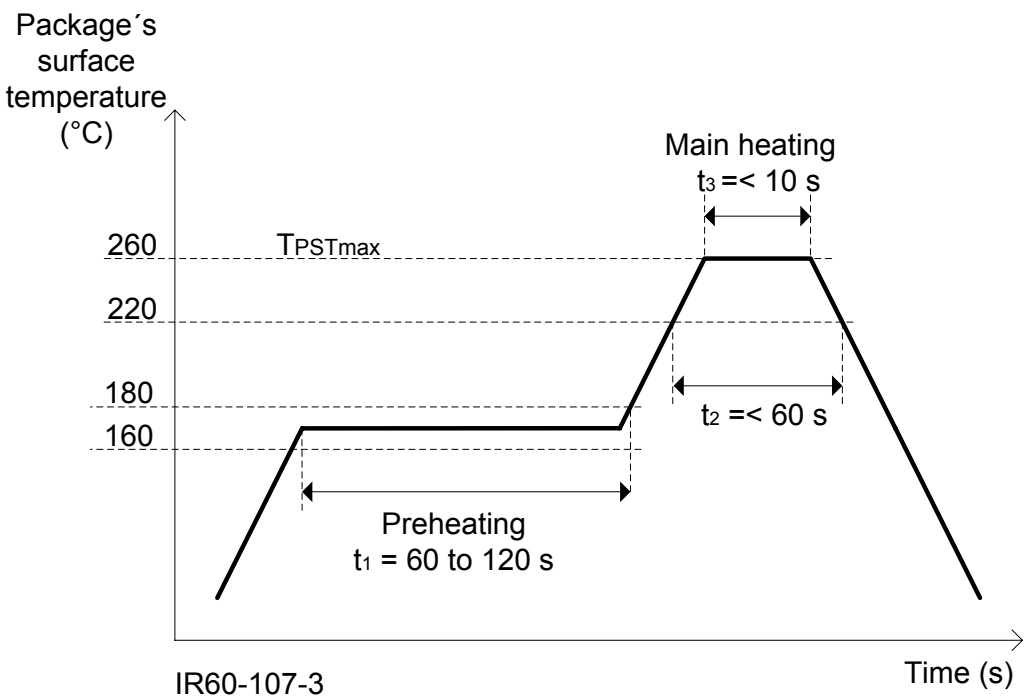


Figure 182: Infrared reflow soldering (surface temperature)

5.8.2 Vapor phase reflow soldering (VPS) characterization

Symbol	Parameter	Value
T _{PSTmax}	Max. temperature, surface temperature of package	215 °C
t ₁	Preheating time	30 s – 60 s
t ₂	Time of temperature > 200 °C	25 s – 40 s
	Max. chlorine content of rosin flux	0.2 %

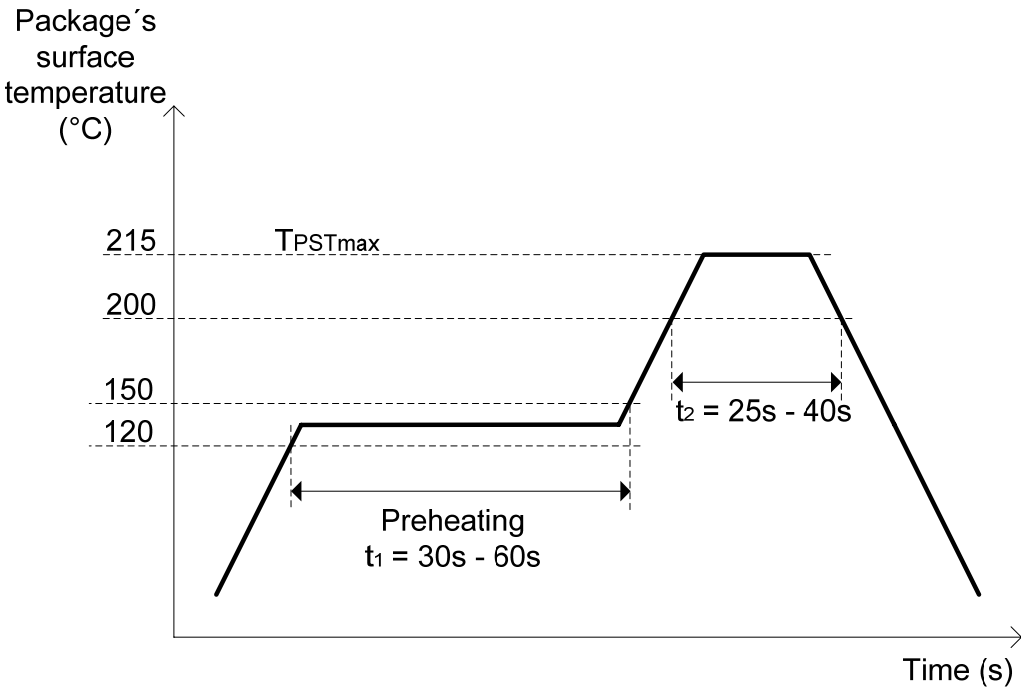


Figure 183: Vapor phase reflow soldering (surface temperature)

5.9 General storage conditions

Parameter	Conditions	Min.	Max.	Unit	Note
Storage temperature	Sealed drypack	5	30	°C	Note 1)
Storage humidity	Sealed drypack	20	70	%RH	Note 1)
Storage time	Sealed drypack		2	years	Note 2)
Storage temperature	Open drypack		< 25	°C	Note 3)
Storage humidity	Open drypack		< 65	%RH	Note 3)
Storage time after opening dry pack	Open drypack		7	days	Note 3)
Baking time	125°C	20	72	hours	
Number of times of mounting			3	times	

Notes

1. Storing the sealed drypacks under other conditions may affect the solderability of the components.
2. When storing sealed drypacks for more than two years, we recommend checking the solder balls for oxidation and performing tests to approve solderability before using the components for production. The 2-year period begins on the seal date printed on the drypack label.
3. Open drypacks/unpacked components may be stored under these conditions for 7 days before soldering. If one of the parameters, namely temperature, humidity or time after opening a sealed drypack is exceeded, the components must be tempered according to parameter "Baking time" before soldering.

5.10 Moisture sensitivity level

The Moisture Sensitivity Level (MSL) is MSL 3, 168 h, 30 °C / 60%RH.

6 netX 52 package and signal information

6.1 Pin table sorted by signals

Pin	Signal	Shared with								PAD type
E3	BSCAN_TRST									IDS
A3	HIF_A0	DPM_A0	EXT_A0	DPM_BE0n	EXT_BE0n		SD_A0	MII3_TXER	PIO73	IOU9
B6	HIF_A1	DPM_A1	EXT_A1	DPM_BE2n	EXT_BE2n		SD_A1	MII3_COL	PIO70	IOU9
C6	HIF_A2	DPM_A2	EXT_A2				SD_A2	MII3_CRS	PIO69	IOU9
C7	HIF_A3	DPM_A3	EXT_A3				SD_A3	MII3_RXD0	PIO66	IOU9
B8	HIF_A4	DPM_A4	EXT_A4				SD_A4	MII3_RXD1	PIO65	IOU9
A7	HIF_A5	DPM_A5	EXT_A5				SD_A5	MII3_RXD2	PIO64	IOU9
B9	HIF_A6	DPM_A6	EXT_A6				SD_A6	MII3_RXD3	PIO61	IOU9
C9	HIF_A7	DPM_A7	EXT_A7				SD_A7	MII3_RXDV	PIO60	IOU9
C13	HIF_A8	DPM_A8	EXT_A8				SD_A8	MII3_TXD0	PIO57	IOU9
A10	HIF_A9	DPM_A9	EXT_A9				SD_A9	MII3_TXD1	PIO56	IOU9
B12	HIF_A10	DPM_A10	EXT_A10				SD_A10	MII3_TXD2	PIO53	IOU9
A14	HIF_A11	DPM_A11	EXT_A11				SD_A11	MII3_TXD3	PIO50	IOU9
B13	HIF_A12	DPM_A12	EXT_A12				SD_A12	MII3_TXEN	PIO49	IOU9
A13	HIF_A13	DPM_A13	EXT_A13				SD_DQM0n	MII3_TXCLK	PIO48	IOU9
A12	HIF_A14	DPM_A14	EXT_A14				SD_BA0		PIO54	IOU9
B11	HIF_A15	DPM_A15	EXT_A15	DPM_ALE32_netX50			SD_BA1		PIO55	IOU9
A17	HIF_A16	DPM_A16	EXT_A16		EXT_A22		SD_RASn	MII1_COL		IOU9
A18	HIF_A17	DPM_A17	EXT_A17	DPM_ALE32	EXT_A23		SD_CASn	MII1_CRS		IOU9
A16	HIF_BHE1	DPM_BE1n	EXT_BE1n	DPM_BHEn	EXT_BHEn	DPM_CS2	SD_DQM1n	MII3_RXER	PIO43	IOU9
B16	HIF_BHE3	DPM_BE3n	EXT_BE3n	DPM_WRHn	EXT_CS3n	EXT_A24	SD_DQM3n	MII2_RXDV	PIO44	IOU9
A15	HIF_CSn	DPM_CSn	EXT_CS0n				SD_CSn		PIO51	IOU9

Table 77: netX 52 – Signals B ... HIF (Pin table sorted by signals)

PAD type explanation, see section 7.1 on page 348.

Pin	Signal	Shared with								PAD type
A1	HIF_D0	DPM_D0	EXT_D0				SD_D0		PIO83	IOU9
C2	HIF_D1	DPM_D1	EXT_D1				SD_D1		PIO82	IOU9
D2	HIF_D2	DPM_D2	EXT_D2				SD_D2		PIO81	IOU9
B3	HIF_D3	DPM_D3	EXT_D3				SD_D3		PIO78	IOU9
B4	HIF_D4	DPM_D4	EXT_D4				SD_D4		PIO77	IOU9
B2	HIF_D5	DPM_D5	EXT_D5				SD_D5		PIO76	IOU9
A5	HIF_D6	DPM_D6	EXT_D6				SD_D6		PIO75	IOU9
B5	HIF_D7	DPM_D7	EXT_D7				SD_D7		PIO74	IOU9
G18	HIF_D8	DPM_D8	EXT_D16	DPM_SPI_MISO / SIO1	EXT_D8_netX50	MMIO40	SD_D16	MII2_COL	PIO32	IOU9
G16	HIF_D9	DPM_D9	EXT_D17	DPM_SPI_MOSI / SIO0	EXT_D9_netX50	MMIO41	SD_D17	MII2_CRS	PIO34	IOU9
F17	HIF_D10	DPM_D10	EXT_D18	DPM_SPI_CS _n	EXT_D10_netX50	MMIO42	SD_D18	MII2_TXER	PIO33	IOU9
D18	HIF_D11	DPM_D11	EXT_D19	DPM_SPI_CLK	EXT_D11_netX50	MMIO43	SD_D19		PIO39	IOU9
C18	HIF_D12	DPM_D12	EXT_D26	DPM_SPI_DIRQ	EXT_D12_netX50	MMIO44	SD_D26		PIO38	IOU9
B18	HIF_D13	DPM_D13	EXT_D27	DPM_SPI_SIRQ	EXT_D13_netX50	MMIO45	SD_D27	MII2_RXD1	PIO37	IOU9
C17	HIF_D14	DPM_D14	EXT_D30	DPM_SPI_SIO2	EXT_D14_netX50	MMIO46	SD_D30	MII2_RXD2	PIO42	IOU9
B17	HIF_D15	DPM_D15	EXT_D31	DPM_SPI_SIO3	EXT_D15_netX50	MMIO47	SD_D31	MII2_RXD3	PIO41	IOU9
G17	HIF_D16	DPM_D16	EXT_D8			IRQ_P36	SD_D8	MII1_TXER	PIO36	IOU9
F18	HIF_D17	DPM_D17	EXT_D9	DPM_ALE16		IRQ_P35	SD_D9		PIO35	IOU9
E18	HIF_D18	DPM_D18	EXT_D10	DPM_SEL19	EXT_A24_netX50		SD_D10	MII1_RXDV	PIO40	IOU9
E17	HIF_D19	DPM_D19	EXT_D11				SD_D11		WDGACT	IOU9
C12	HIF_D20	DPM_D20	EXT_D20	DPM_A16	EXT_A16		SD_D20	MII12_TCLK	PIO58	IOU9
C11	HIF_D21	DPM_D21	EXT_D21	DPM_A17	EXT_A17		SD_D21	MII12_TXD0	PIO59	IOU9
A11	HIF_D22	DPM_D22	EXT_D22	DPM_A18	EXT_A18		SD_D22	MII12_TXD1	PIO62	IOU9
C10	HIF_D23	DPM_D23	EXT_D23	DPM_A19	EXT_A19		SD_D23	MII12_TXD2	PIO63	IOU9

Table 78: netX 52 – Signals HIF_D (Pin table sorted by signals)

Pin	Signal	Shared with								PAD type
A9	HIF_D24	DPM_D24	EXT_D24	DPM_SEL12	EXT_A20		SD_D24	MII12_TXD3	PIO67	IOU9
A8	HIF_D25	DPM_D25	EXT_D25	DPM_SEL13	EXT_A21		SD_D25	MII12_TXEN	PIO68	IOU9
A6	HIF_D26	DPM_D26	EXT_D12	DPM_SEL14	EXT_A22_netX50		SD_D12	MII1_RXD0	PIO71	IOU9
B7	HIF_D27	DPM_D27	EXT_D13	DPM_SEL15	EXT_A23_netX50	IRQ_P72	SD_D13	MII1_RXD1	PIO72	IOU9
A2	HIF_D28	DPM_D28	EXT_D28	DPM_SEL16	EXT_CS2n		SD_D28	MII12_RXCLK	PIO79	IOU9
A4	HIF_D29	DPM_D29	EXT_D29	DPM_SEL17	EXT_CS1n		SD_D29	MII12_RXER	PIO80	IOU9
D3	HIF_D30	DPM_D30	EXT_D14	DPM_SEL18	EXT_CS3n_netX50		SD_D14	MII1_RXD2	PIO84	IOU9
H17	HIF_D31	DPM_D31	EXT_D15	DPM_SEL11			SD_D15	MII1_RXD3	PIO85	IOU9
D16	HIF_DIRQ	DPM_DIRQ	EXT_CS1n_D32		EXT_INT			MII2_RXD0	PIO47	IOU9
B15	HIF_RDn	DPM_RDn	EXT_RDn	DPM_DATADIR		IRQ_P47	SD_DQM2		PIO52	IOU9
C15	HIF_RDY	DPM_RDY	EXT_RDY				SD_CKE	MII3_RXCLK	PIO46	IOU9
C8	HIF_SDCLK	DPM_SIRQ					SD_CLK			IOD9
B14	HIF_WRn	DPM_WRn	EXT_WRn	DPM_WRLn			SD_WEn		PIO45	IOU9
H18	I2C_SCL									IOUS9 (5k pU)
H16	I2C_SDA									IOUS9 (5k pU)
H1	JT_TCLK									IUS
J3	JT_TDI									IUS
M3	JT_TDO									OZ6
K3	JT_TMS									IUS
H3	JT_TRSTn									IDS

Table 79: netX 52 – Signals HIF_D ... JT (Pin table sorted by signals)

Pin	Signal	Shared with								PAD type
H2	MMIO0				FO0_RD					IODS6
K2	MMIO1				FO0_TD					IODS6
J2	MMIO2				FO0_EN					IODS6
L3	MMIO3				FO1_RD					IODS6
L1	MMIO4				FO1_TD					IODS6
M1	MMIO5				FO1_EN					IODS6
M2	MMIO6				FO0_SD					IODS6
N1	MMIO7				FO1_SD					IODS6
N2	MMIO8									IODS6
P2	MMIO9									IODS6
R1	MMIO10									IODS6
T1	MMIO11									IODS6
R2	MMIO12									IODS6
U2	MMIO13									IODS6
T2	MMIO14									IODS6
R3	MMIO15									IODS6
T4	MMIO16									IODS6
V1	MMIO17									IODS6
U4	MMIO18									IODS6
U3	MMIO19									IODS6
T5	MMIO20									IODS6
V3	MMIO21									IODS6
T6	MMIO22									IODS6
U5	MMIO23									IODS6

Table 80: netX 52 – Signals MMIO (Pin table sorted by signals)

Pin	Signal	Shared with								PAD type
T13	N.C.									Reserved
U10	N.C.									Reserved
U11	N.C.									Reserved
U12	N.C.									Reserved
U13	N.C.									Reserved
V10	N.C.									Reserved
V11	N.C.									Reserved
V12	N.C.									Reserved
V13	N.C.									Reserved
V7	OSC_VDDC									PWR
U7	OSC_VSS									GND
U8	OSC_XTI									XTAL
V8	OSC_XTO									XTAL
N16	PHY_ATP									ANA
N17	PHY_EXTRES									ANA
M16	PHY_VDDCAP									APWR
N18	PHY_VDDIOAC									APWR
T17	PHY_VDDIOAT									APWR
M12	PHY_VSSAT									AGND
L16	PHY_VSSCAP									AGND
R17	PHY0_RXN									PHY
R18	PHY0_RXP									PHY
P17	PHY0_TXN									PHY
P18	PHY0_TXP									PHY
U18	PHY0_VDDCART									APWR
P16	PHY0_VSSAR									AGND
L10	PHY0_VSSAT1									AGND

Table 81: netX 52 – Signals OSC ... PHY0 (Pin table sorted by signals)

Pin	Signal	Shared with								PAD type
L11	PHY0_VSSAT2									AGND
K17	PHY1_RXN									PHY
K18	PHY1_RXP									PHY
L17	PHY1_TXN									PHY
L18	PHY1_TXP									PHY
K16	PHY1_VDDCART									APWR
L12	PHY1_VSSAR									AGND
M18	PHY1_VSSAT1									AGND
M17	PHY1_VSSAT2									AGND
C1	PORn									IUS
T14	QSPI_SIO2									IOD6
V15	QSPI_SIO3									IOD6
F1	RDY									IOD6
D1	RSTOUTn									OZ6
F2	RUN									IOD6
V16	SPI0_CLK	QSPI_CLK								IOD6
U15	SPI0_CS0n	QSPI_CSn								IOU6
T15	SPI0_MISO	QSPI_SIO1								IOD6
U16	SPI0_MOSI	QSPI_SIO0								IOD6
G2	TACT_TRST									IDS
B10	TEST									ID
F3	TMC1									I
G3	TMC2									I
U6	USB_DNEG									USB
V6	USB_DPOS									USB
T8	USB_VDDC									PWR
T7	USB_VDDIO									PWR

Table 82: netX 52 – Signals PHY1 ... USB (Pin table sorted by signals)

Pin	Signal	Shared with	PAD type
F6, F7, F8, F9, F10, F11, F12, F13, G6, G13, H6, H13, J6, J13, J17, J18, K6, K13, L6, L13, M6, M13, N6, N7, N8, N9, N10, N11, N12, N13, R16, T10, T12, T16	VDDC	-	PWR
C3, C4, C14, D17, E1, E2, E16, K1, L2, P3, T3, T9, T11, U14, U17, V4	VDDIO	-	PWR
B1, C5, C16, F16, G1, G7, G8, G9, G10, G11, G12, H7, H8, H9, H10, H11, H12, J1, J7, J8, J9, J10, J11, J12, J16, K7, K8, K9, K10, K11, K12, L7, L8, L9, M7, M8, M9, M10, M11, N3, P1, T18, U1, U9, V2, V5, V9, V14, V17, V18	VSS	-	GND

Table 83: netX 52 – Signals VDDC, VDDIO and VSS (Pin table sorted by signals)

6.2 Pin table sorted by pin number

Pins A1 ... A18

Pin	Signal	Shared with								PAD type
A1	HIF_D0	DPM_D0	EXT_D0				SD_D0		PIO83	IOU9
A2	HIF_D28	DPM_D28	EXT_D28	DPM_SELA16	EXT_CS2n		SD_D28	MII12_RXCLK	PIO79	IOU9
A3	HIF_A0	DPM_A0	EXT_A0	DPM_BE0n	EXT_BE0n		SD_A0	MII3_TXER	PIO73	IOU9
A4	HIF_D29	DPM_D29	EXT_D29	DPM_SELA17	EXT_CS1n		SD_D29	MII12_RXER	PIO80	IOU9
A5	HIF_D6	DPM_D6	EXT_D6				SD_D6		PIO75	IOU9
A6	HIF_D26	DPM_D26	EXT_D12	DPM_SELA14	EXT_A22_netX50		SD_D12	MII1_RXD0	PIO71	IOU9
A7	HIF_A5	DPM_A5	EXT_A5				SD_A5	MII3_RXD2	PIO64	IOU9
A8	HIF_D25	DPM_D25	EXT_D25	DPM_SELA13	EXT_A21		SD_D25	MII12_TXEN	PIO68	IOU9
A9	HIF_D24	DPM_D24	EXT_D24	DPM_SELA12	EXT_A20		SD_D24	MII12_TXD3	PIO67	IOU9
A10	HIF_A9	DPM_A9	EXT_A9				SD_A9	MII3_TXD1	PIO56	IOU9
A11	HIF_D22	DPM_D22	EXT_D22	DPM_A18	EXT_A18		SD_D22	MII12_TXD1	PIO62	IOU9
A12	HIF_A14	DPM_A14	EXT_A14				SD_BA0		PIO54	IOU9
A13	HIF_A13	DPM_A13	EXT_A13				SD_DQM0n	MII3_TXCLK	PIO48	IOU9
A14	HIF_A11	DPM_A11	EXT_A11				SD_A11	MII3_TXD3	PIO50	IOU9
A15	HIF_CSn	DPM_CSn	EXT_CS0n				SD_CSn		PIO51	IOU9
A16	HIF_BHE1	DPM_BE1n	EXT_BE1n	DPM_BHEn	EXT_BHEn	DPM_CS2	SD_DQM1n	MII3_RXER	PIO43	IOU9
A17	HIF_A16	DPM_A16	EXT_A16		EXT_A22		SD_RASn	MII1_COL		IOU9
A18	HIF_A17	DPM_A17	EXT_A17	DPM_ALE32	EXT_A23		SD_CASn	MII1_CRS		IOU9

Table 84: netX 52 – Pin A1 ... A18 (Pin table sorted by pin number)

PAD type explanation, see section 7.1 on page 348.

Pins B1 ... B18

Pin	Signal	Shared with								PAD type
B1	VSS									GND
B2	HIF_D5	DPM_D5	EXT_D5				SD_D5		PIO76	IOU9
B3	HIF_D3	DPM_D3	EXT_D3				SD_D3		PIO78	IOU9
B4	HIF_D4	DPM_D4	EXT_D4				SD_D4		PIO77	IOU9
B5	HIF_D7	DPM_D7	EXT_D7				SD_D7		PIO74	IOU9
B6	HIF_A1	DPM_A1	EXT_A1	DPM_BE2n	EXT_BE2n		SD_A1	MII3_COL	PIO70	IOU9
B7	HIF_D27	DPM_D27	EXT_D13	DPM_SELA15	EXT_A23_netX50	IRQ_P72	SD_D13	MII1_RXD1	PIO72	IOU9
B8	HIF_A4	DPM_A4	EXT_A4				SD_A4	MII3_RXD1	PIO65	IOU9
B9	HIF_A6	DPM_A6	EXT_A6				SD_A6	MII3_RXD3	PIO61	IOU9
B10	TEST									ID
B11	HIF_A15	DPM_A15	EXT_A15	DPM_ALE32_netX50			SD_BA1		PIO55	IOU9
B12	HIF_A10	DPM_A10	EXT_A10				SD_A10	MII3_TXD2	PIO53	IOU9
B13	HIF_A12	DPM_A12	EXT_A12				SD_A12	MII3_TXEN	PIO49	IOU9
B14	HIF_WRn	DPM_WRn	EXT_WRn	DPM_WRLn			SD_WEn		PIO45	IOU9
B15	HIF_RDn	DPM_RDn	EXT_RDn	DPM_DATADIR		IRQ_P47	SD_DQM2		PIO52	IOU9
B16	HIF_BHE3	DPM_BE3n	EXT_BE3n	DPM_WRHn	EXT_CS3n	EXT_A24	SD_DQM3	MII2_RXDV	PIO44	IOU9
B17	HIF_D15	DPM_D15	EXT_D31	DPM_SPI_SIO3	EXT_D15_netX50	MMIO47	SD_D31	MII2_RXD3	PIO41	IOU9
B18	HIF_D13	DPM_D13	EXT_D27	DPM_SPI_SIRQ	EXT_D13_netX50	MMIO45	SD_D27	MII2_RXD1	PIO37	IOU9

Table 85: netX 52 – Pin B1 ... B18 (Pin table sorted by pin number)

Pins C1 ... C18

Pin	Signal	Shared with								PAD type
C1	PORn									IUS
C2	HIF_D1	DPM_D1	EXT_D1				SD_D1		PIO82	IOU9
C3	VDDIO									PWR
C4	VDDIO									PWR
C5	VSS									GND
C6	HIF_A2	DPM_A2	EXT_A2				SD_A2	MII3_CRS	PIO69	IOU9
C7	HIF_A3	DPM_A3	EXT_A3				SD_A3	MII3_RXD0	PIO66	IOU9
C8	HIF_SDCLK	DPM_SIRQ					SD_CLK			IOD9
C9	HIF_A7	DPM_A7	EXT_A7				SD_A7	MII3_RXDV	PIO60	IOU9
C10	HIF_D23	DPM_D23	EXT_D23	DPM_A19	EXT_A19		SD_D23	MII12_TXD2	PIO63	IOU9
C11	HIF_D21	DPM_D21	EXT_D21	DPM_A17	EXT_A17		SD_D21	MII12_TXD0	PIO59	IOU9
C12	HIF_D20	DPM_D20	EXT_D20	DPM_A16	EXT_A16		SD_D20	MII12_TCLK	PIO58	IOU9
C13	HIF_A8	DPM_A8	EXT_A8				SD_A8	MII3_TXD0	PIO57	IOU9
C14	VDDIO									PWR
C15	HIF_RDY	DPM_RDY	EXT_RDY				SD_CKE	MII3_RXCLK	PIO46	IOU9
C16	VSS									GND
C17	HIF_D14	DPM_D14	EXT_D30	DPM_SPI_SIO2	EXT_D14_netX50	MMIO46	SD_D30	MII2_RXD2	PIO42	IOU9
C18	HIF_D12	DPM_D12	EXT_D26	DPM_SPI_DIRQ	EXT_D12_netX50	MMIO44	SD_D26		PIO38	IOU9

Table 86: netX 52 – Pin C1 ... C18 (Pin table sorted by pin number)

Pins D1 ... D18, E1 ... E18

Pin	Signal	Shared with								PAD type
D1	RSTOUTn									OZ6
D2	HIF_D2	DPM_D2	EXT_D2				SD_D2		PIO81	IOU9
D3	HIF_D30	DPM_D30	EXT_D14	DPM_SELA18	EXT_CS3n_netX50		SD_D14	MII1_RXD2	PIO84	IOU9
D16	HIF_DIRQ	DPM_INT	EXT_CS1n_D32		EXT_INT			MII2_RXD0	PIO47	IOU9
D17	VDDIO									PWR
D18	HIF_D11	DPM_D11	EXT_D19	DPM_SPI_CLK	EXT_D11_netX50	MMIO43	SD_D19		PIO39	IOU9
E1	VDDIO									PWR
E2	VDDIO									PWR
E3	BSCAN_TRST									IDS
E16	VDDIO									PWR
E17	HIF_D19	DPM_D19	EXT_D11				SD_D11		WDGACT	IOU9
E18	HIF_D18	DPM_D18	EXT_D10	DPM_SELA19	EXT_A24_netX50		SD_D10	MII1_RXDV	PIO40	IOU9

Table 87: netX 52 – Pin D1 ... D18, E1 ... E18 (Pin table sorted by pin number)

Pins F1 ... F18

Pin	Signal	Shared with								PAD type
F1	RDY									IOD6
F2	RUN									IOD6
F3	TMC1									I
F6	VDDC									PWR
F7	VDDC									PWR
F8	VDDC									PWR
F9	VDDC									PWR
F10	VDDC									PWR
F11	VDDC									PWR
F12	VDDC									PWR
F13	VDDC									PWR
F16	VSS									GND
F17	HIF_D10	DPM_D10	EXT_D18	DPM_SPI_CS _n	EXT_D10_netX50	MMIO42	SD_D18	MII2_TXER	PIO33	IOU9
F18	HIF_D17	DPM_D17	EXT_D9	DPM_ALE16		IRQ_P35	SD_D9		PIO35	IOU9

Table 88: netX 52 – Pin F1 ... F18 (Pin table sorted by pin number)

Pins G1 ... G18

Pin	Signal	Shared with								PAD type
G1	VSS									GND
G2	TACT_TRST									IDS
G3	TMC2									I
G6	VDDC									PWR
G7	VSS									GND
G8	VSS									GND
G9	VSS									GND
G10	VSS									GND
G11	VSS									GND
G12	VSS									GND
G13	VDDC									PWR
G16	HIF_D9	DPM_D9	EXT_D17	DPM_SPI_MOSI / SIO0	EXT_D9_netX50	MMIO41	SD_D17	MII2_CRS	PIO34	IOU9
G17	HIF_D16	DPM_D16	EXT_D8			IRQ_P36	SD_D8	MII1_TXER	PIO36	IOU9
G18	HIF_D8	DPM_D8	EXT_D16	DPM_SPI_MISO / SIO1	EXT_D8_netX50	MMIO40	SD_D16	MII2_COL	PIO32	IOU9

Table 89: netX 52 – Pin G1 ... G18 (Pin table sorted by pin number)

Pins H1 ... H18

Pin	Signal	Shared with								PAD type
H1	JT_TCLK									IUS
H2	MMIO0				FO0_RD					IODS6
H3	JT_TRSTn									IDS
H6	VDDC									PWR
H7	VSS									GND
H8	VSS									GND
H9	VSS									GND
H10	VSS									GND
H11	VSS									GND
H12	VSS									GND
H13	VDDC									PWR
H16	I2C_SDA									IOUS9 (5k pU)
H17	HIF_D31	DPM_D31	EXT_D15	DPM_SELA11			SD_D15	MII1_RXD3	PIO85	IOU9
H18	I2C_SCL									IOUS9 (5k pU)

Table 90: netX 52 – Pin H1 ... H18 (Pin table sorted by pin number)

Pins J1 ... J18

Pin	Signal	Shared with								PAD type
J1	VSS									GND
J2	MMIO2				FO0_EN					IODS6
J3	JT_TDI									IUS
J6	VDDC									PWR
J7	VSS									GND
J8	VSS									GND
J9	VSS									GND
J10	VSS									GND
J11	VSS									GND
J12	VSS									GND
J13	VDDC									PWR
J16	VSS									GND
J17	VDDC									GND
J18	VDDC									GND

Table 91: netX 52 – Pin J1 ... J18 (Pin table sorted by pin number)

Pins K1 ... K18

Pin	Signal	Shared with								PAD type
K1	VDDIO									PWR
K2	MMIO1				FO0_TD					IODS6
K3	JT_TMS									IUS
K6	VDDC									PWR
K7	VSS									GND
K8	VSS									GND
K9	VSS									GND
K10	VSS									GND
K11	VSS									GND
K12	VSS									GND
K13	VDDC									PWR
K16	PHY1_VDDCART									APWR
K17	PHY1_RXN									PHY
K18	PHY1_RXP									PHY

Table 92: netX 52 – Pin K1 ... K18 (Pin table sorted by pin number)

Pins L1 ... L18

Pin	Signal	Shared with								PAD type
L1	MMIO4				FO1_TD					IODS6
L2	VDDIO									PWR
L3	MMIO3				FO1_RD					IODS6
L6	VDDC									PWR
L7	VSS									GND
L8	VSS									GND
L9	VSS									GND
L10	PHY0_VSSAT1									AGND
L11	PHY0_VSSAT2									AGND
L12	PHY1_VSSAR									AGND
L13	VDDC									PWR
L16	PHY_VSSCAP									AGND
L17	PHY1_TXN									PHY
L18	PHY1_TXP									PHY

Table 93: netX 52 – Pin L1 ... L18 (Pin table sorted by pin number)

Pins M1 ... M18

Pin	Signal	Shared with								PAD type
M1	MMIO5				FO1_EN					IODS6
M2	MMIO6				FO0_SD					IODS6
M3	JT_TDO									OZ6
M6	VDDC									PWR
M7	VSS									GND
M8	VSS									GND
M9	VSS									GND
M10	VSS									GND
M11	VSS									GND
M12	PHY_VSSAT									AGND
M13	VDDC									PWR
M16	PHY_VDDCAP									APWR
M17	PHY1_VSSAT2									AGND
M18	PHY1_VSSAT1									AGND

Table 94: netX 52 – Pin M1 ... M18 (Pin table sorted by pin number)

Pins N1 ... N18

Pin	Signal	Shared with								PAD type
N1	MMIO7				FO1_SD					IODS6
N2	MMIO8									IODS6
N3	VSS									GND
N6	VDDC									PWR
N7	VDDC									PWR
N8	VDDC									PWR
N9	VDDC									PWR
N10	VDDC									PWR
N11	VDDC									PWR
N12	VDDC									PWR
N13	VDDC									PWR
N16	PHY_ATP									ANA
N17	PHY_EXTRES									ANA
N18	PHY_VDDIOAC									APWR

Table 95: netX 52 – Pin N1 ... N18 (Pin table sorted by pin number)

Pins P1 ... P18, R1 ... R18

Pin	Signal	Shared with								PAD type
P1	VSS									GND
P2	MMIO9									IODS6
P3	VDDIO									PWR
P16	PHY0_VSSAR									AGND
P17	PHY0_TXN									PHY
P18	PHY0_TXP									PHY
R1	MMIO10									IODS6
R2	MMIO12									IODS6
R3	MMIO15									IODS6
R16	VDDC									PWR
R17	PHY0_RXN									PHY
R18	PHY0_RXP									PHY

Table 96: netX 52 – Pin P1 ... P18, R1 ... R18 (Pin table sorted by pin number)

Pins T1 ... T18

Pin	Signal	Shared with								PAD type
T1	MMIO11									IODS6
T2	MMIO14									IODS6
T3	VDDIO									PWR
T4	MMIO16									IODS6
T5	MMIO20									IODS6
T6	MMIO22									IODS6
T7	USB_VDDIO									PWR
T8	USB_VDDC									PWR
T9	VDDIO									PWR
T10	VDDC									PWR
T11	VDDIO									PWR
T12	VDDC									PWR
T13	N.C.									Reserved
T14	QSPI_SIO2									IOD6
T15	SPI0_MISO	QSPI_SIO1								IOD6
T16	VDDC									PWR
T17	PHY_VDDIOAT									APWR
T18	VSS									GND

Table 97: netX 52 – Pin T1 ... T18 (Pin table sorted by pin number)

Pins U1 ... U18

Pin	Signal	Shared with								PAD type
U1	VSS									GND
U2	MMIO13									IODS6
U3	MMIO19									IODS6
U4	MMIO18									IODS6
U5	MMIO23									IODS6
U6	USB_DNEG									USB
U7	OSC_VSS									GND
U8	OSC_XTI									XTAL
U9	VSS									GND
U10	N.C.									Reserved
U11	N.C.									Reserved
U12	N.C.									Reserved
U13	N.C.									Reserved
U14	VDDIO									PWR
U15	SPI0_CS0n	QSPI_CSn								IOU6
U16	SPI0_MOSI	QSPI_SIO0								IOD6
U17	VDDIO									PWR
U18	PHY0_VDDCART									APWR

Table 98: netX 52 – Pin U1 ... U18 (Pin table sorted by pin number)

Pins V1 ... V18

Pin	Signal	Shared with								PAD type
V1	MMIO17									IODS6
V2	VSS									GND
V3	MMIO21									IODS6
V4	VDDIO									PWR
V5	VSS									GND
V6	USB_DPOS									USB
V7	OSC_VDDC									PWR
V8	OSC_XTO									XTAL
V9	VSS									GND
V10	N.C.									Reserved
V11	N.C.									Reserved
V12	N.C.									Reserved
V13	N.C.									Reserved
V14	VSS									GND
V15	QSPI_SIO3									IOD6
V16	SPI0_CLK	QSPI_CLK								IOD6
V17	VSS									GND
V18	VSS									GND

Table 99: netX 52 – Pin V1 ... V18 (Pin table sorted by pin number)

6.3 Pin overview netX 52

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
A	HIF_D0	HIF_D28	HIF_A0	HIF_D29	HIF_D6	HIF_D26	HIF_A5	HIF_D25	HIF_D24	HIF_A9	HIF_D22	HIF_A14	HIF_A13	HIF_A11	HIF_CS _n	HIF_BHE1	HIF_A16	HIF_A17	A
B	VSS	HIF_D5	HIF_D3	HIF_D4	HIF_D7	HIF_A1	HIF_D27	HIF_A4	HIF_A6	TEST	HIF_A15	HIF_A10	HIF_A12	HIF_WR _n	HIF_RD _n	HIF_BHE3	HIF_D15	HIF_D13	B
C	POR _n	HIF_D1	VDDIO	VDDIO	VSS	HIF_A2	HIF_A3	HIF_SDCLK	HIF_A7	HIF_D23	HIF_D21	HIF_D20	HIF_A8	VDDIO	HIF_RDY	VSS	HIF_D14	HIF_D12	C
D	RSTOUT _n	HIF_D2	HIF_D30													HIF_DIRQ	VDDIO	HIF_D11	D
E	VDDIO	VDDIO	BSCAN_TRST													VDDIO	HIF_D19	HIF_D18	E
F	RDY	RUN	TMC1			VDDC	VDDC	VDDC	VDDC	VDDC	VDDC	VDDC	VDDC			VSS	HIF_D10	HIF_D17	F
G	VSS	TACT_TRST	TMC2			VDDC	VSS	VSS	VSS	VSS	VSS	VSS	VDDC			HIF_D9	HIF_D16	HIF_D8	G
H	JT_TCLK	MMIO0	JT_TRST _n			VDDC	VSS	VSS	VSS	VSS	VSS	VSS	VDDC			I2C_SDA	HIF_D31	I2C_SCL	H
J	VSS	MMIO2	JT_TDI			VDDC	VSS	VSS	VSS	VSS	VSS	VSS	VDDC			VSS	VDDC	VDDC	J
K	VDDIO	MMIO1	JT_TMS			VDDC	VSS	VSS	VSS	VSS	VSS	VSS	VDDC			PHY1_VDDCART	PHY1_RXN	PHY1_RXP	K
L	MMIO4	VDDIO	MMIO3			VDDC	VSS	VSS	VSS	PHY0_VSSAT1	PHY0_VSSAT2	PHY1_VSSAR	VDDC			PHY_VSSCAP	PHY1_TXN	PHY1_TXP	L
M	MMIO5	MMIO6	JT_TDO			VDDC	VSS	VSS	VSS	VSS	VSS	PHY_VSSAT	VDDC			PHY_VDDCAP	PHY1_VSSAT2	PHY1_VSSAT1	M
N	MMIO7	MMIO8	VSS			VDDC	VDDC	VDDC	VDDC	VDDC	VDDC	VDDC	VDDC			PHY_ATP	PHY_EXTRES	PHY_VDDIOAC	N
P	VSS	MMIO9	VDDIO													PHY0_VSSAR	PHY0_TXN	PHY0_TXP	P
R	MMIO10	MMIO12	MMIO15													VDDC	PHY0_RXN	PHY0_RXP	R
T	MMIO11	MMIO14	VDDIO	MMIO16	MMIO20	MMIO22	USB_VDDIO	USB_VDDC	VDDIO	VDDC	VDDIO	VDDC	N.C.	QSPI_SIO2	SPI0_MISO	VDDC	PHY_VDDIOAT	VSS	T
U	VSS	MMIO13	MMIO19	MMIO18	MMIO23	USB_DNEG	OSC_VSS	OSC_XT1	VSS	N.C.	N.C.	N.C.	N.C.	VDDIO	SPI0_CS _{0n}	SPI0_MOSI	VDDIO	PHY0_VDDCART	U
V	MMIO17	VSS	MMIO21	VDDIO	VSS	USB_DPOS	OSC_VDDC	OSC_XT0	VSS	N.C.	N.C.	N.C.	N.C.	VSS	QSPI_SIO3	SPI0_CLK	VSS	VSS	V
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	

6.4 Multiplex matrix signals netX 52

Function	Direction	Description	Power-on MMIO mapping	Hilscher default MMIO mapping
GPIOs / IO-LINK				
GPIO00-GPIO31	IO	General Purpose IOs or IOLINK signal blocks	MMIO00-19	MMIO00-19
I²C				
I2C0_SCL	IO	I ² C Interface clock signal	-	-
I2C0_SDA	IO	I ² C Interface data signal	-	-
I2C1_SCL	IO	I ² C Interface clock signal	-	-
I2C1_SDA	IO	I ² C Interface data signal	-	-
MII (Media Independent Interface (Ethernet))				
MII_MDC	O	-	-	-
MII_MDIO	IO	-	-	-
MII0_COL	I	-	-	-
MII0_CRS	I	-	-	-
MII0_LED0	I	-	-	-
MII0_LED1	I	-	-	-
MII0_LED2	I	-	-	-
MII0_LED3	I	-	-	-
MII0_RXCLK	I	-	-	-
MII0_RXD0	I	-	-	-
MII0_RXD1	I	-	-	-
MII0_RXD2	I	-	-	-
MII0_RXD3	I	-	-	-
MII0_RXDV	I	-	-	-
MII0_RXER	I	-	-	-
MII0_TXCLK	I	-	-	-
MII0_TXD0	OZ	-	-	-
MII0_TXD1	OZ	-	-	-
MII0_TXD2	OZ	-	-	-
MII0_TXD3	OZ	-	-	-
MII0_TXEN	O	-	-	-
MII0_TXER	O	-	-	-
MII1_COL	I	-	-	-
MII1_CRS	I	-	-	-
MII1_LED0	I	-	-	-
MII1_LED1	I	-	-	-
MII1_LED2	I	-	-	-
MII1_LED3	I	-	-	-
MII1_RXCLK	I	-	-	-
MII1_RXD0	I	-	-	-
MII1_RXD1	I	-	-	-
MII1_RXD2	I	-	-	-
MII1_RXD3	I	-	-	-
MII1_RXDV	I	-	-	-

Function	Direction	Description	Power-on MMIO mapping	Hilscher default MMIO mapping
MII1_RXER	I	-	-	-
MII1_TXCLK	I	-	-	-
MII1_TXD0	OZ	-		
MII1_TXD1	OZ	-		
MII1_TXD2	OZ	-		
MII1_TXD3	OZ	-		
MII1_TXEN	O	-		
MII1_TXER	O	-		
PIOs				
PIO0 – 3	IO	Programmable IOs 0 – 3	-	MMIO16 – 19
PIO4 – 7	IO	Programmable IOs 4 – 7	-	
PHY LEDs				
PHY0_LED0 – 1	O	Status LEDs 0 – 1 of internal PHY0	-	MMIO12 – 13
PHY0_LED2 – 3	O	Status LEDs 2 – 3 of internal PHY0	-	-
PHY1_LED0 – 1	O	Status LEDs 0 – 1 of internal PHY1	-	MMIO14 – 15
PHY1_LED2 – 3	O	Status LEDs 2 – 3 of internal PHY1	-	-
SPI				
SPI0_CS2n	IO	Chip Select 2 signal of SPI0 interface	-	-
SPI0_SIO2	IO	Serial IO Data 2	-	-
SPI0_SIO3	IO	Serial IO Data 2	-	-
SPI1_CLK	IO	Clock signal of SPI1 interface	-	-
SPI1_CS0n	IO	Chip Select 0 signal of SPI1 interface	-	-
SPI1_CS1n	IO	Chip Select 1 signal of SPI1 interface	-	-
SPI1_CS2n	IO	Chip Select 2 signal of SPI1 interface	-	-
SPI1_MISO	IO	Master In / Slave Out signal of SPI1 interface	-	-
SPI1_MOSI	IO	Master Out / Slave In signal of SPI1 interface	-	-
UARTs				
UART0_CTSn	I	UART0 Clear To Send	MMIO22	MMIO22
UART0_RTSn	OZ	UART0 Request To Send	MMIO23	MMIO23
UART0_RXD	I	UART0 Receive Data	MMIO20	MMIO20
UART0_TXD	OZ	UART0 Transmit Data	MMIO21	MMIO21
UART1_CTSn	I	UART1 Clear To Send	-	-
UART1_RTSn	OZ	UART1 Request To Send	-	-
UART1_RXD	I	UART1 Receive Data	-	-
UART1_TXD	OZ	UART1 Transmit Data	-	-
UART2_CTSn	I	UART2 Clear To Send	-	-
UART2_RTSn	OZ	UART2 Request To Send	-	-
UART2_RXD	I	UART2 Receive Data	-	-
UART2_TXD	OZ	UART2 Transmit Data	-	-
XMAC0 (Fieldbus 0)				
XM0_IO0	IO	XMAC0 Programmable IO 0	-	MMIO03
XM0_IO1	IO	XMAC0 Programmable IO 1	-	MMIO02
XM0_IO2 – 5	IO	XMAC0 Programmable IO2 – 5	-	tbd
XM0_RX	I	XMAC0 Receive Data	-	MMIO00

Function	Direction	Description	Power-on MMIO mapping	Hilscher default MMIO mapping
XM0_TX_OE	O	XMAC0 Transmit Enable	-	-
XM0_TX_OUT	OZ	XMAC0 Transmit Data	-	MMIO01
XMAC1 (Fieldbus 1)				
XM1_IO0	IO	XMAC1 Programmable IO 0	-	MMIO07
XM1_IO1	IO	XMAC1 Programmable IO 1	-	MMIO05
XM1_IO2 - 5	IO	XMAC1 Programmable IO2 – 5	-	tbd
XM1_RX	I	XMAC1 Receive Data	-	MMIO06
XM1_TX_OE	O	XMAC1 Transmit Enable	-	-
XM1_TX_OUT	OZ	XMAC1 Transmit Data	-	MMIO04
CAN				
CAN_RX	I	CAN Receive Data	-	-
CAN_TXD	O	CAN Transmit Data	-	-
RTE Sync Signals				
XC_SAMPLE0	I	-	-	MMIO08
XC_SAMPLE1	I	-	-	MMIO09
XC_TRIGGER0	OZ	-	-	MMIO10
XC_TRIGGER1	OZ	-	-	MMIO11
MEM				
MEM_RDY	I	MEM IF ready/busy input	-	-

Table 100: Multiplex matrix signals (netX 52)

6.5 netX 52 package

The netX 52 has a 244-pin PBGA package with a 0.8 mm grid.

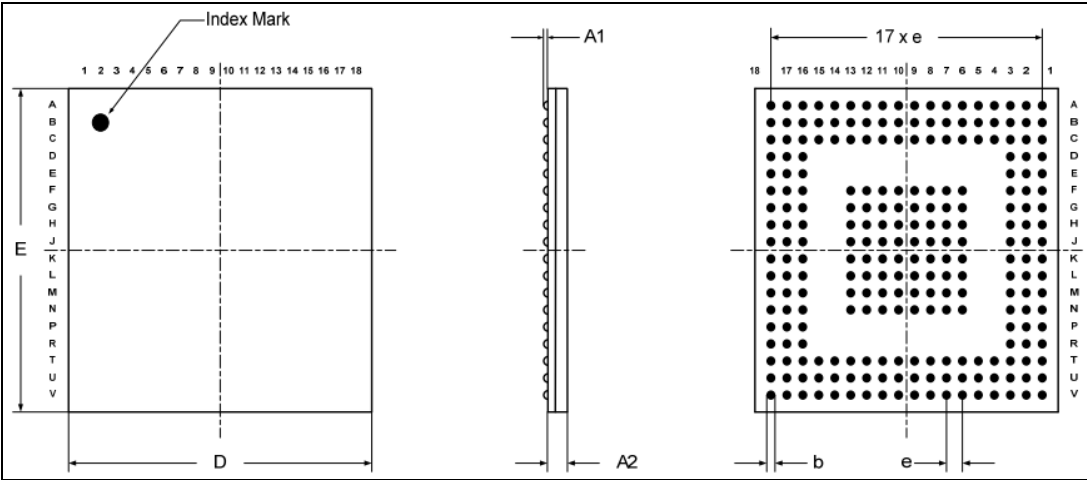


Figure 184: Mechanical dimensions of netX 52

Symbol	Min	Type	Max
A1	0.29 mm	0.35 mm	0.41 mm
A2		1.11 mm	
b	0.40 mm	0.50 mm	0.55 mm
E	14.90 mm	15.00 mm	15.10 mm
e		0.80 mm	
D	14.90 mm	15.00 mm	15.10 mm

Table 101: Mechanical dimensions of netX 52

6.6 Ordering information

Order number	Item	Number of pieces
2232.001	NETX 52 STEP B	1 piece
2232.101	NETX 52 B (BOX)	15 pcs
2232.201	NETX 52 B (TRAY)	119 pcs
2232.952	NETX 52 B (PACKAGE952)	952 pcs

6.7 Thermal package specification

Absolute max. junction temperature: $T_{j\_max} = 125^{\circ}\text{C}$

Absolute min. junction temperature: $T_{j\_min} = -40^{\circ}\text{C}$

Recommended operating ambient temperature

Operating ambient temperature: $-40^{\circ}\text{C} \dots +85^{\circ}\text{C}$ @ typ. 1.5 W (with heat sink $R_{th} \leq 10 \text{ K/W}$)

Operating ambient temperature: $-40^{\circ}\text{C} \dots +70^{\circ}\text{C}$ @ typ. 1.5 W (without heat sink)

Thermal characterization (based on JEDEC-2S2P PCB)

Symbol	Parameter	Air flow [m/s]				Unit
		0	0.2	1	2	
θ_{ja}	Thermal resistance, junction-to-ambient	15.5	13.9	12.1	11.3	$^{\circ}\text{C/W}$
ψ_{jt}	Thermal parameter, junction to the top center of the package surface	0.07	0.11	0.17	0.21	$^{\circ}\text{C/W}$
ψ_{ta}	Thermal parameter, the top center of the package surface to ambient	15.5	13.8	11.9	11.1	$^{\circ}\text{C/W}$
θ_{jc}	Thermal resistance, junction-to-case	4.6				$^{\circ}\text{C/W}$

Case 1 - Without external heat sink	Case 2 - With external heat sink
Heat flow path exists not only on package top	Package surface dominates heat flow path via heat sink
$T_t = T_a + \psi_{ta} \times P_{netX}$	$T_j = T_a + (\theta_{jc} + R_{th}) \times P_{netX}$
$T_j \approx T_t$	

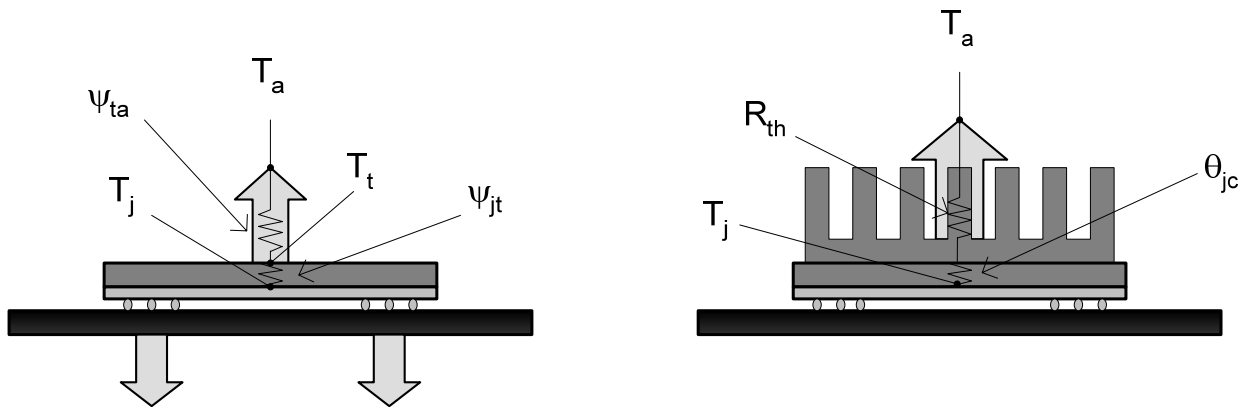


Figure 185: Symbols of thermal characterization

6.8 Soldering conditions

See page 315.

6.9 General storage conditions

See page 317.

6.10 Moisture sensitivity level

The Moisture Sensitivity Level (MSL) is MSL 3, 168 h, 30 °C / 60%RH.

7 Appendix

7.1 PAD type explanation

Symbol	Description
I	Input
O	Output
Z	Output is tristateable or open drain
S	Input provides Schmitt trigger
U	Internal pull-up 50 k (I2C pins: Pull-up 5 k)
D	Internal pull-down 50 k
6	Output buffer can source/sink 6 mA
9	Output buffer can source/sink 9 mA
XTAL	Crystal input or output
USB	USB pad
PHY	PHY pad
ANA	Analog pin
PWR	1.5 V (core) or 3.3 V (I/O)
GND	Digital ground (0 V)
APWR	Analog power (1.5 V or 3.3 V)
AGND	Analog ground (0 V)

Note: PAD types in brackets resemble the logical function of a buffer in a certain situation (if a pure input signal is assigned to an IO buffer, the buffer does not change at all, but it behaves like an input buffer because its output driver is disabled automatically.)

Schematic view of netX PAD types

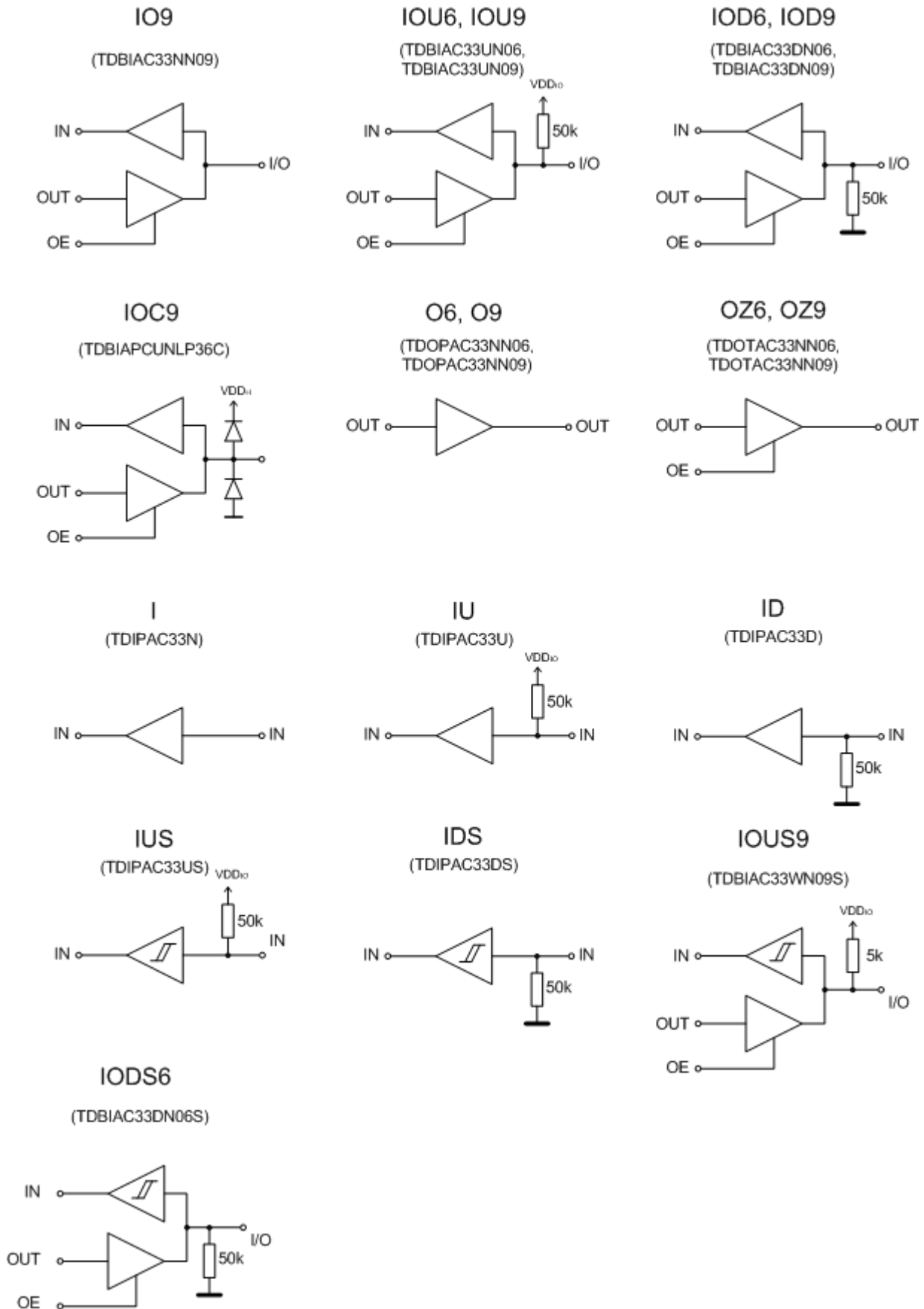


Figure 186: Schematic view of netX PAD types

7.2 netX 51/52 SDRAM bank address metal-fix issues

This section describes the metal-fix for netX 51/52 step B.

Note: The HIF-eMI/HIF-SDRAM is not affected by netX 51/52 metal-fix (MEM-eMI SDRAM bank address change).

The metal-fix was realized for netX 50 pinning compatibility: The SDRAM bank address signals BA0 and BA1 have been changed from MEM_A14 and MEM_A15 to MEM_A16 and MEM_A17 and are now on netX 50-compatible pin positions. However, this has an impact on the SRAM interconnection if SDRAM and SRAM (e.g. FLASH) devices are used simultaneously. For real applications this is not allowed, but still done on some evaluation boards.

Important: Connecting SDRAM and SRAM/FLASH is not netX 50-compatible and explicitly not allowed for netX 51/52 (and netX 50). In that case SDRAM timing cannot be guaranteed because the IO load will exceed specified max. values, see 4.4.7.1).

To reach netX 50 SDRAM bank address IO compatibility, MEM_A14 is swapped with MEM_A16 and MEM_A15 is swapped with MEM_A17 netX 51/52 internally provided that SDRAM is enabled only. To be more precise, the lines are swapped when the MEM-eMI SDRAM controller clock signal is enabled.

Swapping effects both: SDRAM and SRAM/FLASH address lines. I.e. when the SDRAM clock is running netX 51/52 SRAM/FLASH addressing is no longer netX 50-compatible.

Swapped address lines lead to non-linear addressing for SRAM/FLASH (e.g. address 0x04000 will be swapped to 0x10000 or address 0x0C010 will be swapped to 0x30010).

For pure SRAM-memory devices (only used for data storing) swapped addresses normally have no negative impact. However, there is negative impact for FLASH (and all other devices where different addresses have different functions):

Important: Typically, FLASH devices are page-oriented and addressing inside pages must be realized with linearly incrementing addresses during programming. This is no longer possible if address lines are swapped without an address recalculation or data remapping by software (compensating the swapping by software).

Restrictions for netX 51/52 when using SDRAM without SRAM/FLASH:

- No restrictions (SDRAM access always requires enabled SDRAM clock).

Restrictions for netX 51/52 when using SRAM/FLASH without SDRAM:

- The SDRAM controller must not be enabled to avoid swapped addresses and guarantee linear addressing.

Restrictions for netX 51/52 when using both SRAM/FLASH and SDRAM:

- Access to first 16kB of SRAM/FLASH (below offset 0x4000) is possible without any limitations or differences to netX 50 (e.g. for boot options stored in the first bytes of FLASH).
- The following address lines of netX 51/52 must be connected to the following SRAM/FLASH address lines to reach linear addressing:
 - MEM_A14 must be connected to SRAM/FLASH A16
 - MEM_A15 must be connected to SRAM/FLASH A17
 - MEM_A16 must be connected to SRAM/FLASH A14
 - MEM_A17 must be connected to SRAM/FLASH A15

- The MEM-eMI SDRAM controller and external clock must be enabled (bits *ctrl_en* and *extclk_en* must be programmed to 1) and the *sdram_ready* bit must have become active (i.e. SDRAM initialization sequence must be completed by the controller) before complete linear addressing of SRAM/FLASH is possible. The SDRAM must not be programmed to power-down mode (bit *sdram_pwrn* must not be programmed to 1).

Note: All bits are part of register *EXT_SDRAM_CTRL_SDRAM_GENERAL_CTR*.

7.2.1 Interconnections to SDRAM and SRAM/FLASH

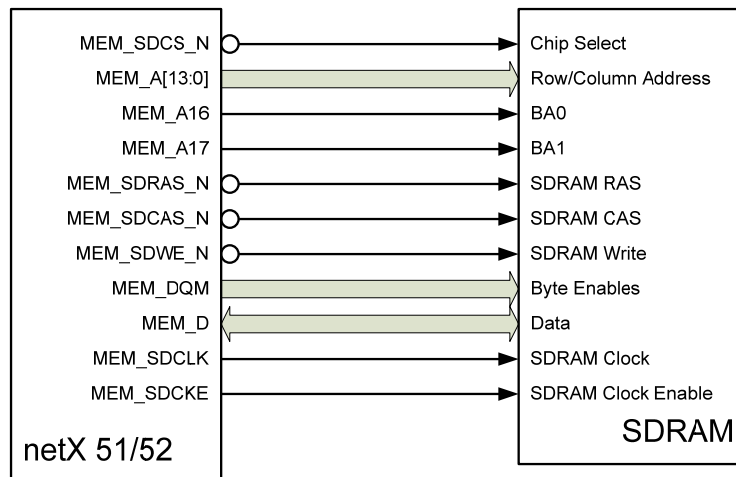


Figure 187: Fixed SDRAM interconnection to MEM-eMI (now netX 50-compatible)

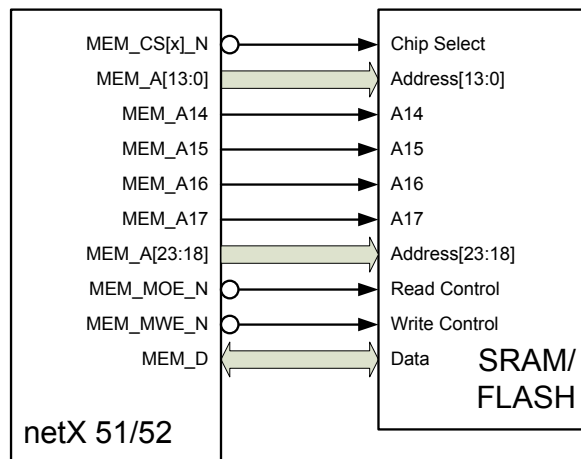


Figure 188: SRAM/FLASH interconnection to MEM-eMI without SDRAM (still netX 50-compatible)

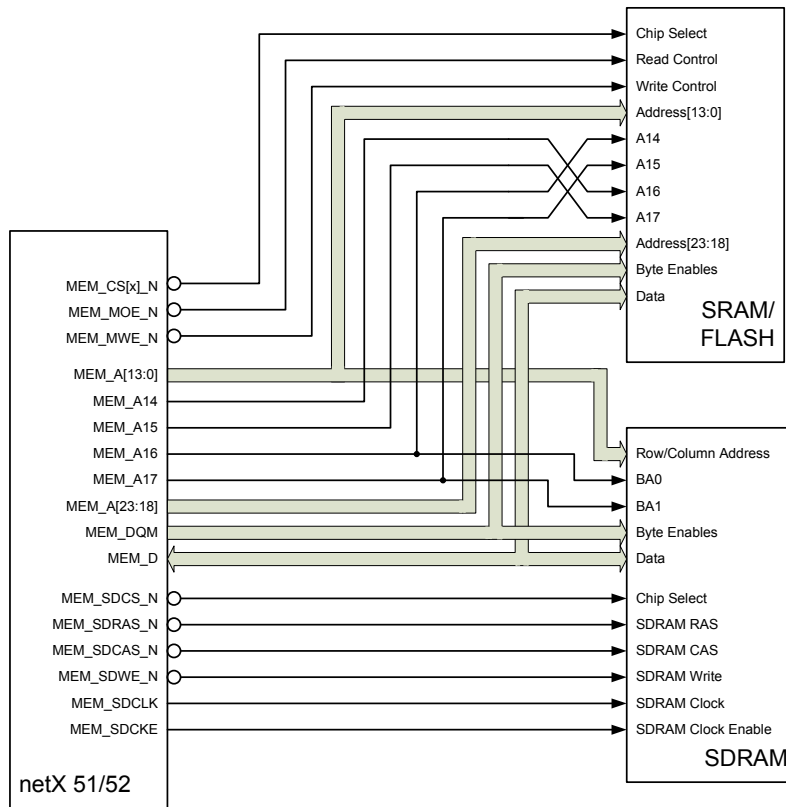


Figure 189: netX 51/52 SRAM/FLASH interconnection to MEM-eMI when used together with SDRAM (not netX 50-compatible)

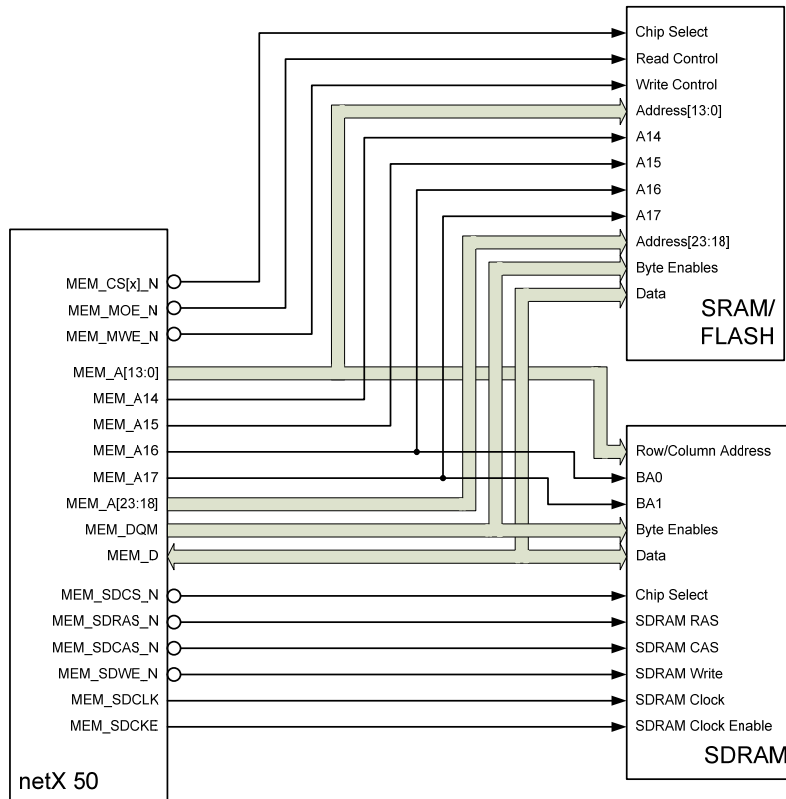


Figure 190: netX 50 SRAM/FLASH interconnection to MEM-eMI when used together with SDRAM (differs from netX 51/52)

7.3 Terms, abbreviations and definitions

Term	Description
ACK	Acknowledge
ADV	Address/Data Valid
AGND	Analog ground (0 V)
AHB	Advanced High-performance Bus
ANA	Analog pin
APM	Asynchronous Page Mode
APWR	Analog power (1.5V or 3.3V)
ARM	Advanced RISC Machines Limited
ASIC	Application-Specific Integrated Circuit
BA	Bank Address
BHE	Byte High Enable
BSCAN	Boundary Scan
CAS	Column Address Strobe
CFG	Configuration
CKE	Clock Enable
CL	CAS-Latency
CLK	Clock
CNT	Counter
CORDIC	COordinate Rotation Digital Computer
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check(sum)
CSn	chip-select
CTS	clear-to-send
CTRL	Control
D	Data
DC	Direct Current
DFP	Dynamic Frame Packing
DHCP	Dynamic Host Configuration Protocol
DIRQ	Dual-Port-Memory (or Data) Interrupt ReQuest
DMA	Direct Memory Access
DPM	Dual-Port Memory
DQM	Data Qualifier Mask
DSP	Digital Signal Processor
ECLK	External Clock
EEPROM	Electrically Erasable Programmable Read Only Memory
ETH	ETHERnet
ETM	Embedded Trace Macrocell
ExtBus	Extension Bus
FB	Fieldbus
FCS	Frame Check Sequence
FIFO	First In First Out
FIQ	Fast Interrupt Request
FIT	Failure Rate
FSS	Frame Slave Select
FTS	Fast-Track Switching

Term	Description
GND	Digital Ground (0 V)
GPIO	General Purpose Input Output
HIF	Host InterFace
HAL	Hardware Abstraction Layer
HIF-eMI	Host InterFace – extended Memory Interface
IEEE	Institute of Electrical and Electronics Engineers
I2C	Inter-Integrated Circuit
ICE	In-Circuit Emulator
IF	Interface
INTRAM	INTernal SRAM
IO or I/O	Input Output or Input/Output
IRQ	Interrupt Request
IRT	Isochronous Real-Time
ISA	Instruction Set Architecture
ISR	Interrupt Service Routine
JTAG	Joint Test Action Group
LED	Light Emitting Diode
LSB	Least Significant Bit
MDIO	Management Data Input Output
MEM	Memory
MEM-eMI	Memory (Interface) – extended Memory Interface
MII	Media Independent Interface
MIIFL	MII Feature Level
MIIMU	MII Management Unit
MISC	Miscellaneous
MISO	Master In Slave Out
MMC/SD	Multi Media Card/Secure Digital (Memory Card)
MMIO	Multiplex Matrix Input/Output
MOSI	Master Out Slave In
MSB	Most Significant Bit
MSYNC	Motion SYNChronization Channel
nADV	not Address/Data Valid
NRZ	Non Return-to-Zero
nVICFIQ	non-Vectored Fast Interrupt Request
NVRAM	Non-Volatile Random Access Memory
OM	Output Mode
OSC	Oscillator
PBGA	Plastic Ball Grid Array
PCB	Printed Circuit Board
pDPM	parallel Dual-Port Memory
PHY	Physical Layer
PIO	Programmable Input/Output
PLL	Phase Lock Loop
PWM	Pulse Width Modulation
PWR	POWER 1.5 V (Core) or 3.3 V (I/O)
OSAC	Operating System ACcelerator
QDP2	Quad (4-bit) Serial Dual-Port Memory, Frame format 2

Term	Description
RAM	Random Access Memory
RAS	Row Address Strobe
RBL	Read Byte Latching
RDn	read enable
RDY	READY
RES	RESet
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RTS	request-to-send
RXD	Receive Data
RXN	Receive Input Negative
RXP	Receive Input Positive
RZI	Return To Zero, Inverted
SCL	Serial Clock
SDA	Serial Data
SDP2	Serial Dual-Port Memory, Frame format 2
sDPM	serial Dual-Port Memory
SDRAM	Synchronous Dynamic Random Access Memory
SELA	SElect Address
SIO	Serial Input Output (Data)
SRAM	Static Random-Access Memory
SPH	SPI Clock Phase
SPI	Serial Peripheral Interface
SIRQ	Serial (or Sync) Interrupt ReQuest
SPO	SPI Clock Polarity
SQI	Serial Quad I/O
SQIROM	Serial Quad I/O Read Only Memory
TC	Threshold Capture
TCM	Tightly Coupled Memory
TCP	Transmission Control Protocol
TDO	Test Data Output
TFTP	Trivial File Transfer Protocol
TIOMAP	Texas Instruments Open Multimedia Application Platform
TMS	Test Mode Select
TOSA	Address Setup-Time
TRDS	Read Data Setup Time
TXD	Transmit Data
TXN	Transmit Output Negative
TXP	Transmit Output Positive
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VDD	Positive supply voltage
VIC	Vectored Interrupt Controller
VPS	Vapor Phase reflow Soldering
VSS	Ground Supply
WBC	Write Byte Collecting
WDG	Watchdog

Term	Description
WE	Write Enable
win	window
WRN	Write Enable
xC	Communication Channel
XiP	EXecution in PPlace
xMAC	Flexible Media Access Controller
xPEC	Flexible Protocol Execution Controller
xPIC	Flexible Peripheral Interface Controller
XTAL	Crystal input or output
XTI	Crystal Input
XTO	Crystal Output
ZLP	Zero Length Packet

Table 102: Terms, abbreviations and definitions

7.4 List of tables

Table 1: List of revisions	5
Table 2: References to documents	6
Table 3: Key features netX 51/52	12
Table 4: Additional features	14
Table 5: netX 51/52 signal description	18
Table 6: Core CPU	19
Table 7: Memory layout	20
Table 8: States of system LED (used from the ROM code)	23
Table 9: Boot sequence	24
Table 10: Host interface mode – Device at host interface is not used for booting	25
Table 11: Host interface mode – SDRAM/SRAM at host interface	25
Table 12: Host interface mode – Dual-port memory is used for booting	26
Table 13: Host interface mode - Reserved	26
Table 14: Memory interface for Quad SPI mode set by MMIO configuration	35
Table 15: SDRAM memory size and organization	43
Table 16: New signal assignment netX 51/52 for extension bus mode	55
Table 17: New signal assignment netX 51/52 for SDRAM mode	58
Table 18: New signal assignment netX 51/52 for MII mode	60
Table 19: New signal assignment netX 51/52 for MII mode	62
Table 20: Signal assignment netX 50 pin-compatible mode	64
Table 21: Pinout functions combinations and exclusions	68
Table 22: Pin functions of parallel DPM functions (pDPM)	70
Table 23: IO functions for parallel DPM	72
Table 24: Pin functions of serial DPM	73
Table 25: IO functions for serial DPM	73
Table 26: Pin functions of HIF extension bus and HIF SDRAM functions	74
Table 27: IO functions of HIF extension bus and HIF SDRAM	76
Table 28: Pin functions of HIF MMIOs	77
Table 29: MMIO core functions	80
Table 30: IO functions for MMIOs	80
Table 31: Ethernet functions of HIF (HIF-ETH)	81
Table 32: IO functions for HIF-ETH	83
Table 33: PIO functions	86
Table 34: Pin functions of parallel DPM	93
Table 35: IRQ table, netX 51/52	108
Table 36: Address and data byte steering	143
Table 37: Extension bus interface signals	144
Table 38: DMA controllers data packing or unpacking depending on the programmed mode	160
Table 39: IO-Link Baud rates	163
Table 40: netX 51/52 – SPI0 and SPI1	182
Table 41: Data placement example of a 6-byte transfer, MSB first	192
Table 42: Available MII feature levels and signals	203
Table 43: MII pin positions	203
Table 44: MMIO min./max. timing values	222
Table 45: Max. power and current values apply at max. voltage conditions	223
Table 46: Failure rate (FIT)	280
Table 47: netX 51 – Signals B ... HIF (Pin table sorted by signals)	281
Table 48: netX 51 – Signals HIF_D (Pin table sorted by signals)	282
Table 49: netX 51 – Signals HIF_D ... JT (Pin table sorted by signals)	283
Table 50: netX 51 – Signals MEM_A (Pin table sorted by signals)	284
Table 51: netX 51 – Signals MEM_D (Pin table sorted by signals)	285
Table 52: netX 51 – Signals MEM_D ... MEMSR (Pin table sorted by signals)	286
Table 53: netX 51 – Signals MMIO00 ... MMIO16 (Pin table sorted by signals)	287
Table 54: netX 51 – Signals MMIO17 ... MMIO39 (Pin table sorted by signals)	288
Table 55: netX 51 – Signals OSC ... PHY (Pin table sorted by signals)	289
Table 56: netX 51 – Signals POR ... USB (Pin table sorted by signals)	290
Table 57: netX 51 – Signals VDDC, VDDIO and VSS (Pin table sorted by signals)	290
Table 58: netX 51 – Pin A1 ... A18 (Pin table sorted by pin number)	291
Table 59: netX 51 – Pin B1 ... B18 (Pin table sorted by pin number)	292
Table 60: netX 51 – Pin C1 ... C18 (Pin table sorted by pin number)	293
Table 61: netX 51 – Pin D1 ... D18 (Pin table sorted by pin number)	294
Table 62: netX 51 – Pin E1 ... E18 (Pin table sorted by pin number)	295
Table 63: netX 51 – Pin F1 ... F18 (Pin table sorted by pin number)	296
Table 64: netX 51 – Pin G1 ... G18 (Pin table sorted by pin number)	297
Table 65: netX 51 – Pin H1 ... H18 (Pin table sorted by pin number)	298
Table 66: netX 51 – Pin J1 ... J18 (Pin table sorted by pin number)	299
Table 67: netX 51 – Pin K1 ... K18 (Pin table sorted by pin number)	300

Table 68: netX 51 – Pin L1 ... L18 (Pin table sorted by pin number)	301
Table 69: netX 51 – Pin M1 ... M18 (Pin table sorted by pin number)	302
Table 70: netX 51 – Pin N1 ... N18 (Pin table sorted by pin number)	303
Table 71: netX 51 – Pin P1 ... P18 (Pin table sorted by pin number)	304
Table 72: netX 51 – Pin R1 ... R18 (Pin table sorted by pin number)	305
Table 73: netX 51 – Pin T1 ... T18 (Pin table sorted by pin number)	306
Table 74: netX 51 – Pin U1 ... U18 (Pin table sorted by pin number)	307
Table 75: netX 51 – Pin V1 ... V18 (Pin table sorted by pin number)	308
Table 76: Multiplex matrix signals (netX 51)	312
Table 77: netX 52 – Signals B ... HIF (Pin table sorted by signals)	318
Table 78: netX 52 – Signals HIF_D (Pin table sorted by signals)	319
Table 79: netX 52 – Signals HIF_D ... JT (Pin table sorted by signals)	320
Table 80: netX 52 – Signals MMIO (Pin table sorted by signals)	321
Table 81: netX 52 – Signals OSC ... PHY0 (Pin table sorted by signals)	322
Table 82: netX 52 – Signals PHY1 ... USB (Pin table sorted by signals)	323
Table 83: netX 52 – Signals VDDC, VDDIO and VSS (Pin table sorted by signals)	324
Table 84: netX 52 – Pin A1 ... A18 (Pin table sorted by pin number)	325
Table 85: netX 52 – Pin B1 ... B18 (Pin table sorted by pin number)	326
Table 86: netX 52 – Pin C1 ... C18 (Pin table sorted by pin number)	327
Table 87: netX 52 – Pin D1 ... D18, E1 ... E18 (Pin table sorted by pin number)	328
Table 88: netX 52 – Pin F1 ... F18 (Pin table sorted by pin number)	329
Table 89: netX 52 – Pin G1 ... G18 (Pin table sorted by pin number)	330
Table 90: netX 52 – Pin H1 ... H18 (Pin table sorted by pin number)	331
Table 91: netX 52 – Pin J1 ... J18 (Pin table sorted by pin number)	332
Table 92: netX 52 – Pin K1 ... K18 (Pin table sorted by pin number)	333
Table 93: netX 52 – Pin L1 ... L18 (Pin table sorted by pin number)	334
Table 94: netX 52 – Pin M1 ... M18 (Pin table sorted by pin number)	335
Table 95: netX 52 – Pin N1 ... N18 (Pin table sorted by pin number)	336
Table 96: netX 52 – Pin P1 ... P18, R1 ... R18 (Pin table sorted by pin number)	337
Table 97: netX 52 – Pin T1 ... T18 (Pin table sorted by pin number)	338
Table 98: netX 52 – Pin U1 ... U18 (Pin table sorted by pin number)	339
Table 99: netX 52 – Pin V1 ... V18 (Pin table sorted by pin number)	340
Table 100: Multiplex matrix signals (netX 52)	344
Table 101: Mechanical dimensions of netX 52	345
Table 102: Terms, abbreviations and definitions	356

7.5 List of figures

Figure 1: Block diagram of netX 51/52	11
Figure 2: RDY/RUN circuit without security memory	21
Figure 3: RDY/RUN circuit with security memory	22
Figure 4: Block diagram of the reset controller	27
Figure 5: Internal structure of the watchdog logic	29
Figure 6: Timing diagram of WDGACT	29
Figure 7: netX 51 mode	30
Figure 8: netX 51 in netX 50 pin compatibility mode	30
Figure 9: Chip mode selection netX 51	30
Figure 10: netX 52 chip mode	30
Figure 11: External memory controller of netX 51/52	31
Figure 12: External memory interface controller high address lines when non-MMIO 4 bit SPI/QSPI is used (e.g. XiP)	33
Figure 13: 8-bit standard SRAM interconnection	34
Figure 14: 16-bit standard SRAM interconnection	34
Figure 15: 32-bit standard SRAM interconnection	34
Figure 16: Interface with 16-bit Flash up to 32 MBytes	36
Figure 17: Interface with two 16-bit Flash up to 64 MBytes	36
Figure 18: Single read (left) and single write (right) access	37
Figure 19: Sequential read pre and post pause not disabled (compatible with netX 50, 100, 500, default)	38
Figure 20: Sequential read pre and post pause disabled (recommended for memory devices)	38
Figure 21: Sequential write	38
Figure 22: Insertion of pre and post-pause at write and data direction change when 0 is configured	39
Figure 23: Insertion of pre and post-pause at chip-select change if 0 is configured	39
Figure 24: Pre and post-pause at chip-select or data direction change if 0 is not configured	40
Figure 25: APM read burst	40
Figure 26: Read access with external wait state generation	41
Figure 27: Read access with external wait state generation	41
Figure 28: 16-bit SDRAM connection of netX 51	44
Figure 29: 32-bit standard SDRAM connection of netX 51	44
Figure 30: Read access from different SDRAM banks	46

Figure 31: Read access from the same SDRAM banks	46
Figure 32: ARM data read burst runaway without caches (avoided by netX MI)	48
Figure 33: netX ARM cached read access	48
Figure 34: netX 51/52 Host interface IO structure (HIF-IO)	51
Figure 35: netX 51/52 DPM interface block diagram	89
Figure 36: DPM interface structure diagram	91
Figure 37: Word and byte addressing for 16-bit data width (read)	97
Figure 38: Word and byte addressing for 32-bit data width (read)	97
Figure 39: Ready signal transitions (non-multiplexed SRAM mode read example)	99
Figure 40: Non-multiplexed mode read access and ready generation (SRAM mode example)	100
Figure 41: Multiplexed mode read access and ready generation (SRAM mode example)	100
Figure 42: Non-multiplexed mode write access ready generation (SRAM mode example)	101
Figure 43: Multiplexed mode write access ready generation (SRAM mode example)	101
Figure 44: Read data setup time and read address setup time (SRAM read)	103
Figure 45: Hazards and spikes which will be suppressed when signal filtering is enabled	104
Figure 46: IRQ signal driving modes	106
Figure 47: DPM IRQ generation	107
Figure 48: Example: Address incrementing 16-bit read accesses (read data latching)	113
Figure 49: Example: Address incrementing 8-bit APM read burst (read data latching)	113
Figure 50: Address transformation example	118
Figure 51: Examples of external address windowing	120
Figure 52: DPM address mapping (example with low configuration window 0)	121
Figure 53: SPI SPO=0 and SPH=0 transfer	131
Figure 54: SPI SPO=0 and SPH=1 transfer	131
Figure 55: SPI SPO=1 and SPH=0 transfer	132
Figure 56: SPI SPO=1 and SPH=1 transfer	132
Figure 57: Serial DPM protocol: straight stream, type 0	134
Figure 58: Serial DPM protocol: dummy cycle stream, byte sequence, type 1	134
Figure 59: Serial DPM protocol: dummy cycle stream, Dword sequence, type 1	135
Figure 60: Serial DPM protocol: ready-polling stream, byte sequence, type 2	136
Figure 61: Serial DPM protocol: ready-polling stream, Dword sequence, type 2	136
Figure 62: Serial mode 0 QSPI frame format example	137
Figure 63: QSPI Serial DPM protocol: straight stream, type 0	138
Figure 64: QSPI serial DPM protocol: status-read, type 3, address 0	139
Figure 65: Extension bus timing without wait states	145
Figure 66: Extension bus timing with wait states	146
Figure 67: Handshake cell DWORD	148
Figure 68: Handshake cell dataflow (16-bit)	149
Figure 69: Handshake cell monitoring of INTRAMHS	150
Figure 70: System time calculation	153
Figure 71: Ongoing correction of time failure	153
Figure 72: Block diagram of the vectored interrupt controller	156
Figure 73: DMA controller	159
Figure 74: Internal structure of the DMA controller interrupts	161
Figure 75: Multiplex matrix	162
Figure 76: IO-Link: Block diagram of a single port	163
Figure 77: netX 51/52 with 8 IO-Link ports	164
Figure 78: netX 51/52 IO-Link master interfacing	164
Figure 79: Block diagram of the UART	166
Figure 80: Typical application scenario with external connection to a USB host	169
Figure 81: USB transaction examples	172
Figure 82: ACK-token receive timeout	175
Figure 83: Resistor switching back when SE0 is detected	176
Figure 84: Resistor switching back after 7 bit-times J-state	176
Figure 85: I2C module block diagram	178
Figure 86: I2C signal condition states	179
Figure 87: I2C 1 byte transfer	179
Figure 88: I2C write transfer	180
Figure 89: I2C read transfer	180
Figure 90: I2C transfer with direction change	180
Figure 91: 10-bit addressing	181
Figure 92: I2C initializing a read transfer from a 10-bit address slave	181
Figure 93: Standard external SPI interconnection	184
Figure 94: Extended external SPI interconnection with DMUX	184
Figure 95: Quad SPI and XiP/SQIROM interconnection	184
Figure 96: Module organization: QSPI peripheral mode and SQIROM/XiP mode	185
Figure 97: SPI SPO=0 and SPH=0 transfer (mode 0)	187
Figure 98: SPI SPO=0 and SPH=0 transfer (mode 0) without chip-select toggling	187
Figure 99: SPI SPO=0 and SPH=1 transfer (mode 1)	187

Figure 100: SPI SPO=1 and SPH=0 transfer (mode 2).....	188
Figure 101: SPI SPO=1 and SPH=0 transfer (mode 2) without chip-select toggling	188
Figure 102: SPI SPO=1 and SPH=1 transfer (mode3).....	188
Figure 103: QSPI mode 0 and 3 transfer example	189
Figure 104: Transfer bit order: MSB or LSB	191
Figure 105: Transfer byte order: Little or big endianness, MSB first	191
Figure 106: Data on wire for data placement example of a 6 byte transfer, MSB first.....	192
Figure 107: SQUIROM header starting with command byte (bit 'addr_before_cmd' not set)	194
Figure 108: SQUIROM header starting with address bytes (bit 'addr_before_cmd' set).....	194
Figure 109: SQUIROM data organization	194
Figure 110: Block diagram of SPI module	197
Figure 111: SPI SPO=0 and SPH=0 transfer (mode 0).....	198
Figure 112: SPI SPO=0 and SPH=1 transfer (mode 1).....	199
Figure 113: SPI SPO=1 and SPH=0 transfer (mode 2).....	199
Figure 114: SPI SPO=1 and SPH=1 transfer (mode 3).....	199
Figure 115: Standard external SPI interconnection	200
Figure 116: Extended external SPI interconnection with DMUX	200
Figure 117: MII Connection in MAC and PHY mode	202
Figure 118: GPIO structure	205
Figure 119: Timer function diagram	206
Figure 120: Power-on reset sampling of PIO 32–85 pins	208
Figure 121: Input sampling of PIO 32–85 pins	208
Figure 122: Block diagram of the special real-time Ethernet features	211
Figure 123: Power-up and down sequencing	214
Figure 124: Reset input signal timing requirements	220
Figure 125: MMIO level change	221
Figure 126: MMIO output enable.....	221
Figure 127: MMIO output disable	221
Figure 128: MMIO input sampling	221
Figure 129: SDRAM power-up and mode register initialization	225
Figure 130: SDRAM auto refresh generation	226
Figure 131: SDRAM self refresh mode entry and exit	227
Figure 132: SDRAM write timing (MEM_SDCKE is always high).....	227
Figure 133: SDRAM read timing (MEM_SDCKE is always high)	228
Figure 134: Read, pre and post-pauses enabled for sequential reads (write enable always high).....	232
Figure 135: Read, pre and post-pauses disabled for sequential reads (write enable always high)	232
Figure 136: APM read burst, pre and post-pauses disabled between sequential reads.....	232
Figure 137: Write (read enable always high).....	233
Figure 138: Detailed timing of DPM SRAM mode read access DPM_RDn controlled	238
Figure 139: Detailed timing of DPM SRAM mode read access DPM_CSn controlled.....	239
Figure 140: Detailed timing of DPM read burst access (DPM_A or DPM_DQM change without DPM_CSn or DPM_RDn toggling).....	239
Figure 141: Detailed timing of DPM SRAM mode write access when internal netX DPM side is idle (no insertion of wait cycles)	240
Figure 142: Detailed timing of DPM SRAM mode write access when internal netX DPM side busy (insertion of wait cycles)	241
Figure 143: Detailed timing of multiplexed SRAM mode read access	242
Figure 144: Detailed timing of multiplexed SRAM mode read access with address phase overlapping into read-active phase.....	243
Figure 145: Detailed timing of multiplexed SRAM mode write access when internal netX DPM side is idle (no insertion of wait cycles).....	244
Figure 146: Detailed timing of multiplexed SRAM mode write access when internal netX DPM side busy (insertion of wait cycles)	245
Figure 147: Detailed timing of multiplexed SRAM mode write access with address phase overlapping into write-active phase.....	246
Figure 148: Detailed timing of an Intel-like byte enable controlled read access	247
Figure 149: Detailed timing of an Intel-like chip-select controlled read access with read byte enables.....	248
Figure 150: Detailed timing of an Intel-like read access not using read byte enables	249
Figure 151: Detailed timing of an Intel-like write access with write byte enables	250
Figure 152: Detailed timing of an Intel-like multiplexed mode read access.....	251
Figure 153: Detailed timing of an Intel-like multiplexed mode read access with address phase overlapping with read-active phase	252
Figure 154: Detailed timing of an Intel-like multiplexed mode read access without byte enables	253
Figure 155: Detailed timing of an Intel-like multiplexed mode write access when internal netX DPM side busy (insertion of wait cycles).....	254
Figure 156: Detailed timing of a Motorola-like byte enable controlled read access.....	255
Figure 157: Detailed timing of a Motorola-like chip-select controlled read access	256
Figure 158: Detailed timing of a Motorola-like write access with write byte enables	257
Figure 159: Detailed timing of a Motorola-like multiplexed mode read access.....	258

Figure 160: Detailed timing of a Motorola-like multiplexed mode read access with address phase overlapping with read-active phase	259
Figure 161: Detailed timing of an Intel-like multiplexed mode write access when internal netX DPM side busy (insertion of wait cycles)	260
Figure 162: DPM SPI slave timing SPO=0 and SPH=0 transfer (data valid on positive edge of serial clock)	266
Figure 163: DPM SPI slave timing SPO=0 and SPH=1 transfer (data valid on negative edge of serial clock)	267
Figure 164: DPM SPI slave timing SPO=1 and SPH=0 transfer (data valid on positive edge of serial clock)	267
Figure 165: DPM SPI slave timing SPO=1 and SPH=1 transfer (data valid on negative edge of serial clock)	267
Figure 166: SPI timing SPO=0 and SPH=0 transfer (data generation on falling edge, sampling on rising edge of SPI_CLK)	269
Figure 167: SPI timing SPO=0 and SPH=1 transfer (data generation on rising edge, sampling on falling edge of SPI_CLK)	269
Figure 168: SPI timing SPO=1 and SPH=0 transfer (data generation on rising edge, sampling on falling edge of SPI_CLK)	269
Figure 169: SPI timing SPO=1 and SPH=1 transfer (data generation on falling edge, sampling on rising edge of SPI_CLK)	270
Figure 170: SPI master signal timing (SPO=0 and SPH=0)	272
Figure 171: SPI master signal timing (SPO=0 and SPH=1)	272
Figure 172: SPI master signal timing (SPO=1 and SPH=0)	273
Figure 173: SPI master signal timing (SPO=1 and SPH=1)	273
Figure 174: SPI slave signal timing (SPO=0 and SPH=0)	275
Figure 175: SPI slave signal timing (SPO=0 and SPH=1)	275
Figure 176: SPI slave signal timing (SPO=1 and SPH=0)	276
Figure 177: SPI slave signal timing (SPO=1 and SPH=1)	276
Figure 178: Timing on I2C bus	278
Figure 179: UART	279
Figure 180: Mechanical dimensions of netX 51	313
Figure 181: Symbols of thermal characterization	314
Figure 182: Infrared reflow soldering (surface temperature)	315
Figure 183: Vapor phase reflow soldering (surface temperature)	316
Figure 184: Mechanical dimensions of netX 52	345
Figure 185: Symbols of thermal characterization	346
Figure 186: Schematic view of netX PAD types	349
Figure 187: Fixed SDRAM interconnection to MEM-eMI (now netX 50-compatible)	351
Figure 188: SRAM/FLASH interconnection to MEM-eMI without SDRAM (still netX 50-compatible)	351
Figure 189: netX 51/52 SRAM/FLASH interconnection to MEM-eMI when used together with SDRAM (not netX 50-compatible)	352
Figure 190: netX 50 SRAM/FLASH interconnection to MEM-eMI when used together with SDRAM (differs from netX 51/52)	352

7.6 Contacts

Headquarters

Germany

Hilscher Gesellschaft für
Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com

Support

Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

China

Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-Mail: info@hilscher.cn

Support

Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

France

Hilscher France S.a.r.l.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr

Support

Phone: +33 (0) 4 72 37 98 40
E-Mail: fr.support@hilscher.com

India

Hilscher India Pvt. Ltd.
Pune, Delhi, Mumbai
Phone: +91 8888 750 777
E-Mail: info@hilscher.in

Italy

Hilscher Italia S.r.l.
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-Mail: info@hilscher.it

Support

Phone: +39 02 25007068
E-Mail: it.support@hilscher.com

Japan

Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-Mail: info@hilscher.jp

Support

Phone: +81 (0) 3-5362-0521
E-Mail: jp.support@hilscher.com

Korea

Hilscher Korea Inc.
Seongnam, Gyeonggi, 463-400
Phone: +82 (0) 31-789-3715
E-Mail: info@hilscher.kr

Switzerland

Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch

Support

Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@hilscher.com

USA

Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-Mail: info@hilscher.us

Support

Phone: +1 630-505-5301
E-Mail: us.support@hilscher.com