MXT3010

Reference Manual

Version 4.1

Order Number: 100108-05

CONTENTS

CHAPTER 3   *The Cell Scheduling System   27*

CHAPTER 4   *The Fast Memory Interface   47*

CHAPTER 11   *Branch Instructions 261*

Section 3   *Signal Descriptions and Electrical Characteristics 341*

CHAPTER 17   *Timing   343*

CHAPTER 18   *Pin Information   367*

CHAPTER 19   *Electrical Parameters   383*

CHAPTER 20   *Mechanical and Thermal Information   395*

APPENDIX A   Acronyms 399

APPENDIX B   Device Initialization 401

APPENDIX C   Quick Reference 411

List of Figures

List of Tables

Preface

Maker Products

Integrated
Circuits

Maker Communications delivers a wide range of ATM solutions based on the **MXT3010** cell processing engine and the **MXT3020** circuit interface coprocessor. The MXT3010 is a high-performance programmable cell processor engine specifically designed to handle ATM cell manipulation and transmission at data rates up to 622 Mb/s. The MXT3020 is an ATM circuit interface coprocessor for the MXT3010 cell processor. It provides flexible interworking between Time Division Multiplexed (TDM) links and the ATM network.

Software
Solutions

The MXT3010 and MXT3020 are complemented with a series of software applications that provide standard cell processing functionality. **CellMaker®-155** and **CellMaker®-622** execute on an MXT3010 and provide ATM Adaptation Layer 5 (AAL5) Segmentation and Reassembly (SAR) at data rates of 155 Mb/s and 622 Mb/s, respectively. **AccessMaker™** executes on an MXT3010 with up to four attached MXT3020 coprocessors. It

provides cell processing functions for both packet and circuit interworking to support multiple services concurrently including AAL1, AAL5, IMA, and cell relay.

Development Tools

Maker Communications offers a full suite of development tools for the MXT3010 Cell Processor including Verilog models of the chips, the **WASM** assembler, CellMaker Simulator (**CSIM**), and Graphical CellMaker Simulator (**GCSIM**). CSIM is a Verilog-based simulator that provides a tightly controlled and fully observable environment to execute and debug both processor applications and external host programs before running them on the target hardware. Maker also provides two development boards. CSIM is complemented with a graphical post processor, GCSIM. The **MXT3016** is a 32-bit, PCI bus-based development board used to test 622Mb/s applications. The **MXT3025** is a 32-bit, PCI bus-based evaluation board used to test OC-3 ATM (MXT3010) and T1 (MXT3020) applications.

Using this manual

This section provides information on the conventions used within this manual.

Typographical conventions

This document uses the following typographical conventions when describing features of the hardware and software, user-machine interactions, and variables.

- Commands appear in mixed case, for example Write_Channel_Map.
- Instruction mnemonics appear in uppercase, for example the SUBBI instruction.
- User input appears in **bold monospace font**.
- System output and code examples appear in `monospace font`.
- Variables, such as user-definable names, appear in *italics*.

Instruction syntax

All the instructions use the following syntax:

- Required values appear between (parentheses).
- Optional values appear between [square brackets].
- Optional descriptions appear in lowercase.
- Literal descriptions appear in UPPERCASE.
- Numbers are denoted by pound signs, #.
- A string of options from which you can only choose one appear as follows: [option1 | option2 | option3]
- A string of options from which you can choose one or all the options appear as follows: [option1] [option2] [option3]
- Bits which should be written as zeroes and ignored on reads appear as *Reserved*

Terminology   Common acronyms and abbreviations are defined in "Acro-
                     nyms" on page 399 and not in the text. In addition, this manual
                     uses the following term as defined:

                     Packets refer to Local Area Network (LAN) information and
                     frames refer to circuit information.

Contacting Maker Support Services

Maker Communications, Inc. has the following forums for com-
municating ideas, questions, and reporting problems:

- Sales and customer support 508-628-0622

- Product support   support@maker.com

- Product inquires   info@maker.com

- Facsimile   508-628-0256

- Web   www.maker.com

Changes Installed in This Version of the Manual

Change Bars

Change bars are provided to indicate revisions made since the previous publication of the manual.

Changes

1. Additional text has been added to "Register access rules" on page 22, and to the paragraph before that, concerning the use of LD and LDD between accesses to rla registers. Cross references to this warning have been added to "Avoiding stale rla values" on page 315, to "LD Load Register" on page 321, to "LDD Load Double Register" on page 322, and to all hardware register descriptions in CHAPTER 9 "Registers" on page 189.

2. Figure 95, "FN and half-speed RX_CLK/TX_CLK," on page 348 and Figure 96, "FN and quarter-speed RX_CLK/TX_CLK," on page 348 have been added to show the relationship of UTOPIA clocks to FN.

3. Figure 22, "Receive cell organization: 52-byte and 56-byte cells," on page 63 has been modified to correctly identify User Header bytes 2 and 3 in the 56-byte cell format.

4. The description of "LIMD Load Immediate" on page 243 has been corrected to indicate that the immediate is loaded into register *rd*, not register *rsa*.

5. Table 47, "Abbreviations used in SWAN instructions," on page 188 has been modified to generalize the definition of *usi*.

6. The caption of Figure 89 on page 316 has been corrected to indicate that it applies to XOR rather than OR.

7. A typographic error ("3020" vs "3010") in the description of out-of-bag floor life in "MXT3010EP mechanical/thermal information" on page 396 has been corrected.

8. The note that explains the enabling/disabling of "R54-R55 Programmable Interval Timer registers" on page 211 has been changed.

Section 1 Subsystems

This section is composed of eight chapters. It provides an overview of the MXT3010 ATM cell processing engine and its major functional subsystems.

CHAPTER 1 *Introduction*

The MXT3010 is Maker Communication's innovative, programmable ATM cell processing engine. The MXT3010 is built around Maker Communication's SWAN processor and specifically designed for use in high-speed ATM cell-processing applications. The MXT3010 delivers throughput at hard-wired speeds while maintaining all of the benefits of programmable approaches.

MXT3010 features

MXT3010-based systems are insulated against changes in ATM standards because firmware modifications can accommodate these changes. The MXT3010 can:

- Scale across both performance and application ranges.

- Run at speeds ranging from 1.5 Mb/s up to 622 Mb/s.

- Handle the ATM Forum's Traffic Management 4.0 Available Bit Rate (ABR) service specification.

- Operate as a self-contained device managing concurrent Constant Bit Rate (CBR), Variable Bit Rate (VBR), and ABR connections, which frees host processing resources for other tasks.

- Support rate-based and Quantum Flow Control-based ABR services with algorithmic implementation of traffic shaping.

- Perform in ATM layer processing applications.

The MXT3010 has a high speed glueless interface to Fast Memory (SRAM) for storage of instructions and control structures, two high-performance data interfaces, and a UTOPIA Level 2 compliant interface.

The MXT3010 device, packaged in a 240-pin plastic quad flat package, is available in three speed grades, 100 MHz, 80 MHz and 66 MHz. Full electrical and mechanical details are provided in Section 3 of this manual.

Figure 1 shows the MXT3010's internal subsystems and their relationship to devices found in a typical ATM application.

FIGURE 1.MXT3010 and surrounding system devices
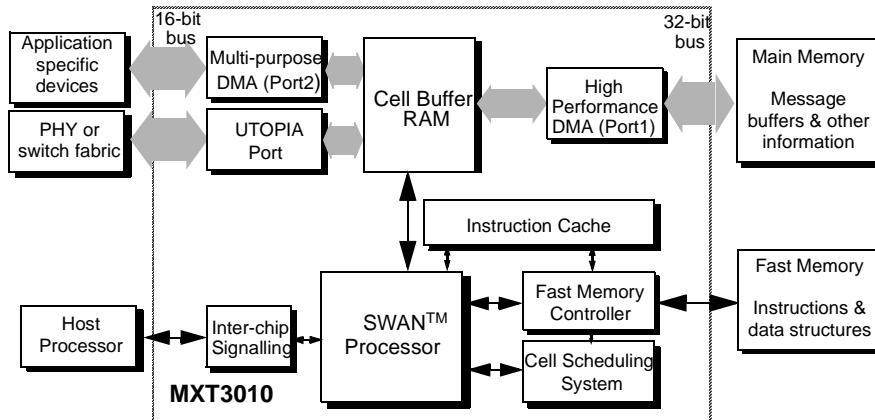
MXT3010 subsystems

While the SWAN processor is the heart of the MXT3010, the device also uses a series of subsystems or hardware agents created to handle ATM-specific tasks. Not only do these subsystems off-load many time-critical functions from the SWAN processor, but they also operate simultaneously with the SWAN processor and with each other, achieving a high degree of parallelism. The subsystems include:

- The Cell Scheduling System (CSS), a hardware-based traffic-shaping subsystem that allows concurrent shaping of dissimilar traffic types.

- The Fast Memory port that provides low latency access to external Channel Descriptors, program code, traffic shaping memory, and the look up tables used for Available Cell Rate calculations.

- The Cell Buffer RAM that buffers cells in both the transmit and receive directions.

- The UTOPIA port that provides connection to an ATM network via a UTOPIA Level 2 Multi-PHY interface.

- The Port1 and Port2 interfaces: Port1 is a high performance 32-bit DMA host system interface and Port2 is a general purpose 16-bit DMA interface.

How the subsystems work together

The Cell Scheduling System, the Fast Memory port, the Cell Buffer RAM, and the port interfaces utilize "dispatched" instructions that operate outside of the CPU such that the SWAN processor does not stall while the instruction is being executed. Not only do dispatched instructions not interfere with the SWAN, but those associated with different subsystems do not interfere with each other, thus permitting simultaneous operation of several dispatched instructions within independent subsystems.

Although the Cell Scheduling System relies on the SWAN processor for direction on required traffic patterns, the CSS manages the traffic-shaping functions of the ATM task. This CSS function provides all of the benefits of algorithmic traffic shaping without decreasing overall performance.

What information is in this manual

This reference manual includes three sections: "Subsystems", "Register and Instruction Reference," and "Signal Descriptions and Electrical Characteristics." Also included are Appendix A "Acronyms," Appendix B "Device Initialization," and Appendix C "Quick Reference."

The "Subsystems" section includes information on:

- The SWAN processor

- The Cell Scheduling System

- The Fast Memory port

- The Cell Buffer RAM

- The UTOPIA port

- The Port1 and Port2 interfaces

- Interchip communications

The "Register and Instruction Reference" section describes the software and hardware registers within the SWAN processor, and includes bit assignments and functions for all of the hardware registers. The "Register and Instruction Reference" section also describes instructions in functional groups and provides an alphabetical list of instructions within each group.

The "Signal Descriptions and Electrical Characteristics" section includes information on:

- Timing information

- Pin out and pin listing

- Signal descriptions

- Electrical parameters

- PLL details

- Thermal characteristics

- Mechanical information

CHAPTER 2   *The SWAN Procesor*
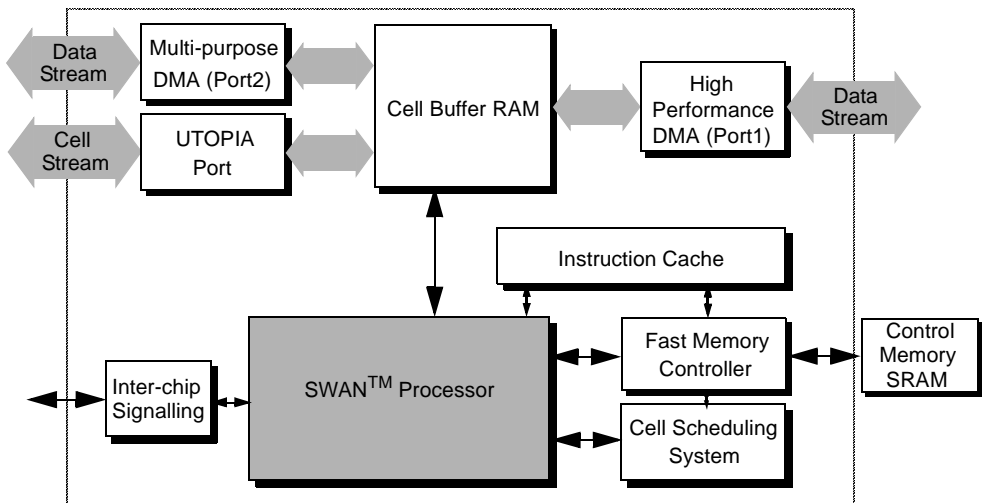
The SWAN processor is used in network protocol processing applications. This chapter describes how the SWAN processor functions and provides functional descriptions of Arithmetic Logic Unit (ALU) and Branch instructions of the SWAN processor.

The SWAN advantage

The SWAN processor was designed using Reduced Instruction Set Computer (RISC) and Complex Instruction Set Computer (CISC) design techniques. By combining the high pipeline speeds of a RISC processor with the instruction set power of a CISC processor, the SWAN processor attains the level of performance required to process a 622 Mb/s ATM cell stream.

SWAN's instructions and address spaces

In addition to utilizing an advanced RISC/CISC design, the SWAN processor employs highly efficient instructions and address spaces optimized for ATM applications.

Instruction features

- The ALU instructions include a memory update feature that can write the results of an ALU operation back into a memory location linked to the destination register.

- The ALU instructions include an integral branching capability that can perform a branch within the ALU instruction cycle if the results of the ALU operation meet selected criteria.

- The ALU instructions can perform modulo arithmetic operations, selectable from 1 bit to 16 bits (full ALU width).

- The Branch instructions can test the status of more than a dozen internal hardware points and two external pins.

- Branch instructions and ALU branching facilities can be programmed to eliminate the performance penalties normally exacted by branch failures in pipeline architectures.

- The Cell Scheduling System provides a powerful set of cell scheduling instructions.

- DMA operations are dispatched with a single instruction, and those for Port1 include flexible CRC capabilities.

- Load and Store instructions include indexing and byte-swapping capability.

Address spaces

The architecture of the SWAN processor, a big-endian design, provides several independent address spaces. The processor accesses each space with instructions specifically designed for optimal performance. Figure 2 shows these address spaces and the instructions which access them. The circled numbers in the figure correspond to the explanatory paragraphs which follow.

FIGURE 2. **SWAN processor address spaces and access instructions**
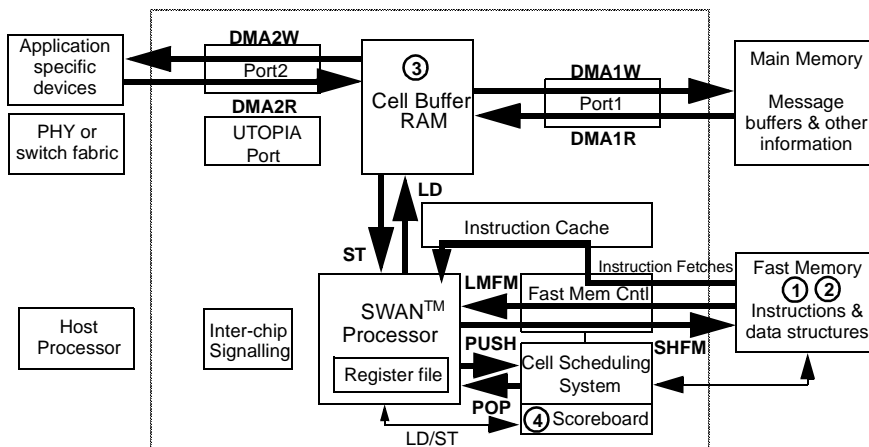
1. Instruction Space - 128K Words

   The SWAN processor executes instructions stored in Fast Memory. Fast Memory instructions are prefetched and optionally cached in a direct mapped on-chip cache to accelerate execution. A 17-bit Program Counter (allowing up to 128K instructions) identifies the current instruction.

The processor executes instructions in a four stage instruction pipeline. The four stages -- Fetch, Decode, Execute and Store -- utilize scoreboarding and feedback to ensure proper operation, minimize stalls, and safeguard against illegal instruction sequences. The Decode stage of the pipeline is the current Program Counter value.

2. Control Memory Space - 1MByte (includes instruction space)

Fast Memory also provides a low latency store for control structures such as descriptors for the applications objects (VC descriptors, packet descriptors). The SWAN register set is tightly coupled to this control memory space through special purpose instructions -- Load Multiple from Fast Memory (LMFM) and Store Halfword to Fast Memory (SHFM). See "Load and Store Fast Memory Instructions" on page 293.

A powerful extension to ALU operations, linking, dynamically associates Fast Memory with the register set. These instructions virtually eliminate the context switching overhead that limits the performance of off-the-shelf processors in ATM systems. See "Automatic memory updates" on page 228.

3. On-Chip Cell Buffer RAM - 1Kbytes

The Cell Buffer RAM on the MXT3010 provides the SWAN processor with low latency access to cells in the ATM data flow and to control information from the host. A flexible Load/Store instruction paradigm provides an efficient memory-register manipulation mechanism. In addition to byte swapping, the extended load/store operations include an indexing method to facilitate control structure parsing. See "Load and Store Internal RAM Instructions" on page 313. This multi-port RAM is accessible to the UTOPIA, Port1 and Port2 DMA engine as well as the

SWAN. Since it is truly multi-ported, it provides very low latency access to all arbiters. See "Direct Memory Access Instructions" on page 283.

4. On-Chip Cell Scheduling System Scoreboard RAM - 2Kbytes

The Cell Scheduling System uses an on-chip RAM to accelerate cell scheduling operations. When not used by the CSS, this RAM is accessible to the SWAN processor through the Load/Store instructions and may be used as general purpose memory. See "The Cell Scheduling System" on page 27.

Instruction execution

All SWAN instructions, except *dispatched instructions,* execute in a single clock cycle. Dispatched instructions include Load Multiple Fast Memory (LMFM), the cell scheduling instructions (PUSHC, POPC), the DMA instructions (DMA1, DMA2), and the load and store double instructions (LDD, STD). Dispatched instructions require more than one cycle to complete, but their execution occurs outside of the CPU such that the processor can accomplish other tasks while dispatched instructions execute.

Since the input clock is doubled in frequency by an on-chip PLL, the SWAN processor executes instructions at twice the frequency of the input clock. Like other high performance RISC processors, the SWAN utilizes a multi-stage pipeline. Delayed branching techniques ensure that Branch instructions also operate at an effective rate of one instruction per cycle by preventing pipeline delays.

Instruction space organization

The SWAN supports an instruction space of 128K 32-bit instructions, which must be 4-byte aligned. The instruction space spans 32 Segments of 4K instructions each. Figure 3 shows the SWAN instruction space.

FIGURE 3. SWAN instruction space
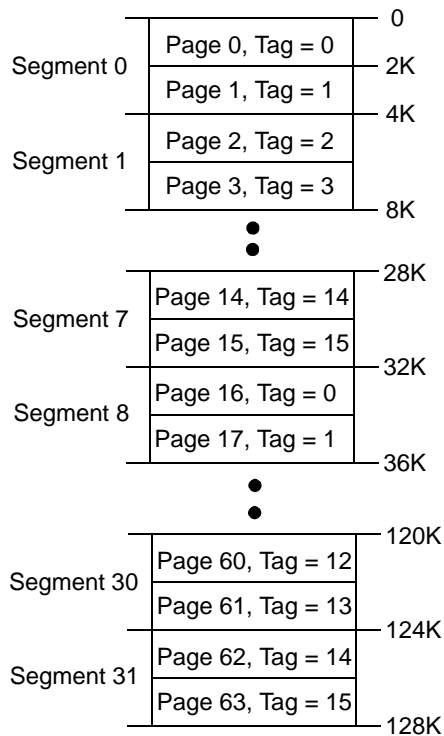
Notes: 1. The tag numbers wrap every 32K instructions
2. Page size is defined by the instruction cache size. Therefore, the MXT3010 EP has sixty-four 2K pages.

Segments are defined by the branching range of the instruction set. Since the Branch instruction has a 12 bit instruction address range, it may jump anywhere within a 4K segment. See "Target field" on page 20.

Instruction cache

The internal Instruction Cache is 2048 instructions. The cache is a direct-mapped cache, with each 32-bit entry having an independent 4-bit tag. There are no separate *valid* bits for the cache entries. At device initialization time, all of the cache tags are written to 0xF. After the micro-boot routine downloads the firmware, the SWAN processor jumps to the specified starting address. The address must not map onto a cache tag of 0xF, as these fetches would cause incorrect cache hits. For simplicity's sake, consider the code space of 32K instructions as an executable space of 30K instructions and with a top 2K of instructions inaccessible for execution.

Cache organization and mapping

The line size of the MXT3010 cache (i.e. the amount of cache replaced on a cache miss) is 1 instruction. Each entry in the cache is therefore a single instruction. Each entry or instruction in the cache is 'tagged' with a 4 bit value that represents the cache page. As shown in Figure 3 on page 14, each 4K instruction segment contains two 2K cache pages.

The NC (No-Cache) bit in the Instruction Base Address register (R53) disables the cache. If this bit is set (one), the SWAN fetches all instructions from Fast Memory, and these instructions are not stored in the on-chip cache. Since the Fast Memory interface runs at 1/2 of the processor speed, it delivers an instruction every other cycle. Therefore, while running out of Fast Memory, the SWAN will stall, at a minimum, every other cycle.

While NC is clear (zero), the cache is enabled. When the SWAN fetches an instruction, the tag of the cache entry at the page offset of the instruction is compared with the tag of the instruction address. Figure 4 details the formation of the page offset and the instruction tag.

FIGURE 4. Formation of the page offset and the instruction tag
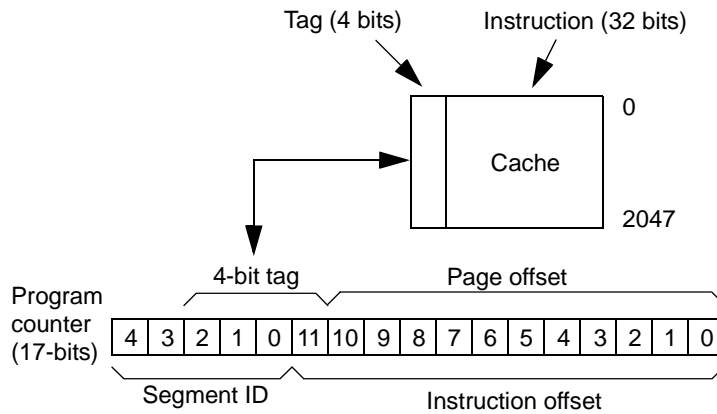
Note: The Instruction Offset is a word offset, as opposed to a byte offset. The byte instruction address in Fast Memory will be ((Segment_ID << 14)+ (Instruction Offset << 2))

If the instruction tag matches the corresponding cache tag, a cache hit has been achieved and the cache returns the instruction within a single cycle. The processor continues execution without stalling. However, if the tag does not match, a cache miss has occurred and the instruction must be fetched from Fast Memory. This will cause a processor stall as it awaits the instruction. Once Fast Memory returns the instruction, it is stored in the cache and the tag is updated. Because the cache line size is a single instruction, only a single instruction is replaced in the cache on a cache miss. Subsequent cache misses may replace other instructions in the cache. With an empty cache, such as when exiting the bootstrap, every instruction must be fetched from Fast Memory. Therefore, every other cycle will be a stall as the cache is cold filled.

The firmware designer controls which segments are cacheable. The NC bit in the Instruction Base Address register (R53) controls the cache and is typically modified by firmware when a code path jumps off the current segment. The firmware must ensure that for each cache tag value (0x0-0xE), only a single

cache page is made cacheable. Otherwise, stale cache entries prevent proper operation. The SWAN's bootstrap program pre-loads a tag of 0xF into all cache entries at initialization. It is recommended that no cacheable code be placed at a location with a tag of 0xF.

Using the Cache

Code that is always executed, referred to as the 'fast path', should be placed in cacheable space, preferably within a single cache page. Infrequently executed code (slow path) and performance insensitive code (for example, initialization code) should be located in non-cacheable segments. Maker's development tools provide code location features.

Many applications do not require more than 2K instructions. In this case, the application may be located on a single cache page. The entire page will be mapped into cache. Obviously, this will provide an optimal level of performance. However, it is not a requirement, as a program can easily jump to a new segment using the following instruction sequence:

```
LIMD R53 new-segment
BI offset_in_new_segment n
```

Instruction prefetch

The SWAN architecture is highly pipelined. The hardware may prefetch instructions from Fast Memory in anticipation of execution. These prefetches may be cached. However, changes in program flow (branches) may prevent the instructions from being executed. This behavior is expected and does not cause improper operation. Prefetches are mentioned here to alert the user that fetches from Fast Memory do not correlate exactly to the sequence of the Program Counter.

Observing cached program flow

When the processor is executing out of cache, it does not need to access Fast Memory. However, if Fast Memory is not being used, the MXT3010 presents the program counter address on the Fast Memory address lines. This helps to monitor code execution from cache.

SWAN processor instruction classes

The SWAN processor includes powerful 32-bit instructions in six functional areas or classes. Descriptions of each class of instruction are divided into two sections — one which describes the subsystem that uses that instruction and one which describes the bit utilization and format for each instruction. These descriptions appear in the chapters listed in Table 1.

TABLE 1.   SWAN processor instruction classes

Functional Area	*Subsystem Description*	*Instruction Description*
Arithmetic Logic Unit Instructions	"The SWAN Processor" (this chapter)	"Arithmetic Logic Unit Instructions" on page 223
Branch Instructions	"The SWAN Processor" (this chapter)	"Branch Instructions" on page 261
Cell Scheduling Instructions	"The Cell Scheduling System" on page 27	"Cell Scheduling Instructions" on page 277
Direct Memory Access Instructions	"The Port1 and Port2 Interfaces" on page 97	"Direct Memory Access Instructions" on page 283
Load and Store Internal RAM Instructions	"The Cell Buffer RAM" on page 59	"Load and Store Internal RAM Instructions" on page 313
Load and Store Fast Memory Instructions	"The Fast Memory Interface" on page 47	"Load and Store Fast Memory Instructions" on page 293

Arithmetic Logic Unit (ALU) instructions

Basic ALU
instructions

The SWAN processor instruction set includes a complete suite of arithmetic, logical, and shifting instructions implemented in a high performance ALU. The format of a typical ALU instruction is shown below:

ADD (rsa, rsb) rd [MODx][abc][AE][UM]

In the example shown, input data is stored in rsa and rsb, while the result is delivered to register rd. The notations shown in square brackets represent the special features that optimize the SWAN ALU for ATM cell processing. These features, referred to as instruction field options (IFOs), include the modulo field (MODx), the ALU branch condition field (abc), the always execute bit (AE), and the update memory feature (UM). For more information see "Arithmetic Logic Unit Instructions" on page 223.

Branch instructions

The SWAN processor includes two basic branch control mechanisms:

- A suite of ALU instructions that includes conditional branching capabilities. See "Arithmetic Logic Unit Instructions" on page 223.

- A suite of three basic branch instructions, each of which is available with a return address linking version.

Basic Branch
instructions

The format of a typical Branch instruction (Branch Fast Memory) is shown below:

BF [ESS#/(0|1)[/C]][cso][N]

Branch instructions allow the programmer to specify conditional branching decisions which will alter the instruction execution sequence. The branching decisions are based on the state of the MXT3010 subsystems, as indicated in the External State Signals (ESS) register. The point to be tested is specified by the ESS field (ESS#). If the branch is to be taken when the point tested is a 1, the ESS# is followed by a /1. If the branch is to be taken when the point tested is a 0, the ESS# is followed by a /0. Branch instructions can also be used to manipulate the UTOPIA port's control counters via the counter system operation (cso) field. The C and N options optimize the performance of Branch instructions in special circumstances. Descriptions of these options appear in "The Conditional operator (C-bit)" on page 265. Complete information on Branch instructions appears in "Branch Instructions" on page 261.

Target address
The branch *target address* is the address at which execution continues if the specified branch condition is satisfied. The full branch target address within Fast Memory is formed from the Segment ID in the Instruction Base Address register (R53) and the branch target field. Figure 5 shows the format of the target address.

FIGURE 5. Target address format in Fast Memory
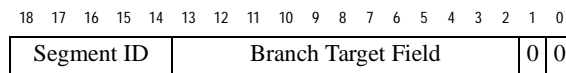
18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Segment ID				Branch Target Field													0	0

Target field
The branch *target field* is a 12-bit field that specifies the absolute word address within the current code segment (4096 words) at which execution is to continue. The three basic branch instructions differ only in their method of specifying the branch target address field. Table 2 summarizes the methods used.

TABLE 2.   Methods of specifying the branch target field

Instruction	*Method of specifying the branch target field*
Branch Immediate (BI)	As bits [11:0] of the instruction
Branch Fast Memory Shadow Register (BF)	As bits [11:0] of the Fast Memory Shadow register (R58). (Note 1)
Branch Register (BR)	As bits [11:0] of the Branch register (R59)

Note 1:The Fast Memory shadow register is loaded with the first halfword returned from memory during a Fast Memory read operation that specifies the LNK Instruction Field Option.

For a complete description of the three basic branch instructions and the versions which include return address linking, see "Branch Instructions" on page 261.

Registers

Register types

The SWAN processor contains 64 software-visible registers of two types, general-purpose and control/status. The general-purpose registers are classified as *software registers* because their usage and content is firmware dependent. The registers that control functions and provide status information are classified as *hardware registers*.

The SWAN processor has 32 general-purpose software registers, R0-R31, each 16-bits wide. The SWAN also has 32 control and status hardware registers, R32-R63.

Pipeline feedback

The SWAN processor includes two pipeline feedback features. One of the feedback paths takes results from the execution stage of the instruction pipeline and delivers those results to the decode stage. A second feedback path takes results from the storage stage of the pipeline and also delivers those results to the decode stage. Figure 6 shows the general concept:

FIGURE 6. Pipeline feedback
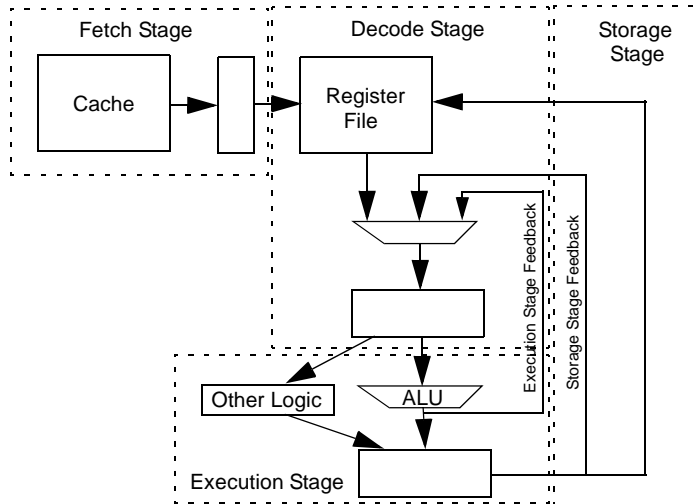
Using the execution stage feedback facility, an ALU instruction that modifies a register can be followed immediately by another ALU instruction that accesses that same register. Using the storage stage feedback facility, other instructions that modify a register can be followed, after an intervening instruction, by an instruction that accesses the same register. This intervening instruction must not be an LD or LDD to a hardware register.

Register access rules

Do not perform a load (LD, LDD) to a hardware register immediately between an instruction that accesses an rla register (R48-R51, GA-GD) and an instruction that stores to that rla register.

The number of processor cycles which must intervene between an instruction that alters a register and an instruction which uses the data in the altered register depends upon two factors:

- The instruction used

   When a POPC is issued, the destination register, rd, does not contain the requested data until eight cycles after the POPC instruction is decoded.

When a Load (LD) instruction is issued, the destination register, rd, does not contain the requested data until one cycle after the LD instruction is decoded.

When a Load Double (LDD) instruction is issued, the second destination register, rd + 1, does not contain the requested data until two cycles after the LDD instruction is decoded.

When a Load Multiple Fast Memory (LMFM) instruction is issued, the destination registers are updated after delays described in "LMFM Load Multiple from Fast Memory" on page 308.

• The register accessed

A write to a software register, R0-R31, can be immediately followed by an instruction that uses the data in that register.

Writes to the following hardware registers should be followed by at least one other instruction before the new information in the register is used for load, store, or branch instructions. This restriction does not apply to their use in ALU or DMA instructions.:

TABLE 3.   Hardware registers requiring one instruction delay

Location	*Name*	*Read/Write*
R44-R47	CRCX/CRCY (when used a general purpose registers)	R/W
R48	rla Address register	R/W
R49	rla Address register	R/W
R50	rla Address register	R/W
R51	rla Address register	R/W
R57-write	Sparse Event/ICS register	Set/Clear
R58	Fast Memory Shadow register	R/W
R59	Branch register	R/W

Writes to the following hardware registers should be followed by at least two other instructions before the new information in the register is used:

TABLE 4.   Hardware registers requiring two instruction delays

Location	Name	Read/Write
R42-write	Mode Configuration register	Set/Clear
R43-read	Fast Memory Bit Swap register	R
R60	CSS Configuration register	R/W
R62	UTOPIA Configuration register	R/W
R63	System register	R/W

Flag registers

Flag registers include the Assigned Cell Flag register and the Overflow Flag register. These registers are internal state bits; programs do not manipulate them directly, but can use the status of the flags to modify program flow.

Assigned Cell flag register

The Cell Scheduling System manipulates this register at the conclusion of a POPC operation. The state of the Scoreboard bit targeted by the POPC operation is copied into this register, which is connected to ESS4 and can be tested by ALU Conditional Branch instructions.

Overflow flag register

Add and Subtract instructions that cause arithmetic overflow set this register. ALU Conditional Branch instructions test this register.

For more information

For a complete description of the registers within the SWAN processor, please see "Registers" on page 189.

HEC generation and check circuit

The MXT3010 provides two HEC generation and checking methods:

1. HEC generation and checking is provided in the UTOPIA port. See "Receive cell flow" on page 77.

2. For applications which do not use the UTOPIA port, HEC generation and checking is provided in the SWAN processor.

SWAN HEC operation

Bit 9 of the Mode Configuration Register (R42) enables HEC generation mode in the SWAN processor and changes the definition of General Purpose register R33. In normal operation, R33 is a read/write register and is initialized to 0xFFFF. In HEC-enabled mode, R33 is redefined to include the output from the HEC generation circuitry and is therefore no longer available as a simple read/write register or as a constant value.

Additionally, the HEC circuitry uses R32 as a source of data for HEC generation. This register is read/write and is initialized to 0x0000. When not used for HEC purposes, R32 can continue to function as a 16-bit read/write register.

The HEC generation logic takes in two 16-bit data values and produces an 8-bit result. In normal ATM cell processing, the first data value would be the first two bytes of the ATM cell header. The second data value would be the second two bytes of the ATM cell header, and the 8-bit result of that second operation would be the HEC inserted (or checked) for the current cell.

The HEC circuitry initializes its 8-bit seed value when new data is written to R32. A subsequent write to R33 completes the input of data to the HEC circuit. After appropriate pipeline delays, the resultant HEC is available right justified in R33. The following code segment illustrates this.

```
LIMD r32 #first_two_bytes    ;load first half of cell
                              header

                             ;this also resets the
                              SEED value

LIMD r33 #second_two_bytes   ;load second half of
                              cell header

NOP                          ;execute stage

NOP                          ;store stage

NOP                          ;HEC processing

CMP r33, x                   ;HEC is returned in low
                              byte R33
```

The HEC result can be used directly in a transmitted cell, or compared to the fifth byte in a received cell. The NOP instructions can be replaced with useful operations, but the HEC result in R33 is not valid until the fourth instruction after data is written to R33.

CHAPTER 3 *The Cell Scheduling System*
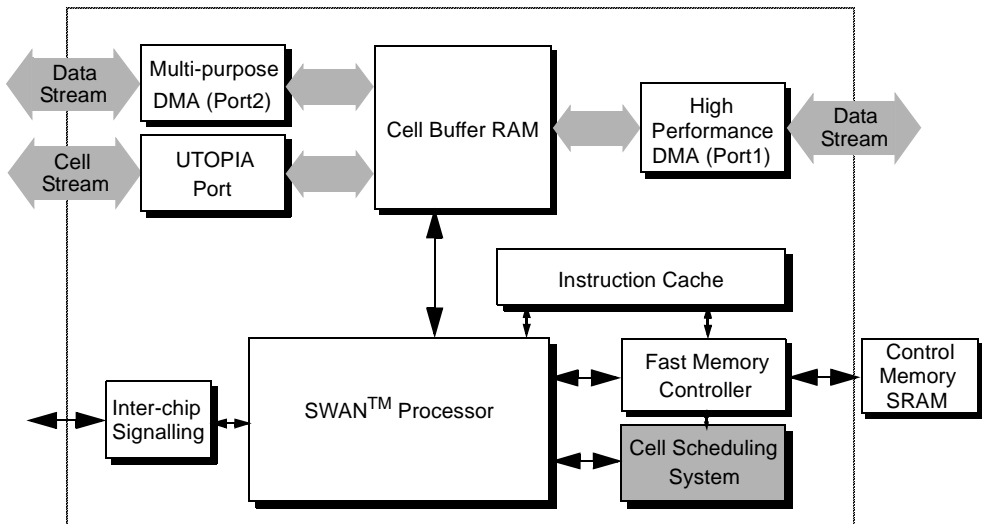
The Cell Scheduling System (CSS) is a traffic-shaping system that operates as a combination of algorithmic- and hardware-assisted functions. The SWAN processor implements the algorithmic-assisted portion of the scheduling function, and the cell scheduler performs the hardware-assisted portion. By implement-

ing traffic shaping as a combination of algorithmic- and hardware-assisted functions, the programmer has complete control over the traffic-shaping algorithms used.

This chapter includes the following information:

- How the Cell Scheduling System works

- Data transmission - servicing and scheduling

- Pacing the transmission rate of cells

- Programming the Cell Scheduling System

How the Cell Scheduling System works

The Cell Scheduling System works by dividing the ATM cell payload capacity of the transmission link into periodic containers of cells. The boundary of the periodic containers relative to the transmission convergence framing structure is arbitrary. To schedule cell usage within the containers, the MXT3010 creates a Scoreboard (schedule) on-chip and a Connection ID table in Fast Memory. The Scoreboard can contain up to eight sections, each of which represents an independent periodic container for a separate physical link or priority level. Each location within a periodic container corresponds to a single bit in the Scoreboard section and a single entry in the corresponding Connection ID table. Bits [13:12] of the Cell Scheduling System (CSS) Configuration register (R60) control the number of sections in the Scoreboard.

TABLE 5.   Scoreboard sectioning control

Bits	Name	Description
13:12	SZ	Scoreboard Section Size
		00 = 2,048 bits/entries per section; up to 8 sections
		01 = 4,096 bits/entries per section; up to 4 sections
		10 = 8,192 bits/entries per section; up to 2 sections
		11 = 16,384 bits/entries per section; 1 section

To clarify the discussion which follows, it will be assumed that the Scoreboard contains only a single section of 16,384 bits/ entries.

The Scoreboard and Connection ID table are maintained by the SWAN processor working with a specialized control circuit referred to as the *cell scheduler*. The cell scheduler modifies the Scoreboard and Connection ID table in response to servicing and cell scheduling requests issued by the SWAN processor.

Successive bits in the Scoreboard and locations in the Connection ID table represent successive cell time slots on a transmission link. If the transmission link is not fully loaded with traffic, only some of the entries in the table have a virtual circuit (VC) assigned to them, as indicated by a bit set to one in the Scoreboard. Others are labeled *Available*, as indicated by a Scoreboard bit that is zero. Figure 7 shows an example of Connection ID table entries.

FIGURE 7. Connection ID entries

Connection ID 145
Available
Connection ID 47

1 0 1

Scoreboard

Connection ID Table

Note

The MXT3010 accommodates Scoreboards of 2,048 through 16,384 bits and Connection ID tables of 2,048 through 16,384 16-bit halfwords.
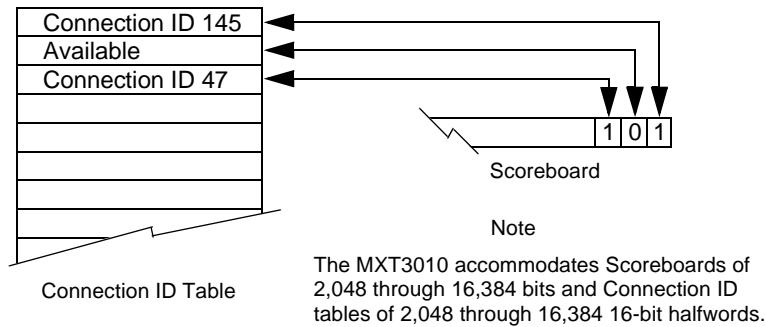
During the cell-scheduling process, status bits in the Scoreboard table summarize the assigned or available status of each Connection ID table entry. Since a bit in the Scoreboard represents the status of a 16-bit entry in the Connection ID table, the Scoreboard is only 1/16th the size of the Connection ID table. This compaction of table status accelerates the cell scheduler task of searching for available time slots. The searching task is further accelerated by a proprietary algorithm that guarantees to identify an available cell-time slot from anywhere within the Scoreboard and write the Connection ID into that slot within 12[1] processor cycles. High-speed searching is especially important for high-speed ATM links and/or those links that carry a large number of VCs, as larger Connection ID tables are used in such systems.

---

1. Under ideal conditions (Fast Memory write pipe empty), this number could be as low as 10, but it is highly likely that a write pipe entry will need to be displaced, raising the number to 12.

Data transmission - servicing and scheduling

The data transmission process consists of two major steps:

1. *Servicing* the Connection ID table to find entries representing assigned time slots that are scheduled for transmission on an established virtual circuit.

2. *Scheduling* time slots for existing virtual circuits or establishing new VCs by placing entries into Connection ID table locations that ensure the proper service quality for that VC.

Servicing

The SWAN processor services the Connection ID table and Scoreboard linearly and services the VCs that have reserved the various locations. The SWAN processor determines which VC reserved a time slot by examining the corresponding Connection ID table entry. The SWAN processor reads the Connection ID table entry by executing the POPC instruction. When POPC executes, the cell scheduler returns the addressed Connection ID table entry, copies the value of the Scoreboard bit corresponding to the entry into the Assigned Cell flag bit of the External State Signals register (R42, bit 4), and clears the Scoreboard bit.

The processor maintains a pointer into the Connection ID table that represents the current cell time slot. In a normal application, the processor increments this pointer each time it issues the POPC instruction. Because the Scoreboard and Connection ID table represent periodic containers, the SWAN processor is responsible for manipulating its Connection ID table pointer modulo the container size. If multiple Scoreboards and Connection ID tables are used, the SWAN is responsible for manipulating multiple Connection ID table pointers, each modulo its respective container size.

The POPC instruction is a dispatched instruction operating outside of the CPU such that the SWAN processor does not stall while the cell scheduler executes the POPC instruction. The SWAN processor can determine when the POPC operation is complete by testing the state of bit 5 in the External States Signals register (R42). ESS5 is set while a cell scheduling operation is in progress. Alternatively, the SWAN processor can determine when the POPC operation is complete by accessing the destination register, although this method can result in a processor stall. Register scoreboarding guarantees that the processor will stall if the processor tries to access the destination register (rd) before the cell scheduler has written the POPC result to that register. However, the instruction immediately following the POPC is not register scoreboarded and should not access register rd.

When a POPC instruction has executed, and the Assigned Cell Flag indicates that the selected time slot had an assigned Connection ID table entry, the program can read the destination register of the POPC instruction to obtain a pointer to the Channel Descriptor for the VC associated with that time slot. The Channel Descriptor contains the application-defined state information needed to process the cell transmission event for the associated VC. This data normally includes the pointer to the data to be transmitted plus rate or flow control information used in scheduling future activity for the VC.

If the Assigned Cell Flag indicates that the selected time slot is unassigned, the program must employ measures to ensure that an appropriate transmission rate is maintained. See "Pacing the transmission rate of cells" on page 37.

Scheduling

The SWAN processor schedules a VC when adding a new connection or when servicing an existing VC. The SWAN processor initiates a scheduling operation by executing a PUSHC instruc-

tion. PUSHC specifies a 16-bit Connection ID and a target location within the periodic container (Scoreboard). The cell scheduler responds to PUSHC by scanning the Scoreboard looking for the first available location at or after the targeted location. If an available location is not found by the time the last bit of the Scoreboard is reached, the cell scheduler loops back to the beginning of the Scoreboard to continue the search. When the cell scheduler finds an available location, it sets the bit in the Scoreboard and writes the Connection ID into the corresponding Connection ID table entry. In general, the Connection ID identifies the Fast Memory address of the Channel Descriptor for the VC.

Like POPC, PUSHC is a dispatched instruction operating outside of the CPU such that the SWAN processor does not stall while the cell scheduler executes the PUSHC instruction. The SWAN processor determines when the PUSHC operation is complete by testing the state of bit 5 in the External Signal Status register (R42). ESS5 is set while a cell scheduling operation is in progress. When the scheduling operation is complete, the processor reads the scheduled address in the Cell Scheduling System Scheduled Address register (R61). This address differs from the target address if the target address was previously scheduled.

Software cannot depend upon the state of register R61 until the PUSH/PUSHF instruction is complete, as no register scoreboarding mechanism protects access to this register during PUSH/PUSHF instruction operation.

For example, Figure 8 shows the SWAN processor servicing the third location in the Connection ID table and scheduling a new time slot for Connection ID 47.

FIGURE 8. Servicing and scheduling

Connection ID Table
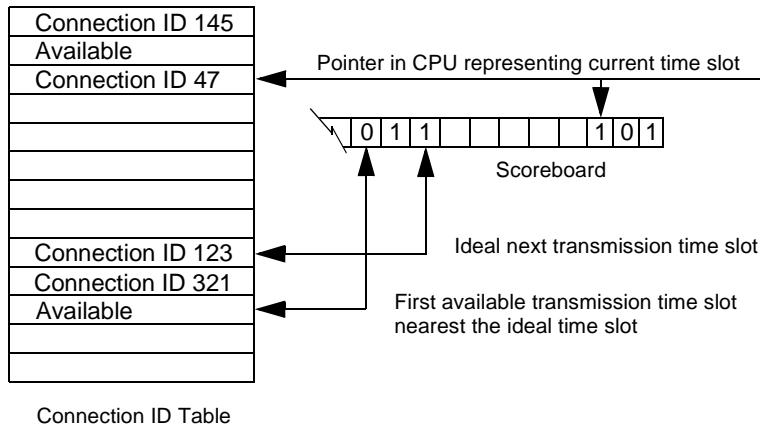
In the example shown in Figure 8, the requested location was six entries away from the entry being serviced. However, that location was assigned, and the nearest available location was eight entries away. The cell scheduler reserves the available location and reports the location to the SWAN processor via the Cell Scheduling System Scheduled Address register (R61). This report-back feature is important when creating controlled delay connections, as it enables the program to determine whether the chosen location meets the cell delay variation (CDV) requirements. If the CDV requirements are not met, the SWAN processor can make another scheduling attempt or otherwise reschedule or reject the connection.

Calculating target time slots

The SWAN processor uses the Channel Descriptor information to calculate, via an algorithm, a target time slot location for the next transmission to serve the VC. A variety of methods can be employed.

Using GCRA to calculate time slots

The scheduling of cells on a per-connection basis is completely implementation dependent. For example, an implementation can use the Generic Cell Rate Algorithm[1] as defined by Increment and Limit ((GCRA(I,L)) to schedule cells on a VC. The Increment represents the minimum *inter-cell emission interval* for the VC.

The scheduling algorithm calculates target time slots for various types of connection as follows:

- For an Available Bit Rate (ABR) connection, the inter-cell emission interval is based on feedback from the network (flow control information in the Channel Descriptor) and is equal to 1/ACR. The implementation can calculate the Increment for ABR connections in accordance with the ATM Forum's rate-based ABR service specification, but other methods can be used.

- For a Variable Bit Rate (VBR) connection, the target time slot calculation can use an algorithm that allows burst transmission of a specified number of cells (Maximum Burst Size) at a peak cell rate (Peak Cell Rate), not to exceed a sustained cell rate (Sustained Cell Rate) over time. In this case, the Increment depends upon the above three parameters.

- For an Unspecified Bit Rate (UBR) connection, the target time slot calculation is based on the information in the Channel Descriptor without regard to flow control, but with no effort at reliable transport.

- For a CBR connection, the algorithm can schedule all the required time slots in the Scoreboard when the connection is initially established. The quantity and spacing of these time slots depends on the bandwidth and Cell Delay Variation (CDV) requirements associated with the connection. Therefore, when a target time slot calculation is created for

---

1. Consult ATM Forum's Traffic Management 4.0 for GCRA information.

an established CBR connection, the target time slot is the current time slot. Maintaining the currently assigned time slots ensures consistent CBR connection performance.

For CBR connections, the inter-cell emission interval is not time varying and is equal to 1/Peak Cell Rate (PCR).

For VCs with dynamically allocated time slots, such as VBR and ABR VCs, a single time slot can exist on the Connection ID Table/Scoreboard for each VC. VCs that use permanent reservation of bandwidth, such as CBR VCs, can have multiple time slots.

All of the information required to calculate the inter-cell emission intervals can be stored in Fast Memory. Inter-cell emission intervals can be stored as fractional integers to support high connection rates. The program can store the inter-cell interval as a fractional integer and maintain a remainder. The SWAN processor can then schedule cells using the integer portion of the result, saving the remainder for use in the next scheduling event on that VC.

The SWAN processor can recover bandwidth lost due to cell scheduling collisions by scheduling connections at the calculated Theoretical Arrival Time minus the Limit. A copy of the scheduled time must be stored in the Channel Descriptor for each VC scheduled in this fashion for proper operation of the GCRA.

Pacing the transmission rate of cells

The MXT3010 can pace the transmission rate of cells in either of two ways:

- Back pressure through the UTOPIA port

- Use of an external clock

Back pressure method

When the back pressure method is used, the Cell Scheduling System is a self-pacing system—no external clock is required. Back pressure from the transmission link through the UTOPIA port limits the rate at which the SWAN processor can queue cells for transmission. Therefore, the processor must maintain a continuously scheduled cell stream at the UTOPIA port. The processor maintains this cell stream by issuing idle or unassigned cells when no active VC is scheduled.

As indicated in "Servicing" on page 31, the SWAN processor determines if a time slot is assigned or unassigned by testing the state of the Assigned Cell flag bit of the ESS register following a POPC instruction. If the Assigned Cell flag bit is 0, the time slot is unassigned and an unassigned cell must be queued to maintain the necessary back pressure. The queuing of unassigned cells guarantees that inter‑cell emission intervals on the transmission link remain synchronous with the intervals programmed into the schedule.

External clock method

When the external clock method is used, the Cell Scheduling System is no longer a self-pacing system, as an external clock[1] is required to indicate cell transmission opportunities. If there are no cells to be sent, no cells are presented to the PHY. Only user data cells are presented to the UTOPIA Port for transmission; no idle cells are sent.

---

1. Either of the Programmable Interval Timers (PIT0 or PIT1) can be used. See "R54-R55 Programmable Interval Timer registers" on page 211.

Advantages of
each method

The back pressure method is preferable when transmitting cells over an ATM transmission link, as the ATM transmission link must be kept full, and the transmission of idle cells is required. The external clock method is preferable when the MXT3010 is connected to a switch fabric, as it saves the switch the overhead of dealing with idle cells.

Programming the Cell Scheduling System

The Cell Scheduling System example in Figure 9 shows the SWAN processor maintaining a pointer that represents the present transmission time slot, such as the service address, in R7. In this example R7= 02, the halfword address of the third location in the Connection ID table.

FIGURE 9. Scoreboard operation

Connection ID Table
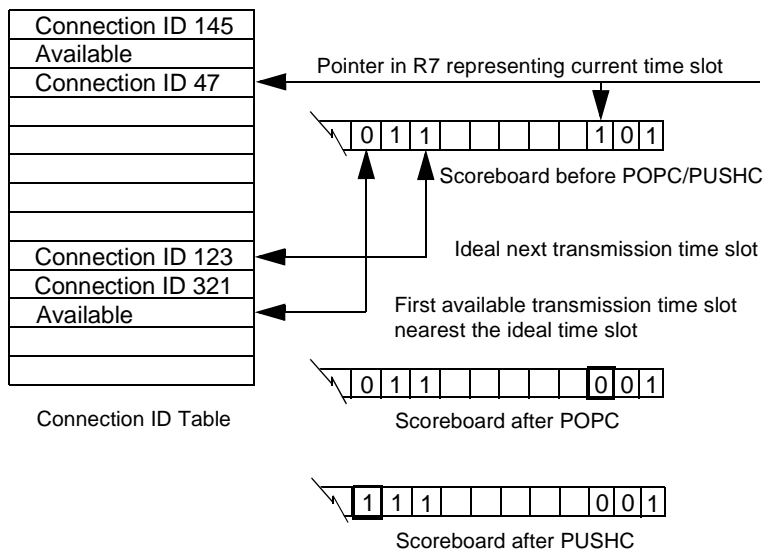
The following instructions represent typical cell scheduling operation:

	`POPC R10 @R7`	If the UTOPIA Port Transmit queue is not full, the SWAN processor executes a POPC requesting that the cell scheduler access the Connection ID table entry that R7 references (location 02), and place that Connection ID into R10. The cell scheduler copies the Scoreboard bit associated with this Connection ID table entry into the Assigned Cell Flag register, then clears the Scoreboard bit (see "Scoreboard after POPC" in Figure 9). Because the relevant Scoreboard bit was set to 1 at the time that POPC was executed, the Assigned Cell Flag register is set to 1.
	`BI $RDY ESS5/0`	The SWAN processor first tests for completion of the POPC instruction (bit 5 in the External Signals State (ESS) register) using a Branch Immediate (BI) instruction. The BI instruction specifies a branch to location $RDY if the point tested (ESS5) is a 0, indicating the CSS scheduling operation is no longer in progress.
$RDY	`BI $SAC ESS4/1`	The SWAN processor then tests the Assigned Cell Flag register (bit 4 in the External Signals State (ESS) register) using a Branch Immediate (BI) instruction. The BI instruction specifies a branch to location $SAC if the point tested (ESS4) is a 1.
$SAC	`LMFM R16 @R10/` `R10 16HW LNK`	Since the time slot was assigned, the SWAN processor uses the connection ID returned in R10 to retrieve the Fast Memory-based Channel Descriptor for the VC that reserved the time slot. The Load Multiple Fast Memory (LMFM) instruction is used to copy 16 halfwords beginning at the Fast Memory Address specified in R10 into 16 SWAN registers starting with register R16. To ensure that any changes to the Fast Memory locations can be automatically copied into the entries stored in R16-R31, the Link (LNK) instruction field option is invoked.

The SWAN processor uses the information stored in the Channel Descriptor to build or retrieve a cell for the VC. In a SAR application that uses dynamic scheduling as part of the service routine, the SWAN processor determines when to service the next connection. The SWAN processor does this by executing a scheduling algorithm using parameters stored in the Channel Descriptor. The Channel Descriptor contains the parameters necessary to determine the connection scheduling rate.

From the information in the Channel Descriptor, the SWAN processor determines the next location within the Connection ID table that should be scheduled for this VC. Then the SWAN processor places the result into a software register, for example R22. The SWAN processor activates the connection by executing the PUSHC instruction. The PUSHC instruction requests that the cell scheduler find an available time slot at or after the target address specified in R22, assign the chosen time slot, and write the Connection ID from register R10 into the Connection ID table location corresponding to that time slot.

```
PUSHC R10 @R22
```

The cell scheduler translates the target address indicated by R22 into a Scoreboard bit position and searches the Scoreboard, beginning at that bit position. In the example shown in Figure 9, the cell scheduler discovers that a previous connection reserved the target location. Therefore, the cell scheduler examines the Scoreboard until it finds an available location. This location is found two cell slots away from the target location. The cell scheduler reserves the location for the present connection by setting the Scoreboard bit to 1 (see "Scoreboard after PUSHC" in Figure 9) and by writing the Connection ID provided by the SWAN processor in R10 into the selected location. When the scheduling operation is complete, the cell scheduler reports the scheduled address in the Cell Scheduling System Scheduled Address register (R61). The SWAN processor can read this register to determine whether the scheduled address meets the CDV requirements for the service being provided.

The SWAN processor completes servicing the connection by incrementing the service address contained in R7, modulo the Connection ID table size. For example, the SWAN processor can use the Add Immediate (ADDI) instruction to add 0x0002 to the address contained in R7 and place the result in R7. If the Connection ID table size is 4096 entries, the ADDI instruction can include 4096 as a modulo value, limiting the incrementation process to the lowest order twelve bits. This limitation causes the incrementation process to cycle through the table locations.

```
ADDI R7 0x0002 R7 MOD4096
```

Guaranteeing the availability of a location in the Connection ID table

If the Scoreboard is full while the cell scheduler is servicing or adding a new connection, the Cell Scheduling System returns an error by setting bit 15 in R60, the Cell Scheduling System Configuration register. Constant checking for this error bit slows down the effective operating rate of the device. Rather than check the error bit setting, use either of these two methods to ensure that a location is available:

1. Add new connections or activate inactive connections only when unassigned slots are encountered.

2. Maintain a count of the active VCs on the scoreboard, being careful to adjust for connections (such as pre-allocated CBR connections) that consume more than one slot in the Scoreboard. Do not admit a new connection that exceeds the capacity of the Scoreboard.

The PUSHC/POPC instruction buffer

The cell scheduler contains a two-deep PUSHC/POPC instruction buffer. The SWAN processor can issue the following cell scheduling instructions without entering a stall condition:

- A PUSHC or PUSHF followed by a PUSHC or PUSHF

- A PUSHC or PUSHF followed by a POPC or POPF

Execution of a cell scheduling instruction while the buffer is full results in a SWAN processor stall until the first operation finishes.

POPC, PUSHC, POPF, and PUSHF instruction operation

POPC and PUSHC timing

The POPC operation completes in eight cycles from the instruction decode to loading of the rd register. The worst case PUSHC time is 12 cycles from the instruction decode to the Fast Memory write acknowledge from the write buffer. If the four-stage write buffer is full at the time of the PUSHC operation, this cycle count increases so that the buffer can be flushed of one entry, and space for the new write information can be provided.

POPF and PUSHF timing

Both the POPF (Pop Fast) and PUSHF (Push Fast) instructions manipulate the internal Scoreboards without accessing the Connection ID table in Fast Memory. By eliminating unnecessary accesses to Fast Memory, memory read/write latencies are avoided.

In POPF, as in POPC, the Cell Scheduling System translates the target address into a Scoreboard bit position. The Cell Scheduling System copies the state of that bit into the Assigned Cell flag (see below), and clears the bit location. However, POPF differs from POPC in that the Cell Scheduling System does not access the Fast Memory and does not provide a Connection ID in the destination register. The POPF operation completes in five cycles from the instruction decode.

In PUSHF, as in PUSHC, the Cell Scheduling System translates the target address into a Scoreboard bit position. The Cell Scheduling System searches for the first available location in the Scoreboard at or after that bit position and sets the bit for that location to reserve it. However, PUSHF differs from PUSHC in that the Cell Scheduling System does not write a new Connection ID into the Connection ID table location corresponding to the reserved Scoreboard bit. Rather, the existing Connection ID at that location is scheduled. The PUSHF operation completes in 12 cycles from the instruction decode. This is the same speed as an optimum PUSHC that experiences no write buffer delays. Unlike the PUSHC instruction, PUSHF will never experience write buffer delays, as it does not perform a Fast Memory write.

When servicing a Scoreboard where time slot assignments rarely vary, a combination of POPF and PUSHF can be used to service and schedule connections without the overhead of Fast Memory access.

Connection ID table and Scoreboard addressing

The Cell Scheduling System Configuration register specifies bits(18:15) of the Connection ID table address. Bits (14:1) of the Connection ID table address are provided by software in bits(13:0) of the rsb register specified by POPC and PUSHC instructions.

FIGURE 10.Connection ID table address generation

18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
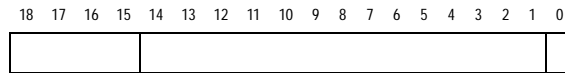
TABLE 6.   Connection ID table address bits

Bits	*Source*
[0]	Fixed as zero (0)
[14:1]	Bits [13:0] of the rsb register in POPC or PUSHC instruction
[18:15]	Bits [11:8] of "R60 The Cell Scheduling System (CSS) Configuration register" on page 217

The Connection ID table entry generates the Scoreboard address corresponding to the specified Connection ID table entry as follows:

FIGURE 11.Scoreboard address generation

18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																		
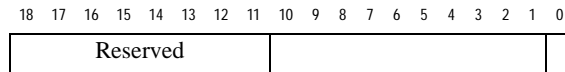
TABLE 7.   Scoreboard address bits

Bits	Source
18:11	Reserved. Write as zeros; ignore on reads
10:1	Bits [13:4] of the rsb register in POPC or PUSHC instruction
0	Fixed as zero (0)

Note:Bits [3:0] of the rsb register in POPC or PUSHC instruction select a target bit within the 16-bit Scoreboard entry. (While the Cell Scheduling System searches the Scoreboard on the basis of 32-bit quantities, the SWAN processor addresses the Scoreboard on a 16-bit basis.)

Initializing the Scoreboard

The SWAN processor clears the Scoreboard during its system initialization routine. The SWAN processor initializes the Scoreboard by executing POPF instructions to all of the locations in the Connection ID table. Once the SWAN processor has cleared the Scoreboard, it can execute cell-scheduling instructions. For those portions of the Scoreboard used for cell scheduling, the program must perform all scheduling changes through the PUSHC and POPC instructions to ensure that the MXT3010's internal mechanisms remain consistent. However, the SWAN processor can read Connection ID table entries at any time with the LMFM instruction, or read the Scoreboard using the LD instruction, without affecting the internal mechanisms.

Selecting a Scoreboard size

The Cell Scheduling System Configuration register includes the desired Scoreboard size, rounded up to the nearest power of two. The SWAN processor can mark certain locations as unavailable to support Scoreboard sizes other than powers of two.

For example, assume the desired Schedule size is 2304 bits. The program can execute a series of PUSHC operations to select a 4096 bit schedule and to mark bits 2304 to 4095 as unavailable. From that point on, the cell scheduler will not try to reserve those locations in response to cell scheduling requests. As a program executes POP instructions to the Scoreboard, it must return to the beginning of the Scoreboard when it reaches location 2303. In other words, once the unwanted locations are reserved, the program must not specify them as the target address of a POP operation. Also, the program must calculate the PUSHC target addresses modulo 2304 instead of modulo 4096.

Supporting multiple Scoreboard sections

As indicated in Table 5, "Scoreboard sectioning control," on page 29, the MXT3010 supports multiple Connection ID tables/Scoreboard sections. The device supports a maximum of:

- Eight 2K Connection ID tables/Scoreboard sections

- Four 4K Connection ID tables/Scoreboard sections

- Two 8K Connection ID tables/Scoreboard sections

- One 16K Connection ID table/Scoreboard section

If eight schedules are used, bits [13:11] of the rsb register in POPC or PUSHC instruction select a schedule within the block of eight. If four schedules are used, rsb bits [13:12] select a schedule within the block of four, and so on.

PUSHC/POPC rsb register address bit(s)	*Select(s) which schedule for*
13	2 x 8K
13:12	4 x 4K
13:11	8 x 2K

The Fast Memory Interface
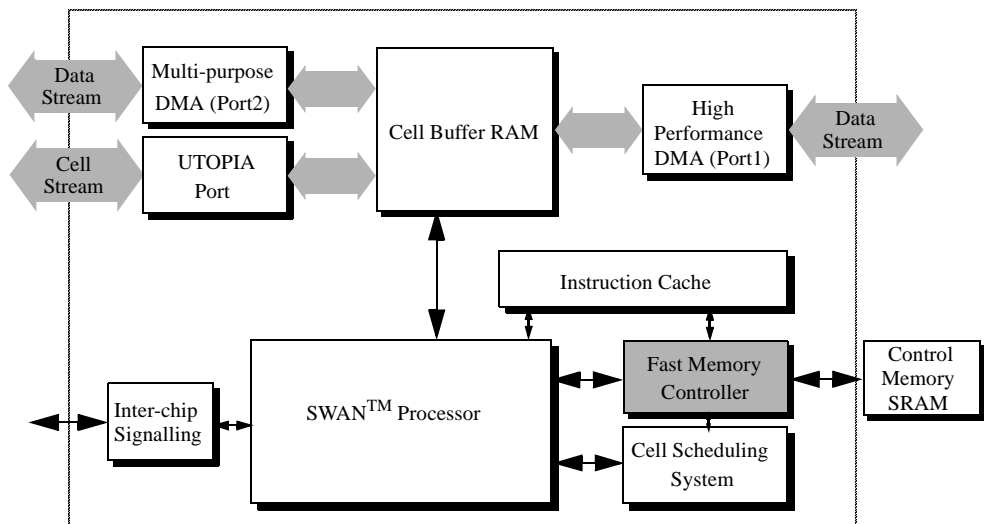
The Fast Memory port provides the SWAN processor and the Cell Scheduling System with low latency access to external Channel Descriptors, program code, traffic shaping memory, and the look up tables used for Available Bit Rate calculations. The Fast Mem-

ory controller provides a glue-less interface to synchronous, flow-through, burst-mode cache RAMs. The Samsung KM718B90 and compatible parts are examples of suitable RAMs.

This chapter describes:

- SWAN processor accesses to Fast Memory

- Cell Scheduling System accesses to Fast Memory

- SWAN executable fetches from Fast memory

- Fast Memory configuration

SWAN processor accesses to Fast Memory

The processor accesses Fast Memory with Load Multiple Fast Memory (LMFM) and Store Halfword (SHFM) instructions. A specialized Fast Memory access and update protocol in the Fast Memory controller accelerates access to and update of Fast Memory-based data structures.

Loading

The software tables and data structures stored in Fast Memory are accessed by the SWAN processor through the LMFM (Load Multiple Fast Memory) instruction. A simplified version of the LMFM instruction is shown below.

FIGURE 12.Load Fast Memory instruction

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Op Code							rd					LNK	00		Z	rsa						#HW					rsb				

LMFM rd @rsa/rsb #HW [LNK]

The SWAN processor uses the #HW field to specify the number of halfwords to be fetched and the rsa and rsb fields to specify the Fast Memory byte address at which the transfer will begin. In response to the LMFM instruction, the Fast Memory interface controller will write the halfwords returned from memory into the SWAN's register file beginning with register rd and continuing with rd+1, rd+2, etc. until the designated number of halfwords have been transferred. Thus, the LMFM instruction allows the SWAN processor to transfer up to 16 halfwords[1] from the Fast Memory into the register file in a single instruction.

Memory update protocol

If the LNK instruction field option is specified, the fast memory interface controller links the loaded registers to the locations in Fast Memory from which their contents were read. ALU instructions which modify these registers can force the modifications to be written back to Fast Memory by specifying the update memory (UM) option. Thus, the UM function allows the SWAN processor to update the data structure in Fast Memory without executing a dedicated Store instruction. In addition, use of the LNK option causes the first halfword read from memory to be read into the Fast Memory Shadow Register (R58), where it can be used by BF/BFL instructions.

Once a linking relationship has been set up by an LMFM instruction, subsequent LMFM instructions do not need to specify a linking option, as the links remain in place. When it is desired that the links be changed, a new LMFM with linking option enabled can change the links. An LMFM used to change links does not have to specify a data transfer (#HW can be zero).

---

1. Since the number of halfwords that can be transferred range from 0 to 16 halfwords, there are 17 possible values for the #HW field. Therefore, the #HW field is 5 bits wide.

Additional information on the LNK option and memory updating, including restrictions, appears in "Linking (the LNK bit)" on page 299 and following pages.

Further information

Further information about the LMFM instruction is provided in "Load and Store Fast Memory Instructions" on page 293. Examples of LMFM instruction usage are provided in that chapter and in "Swan Instruction Reference Examples" on page 325.

Storing

Fast Memory writes can be accomplished utilizing the memory update function described above or by utilizing the Store Halfword to Fast Memory (SHFM) instruction. A simplified version of the SHFM instruction is shown below.

FIGURE 13.Store Fast Memory instruction

31 30 29 28 27 26 25	24 23 22 21 20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
Op Code	000000000	rsa	#HW	rsb

SHFM @rsa/rsb

Execution of the SHFM instruction causes the Fast Memory interface controller to write the halfword contained in the Fast Memory Data register (R56) into the halfword addressed by the byte address contained in registers rsa and rsb. A more powerful store instruction, Store Register Halfword (SRH) is also available. The SRH instruction is especially useful for accelerating CRC operations. See "Cyclical Redundancy Check operation

acceleration" on page 104 and "Instructions for accelerating CRC operations" on page 305.

Further Information	Further information about the SHFM and SRH instructions is provided in "Load and Store Fast Memory Instructions" on page 293. Examples of SHFM and SRH instruction usage are provided in that chapter and in "Swan Instruction Reference Examples" on page 325.

Cell Scheduling System accesses to Fast Memory

The Cell Scheduling System maintains one or more Connection ID tables in Fast Memory. The Cell Scheduling System accesses Fast Memory with PUSHC and POPC instructions issued by the SWAN processor. PUSHC instructions cause a halfword write to a Connection ID table, and POPC instructions cause a halfword read to a Connection ID table.

Cell Scheduling System operations are a lower priority than LMFM burst data reads. If a Cell Scheduling System operation is in progress when a LMFM is issued, the Cell Scheduling System operation finishes but the LMFM is serviced before the next Cell Scheduling System operation proceeds.

SWAN executable fetches from Fast Memory

The SWAN processor fetches all instructions from the Fast Memory using 32-bit word read accesses. These accesses are higher priority than any other access to the Fast Memory. If an LMFM or Cell Scheduling System operation is in progress when the SWAN processor makes a Fast Memory read request, the LMFM or Cell Scheduling System operation finishes but the read request is serviced before the next LMFM or Cell Scheduling System operation proceeds.

Fast Memory configurations

This section describes these configuration features:

- Memory sizes supported

- RAM selection and configuration

- Mode 0 operation for chips with single or multiple Chip Enable inputs

- Mode 1 operation for chips with multiple Chip Enable inputs only, allowing 512K banks

- Bus contention avoidance

Memory sizes supported

The Fast Memory interface supports memory sizes from 128 Kbytes to 1 Mbyte in configurations of one or two banks. In all configurations Fast Memory is 32 bits wide.

FIGURE 14.Fast Memory SRAM options
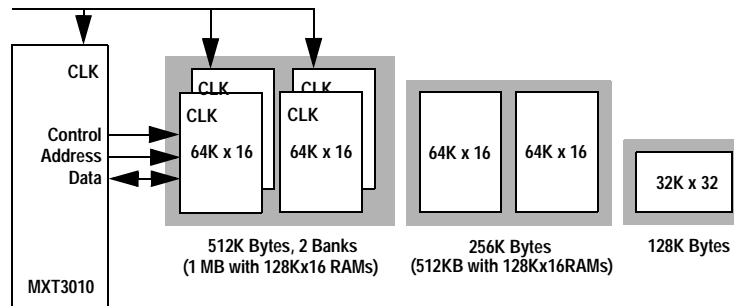
RAM selection and configuration

The MXT3010 supports the following RAM configurations:

Memory Size	*RAM*	*Banks*	*Mode*
128K Bytes 32Kx32	1 – 32Kx32	1	0
256K Bytes 64Kx32	2 – 64Kx16	1	0
512K Bytes 128Kx32	4 – 64Kx16	2	0
512K Bytes 128Kx32	2 – 128Kx16	1	1
1M Byte 256Kx32	4 – 128Kx16	2	1

Mode 0 operation

The MXT3010 provides two operation modes for the Fast Memory. Table 8 compares the two modes, which can be selected by modifying target bit 4 in the Mode Configuration Register (R42):

TABLE 8.   Comparison of Mode 0 and Mode 1 operation

Attribute	*Mode 0*	*Mode 1*
Types of access supported	Byte or halfword	Byte or halfword
Device Chip Enable configurations	Single or multiple	Multiple only
Maximum addressable memory	512 Kbytes	1 Mbyte

In Mode 0 the MXT3010 drives 16 address bits and four independent byte enables to permit direct addressing of 64K 32-bit words in each of two memory banks. Two chip select signals and two output enable signals provide independent bank selection and output drive enable signals for the two memory banks. Address bit FADRS[18] internally generates the select signal for the active bank. Physical memory appears as two contiguous 64Kx32 banks starting in address space at 0x00000, going to 0x7FFFF.

FIGURE 15.Mode 0 design example

Bank 0   0x00000-0x3FFFF   Bank 1   0x40000-0x7FFFF
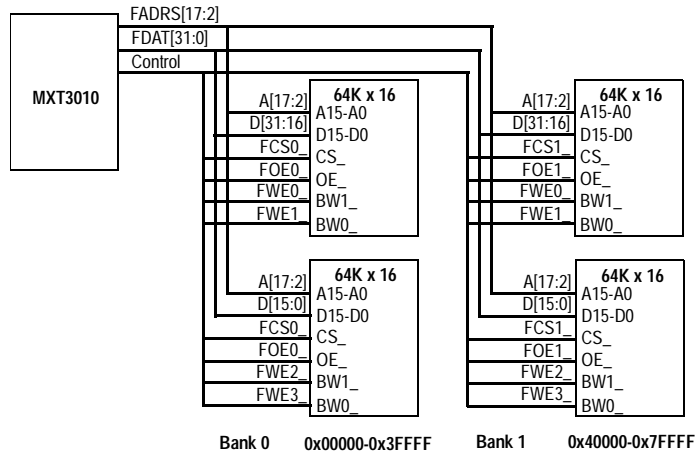
Mode 1 operation

In Mode 1 the MXT3010 drives 18 address bits and four independent byte enables to directly address 256K 32-bit words in each of two memory banks. Two output enable signals provide independent output drive enable signals for the two memory banks. No chip enable signals are used; address bit FADRS[19] selects the active bank. Physical memory appears as two contiguous 128Kx32 banks starting in address space at 0x00000 and ending at 0xFFFFF. One of the two banks is always enabled since no independent chip enable signals are used.

In a single bank configuration, Mode 1 can configure only 512 Kbytes of Fast Memory using 128Kx16 RAMs. In this configuration, physical memory appears as a single 128Kx32 bank in address space from 0x00000 to 0x7FFFF.

The Connection ID table and the executable code space (as set by the Segment ID field in the Instruction Base Address register) can only reside in the first 512K bytes of Fast Memory.

FIGURE 16.Mode 1 design example

0x00000-0x7FFFF   0x80000-0xFFFFF
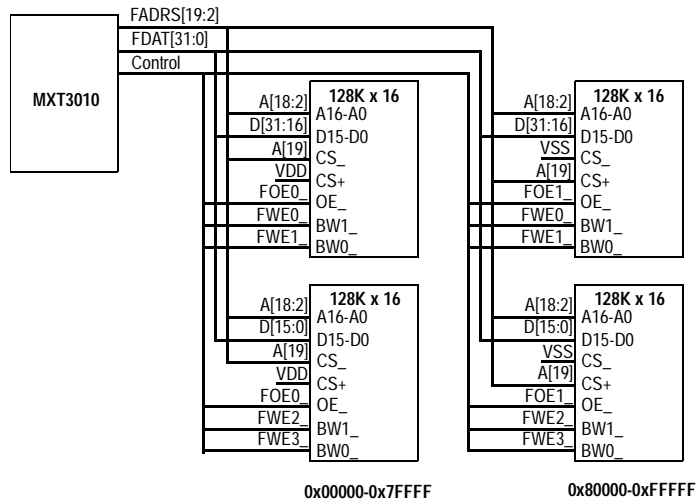
When operating in Mode 1, the Chip Select pins are used as Fast Memory Address lines 19 and 18.
FCS1_ = FADRS[19]. FCS0_ = FADRS[18]

Bus contention avoidance

The timing of the two output enable signals (FOE1_ and FOE0_) is skewed when the addresses of consecutive memory accesses cross bank boundaries to prevent bus contention on back-to-back read cycles. The MXT3010 guarantees a window[1] between disabling one bank and enabling an alternate bank. This allows both banks to be directly wired to the data bus without external buffers or transceivers.

---

1. Please refer to "Timing" in Section 3 for further information.

Fast Memory sequence diagrams

This section shows sequence diagrams for the following Fast Memory operations:

- Read operations, single bank (Figure 17 on page 56)

- Write operations, single bank (Figure 18 on page 57)

- Read and write operations, back-to-back operation and dual bank (Figure 19 on page 57)

Set-up times, propagation times, and other timing information for the Fast Memory interface are provided in "Timing" on page 343.

FIGURE 17.Fast Memory read operations - single bank
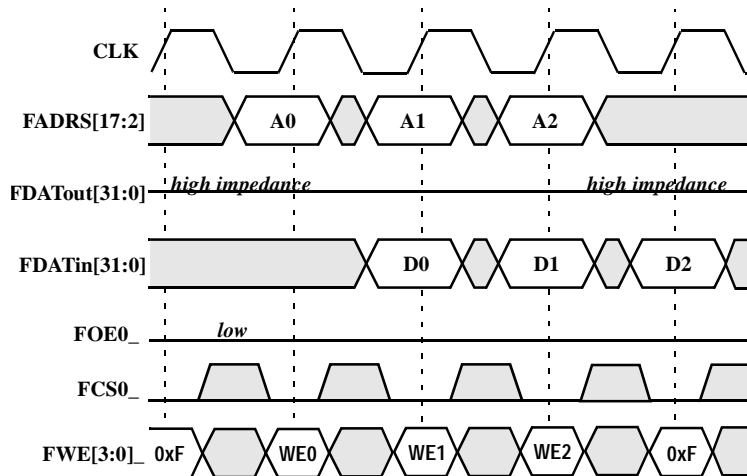
FIGURE 18.Fast Memory write operations - single bank
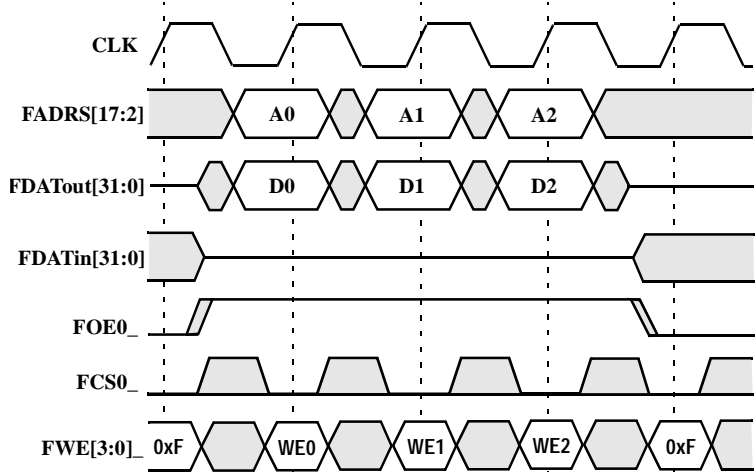
FIGURE 19.Fast Memory reads and writes - back-to-back and dual bank
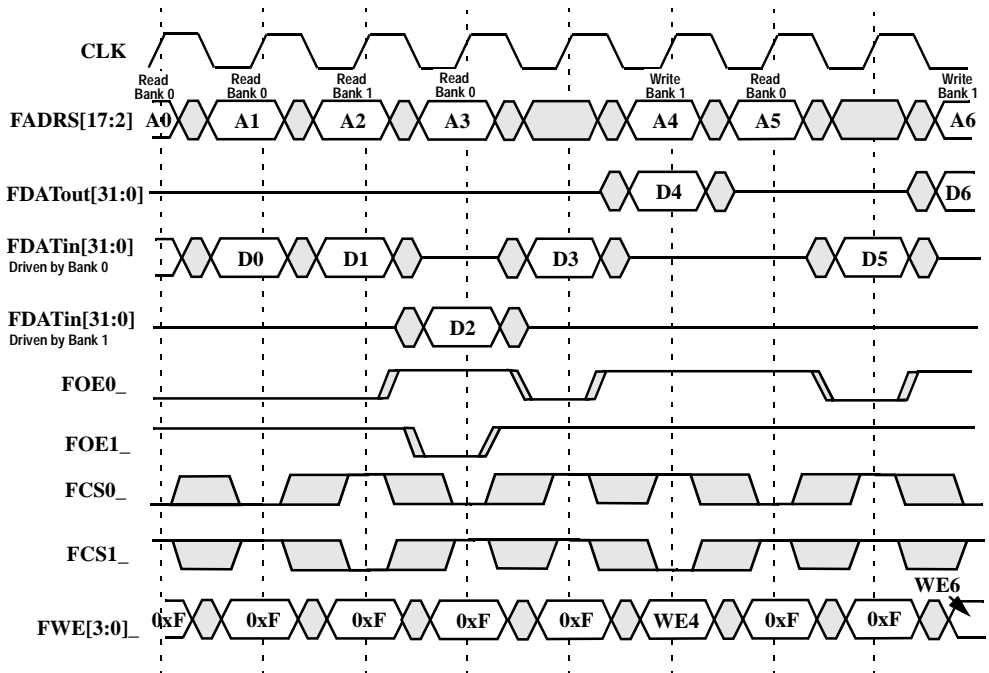
CHAPTER 5 *The Cell Buffer RAM*
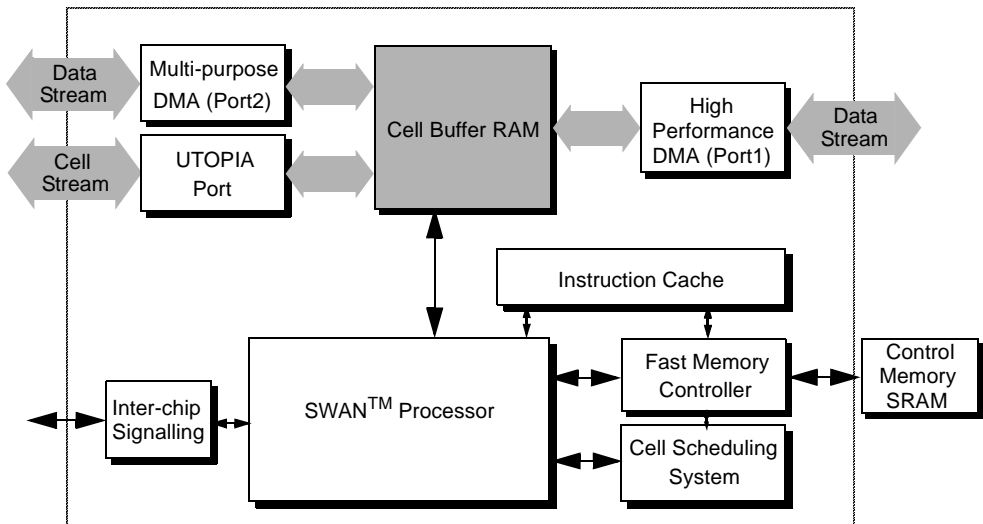
The MXT3010's internal Cell Buffer RAM buffers cells in both
the transmit and receive directions. The CPU and the DMA unit
can access the Cell Buffer RAM through memory access proto-
cols. This chapter describes the Cell Buffer RAM and the memory
access protocols.

Internal cell storage in the Cell Buffer RAM

To store cells, the Cell Buffer RAM is configured into a number of 64 byte blocks referred to as cell holders. During reception, cells are written into cell holders as they are received from the physical layer. During transmission, cells are built in cell holders before being transmitted to the physical layer.

At device initialization, the Cell Buffer RAM is segmented into sections for receive cell storage, transmit cell construction, and general purpose scratch pad use. As shown in Table 9, bits [6:1] of the UTOPIA Configuration register(R62) control the segmentation of the Cell Buffer RAM.

TABLE 9.   UTOPIA Configuration control of the Cell Buffer RAM

Bits	*Description*	
3:1	Receive Cell Buffer Size in the Cell Buffer RAM	
	000	UTOPIA Port Receiver in Reset Mode. All Rx outputs are tristated. This includes RXDATA (a bidirectional signal), but does not include RXCLK. All inputs are pulled to their inactive states by the MXT3010.
	001	Receiver Buffer Size in the Cell Buffer RAM = 2 cells
	010	Receiver Buffer Size in the Cell Buffer RAM = 3 cells

	110	Receiver Buffer Size in the Cell Buffer RAM = 7 cells
	111	Receiver Buffer Size in the Cell Buffer RAM = 8 cells
6:4	Transmit Cell Buffer Size in the Cell Buffer RAM	
	000	UTOPIA Port Transmitter in Reset Mode. All Tx outputs are tristated except TXCLK. All inputs are pulled to their inactive states by the MXT3010.
	001	Transmitter Buffer Size in the Cell Buffer RAM = 2 cells
	010	Transmitter Buffer Size in the Cell Buffer RAM = 3 cells
	----	
	110	Transmitter Buffer Size in the Cell Buffer RAM = 7 cells
	111	Transmitter Buffer Size in the Cell Buffer RAM = 8 cells

The minimum allocation for receive cell holders is two, the maximum is eight, and receiver cell holder addressing begins at location 0x0000. The minimum allocation for transmit cell holders is two, the maximum is eight, and transmit cell holder addressing begins at location 0x0200. As an example, Figure 20 shows a Cell Buffer RAM organization with eight receive cell holders, four transmit cell holders, and the remaining space available as scratch pad space.

FIGURE 20.Cell Buffer RAM organization

Address	Type	Size
0x0000	Rx Cell	64 bytes
0x0040	Rx Cell	64 bytes
0x0080	Rx Cell	64 bytes
0x00C0	Rx Cell	64 bytes
0x0100	Rx Cell	64 bytes
0x0140	Rx Cell	64 bytes
0x0180	Rx Cell	64 bytes
0x01C0	Rx Cell	64 bytes
0x0200	Tx Cell	64 bytes
0x0240	Tx Cell	64 bytes
0x0280	Tx Cell	64 bytes
0x02c0	Tx Cell	64 bytes
0x0300		64 bytes
0x0340		64 bytes
0x0380		64 bytes
0x03C0		64 bytes

Cell fields

Independent of the specific cell format used, certain fields (if provided) occupy certain positions. Figure 21 shows these fields, and Table 10 summarizes their functions.

FIGURE 21.Cell fields defined

4 bytes	User Header	Present in proprietary 56-byte cells only
4 bytes	ATM Header	Present in all cells
1 byte	HEC	Optionally present in cells
48 bytes	SAR PDU	Present in all cells
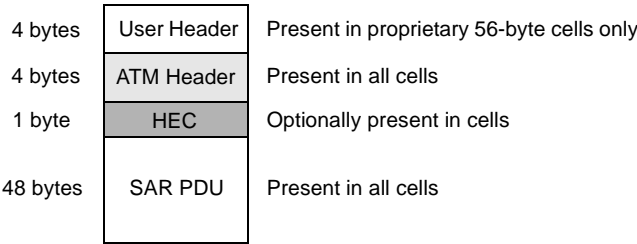
TABLE 10.   Cell field functions

Field	*Function*
User Header	The User Header is a four-byte field that can be inserted before the ATM header, adding four bytes to the front of a cell.
ATM Header	The ATM Header is a four-byte field specified by relevant ATM standards and consists of GFC, VPI, VCI, and PTI sub-fields. It is generally present in all but a few proprietary schemes. The VPI and VCI sub-fields are interpreted by UTOPIA Receive Header Reduction hardware in the MXT3010 to form the Channel Identifier for the cell. See "Receive Header Reduction hardware" on page 91.
HEC	The Header Error Control (HEC) is a one-byte CRC accumulated across the ATM Header. The MXT3010 can be configured to transmit and receive cells with or without HEC.
SAR PDU	The SAR PDU is a 48-byte field that is present in every cell.

Cell formats

The format of the information in the cell holders is a function of the selection of 52-byte or 56-byte cell operation via bit 1 of the "R42-write Mode Configuration register" on page 201.

Bit	*Bit State and Function*	
1	Cell Length Control	
	0	52 byte cells
	1	56 byte cells

Figure 22 compares the 52-byte and 56-byte cell formats.

FIGURE 22.Receive cell organization: 52-byte and 56-byte cells

	52-byte cell	56-byte cell
0x0000	Unused	User Header bytes 0, 1
0x0002	Receive Cell Status Word	User Header bytes 2, 3
0x0004	ATM Header bytes 0, 1	ATM Header bytes 0, 1
0x0006	ATM Header bytes 2, 3	ATM Header bytes 2, 3
0x0008	SAR PDU bytes 0, 1	SAR PDU bytes 0, 1
0x000A	SAR PDU bytes 2, 3	SAR PDU bytes 2, 3
0x000C	SAR PDU bytes 4, 5	SAR PDU bytes 4, 5
0x000E	SAR PDU bytes 6, 7	SAR PDU bytes 6, 7
0x0010	SAR PDU bytes 7, 8	SAR PDU bytes 7, 8
• • •		
0x0034	SAR PDU bytes 44, 45	SAR PDU bytes 44, 45
0x0036	SAR PDU bytes 46, 47	SAR PDU bytes 46, 47
0x0038	Unused	Receive Cell Status Word
0x003A	Unused	Unused
0x003C	Unused	Unused
0x003E	Unused	Unused

Figure 22 does not show the HEC byte, because the HEC byte (if enabled) is never written to or read from the Cell Buffer RAM. Rather, HEC generation/insertion on transmission and HEC checking/removal on reception are performed at the UTOPIA port[1]. The result of HEC verification is available in the Receive Cell Status Word. See Figure 22 and "Receive cell flow" on page 77.

Receive Cell Status location

While the ATM Header bytes and the SAR PDU bytes are fixed with respect to the cell holder in both the 52-byte and 56-byte mode, the location of the Receive Cell Status Word does change.

---

1. The MXT3010 also provides HEC generation and checking logic for devices not using the UTOPIA port.

In 52-byte mode, it precedes the ATM Header field, while in 56-byte mode, the four-byte User Header precedes the ATM Header, and the Receive Cell Status Word follows the last byte of the SAR PDU. The placement of the Receive Cell Status Word beyond the last byte of the SAR PDU in 56-byte mode conflicts with the concept of Cell Buffer RAM memory gathering as described in "Gather method accesses" on page 65. Memory gathering is still a valid means of addressing the unused Cell Buffer RAM space, however the presence of the Receive Cell Status Word within each receive cell holder must be accommodated.

Cell Buffer RAM memory construction

As shown in Figure 20, "Cell Buffer RAM organization," on page 61, the Cell Buffer RAM is logically constructed as sixteen 64-byte cell holders. As shown in Figure 22, "Receive cell organization: 52-byte and 56-byte cells," on page 63, a cell occupies no more than the top 56 or 58 bytes of a cell holder. This leaves approximately eight bytes of RAM at the bottom of each cell holder location. This space is discontinuous and therefore difficult to use. So that the CPU can regain access to this unused memory as a single linear space, the Cell Buffer RAM interface supports both a linear access and a memory gathering protocol.

Selecting an access method

Both the CPU and the DMA controllers can access the Cell Buffer RAM by using either linear or gather access methods. The CPU uses register rla and the Index field (IDX) in the LD (Load), LDD (Load Double), ST (Store), and STD (Store Double) instructions to form an address in the Cell Buffer RAM. See "Register load address (rla field)" on page 314 and "The index field (IDX)" on page 315. DMA controllers use register rla in the DMA1 or DMA2 instruction to form an address in the Cell Buffer RAM. See "Direct Memory Access Instructions" on page 283.

Whether generated by a CPU instruction or a DMA controller, Bit [10] of the local address selects the access method of the Cell Buffer RAM.

Bit 10	*Cell Buffer RAM method selected*
0	Linear
1	Gather

Linear method accesses

In linear method accesses, the Cell Buffer RAM is treated as a simple contiguous memory 1024 bytes in length. Bits [9:1] of the target address select the 16-bit halfword within this space.

Gather method accesses

In gather method accesses, the last eight bytes of each 64-byte section appear as a contiguous 128-byte block of memory. The first 16-bit halfword of this block is at address 0x0400 of the gather address method. The last 16-bit halfword is at address 0x047E. Thus, gather access recovers discontinuous regions of Cell Buffer RAM memory into one continuous address space. This is not additional space, but rather a method of making use of small pieces of existing space. Figure 23 illustrates this addressing method.

FIGURE 23.Gather method accesses
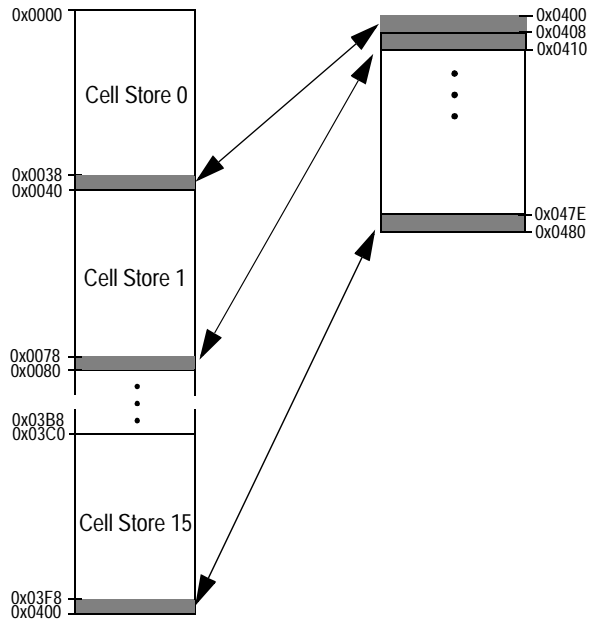
Please note the restrictions on gather access in 56-byte mode (see "Receive Cell Status location" on page 63). For additional information, please see "Cell Buffer RAM accesses" on page 317.

Cell Buffer RAM access

The MXT3010EP Cell Buffer RAM has five independent 16-bit ports, each capable of moving data at the internal clock frequency. The arrangement of data ports is shown in Figure 24.

FIGURE 24.Cell Buffer RAM access
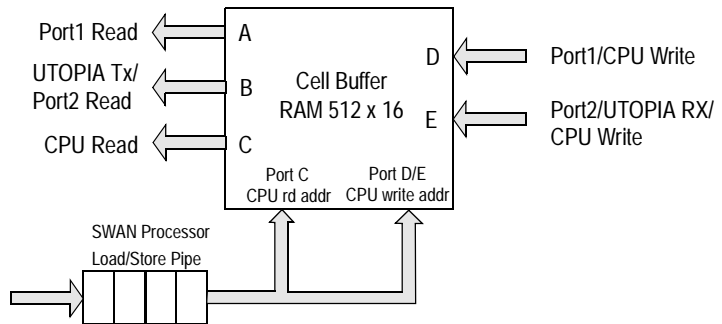
On a 100 MHz device, the three read ports can deliver data at a total rate of 600 MB per second, and the two write ports can accept a total of 400 MB per second.

Making optimum use of this high performance design requires some programming care, however. While Load and Store instructions from the SWAN processor are guaranteed to be ordered with respect to one another, ordering is not guaranteed between Load/Store instructions and DMA operations to Port1 or Port 2. Consider the following example:

```
STD R0/R1 @R48
STD R2/R3 @R49
STD R4/R5 @R50
DMA1W rsa/rsb R50
```

The Port1 write operation is not guaranteed to see the new values of R4/R5. This is true because Store Double (STD) instructions are retired in the Cell Buffer RAM at half the rate they can be issued by the SWAN, and the SWAN does not have a dedicate write pipe into the Cell Buffer RAM.

To guarantee correct behavior, the program must do one of the following:

1. Guarantee that at least one of the write ports is always available to ensure that the DMA from Port1 or Port2 can never fetch stale data. The pipelining of the DMA operation guarantees that it will not fetch data before it is flushed from the SWAN Load/Store pipe into Cell Buffer RAM.

2. Follow all stores by a dummy read prior to issuing a DMA command. The read ensures that all preceding writes are flushed in the pipe. Note that since the Load Double (LD) is offloaded from the host, it must be followed by an instruction that uses the destination of the load to invoke the hardware register scoreboarding mechanism.

3. Use successive writes to ensure that preceding writes are flushed through the pipe into the Cell Buffer RAM.

CHAPTER 6 *The UTOPIA port*
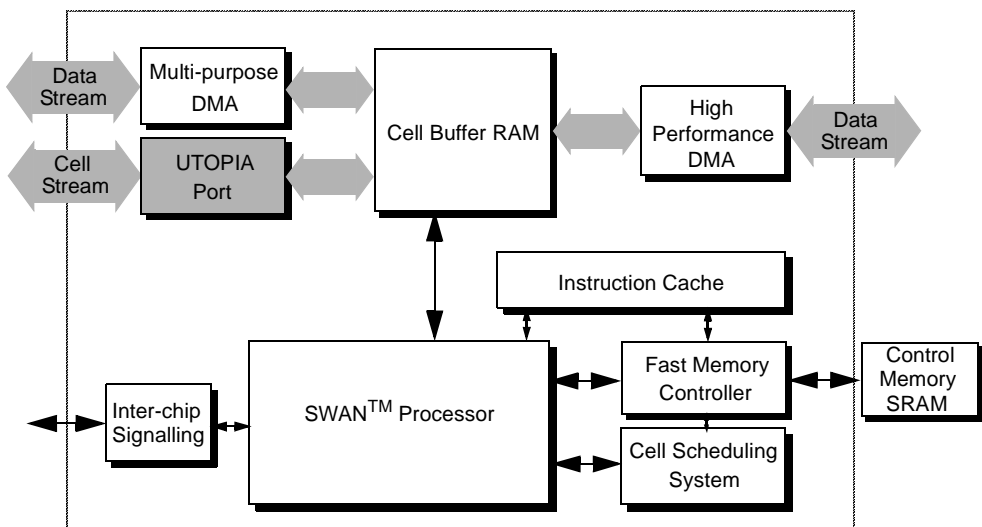
The UTOPIA port implements the ATM Forum's UTOPIA Level
1 and Level 2 protocol for interfacing ATM Layer devices, such
as the MXT3010, to PHY Layer devices, such as SONET framers.
The UTOPIA port supports the direct attachment of up to 16 sin-
gle PHY or multiple logical PHY devices. In addition, the UTO-
PIA port supports the direct attachment of a Level 2-compliant

Multi-PHY device with up to 16 ports. In compliance with the ATM Forum specification, the UTOPIA connection operates as the Master device.

This chapter includes:

- UTOPIA port interface overview

- Receive cell flow

- Transmit cell flow

- The control byte and special operations

- Multi-PHY support

- Receive Header Reduction hardware

- UTOPIA port configuration summary

UTOPIA port interface overview

Features

The UTOPIA port interface includes the following features:

- Two modes of operation are supported, 8-bit bi-directional mode and 16-bit unidirectional mode (either transmit or receive).

- The UTOPIA port supports up to 16 physical ports in 8-bit bi-directional mode. The UTOPIA port complies with the ATM Forum's *Level 2 Specification for Multi-PHY Operations*.

- Cell-level handshaking is supported. No wait states are inserted, and no wait states are expected.

- 56-byte cell mode over the UTOPIA interface is supported for applications where a field is prepended to an ATM cell.

- HEC insertion and checking can be enabled.

Operating modes

The UTOPIA port can be configured to operate in bi-directional mode with an 8-bit Receive (Rx) data path and an 8-bit Transmit (Tx) data path, or in unidirectional mode as either a 16-bit Transmitter or a 16-bit Receiver. The 16-bit mode supports 622 Mb/s data rates.

Selecting 8-bit or 16-bit mode

Bit [8] of the UTOPIA Configuration register (R62) controls the operating mode.

TABLE 11.   UTOPIA port data bus width selection

Bit	Description
8	UTOPIA Port Data Bus Width
	0 16 Bits Wide
	1 8 Bits Wide

In 16-bit transmit mode, the TxData pins carry data [7:0]. The RxData pins are configured as outputs and carry data [15:8]. In 16-bit receive mode, the RxData pins carry data [7:0]. The TxData pins are configured as inputs and carry data [15:8].

TABLE 12.   UTOPIA port Tx and Rx pin utilization in 16-bit mode

Mode	Tx Data Pins	Rx Data Pins
16-bit transmit	Data [7:0]	Data (outputs) [15:8]
16-bit receive	Data (inputs) [15:8]	Data [7:0]

Resetting the transmitter and receiver

Bits [6:4] of the UTOPIA Configuration register (R62) control the Transmit Cell Buffer size in the Cell Buffer RAM, and bits [3:1] control the Receive Cell Buffer size. Definitions for these bits appear in Table 9, "UTOPIA Configuration control of the Cell Buffer RAM," on page 60. While these bits primarily affect the operation of the Cell Buffer RAM, a buffer size selection of 0 cells places the corresponding transmit or receive UTOPIA interface in reset mode. In reset mode the corresponding output signals are placed into their inactive states.

Selecting transmit or receive mode	Transmit-only operation is selected by setting bits [3:1] of the UTOPIA Configuration register (R62) to zeroes, thus placing the UTOPIA port receiver in reset mode. Receive-only operation is selected by setting bits [6:4] of the UTOPIA Configuration (R62) to zeroes, thus placing the UTOPIA port transmitter in reset mode. Figure 25 shows a UTOPIA port using 8/8- and 16-bit modes.

FIGURE 25.The UTOPIA port: 8/8 and 16-bit modes
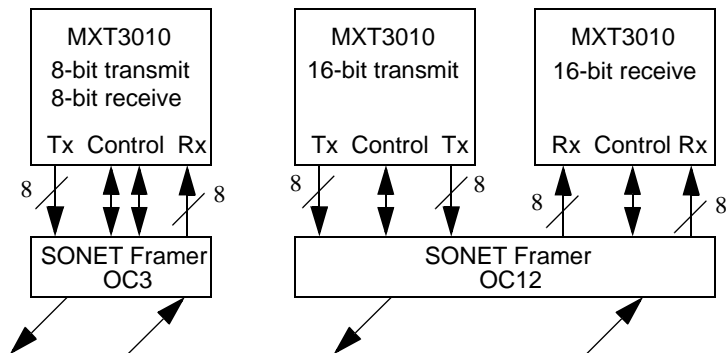
Selecting cell length and HEC operation	The UTOPIA configuration operation uses R42, bits 0 and 1, to select HEC operation and cell length.

TABLE 13.   Cell length and HEC control

Bit	*Description*	
0	HEC Control	
	0	HEC is generated (Tx), inserted (Tx), and checked (Rx)
	1	HEC is omitted
1	Cell Length Control	
	0	52-byte cells
	1	56-byte cells

UTOPIA speed select	The MXT3010 can operate each UTOPIA interface (transmit and receive) either at the input clock frequency or at one-half that frequency. Since the SWAN processor internal clock runs at

twice the input clock frequency, these selections correspond to one-half or one-quarter of the internal clock frequency. The MXT3010 generates a UTOPIA output clock for each of the transmit and receive interfaces based on the setting of the clock selection bit in the UTOPIA Configuration register (R62). All PHY to ATM layer transfers should be controlled from the rising edge of these clocks.

TABLE 14.   UTOPIA port clock selection

Bit	Description
7	UTOPIA Port operational/output clock selection
	0 TXCLK and RXCLK operate at 1/2 of internal clock frequency.
	1 TXCLK and RXCLK operate at 1/4 of internal clock frequency.

UTOPIA Port clock phases

Figure 26 and Figure 27 show the relationship between the chip input clock, the internal clock, and TXCLK and RXCLK operating at 1/2 and 1/4 of the internal clock frequency, respectively.

FIGURE 26.Clock phases for RX/TX CLK = 1/2 Internal Clock
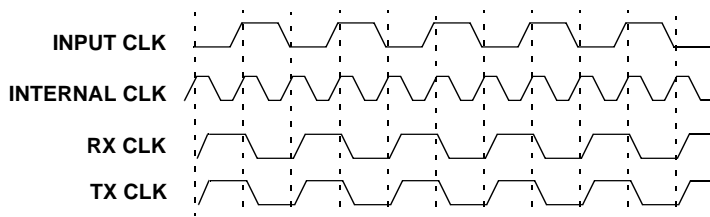
FIGURE 27.Clock phases for RX/TX CLK = 1/4 Internal Clock
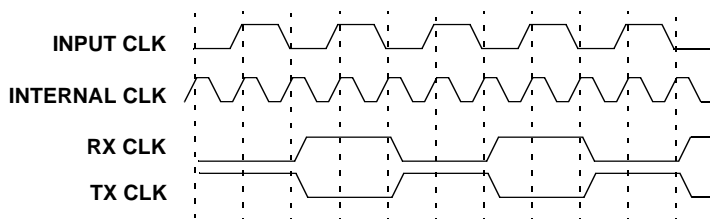
UTOPIA cell formats

Two standard formats for cells are defined for UTOPIA interfaces depending on the width of the data bus in use. Additionally, proprietary schemes may define arbitrary cell lengths and formats as long as the format is commonly understood by the components on the bus. Figure 28 shows the standard cell formats for UTOPIA interfaces.

FIGURE 28.UTOPIA 8-bit and 16-bit cell formats
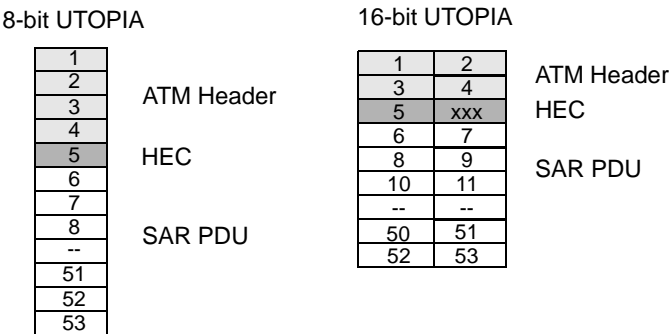
In the case of cells received through a 16-bit UTOPIA port, the PHY inserts an additional byte after the HEC to ensure that the SAR PDU data structure presented to the UTOPIA port is 16-bit aligned. After the HEC has been checked by the UTOPIA port, both the HEC and the extra byte are deleted before the cell is stored in the Cell Buffer RAM. In the case of cells transmitted through a 16-bit UTOPIA port, the UTOPIA port inserts an additional byte after the HEC to ensure that the data structure presented to the PHY is 16-bit aligned. The PHY transmits the HEC over the SONET interface, but discards the extra byte.

Cell format examples

The following figures show examples of HEC-enabled 52-byte mode, HEC-disabled 52-byte mode, HEC-enabled 56-byte mode, and HEC-disabled 56-byte mode.

FIGURE 29.HEC-enabled 52-byte mode

52-byte cell	
0x0000	Unused
0x0002	Receive Cell Status Word
0x0004	ATM Header bytes 0, 1
0x0006	ATM Header bytes 2, 3
0x0008	SAR PDU bytes 0, 1
0x000A	SAR PDU bytes 2, 3
0x000C	SAR PDU bytes 4, 5
0x000E	SAR PDU bytes 6, 7
0x0010	SAR PDU bytes 7, 8

0x0036	SAR PDU bytes 46, 47
0x0038	Unused
0x003A	Unused
0x003C	Unused
0x003E	Unused

8-bit UTOPIA

1
2
3
4
5
6
7
8
--
51
52
53

ATM Header

HEC

SAR PDU

16-bit UTOPIA

1	2
3	4
5	xxx
6	7
8	9
10	11
--	--
50	51
52	53

ATM Header
HEC

SAR PDU

FIGURE 30.HEC-disabled 52-byte mode

52-byte cell	
0x0000	Unused
0x0002	Receive Cell Status Word
0x0004	ATM Header bytes 0, 1
0x0006	ATM Header bytes 2, 3
0x0008	SAR PDU bytes 0, 1
0x000A	SAR PDU bytes 2, 3
0x000C	SAR PDU bytes 4, 5
0x000E	SAR PDU bytes 6, 7
0x0010	SAR PDU bytes 7, 8

0x0036	SAR PDU bytes 46, 47
0x0038	Unused
0x003A	Unused
0x003C	Unused
0x003E	Unused

8-bit UTOPIA

1
2
3
4
5
6
7
--
50
51
52

ATM Header

SAR PDU

16-bit UTOPIA

1	2
3	4
5	6
7	8
9	10
11	12
--	--
49	50
51	52

ATM Header

SAR PDU

FIGURE 31.HEC-enabled 56-byte mode

56-byte cell

Address	
0x0000	User Header bytes 0, 1
0x0002	User Header bytes 2, 3
0x0004	ATM Header bytes 0, 1
0x0006	ATM Header bytes 2, 3
0x0008	SAR PDU bytes 0, 1
0x000A	SAR PDU bytes 2, 3
0x000C	SAR PDU bytes 4, 5
0x000E	SAR PDU bytes 6, 7
0x0010	SAR PDU bytes 7, 8
⋮	
0x0036	SAR PDU bytes 46, 47
0x0038	Receive Cell Status Word
0x003A	Unused
0x003C	Unused
0x003E	Unused

8-bit UTOPIA

#	
1	
2	User Header
3	
4	
5	
6	ATM Header
7	
8	
9	HEC
10	
11	SAR PDU
12	
--	
55	
56	
57	

16-bit UTOPIA

1	2	
3	4	User Header
5	6	
7	8	ATM Header
9	xxx	HEC
10	11	
12	13	
14	15	SAR PDU
--	--	
54	55	
56	57	

FIGURE 32.HEC-disabled 56-byte mode

56-byte cell

Address	
0x0000	User Header bytes 0, 1
0x0002	User Header bytes 2, 3
0x0004	ATM Header bytes 0, 1
0x0006	ATM Header bytes 2, 3
0x0008	SAR PDU bytes 0, 1
0x000A	SAR PDU bytes 2, 3
0x000C	SAR PDU bytes 4, 5
0x000E	SAR PDU bytes 6, 7
0x0010	SAR PDU bytes 7, 8
⋮	
0x0036	SAR PDU bytes 46, 47
0x0038	Receive Cell Status Word
0x003A	Unused
0x003C	Unused
0x003E	Unused

8-bit UTOPIA

#	
1	
2	User Header
3	
4	
5	
6	ATM Header
7	
8	
9	
10	
11	SAR PDU
--	
54	
55	
56	

16-bit UTOPIA

1	2	
3	4	User Header
5	6	
7	8	ATM Header
9	10	
11	12	SAR PDU
13	14	
--	--	
53	54	
55	56	

Receive cell flow

The UTOPIA Receiver transfers cells from an external framing device into the Cell Buffer RAM. All cells received from the physical layer device are written into the Cell Buffer RAM. If HEC insertion and checking is enabled in the Mode Configuration Register (R42), the validity of the cell's HEC byte is marked in bit 9 of the Receive Cell Status field. If HEC insertion and checking is disabled, bit 9 should be ignored.

Since Header Error Control (HEC) generation and checking is a PHY Layer function[1], the MXT3010 discards the HEC field before copying the cell into the Cell Buffer RAM. As a result, a cell in the Cell Buffer RAM consists of 52 contiguous bytes with no gap existing between the ATM Header and the SAR PDU.

The UTOPIA port writes receive cells into the Cell Buffer RAM beginning at location 0x0000. The UTOPIA Receiver writes successive cells into successive cell buffers in the Cell Buffer RAM. During device initialization, the programmer can specify how many cells the UTOPIA Receiver can use. If the Receiver buffer size is set to six cells, for example, the Receiver loops around after the sixth cell and begins writing cells again at location 0x0000 in the Cell Buffer RAM. If the Receiver buffer size is set to eight cells, the receiver loops around after the eighth cell and begins writing cells again at location 0x0000 in the Cell Buffer RAM.

---

1. For applications which do not use the UTOPIA port, HEC generation and checking is provided in the SWAN processor. See "HEC generation and check circuit" on page 25.

A Receive Cell Status word is stored in Cell Buffer RAM at the completion of each receive cell. The format of the Receive Cell Status word is:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Reserved				HE	CE		PTI Copy				PHY Addr		

Bits	Name	Function
4:0	PHY Address	The address of the PHY from which this cell was received.
7:5	PTI Copy	A copy of the Payload Type Indicator field from the received cell header.
8	CE	CRC10 Error When set to one (1), this bit indicates an erroneous CRC was received. Otherwise, this bit is zero (0). This bit should be ignored when CRC10 is not in use.
9	HE	HEC Error When set to one (1), this bit indicates an erroneous HEC was received. Otherwise, this bit is zero (0). This bit should be ignored when HEC is not in use.
15:10	Reserved	These bits should be ignored on reads.

The location of the Receive Cell Status word in the Cell Buffer RAM is dependent on the configured cell length, 52 or 56 bytes. For more information, see "Receive Cell Status location" on page 63.

UTOPIA receiver counters

The UTOPIA Receiver contains two counters, RXBUSY and RXFULL, that track cells received from the PHY layer and stored in the Cell Buffer RAM. Figure 33 on page 79 and Figure 34 on page 81 show how these counters are used in the reception process. A written description follows in "The RXBUSY counter" on page 79 and in "The RXFULL counter" on page 81.

FIGURE 33.The RXBUSY counter

The RXBUSY counter drawing shows two counters. On the left, a "Receiver Busy Counter" with Decrement input from CPU and Increment input from UTOPIA Cell Received. On the right, a "Receiver Full Counter" with Decrement inputs from Port1 Done, Port2 Done, CPU and Increment input from UTOPIA Cell Received. Both feed into Control Logic which has inputs RXCLAV and CPU, and outputs to RXENB_, ESS3 Rx Attention, and ESS9 Rx Busy.

The RXBUSY counter

Function
: The RXBUSY counter tracks the arrival of new cells in the Cell Buffer RAM awaiting CPU servicing. The device initialization process clears the RXBUSY counter to zero.

Incrementing
: As the UTOPIA receiver places the last byte of a cell into the receive section of the Cell Buffer RAM, it increments the RXBUSY counter.

Signals driven
: The RXBUSY signal of the RXBUSY counter drives bit 9 of the External State Signals (ESS) register (R42). The SWAN processor tests ESS9 to determine when one or more cells are awaiting processing by the CPU in the Cell Buffer RAM:

- If ESS9 = 0, no cells are awaiting processing.

- If ESS9 = 1, one or more cells are awaiting processing.

The CPU can use the ESS9 signal to conditionally branch to a receive cell service routine. For example, a Branch Immediate

instruction ("BI Branch Immediate" on page 272) can specify a conditional branch to $RECV_CELL if ESS9 is a 1:

```
BI $RECV_CELL ESS9/1
```

In addition to the RXBUSY indication on ESS9, a receiver attention output of the RXBUSY counter drives ESS3. This signal indicates that the receive buffer is almost full:

- If ESS3 = 0, the Cell Buffer RAM receive buffer contains less than 4 cells.

- If ESS3 = 1, the Cell Buffer RAM receive buffer contains 4 or more cells.

Decrementing   As the CPU services the newly arrived cell, the CPU decrements the RXBUSY counter. The CPU decrements the RXBUSY counter using the Counter System Operation feature of the Branch instructions. For example, a Branch Immediate (BI) instruction can specify an unconditional branch to $MAIN and decrement the RXBUSY counter using a counter system operation option. See "Counter system operation" on page 269.

```
BI $MAIN DRXBUSY
```

FIGURE 34.The RXFULL counter
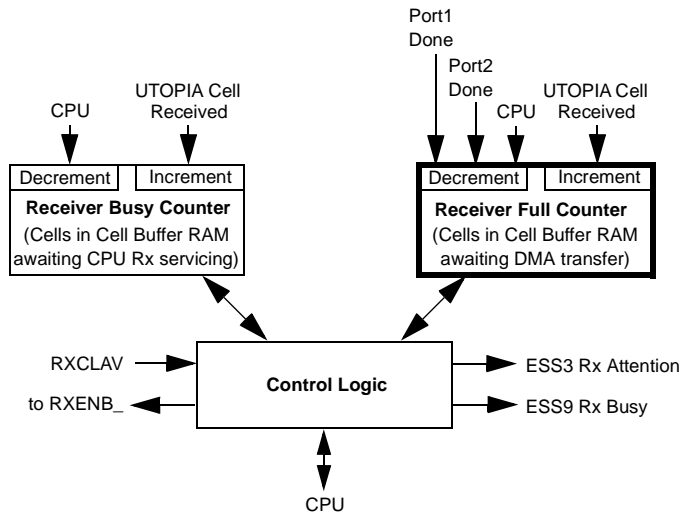
The RXFULL counter

Function

The RXFULL counter indicates to the DMA engines and the CPU that cells are in the Cell Buffer RAM awaiting transfer. The RXFULL counter also drives the RXENB_ signal to the PHY devices. The device initialization process:

- Partitions the Cell Buffer RAM.
- Establishes a value for the number of cells that can be stored in the Cell Buffer RAM receive section.
- Clears the RXFULL counter to zero.

Incrementing

As the last byte of a cell is placed into the receive section of the Cell Buffer RAM by the UTOPIA receiver, it increments the RXFULL counter.

When the MXT3010 receives a cell, it tests the RXCLAV signal from the PHY device, which signals the presence of a cell ready for transfer. The UTOPIA Port controller tests the availability of space in the Cell Buffer RAM by examining the count kept by

the RXFULL counter. If the MXT3010 can accept a cell and the PHY has a cell to send, the UTOPIA port enables the transfer by asserting the ATM layer Receiver Enable (RXENB_) output.

Decrementing   The Port 1/Port 2 DMA controllers can decrement the RXFULL counter at the completion of a data transfer operation if the DMA command specifies the UTOPIA post-DMA operative directive (POD) option in a memory write operation. For further information on POD and other DMA instruction options, see "The Control instruction field option" on page 287.

Alternatively, the CPU can decrement the RXFULL counter after the received cell has been processed. As with the RXBUSY counter, the CPU decrements the RXFULL counter by specifying a Branch instruction with an option. For example, a Branch Immediate (BI) instruction can specify an unconditional branch to $MAIN and decrement the RXFULL counter using the DRX-FULL counter system operation option.

```
BI    $MAIN DRXFULL
```

Whether RXFULL is decremented by use of a Branch instruction or by use of a DMA instruction, the CPU must decrement the RXBUSY counter whenever it finishes handling a cell.

Transmit cell flow

The UTOPIA transmitter transfers cells from the Cell Buffer RAM to an external framing device. All cells to be transmitted must reside in the Cell Buffer RAM before transmission. As part of the transmit operation when HEC generation is enabled, the UTOPIA transmitter inserts a valid HEC between the last byte of the ATM Header and the first byte of the SAR-PDU. In 8-bit mode, only the 8-bit HEC is inserted; in 16-bit mode, the 8-bit HEC plus an 8-bit stuffer are inserted.

The UTOPIA port transfers cells from the Cell Buffer RAM beginning at location 0x0200. The UTOPIA transmitter reads successive cells from the Cell Buffer RAM. During device initialization, the programmer can specify how many cells the UTOPIA transmitter should use. If the transmitter buffer size is set to two cells, the transmitter loops around after the second cell and begins reading cells again at location 0x0200 in the Cell Buffer RAM. For example, if the transmitter buffer size is set to six cells, the transmitter loops around after the sixth cell and begins reading cells again at location 0x0200 in the Cell Buffer RAM.

Each transmit cell buffer is associated with an 16-bit control word. The transmit control word is written through a FIFO-like internal memory mapped into R43. Writes to R43 push control words onto the control byte FIFO for use when the transmit operation is executed.

The format of the transmit control word is:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved									I	CG	TXPHY				

Bits	Name	Function
4:0	TXPHY	Select the address of the target PHY in a multi-PHY system
5	CG	Generate and insert a CRC10 for this cell
6	I	Insert unassigned cell
15:7	Reserved	Programs should write a zero to these bits.

The UTOPIA Control FIFO register recirculates its output back to its input. For applications that only transmit one type of cell, the 8 locations in R43 can be loaded at initialization time and need not be written again.

UTOPIA transmitter counters

The UTOPIA transmitter contains two counters, TXBUSY and TXFULL, that track cells in the transmit section of the Cell Buffer RAM. Figure 35 on page 84 and Figure 36 on page 85 show how these counters are used in the transmission process. A written description follows in "The TXBUSY counter" on page 84 and in "The TXFULL counter" on page 86.

FIGURE 35.The TXBUSY counter
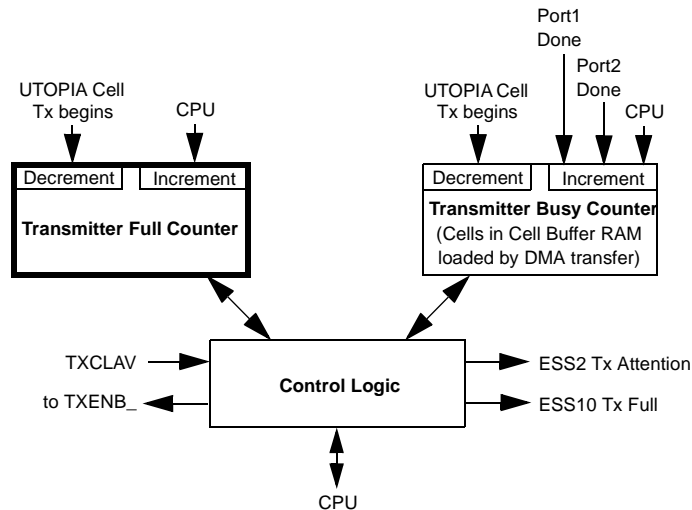
The TXBUSY counter

Function

The CPU uses the TXBUSY counter to inform the UTOPIA transmitter that a new cell in the Cell Buffer RAM is ready for transmission. When the TXBUSY counter is non-zero, the MXT3010 generates the ATM layer Transmit Enable (TXENB_) output signal that informs the PHY device that the MXT3010 is ready to send a cell.

Incrementing

Program execution can be accelerated if the DMA controller increments the TXBUSY counter after it reads data. This technique requires the DMA command to specify a memory read operation with the POD option (see "Post-DMA Operation Directives (PODs)" on page 109). For example:

```
DMA1R rsa/rsb, rla[BC/#, CRC{X Y}, POD, ST]
DMA2R rsa/rsb, rla[BC/#, POD, ST]
```

Alternatively, the CPU can increment the TXBUSY counter by specifying a Branch instruction with a counter system operation option. For example:

```
BI   $MAIN ITXBUSY
```

Decrementing

As the UTOPIA transmitter processes the first byte of a cell, the transmitter decrements the TXBUSY counter.

FIGURE 36.The TXFULL counter
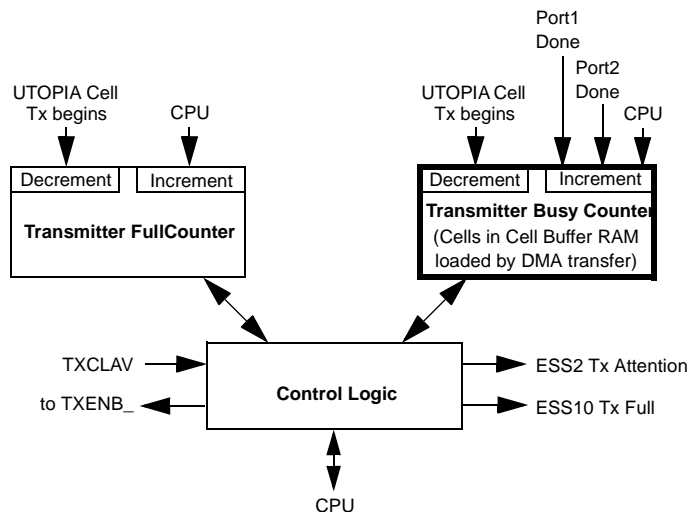
The TXFULL counter

Function

The TXFULL counter tracks the number of cells that are available in the Cell Buffer RAM for transmission. The CPU uses this counter to determine if space is available in the Cell Buffer RAM to assemble a cell via DMA transfers.

Signals driven

The TXFULL output of the TXFULL counter is connected to ESS 10.

- If ESS 10 = 0, the Cell Buffer RAM transmit buffer has space available.

- If ESS 10 = 1, the Cell Buffer RAM transmit buffer queue is full, and the CPU does not build a cell.

The CPU can use the ESS 10 signal to conditionally branch to a transmit cell routine, since available space allows cell building to begin. For example, a Branch Immediate instruction ("BI Branch Immediate" on page 272) can specify a conditional branch to $TRANSMIT_CELL if ESS10 is a 0:

```
BI $TRANSMIT_CELL ESS10/0
```

The Tx Attention output of the TXFULL counter is connected to ESS 2. This signal indicates that the transmit buffer is approaching a drained state.

- If ESS 2 = 0, the Cell Buffer RAM transmit buffer contains more then 2 cells.

- If ESS 2 = 1, the Cell Buffer RAM transmit buffer contains 2 or fewer cells.

Incrementing

As the CPU begins processing a new cell, the CPU increments the TXFULL counter. Now, the CPU can queue the next cell for transmission as a background task. The CPU increments the TXFULL counter by specifying a branch instruction with an option. For example:

```
BI    $MAIN ITXFULL
```

Decrementing   As the UTOPIA transmitter processes the last byte of a cell, the transmitter decrements the TXFULL counter.

CRC10 generation and checking support

The UTOPIA port can perform CRC10 generation and checking in support of AAL3/4, OAM cells, and RM cells. Generation of CRC10 is controlled on a cell-by-cell basis for transmission. CRC10 checking is performed on all receive cells.

The UTOPIA transmitter generates and inserts a CRC10 for a cell if a 1 was written into the CG bit of the control byte for the cell buffer (see "R43-write UTOPIA Control FIFO register" on page 205). If CG = 1, the UTOPIA transmitter generates and inserts a CRC10 field into the cell. If CG = 0, the CPU does not insert a CRC10 field into the cell. The CG bit should be set when queuing an AAL3/4, OAM cell, or RM cell for transmission.

The SWAN processor can determine if a CRC10 error exists for a received cell by checking the CE bit in the second halfword of the received cell. If CE = 1, a CRC10 error exists in the cell. If CE = 0, no CRC10 error exists in the cell. If a cell is from a connection that does not use a CRC10 field, the state of CE is irrelevant and should not be checked.

Multi-PHY support

The MXT3010 supports the connection of up to 16 physical ports to a single UTOPIA port. The UTOPIA port is compliant[1] with the ATM Forum's Level 2 specification for Multi-PHY operations, but can also support Level 1 devices. Bits [15:9] of the UTOPIA Configuration register (R62) are used to support multi-PHY operation. The bit assignments are as follows:

TABLE 15.   Bit assignments for multi-PHY operation

Bits	*Description*
13:9	UTOPIA Port Most Significant PHY Address
	The UTOPIA Port Receiver polls PHY devices searching for an RXCLAV by incrementing the polled address according to the UTOPIA Level 2 specification. The UTOPIA Port Receiver knows that it has reached the last address and should begin at zero again when it reaches this address. For examples of the use of these bits, see Figure 37 on page 89 and Figure 38 on page 90.
15:14	Number of Physical PHY devices present
	This value tells the UTOPIA Port Receiver the number of physical PHY devices present. This in turn determines the number of RXCLAV/TXCLAV and RXENB_/TXENB_ signals that should be used.
	00 Reserved
	01 1-PHY mode
	10 2-PHY mode
	11 Reserved

---

1. While the MXT3010 is compliant with the ATM Level 2 specification, it uses a three-clock polling cycle rather than the two-clock polling cycle shown in the specification diagrams. In addition, the MXT3010 does not provide an idle address of 1F between addresses. The three clock polling cycle reduces the number of PHY devices that can be polled during a 52-clock cell time from 26 to 14. Thus, if 16 PHY devices are used, more than one cell time is needed to poll them all. Refer to *Application Note 20, "MXT3010EP UTOPIA Level 1 and Level 2 Interface Operation"* for further infomration.

The use of bits [15:9] is best understood by considering the configurations shown in Figure 37 and Figure 38.

FIGURE 37.Level 2 PHY configurations

Note: While only transmit control signals (TX) signals are shown, a corresponding set of receive (RX) signals is also used.

Address control (i.e. which device is 0-3, which is 4-7, etc.) provided by host or other external device
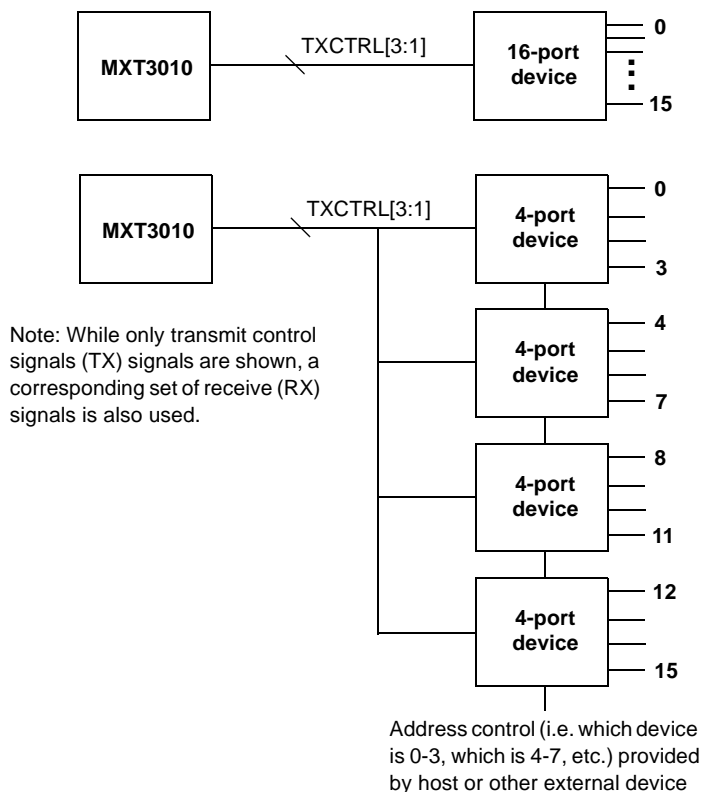
The two implementations shown are logically equivalent Level 2 configurations. Since there is only one logical PHY, bits [15:14] should be 01 to select 1-PHY mode. Since the most significant PHY address is 15, bits [13:9] should be 1111.

FIGURE 38.Mixed Level 1 and Level 2 PHY configuration
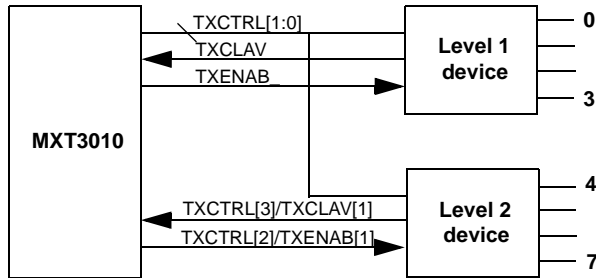
Notes:1.While only transmit control signals (TX) are shown, a corresponding set of receive (RX) signals is also used.

2. In this configuration, the Level 1 device must tri-state the RXDATA/RXSOC leads when its RXENB_ pin is de-asserted.

The implementation shown has two logical PHYs. In this configuration, TX/RXCTRL[3] is used as TX/RXCLAV for the second PHY, and TX/RXCTRL[2] is used as TX/RXENB_ for the second PHY. Since there are two logical PHYs, bits [15:14] should be 10 to select 2-PHY mode. Since the most significant PHY address is 7, bits [13:9] should be 0111.

Mode	TX/RX CLAV	TX/RX ENB	ADRS
1 PHY	TX/RX_CLAV	TX/RX_ENB_	TX/RX CTRL [3:0]
2 PHY			
PHY 0	TX/RX_CLAV	TX/RX_ENB_	TX/RX CTRL [1:0]
PHY 1	TX/RX CTRL [3]	TX/RX CTRL [2]	TX/RX CTRL [1:0]

Receive Header Reduction hardware

The MXT3010 provides receive header reduction via bits [6:0] of the System register (R63). The results of this reduction can be used as a Channel ID. The bit definitions are as follows:

TABLE 16. Receive Header Reduction control

Bits	Name	Function	
6:0	VPI/VCI	Utopia Receiver Reduction Mask	
		Setting	Value written into ATM Header lower halfword in CBR
		0000001	{0,0,0,0,0,0, vpi(0), vci(7:0), clp}
		0000011	{0,0,0,0,0, vpi(1:0), vci(7:0), clp}
		0000111	{0,0,0,0, vpi(2:0), vci(7:0), clp}
		0001111	{0,0,0, vpi(3:0), vci(7:0), clp}
		0000010	{0,0,0,0,0,0, vpi(0), vci(8:0), clp}
		0000110	{0,0,0,0,0, vpi(1:0), vci(8:0), clp}
		0001110	{0,0,0,0, vpi(2:0), vci(8:0), clp}
		0011110	{0,0, vpi(3:0), vci(8:0), clp}
		0000100	{0,0,0,0, vpi(0), vci(9:0), clp}
		0001100	{0,0,0, vpi(1:0), vci(9:0), clp}
		0011100	{0,0, vpi(2:0), vci(9:0), clp}
		0111100	{0, vpi(3:0), vci(9:0), clp}
		0001000	{0,0,0, vpi(0), vci(10:0), clp}
		0011000	{0,0, vpi(1:0), vci(10:0), clp}
		0111000	{0, vpi(2:0), vci(10:0), clp}
		1111000	{vpi(3:0), vci(10:0), clp}
		0010000	{0,0,vpi(0), vci(11:0), clp}
		0110000	{0,vpi(1:0), vci(11:0), clp}
		1110000	{vpi(2:0), vci(11:0), clp}
		0100000	{0,vpi(0), vci(12:0), clp}
		1100000	{vpi(1:0), vci(12:0), clp}
		1000000	{vpi(0), vci(13:0), clp}
		0000000	{vci(14:0), clp}

Receive header reduction mode is enabled by bit 0 of the
UTOPIA Configuration register (R62).

TABLE 17.   Receive Header Reduction enable bit

Bits	*Description*	
0	UTOPIA Receiver Reduction Mode Enable Bit	
	0	Reduction Function Disabled (ATM Header bytes [2:3] written into the Cell Buffer RAM unchanged)
	1	Reduction Function Enabled (ATM header bytes [2:3] written into the Cell Buffer RAM after reduction function performed according to Reduction Mask Setting selected by R63[6:0]).

UTOPIA port configuration summary

UTOPIA configuration information is stored in the UTOPIA Configuration register, R62. The SWAN processor passes this information to the UTOPIA Port at system initialization. Two bits (0,1) in the ESS register (R42) are also used in the programming of the UTOPIA port. Descriptions of these bits appear in the tables referenced in Table 18. For a complete listing and description of all of the bits in these registers, see "R42-read External State Signals (ESS) register" on page 200 and "R62 The UTOPIA Configuration register" on page 219.

TABLE 18.   UTOPIA configuration information

Bits	*Function*	*Reference*
R42 [0]	HEC control	Table 13 on page 72
R42 [1]	Cell length control	Table 13 on page 72
R62 [0]	UTOPIA Receiver Reduction mode enable bit	Table 17 on page 92
R62 [3:1]	Receive Cell Buffer size in the Cell Buffer RAM (000 = Receiver Reset)	Table 9 on page 60
R62 [6:4]	Transmit Cell Buffer size in the Cell Buffer RAM (000 = Transmitter Reset)	Table 9 on page 60
R62 [7]	UTOPIA Port operational / output clock frequency selection	Table 14 on page 73
R62 [8]	UTOPIA Port data bus width	Table 11 on page 71
R62 [13:9]	UTOPIA Port most significant PHY address	Table 15 on page 88
R62 [15:14]	Number of physical PHY devices present	Table 15 on page 88
R63 [6:0]	Receiver Header Reduction control	Table 16 on page 91

UTOPIA port sequence diagrams

This section shows sequence diagrams for the following UTOPIA Port operations:

- Receive timing for single PHY, 8-bit mode (Figure 39 on page 94)

- Transmit timing for single PHY, 8-bit mode (Figure 40 on page 95)

- Receive full timing for single PHY, 8-bit mode (Figure 41 on page 95)

- Transmit full timing for single PHY, 8-bit mode (Figure 42 on page 95)

Set-up times, propagation times, and other timing information for the UTOPIA Port interface are provided in "Timing" on page 343.

FIGURE 39. UTOPIA Port receive timing - single PHY, 8-bit mode
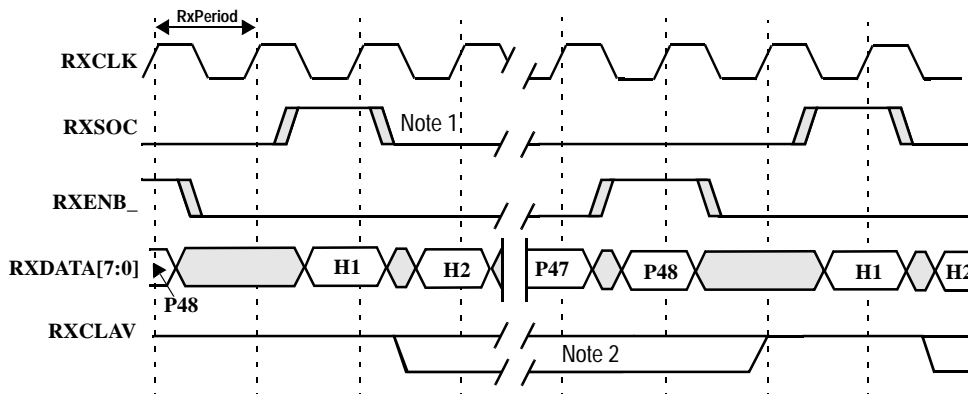
Notes for Figure 39, Figure 40, Figure 41, Figure 42:

1. RXSOC must not be asserted outside the scope of a valid cell. That is, it can only be asserted while RXENB_ is asserted (low).
2. RXCLAV/TXCLAV must be stable during octets 45 through 48.

FIGURE 40.UTOPIA Port transmit timing - single PHY, 8-bit mode
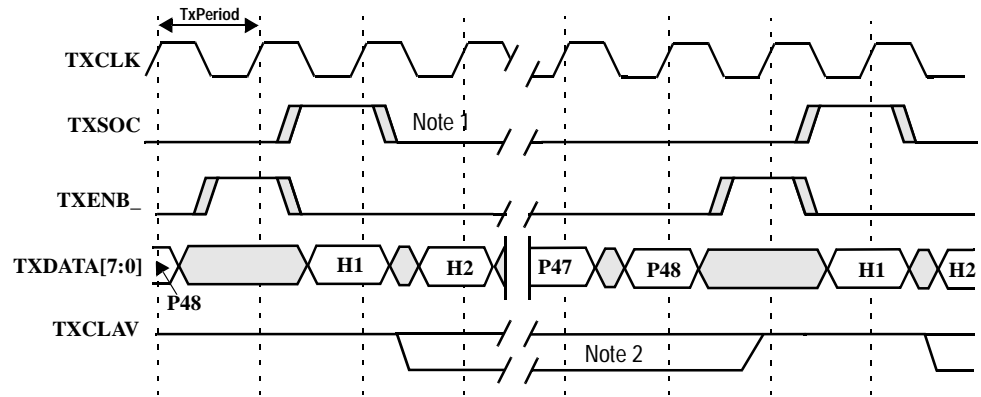
FIGURE 41.UTOPIA Port receive full timing - single PHY, 8-bit mode
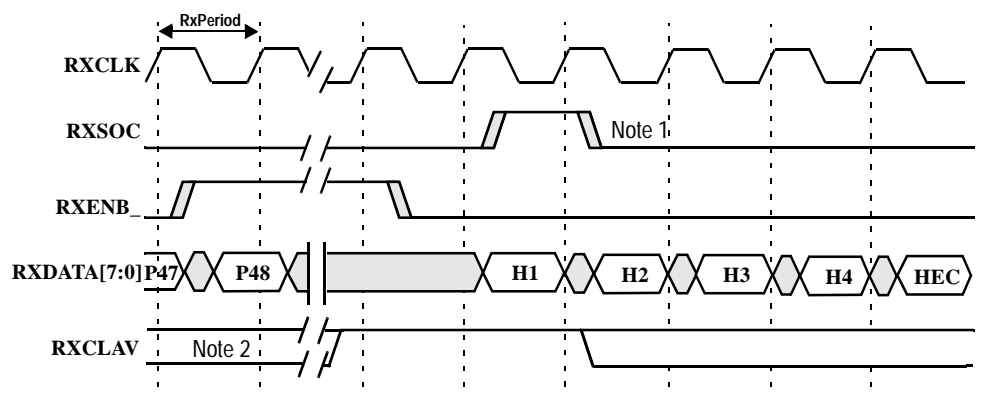
FIGURE 42.UTOPIA Port transmit full timing - single PHY, 8-bit mode
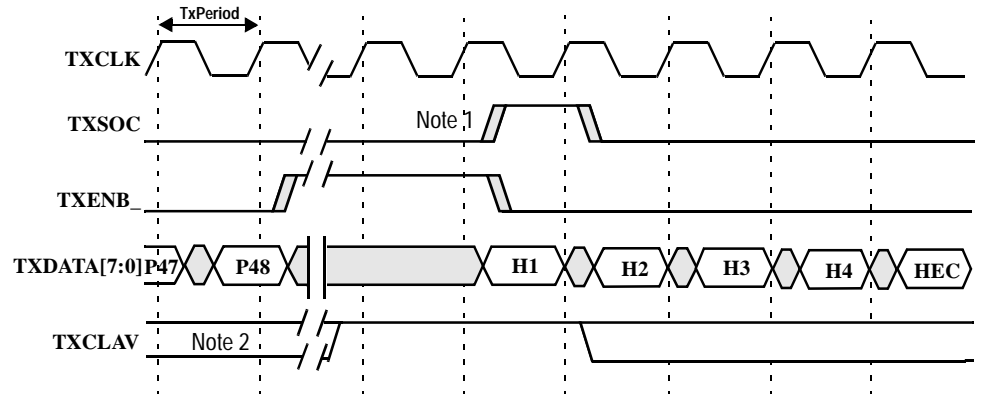
CHAPTER 7 *The Port1 and Port2 Interfaces*
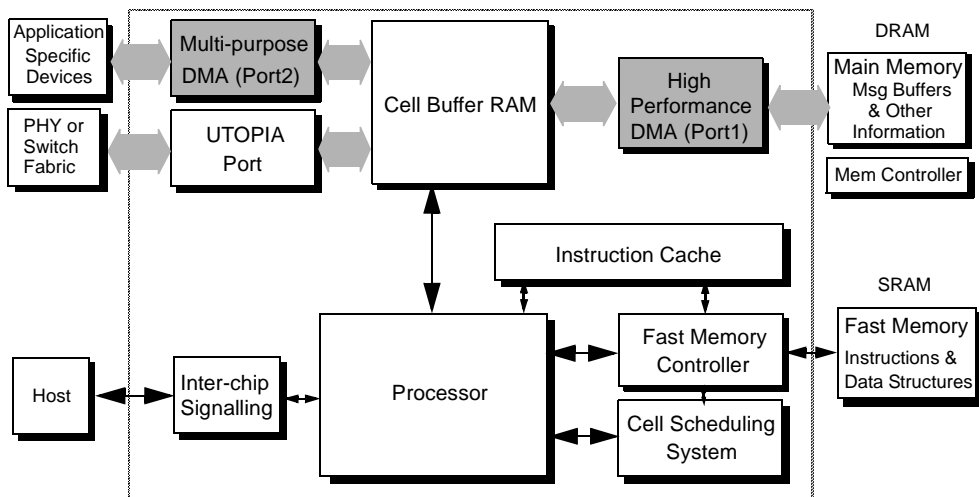
Port1 and Port2 are high-speed interface ports. For each port, this chapter includes:

- Port interface overview
- Control signals
- Port operations
- Burst and non-burst operations
- Port DMA controllers
- Data flow to Cell Buffer RAM

Port interface overview

Both Port1 and Port2 provide high speed transfer paths to and from the MXT3010 Cell Buffer RAM. The characteristics of the two ports differ, however, and are shown in Table 19.

TABLE 19. Characteristics of Port1 and Port2

Port1	*Port2*
Supports only burst mode operations.	Supports both burst and non-burst mode operations.
Provides a 32-bit, multiplexed address and data bus.	Provides a 16-bit, multiplexed address and data bus
Provides access for COMMIN and COMMOUT register I/O transfers.	Provides a mechanism for memory-mapped I/O via non-burst mode. This can be used to access the programming interface of a PHY device or a CAM.
Provides CRC-32 generation and checking.	

SWAN processor memory access

The SWAN processor does not access Port1 or Port2 address space directly. The processor programs the Port DMA Controllers to perform a DMA read or write operation to move the data between the Cell Buffer RAM and Port1/Port2 address space. The SWAN processor initiates all port operations. It initiates a port transfer by executing a DMA instruction that writes a DMA command into the port's command queue.

DMA commands

A typical DMA command format is shown below:

DMA1R rsa/rsb rla [BC/#] [CRCX|CRCY] [POD] [ST]

A single DMA command can transfer of up to 255 bytes of information. The DMA command specifies:

- The transfer's starting address in memory (from registers rsa and rsb)

- The transfer's starting address in the Cell Buffer RAM (from register rla)

- The size of the transfer (from BC/#, or if no BC/# value is specified, from the Alternate Byte Count/ID register, R52)

- The direction of the transfer (Read or Write) (from the choice of DMA1*R* or DMA1*W*, for example)

- A series of instruction field options (IFOs) that control certain aspects of the transfer (BC/#, CRCX|CRCY, ST, POD)

For more information about the DMA commands, see page 283.

Instruction field options supported with the DMA instruction

The Port1 instruction field options supported within the DMA instruction include Byte Count (BC/#), Cyclical Redundancy Check (CRCX or CRCY), Silent Transfer (ST), and Post-DMA Operation Directive (POD). If no BC/# is specified in the command line, the command executes using a subset of the options (BC/#, CRX/CRY) from the Alternate Byte Count/ID register, R52.

The Port2 instruction field options supported within the DMA instruction include Byte Count (BC/#) and Post-DMA Operation Directive (POD). If no BC/# is specified in the command line, the command executes using the byte count (BC/#) field from the Alternate Byte Count/ID register, R52.

Detailed descriptions for instruction field options supported with Port1 and Port2 DMA instructions appear in the sections cited in the following table.

IFO	*For Further Information, See*
BC	"The Byte Count instruction field option (BC)" on page 286
CRCX, CRCY	"CRC partial result registers and the CRCX/CRCY instruction field option" on page 103 (Port1 only)
ST	"Silent transfers" on page 105
POD	"Post-DMA Operation Directives (PODs)" on page 109

The Port DMA command queues

The Port1 and Port2 DMA Controllers each contain a two-deep command queue. The MXT3010 processor contains an additional single-entry command queue for each Port1 and Port2 interface. With these queues, software can have up to three DMA Controller commands outstanding to each port simultaneously.

Port1 and Port2 DMA command queues

The Port1 and Port2 command queues each have two stages referred to as the queue stage and the active stage. Figure 43 shows the DMA command queues for the MXT3010.

FIGURE 43.DMA command queues for the MXT3010EP
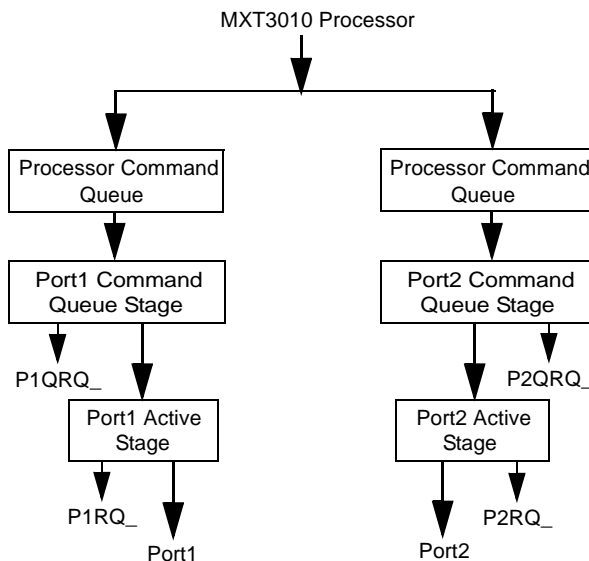
The DMA command information generated by a DMA read or write instruction is written into the selected SWAN command queue. The command is transferred into the associated DMA controller's command queue as soon as the queue is available. If the DMA controller's active stage is not busy, the DMA command is transferred from the command queue stage to the active stage and a port bus DMA operation begins.

If, however, the DMA active stage is busy, the operation remains in either the SWAN or port DMA Controller queue stage until the active operation completes. One cycle after the active operation completes, the DMA command queue operation is transferred into the active stage, and a new operation begins. If a port DMA Controller command is issued while the SWAN command queue is full, the processor stalls until the active operation finishes.

Bus parking

The QRQ_ output signal is asserted whenever a command is present a port's DMA queue. An external arbiter can use this signal to allow the MXT3010 to maintain ownership of a bus until all commands are drained from both the active and queue stages of the port DMA Controller. The arbiter does this by not responding to requests from other devices as long as there is a QRQ_ assertion on the port currently being serviced. Allowing the MXT3010 to maintain bus ownership in this fashion is referred to as "bus parking". See "Port2 bus parking" on page 158.

Testing DMA Controller queues with the ESS bits

Software can monitor the status of the DMA Controller command queues by testing External State Signals (ESS) 13 and 11 for Port1 and ESS12 and 14 for Port2. Table 20 and Table 21 show how ESS status bits are used to indicate DMA Controller status.

TABLE 20.   ESS Bits for DMA Controller status

ESS bit	State	Function
11	0	Port1 DMA Controller queue stage and active stage empty
	1	Port1 DMA Controller queue stage or active stage busy
12	0	Port2 DMA Controller queue stage and active stage empty
	1	Port2 DMA Controller queue stage or active stage busy
13	0	Port1 DMA Controller queue stage empty
	1	Port1 DMA Controller queue stage busy
14	0	Port2 DMA Controller queue stage empty
	1	Port2 DMA Controller queue stage busy

TABLE 21.   Example of DMA Controller status bit utilization

ESS 13 & 11	Port1 queue status
00	Controller queue stage and active stage empty
01	Controller queue stage empty; active stage busy
10	Invalid combination
11	Controller queue stage busy; active stage busy

Branch instructions can be used to control program flow based on the status of these bits. For more information, see "Branch Instructions" on page 261.

Port Controller features

The Cyclical Redundancy Check 32 generator for Port1

A CRC32 generator is provided in the Port1 DMA Controller to generate and to check AAL5 CRC32 polynomials during segmentation and reassembly operations. The CRC32 circuit generates a CRC32 for a Convergence Sublayer (CS) Protocol Data Unit (PDU) as it is transferred between the host memory and the Cell Buffer RAM. The CRC32 generator operates on data 16-bits at a time.

CRC partial result registers and the CRCX/CRCY instruction field option

To support pipelined DMA operations, the MXT3010 implements two CRC32 partial result registers, CRC32PRX (R44, R45), and CRC32PRY (R46, R47). Each DMA instruction specifies, via an instruction field option (IFO), whether a CRC32 calculation occurs and if so, which of the two CRC32 partial result registers to use. Specification of the CRCX/CRCY IFO is summarized in Table 22.

TABLE 22. Specification of the CRCX/CRCY instruction field option

IFO	*Action*
none	CRC32 partial result registers are not modified
CRCX	A CRC32 partial result is generated based on the CRC32PRX register's value and the result is deposited into CRC32PRX (R44/R45).
CRCY	A CRC32 partial result is generated based on the CRC32PRY register's value and the result is deposited into CRC32PRY (R46/R47).

Using partial result registers

To generate CRC32, a program must initialize the CRC32 logic with any prior partial result before the DMA data transfer begins. For the first cell of a CS-PDU, initialize the selected partial result register to 0xFFFFFFFF. At the completion of the

DMA transfer, read the CRC32 partial result from the selected partial result register and save those results in the Channel Descriptor to use the next time that a cell arrives. On transmits, for the last cell of an AAL5 CS-PDU, invert the partial result before placing it into the last four bytes of the cell. On receives, the program can test for a CRC32 error by testing the appropriate CRC32 error bit in the Sparse Event Register (R57) when the final DMA transfer operation finishes.

Pipelined operation

Although the use of command queueing and pipelined DMA operations requires that care be taken with CRC32 partial generation, these operations greatly enhance system performance. Using dual partial result registers, firmware can keep the DMA command queue full by managing the CRC32 partial result registers and the CRC32X/CRC32Y instruction field option bits of the DMA command.

Cyclical Redundancy Check operation acceleration

The registers for the Cyclical Redundancy Check (CRC) and the MXT3010 instruction Store Register Halfword (SRH) accelerate the handling of CRC results during AAL5 packet segmentation or reassembly. Direct Memory Access operations function independently of MXT3010 code execution once they have been started. Because of this functional independence, firmware can process the next channel descriptor in parallel with the DMA transfer (and CRC accumulation) of the previous channel as soon as the DMA operation has been committed for that previous channel. This parallelism provides processing time to the SWAN that might otherwise be wasted waiting for the transfer to complete. However, the program must still save the results of the partial CRC accumulation at the conclusion of a DMA transfer in the previously serviced Channel Descriptor.

CRCX and CRCY
address holding
registers

At the time that a DMA read or write operation with CRCX or CRCY indicated is initiated to Port1, the MXT3010 automatically stores the address contained in the internal Fast Memory Link Address register into one of two temporary holding registers – either that for CRCX operations or that for CRCY operations. Typically, the address stored is the current Channel Descriptor address.

Upon completion of the DMA transfer, the SWAN instruction, SRH, writes the contents of the CRC partial result registers (R44/ R45 or R46/ R47) to Fast Memory using the address contained in either the CRCX holding register or the CRCY holding register as the base address for the transfer. The programmer must specify an offset with the SRH instruction to place the partial results at the appropriate field within the Channel Descriptor.

Silent transfers

On some occasions, it is desirable to perform CRC calculations on data that did not traverse Port1.

- For LAN emulation purposes, the MXT3010 program may need to add a header to a message from Port1 memory before transmitting the message.

- In AAL5, the MXT3010 program may need to add a trailer to a message from Port1 memory before transmitting the message.

The MXT3010 provides this capability via the *silent transfer* instruction field option. When a Port1 DMA instruction is issued with the silent transfer (ST) option specified in the command line, data is transferred into the Port1 CRC logic, and the CRCX or CRCY partial result is updated, as selected by the CRCX/ CRCY instruction field option. During a silent transfer, the Port1 state machine operates with the same timing as an ordi-

nary Port1 DMA transfer (see Figure 45 on page 119), but no
P1QRQ_ or P1RQ_ signals are generated, the external arbiter
does not manipulate P1ASEL_ or P1TRDY_, and data is not
transferred on the Port1 bus.

Example 1 of
silent transfer use

Transmission of a typical AAL5 end of message cell typically
involves at least three DMA instructions, represented by the fol-
lowing pseudocode:

DMA1R, 40 bytes, CRCX   ;   This moves 40 bytes of data into the Cell
                                          ;   Buffer RAM in preparation for transmission.
                                          ;   A CRC is accumulated on this data.

DMA1W, 4 bytes, CRCX, ST ;   This instruction uses an rla value that points
                                          ;   to the UU, CPI, and length bytes at byte 40.
                                          ;   This instruction incorporates previously
                                          ;   written UU, CPI, and length information
                                          ;   into the CRC without modifying the contents
                                          ;   of the Cell Buffer RAM.

DMA1R, CRCX, ST, POD   ;   This instruction complements the CRC and
                                          ;   writes the result to the location specified in
                                          ;   rla (typically byte 44 of the cell). The POD
                                          ;   option increments TXBUSY, informing the
                                          ;   UTOPIA transmitter that a new cell in the
                                          ;   Cell Buffer RAM is ready for transmission.
                                          ;   (See "The TXBUSY counter" on page 84.)

Example 2 of
silent transfer use

It is occasionally useful to provide a silent transfer without CRC
so that the queue state can be predicted. It may be desirable
under some circumstances to fill the SWAN processor's DMA
command queue and intentionally stall the processor. If the com-
mand queue is full, a processor stall can be achieved by intro-
ducing an additional DMA request. In this case, since a stall is
desired rather than a real DMA action, a zero byte silent transfer
should be specified, as such a request reaches the command
queue, but does not reach the designated port queue.

Post-increment option on rla operations

The target rla register can automatically increment when the DMA transfer is completed with the DMA Plus instruction. The increment is 64 modulo 512. If eight 64-byte Cell Buffer registers are used, this saves the SWAN processor the code needed to advance the rla register to the next cell buffer in Cell Buffer RAM following each DMA transfer.

DMA Plus instructions

Two steps are necessary to utilize the rla increment option:

1. To enable the rla increment option, set mode bit 5 in the Configuration Register (R42).

2. Create a DMA Plus instruction by adding a plus sign to any DMA instruction for which the rla increment option is desired. The DMA Plus instructions are DMA1R+, DMA1W+, DMA2R+, and DMA2W+.

If the rla increment mode is not enabled, do not use the DMA Plus instructions. For more information on the DMA Plus instructions, see "The RLA increment bit (i-bit)" on page 285.

Data alignment

The port DMA Controller operates on halfword-aligned data located locally, such as in the Cell Buffer RAM, and in host memory. On memory reads and writes, the local halfword address is specified in the chosen local address register, rla. The DMA controller programs the external halfword address into the DMA instruction. The DMA controller performs data alignment dynamically in those cases where the starting address of the source and destination locations are not halfword aligned.

Byte manipulations on Port1

The Port1 bus supports only word and halfword DMA writes, with P1HWE [1:0] being used as selects for the half-words. However, the P1 address leads (P1AD [31:0]) provide a byte address, and the P1 bus supports byte DMA reads. The DMA read operations can transfer even or odd byte counts and start or end transfers on even or odd boundaries in system memory or in Cell Buffer RAM.

Transfers that end on odd byte boundaries in the Cell Buffer RAM result in the single byte of the last halfword address being stored aside in anticipation of the next transfer. This byte is not accumulated in CRC calculations until the subsequent transfer is initiated. Multiple transfers must be certain to end with a half-word-aligned boundary to properly complete the CRC calculation. Therefore, the beginning and ending of any multi-burst cell accumulation process must occur on even byte boundaries. New CRC accumulations that do not begin on an odd byte address in Fast Memory might include some arbitrary stale byte in the transfer.

Table 23 shows the valid and invalid transfers to use for the first, mid-cell, and last transfer for any given cell during cell construction from system memory. Follow the rules in Table 23 to ensure that the cell accumulation begins and ends on proper boundaries.

TABLE 23. Valid and invalid first, mid-cell, and last transfers.

rla Start Address	*Byte Count*	*Use of this start address and byte count is valid for:*		
		First Transfer	*Mid-Cell Transfer*	*Last Transfer*
Even	Even	Yes	Yes	Yes
Even	Odd	Yes	Yes	No[a]
Odd	Even	No[b]	Yes	No[a]
Odd	Odd	No[b]	Yes	Yes

a. Causes cell to end on odd boundary

b. Causes cell to start on odd boundary

Post-DMA Operation Directives (PODs)

The port DMA Controllers support a feature referred to as Post-DMA Operation Directives, or PODs. PODs instruct the DMA controller to perform UTOPIA port counter manipulations when the operation ends. For instance, during reassembly, firmware can instruct the DMA controller to decrement the RXFULL counter by specifying the POD instruction field option in the DMA1W command.

When a POD is specified with a DMA write operation, the DMA controller decrements the UTOPIA port's RXFULL counter. (See "UTOPIA receiver counters" on page 78.) The RXFULL counter is used for UTOPIA RxENB_ assertion at the conclusion of the DMA operation. The CPU is finished with a received cell once the DMA command required to transfer the cell's SAR SDU to memory is written into the DMA queue. CRC32 partial results might still need handling.

If the POD directive is specified with a DMA read operation, the DMA controller increments the TXBUSY counter when the DMA operation finishes. (See "UTOPIA transmitter counters" on page 84.) The CPU is finished with a transmit cell once it initiates the data transfer from memory. The CPU must be sure that the appropriate command byte, ATM headers, and AAL Headers/Trailers (if any) are written into the cell holder before the data transfer completes.

Burst and non-burst operation (Port2)

A burst mode transfer includes an address cycle and one or more data cycles. Burst mode is used with high-speed synchronous transfer devices such as SRAMs. In contrast, a non-burst transfer includes an address cycle and a single data cycle that models

transfers on a typical asynchronous multiplexed bus. Non-burst mode is used with low-speed non-synchronous transfer devices such as PHYs, CAMs, and FLASH memories.

While Port1 supports only burst mode operations, Port2 supports both burst mode and non-burst mode operations. Port2 burst mode DMA operations proceed similarly to Port1 DMA burst operations. During non-burst transfers, the Port2 DMA controller can insert a programmable number of wait states and can also generate control signals that allow direct connection to external devices.

Selection of burst mode or non-burst mode operation is accomplished via the Port2 basic protocol. See "Port2 basic protocol" on page 137.

Port Operations

Port1 basic protocol

The Port1 DMA interface supports two transfer mechanisms: DMA burst-mode transfers initiated by the MXT3010 and communication register I/O transfers initiated by an external device. For information on DMA burst-mode transfers, see "Burst and non-burst operation (Port2)" on page 109. For information on communication register transfers, see "Communications" on page 177.

Figure 44 and Table 24 illustrate the correspondence between rsa/rsb register values and the Port1 bus signals for Port1 DMA transfers.

FIGURE 44.Diagram of Port1 DMA instruction bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

rsa — P1AD[31:16]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

rsb — P1AD[15:0]
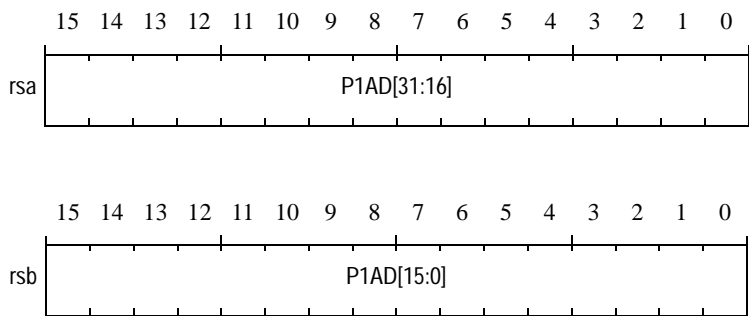
TABLE 24. Port 1 DMA instruction bit mapping

Reg	*Bits*	*Function*	*Port2 Bus*
rsa	15:0	Address	P1AD[31:16]
rsb	15:0	Address	P1AD[15:0]

Port1 control signals

Table 25 describes the signals that control Port1 transfers. Two additional status signals, CIN_BUSY and COUT_RDY, allow both an external device and the SWAN processor to determine the state of the COMMIN/COMMOUT register set.

Restrictions on Port1 Addressing

The address counter for Port1 does not increment across the boundary between P1AD[15] (rsb[15]) and P1AD[16] (rsa [00]). Therefore, firmware running on the MXT3010 must ensure that DMA transfers to/from Port1 do not cross 64K boundaries.

TABLE 25. Signals to control Port1 transfers

Signal	Purpose
P1QRQ_	When the Port1 state machine detects the presence of a command in the queue stage of the Port1 DMA command queue, the state machine asserts this signal to an external device. This provides an advance indication that P1RQ_ will soon be asserted.
P1RQ_	When the Port1 state machine is in the Idle state and detects the presence of a command in the active stage of the Port1 DMA command queue, the state machine asserts this signal to an external device. The external device responds by manipulating P1ASEL_ and P1TRDY_ to control a DMA transfer.
P1ASEL_	This signal is an input to the MXT3010 and is driven by an external device. The external device uses this signal to select between address and data cycles. The external device can also use this signal in conjunction with P1TRDY_ to deselect (tri-state) the Port1 DMA engine.
P1TRDY_	This signal is an input to the MXT3010 and is driven by an external device. The external device uses this signal to insert wait states. The external device can also use this signal in with P1ASEL_ to deselect (tri-state) the Port1 DMA engine.
P1RD	During a DMA transfer, this signal is an output driven by the MXT3010. During a communication register transfer, this signal is an input to the MXT3010 and is driven by an external device. In either use, this signal indicates whether the transfer is a read (1) or a write (0) transfer.
P1END_	This signal indicates the last cycle of a DMA operation.
P1AD[31:0]	This is a multiplexed, bi-directional 32-bit bus. Data is read into and out of the MXT3010 during DMA transfers and during communication register I/O transfers. P1AD [31] is the most significant bit, and P1AD[0] is the least significant bit.
P1HWE[1:0]	During data cycles, P1HWE[1] and P1HWE[0] act as Half Word Enables. If P1HWE[1] is asserted, P1AD[15:0] should contain valid data. If P1HWE[0] is asserted, P1AD[31:16] should contain valid data.
P1IRDY_	During DMA write data cycles, the MXT3010 asserts P1IRDY while it is sourcing valid data on P1AD[31:0]. During DMA read data operations, the MXT3010 asserts P1IRDY_ if it is able to sample P1AD[31:0] on the next rising edge of clock.
COMMSEL	This signal is an input to the MXT3010 and is driven by an external device. The external device uses this signal to perform communication register I/O.
P1ABORT_	This input signal causes the termination of the data transfer at the completion of the next data phase. It is used only by the P1 DMA engine, and the SWAN processor has no knowledge of P1ABORT_ signal indications.
CIN_Busy	Driven high when host writes COMMIN; cleared when MXT3010 reads COMMIN.
COUT_Ready	Driven high when MXT3010 writes COMMOUT; cleared when host reads COMMOUT.
LTN	This is an internal signal indicating that the Last Transfer will occur Next (LTN).

The Port1 control state machine

General information concerning DMA transfers

As indicated in "The Port DMA command queues" on page 100, the MXT3010EP asserts a port's RQ_ signal if that port has a DMA command active. Additionally, it asserts the associated QRQ_ signal if there is an additional DMA command enqueued behind the active command. The port's RD signal indicates whether the requested DMA transfer is to be a read or a write. Arbitration logic external to the MXT3010EP monitors these signals and, in the case of a shared bus, other requestors to determine whether to start a DMA transfer. Upon deciding to start a transfer, the external logic steps ASEL_ and TRDY_ through the various states of a DMA transfer, concluding with a *Last Transfer* during which the MXT3010EP dismisses the current RQ_ request, and during which the arbitration logic again determines subsequent bus utilization. The MXT3010EP re-asserts RQ_ at the conclusion of the "Clean Up" state (which follows *Last Transfer)* if a queued command exists at that time.

Port1 DMA read transfers

Table 26 shows the state table for Port1 DMA read transfers, and Figure 45 shows a sequence diagram for a DMA read transfer. The table and the figure are best understood by considering the function of the various inputs, outputs, and states for read transfers.

Inputs

- ASEL_ and TRDY_

  These inputs are manipulated by an external device to step the state machine through various states.

- LTN

  This input (Last Transfer Next) is an internal MXT3010EP signal based on the byte count. When asserted, it indicates that the next DMA transfer state will be the last.

Outputs

- P1AD

  This is a bi-directional address/data bus. It has four possible states: Out-Address, In-Data, In-X (Don't Care), and Tri-state.

- IRDY_

  When this output is asserted, the MXT3010EP will sample data on the rising edge of clock. Thus, in Table 26, IRDY_ is asserted only when P1AD is presenting *In-Data*.

- END_

  The END_ output is asserted by the MXT3010EP during the Last Transfer state.

Although not shown in Table 26, the state machine also has a COMMSEL input. During DMA transfers, the COMMSEL signal is low for all states shown. Please see "Communication register I/O transfers" on page 133 for COMMSEL high.

States

- Address 1, Automatic-turnaround, and Address 2

  The state machine differentiates between two types of Address state, Address 1 and Address 2.

  - When a DMA transfer begins, and the MXT3010EP samples both ASEL_ and TRDY_ as asserted (low), the MXT3010EP drives address information onto the P1AD bus; this is referred to as an *Address 1* state.

  - At some point after the Address 1 state begins, the external controller that drives the ASEL_ and TRDY_ leads will de-assert ASEL_ (high) while maintaining TRDY_ in the asserted state (low). This step prepares the MXT3010EP for data transfer. When the MXT3010EP is thus switched, it will interject an *Automatic-turnaround* state during which it will not

accept data (IRDY_ will be de-asserted (high)). The *Automatic-turnaround* state provides time for the MXT3010EP to turn off its bus drivers and for the external device to turn on its bus drivers.

- If the system using the MXT3010EP requires that address cycles be inserted during a DMA transfer at some point after data reads have begun, i.e. after the automatic -turnaround state, these are referred to as *Address 2* states. Address 2 states differ from Address 1 states, as Address 2 states require that the external controller manipulate ASEL_ and TRDY_ appropriately to insert Tri-state intervals between the Address 2 states and any data read states to allow time for the bus direction to be changed.

• Data Read

During a Data Read, an external device drives data onto the P1AD bus and the MXT3010EP reads that data. Thus, the P1AD column in the state table shows *In-Data*, and the IRDY_ column shows assertion (low) indicating that the MXT3010EP will read the data. There are three common cases for what happens after a Data Read:

- If ASEL_ remains de-asserted (high) and TRDY_ remains asserted (low), the Data Read is followed by another Data Read.
- If ASEL_ remains de-asserted (high) and TRDY_ is de-asserted (high), the Data Read is followed by a Data Wait.
- If ASEL_ remains de-asserted (high) and TRDY_ remains asserted (low), and LTN is asserted (high), the Data Read is followed by a Last Transfer.

There are two other cases for what happens after a Data Read, but these are used less often than the three listed above.

- If the states of ASEL_ and TRDY_ are switched to ASEL_ asserted (low) and TRDY_ de-asserted (high), the Data Read is followed by Tri-state (Data)[1] and all outputs are tri-stated.

- If ASEL_ is asserted (low) and TRDY_ remains asserted (low), the Data Read is followed by an Address cycle. To avoid bus contention when inserting an Address cycle, it is preferable that ASEL_ /TRDY_ be low/high such that the Data Read is followed by a Tri-state (Data) (see previous paragraph), and that the Tri-state (Data) then be followed by an Address 2 state (ASEL_/TRDY_ both low).

- Data Wait

  During a Data Wait, an external device drives data onto the P1AD bus, but the MXT3010EP ignores that data. Thus, the P1AD column in the state table shows *In-X*, and the IRDY_ column shows de-assertion (high) indicating that the MXT3010EP will not read the data. There are three common cases for what happens after a Data Wait:

  - If ASEL_ remains de-asserted (high) and TRDY_ remains de-asserted (high), the Data Wait is followed by another Data Wait.
  - If TRDY_ is asserted (low) and LTN is de-asserted (low), the Data Wait is followed by a Data Read.
  - If TRDY_ is asserted (low) and LTN is asserted (high), the Data Wait is followed by a Last Transfer.

  There are two other cases for what happens after a Data Wait, but these are used less often than the three listed above.

  - If the states of ASEL_ and TRDY_ are switched to ASEL_ asserted (low) and TRDY_ de-asserted (high), the Data Wait is followed by Tri-state (DW) and all outputs are tri-stated.
  - If ASEL_ is asserted (low) and TRDY_ remains asserted (low), the Data Wait is followed by an Address cycle.

---

1. The state machine keeps track of several versions of the tri-state condition. For example, Tri-state (Data) refers to a tri-state condition entered from the Data Read state. See "Tri-state" on page 117.

- Tri-state

  During a tri-state condition, all outputs are tri-state. This condition is always entered whenever ASEL_ is asserted (low) and TRDY_ is de-asserted (high). The state machine maintains separate versions of the tri-state condition depending upon the state from which the state machine entered the tri-state condition. The versions are Tri-state (Address 1), Tri-state (Address2), Tri-state (Automatic-turn-around), Tri-state (Data), Tri-state (Data Wait), Tri-state (Last Transfer), Tri-state (Clean-up), and Tri-state (Turn-around Wait). As shown in Table 26, four of these states are identical, transitioning to Data Read or Last Transfer depending upon the state of the LTN input.

- Last Transfer

  Last Transfer is a special case of Data Read. It differs from Data Read in three ways:
  - The END_ output is asserted (low) during Last Transfer
  - During this state, external logic decides how to condition the ASEL_ and TRDY_ leads during the Clean Up state that follows. This, in turn, will determine the state that follows the Clean Up state.
  - It is followed by the Clean Up state.

- Clean Up

  During the Clean Up state, the IRDY_ and END_ outputs are de-asserted (high) and P1RQ_ is de-asserted (high). The state which follows Clean Up is determined by the condition of ASEL_ and TRDY_. As indicated above, the condition of these inputs was determined by the external arbitration logic during the Last Transfer state.

TABLE 26. State table for the Port1 DMA burst read state machine

Table Line	Input Signals			Current State	Next State	Outputs in the Next State		
	*ASEL_*	*TRDY_*	*LTN*			*P1AD*	*IRDY_*	*END_*
1	L	H	X[a]	Any	Tri-state (current_state)	Tri-state		
2	L	L	X	Any pre Auto-turnaround[b]	Address 1	Out-Addr	H	H
3	L	L	X	Any post Auto-turnaround	Address 2	Out-Addr	H	H
4	H	L	X	Address 1	Auto-turnaround[c]	Tri-state	H	H
5	H	L	L	Address 2	Data Read	In-Data	L	H
6	H	L	H	Address 2	Last Transfer	In-Data	L	L
7	H	L	L	Auto-turnaround	Data Read	In-Data	L	H
8	H	L	H	Auto-turnaround	Last Transfer	In-Data	L	L
9	H	L	L	Data Read	Data Read	In-Data	L	H
10	H	L	H	Data Read	Last Transfer	In-Data	L	L
11	H	L	X	Auto-turnaround Wait	Auto-turnaround[b]	Tri-state	H	H
12	H	L	L	Data Wait	Data Read	In-Data	L	H
13	H	L	H	Data Wait	Last Transfer	In-Data	L	L
14	H	L	X	Last Transfer	Clean Up	In-X	H	H
15	H	L	X	Clean Up, Tri-state (Last Transfer)	Auto-turnaround Wait	Tri-state	H	H
16	H	L	X	Tri-state (Address 1, Clean Up, Auto-turnaround Wait)	Auto-turnaround[b]	Tri-state	H	H
17	H	L	L	Tri-state (Address 2, Auto-turnaround, Data Read, or Data Wait)	Data Read	In-Data	L	H
18	H	L	H	Tri-state (Address 2, Auto-turnaround, or Data Wait)	Last Transfer	In-Data	L	L
19	H	H	X	Address 1, Auto-turnaround Wait, Tri-state (Address 1, Auto-turnaround Wait)	Auto-turnaround Wait	Tri-state	H	H
20	H	H	X	Address 2, Auto-turnaround, Data Read, Data Wait, Clean Up, Tri-state (Auto-turnaround, Address2, Data, Data Wait, Last Transfer, or Clean Up)	Data Wait	In-X	H	H
21	H	H	X	Last Transfer	Clean Up	In-X	H	H

a. X = don't care

b. A pre Auto-turnaround state is any state between the most recent Last Transfer state and the Auto-turnaround state of a DMA transfer. A post turnaround state is any state between the most recent auto turnaround and the next Last Transfer state.

c. This Auto-turnaround will be Auto-turnaround Wait if there is no RQ_ assertion at this time.

FIGURE 45.Port1 DMA Read transfer with a Wait state
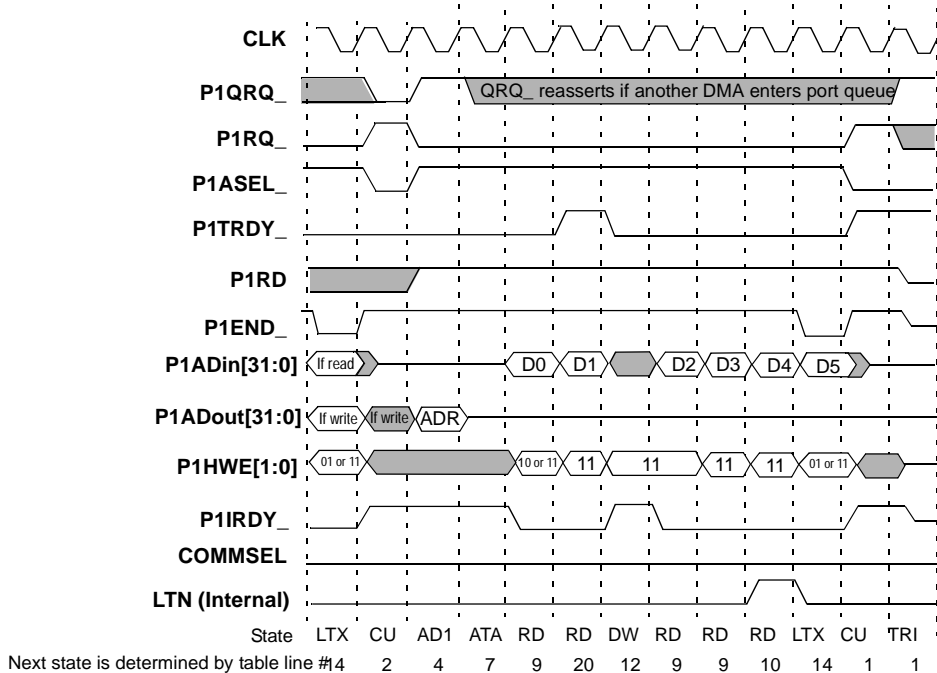
Figure 45 shows the Last Transfer (LTX) and Clean Up (CU) states of a previous DMA read or write transfer. During the Last Transfer state, the external logic that controls the ASEL_ and TRDY_ leads makes a decision as to whether it should:

1. Allow a COMM SEL transfer (having detected a communication request)

2. Prepare for another DMA transfer (having detected P1QRQ_ asserted)

3. Relinquish the bus by entering a tri-state condition

4. Perform some other type of bus operation

Having decided on the appropriate course of action, the external control logic conditions ASEL_ and TRDY_ at the beginning of the Clean Up state so that the Port1 state machine will enter the desired state after the Clean Up state.

In the example shown in Figure 45, the external logic elects to perform a new DMA transfer, and drives both ASEL_ and TRDY_ to the asserted (low) state during the Clean Up state. With ASEL_ and TRDY_ low, the Clean Up state qualifies as *Any pre Auto-turnaround* state in the Table 26 state table. Thus, when the state machine samples the ASEL_ and TRDY_ leads, line 2 of the state table causes the next state to be Address 1 (AD1).

During Address 1, the MXT3010EP puts address information onto the P1AD leads. IRDY_ and END_ are high. If, during Address 1, the external logic drives ASEL_ high while leaving TRDY_ low, the state machine samples that condition, and line 4 of the state table causes the next state to be *Auto-turnaround*.

During Auto-turnaround, the MXT3010EP prepares the P1AD leads for data transfer. IRDY_ and END_ are still high. With ASEL_ still high and TRDY still low during the Auto-turnaround state, line 7 of the state table causes the next state to be a Data Read state.

During Data Read, an external device places data on the P1AD leads, and the MXT3010EP asserts the IRDY_ lead low to indicate it is going to sample that data. Since ASEL_ is still high and TRDY_ is low, the state machine samples that condition, and line 9 of the state table causes the next state to be a Data Read state.

The second Data Read in the above example is identical to the previous Data Read, except that the external logic has driven TRDY_ high. Since TRDY_ has gone high, line 20 of the state table causes the next state to be a Data Wait state. The use of a Data Wait state is optional. It is shown in this figure and subsequent figures only to illustrate the waveforms that occur during a Data Wait.

During Data Wait, the incoming data is ignored ("In-X"), and the IRDY_ lead goes high to indicate that the MXT3010EP is not going to sample the data. In the example, TRDY_ returns is returned to the low state, the state machine samples that condition, and line 12 of the state table causes the next state to be a Data Read state.

During Data Read, an external device places data on the P1AD leads, and the MXT3010EP asserts the IRDY_ lead (low) to indicate it is going to sample that data. Since ASEL_ is still high and TRDY_ is low, line 9 of the state table causes the next state to be a Data Read state.

This Data Read is similar to the previous Data Reads. Since ASEL_ is still high and TRDY_ is low, line 9 of the state table causes the next state to be a Data Read state.

During this Data Read, the LTN signal (generated by the byte count logic within the MXT3010EP) is asserted. Therefore, line 10 of the state table causes the next state to be a Last Transfer (LTX).

Last Transfer is similar to a Data Read, except that the END_ output is asserted (low). Line 14 of the state table causes the next state to be a Clean Up (CU) state. During the Last Transfer state, the external controller decides what to do with the ASEL_ and TRDY_ inputs during the Clean Up state. The state that follows the Clean Up state depends upon that decision. In this example, the decision was to tri-state the bus. Thus, during Clean Up, the external controller has driven ASEL_ low and TRDY high. When the state machine samples that condition, line 1 of the state table causes the next state to be Tri-state (CleanUp).

FIGURE 46.Port1 DMA Read transfer without a Wait state
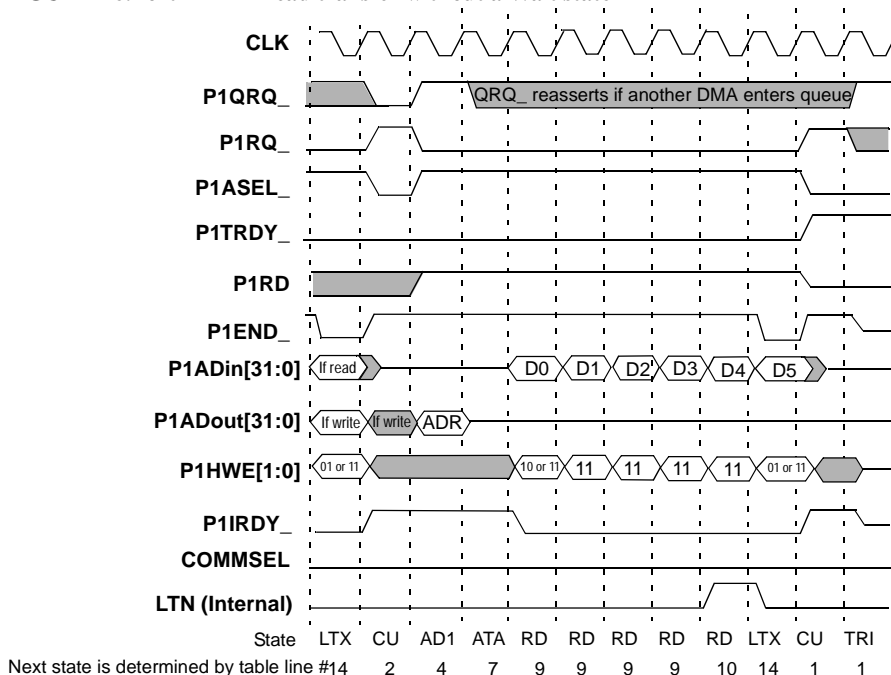
Figure 46 is similar to Figure 45, but without a wait state. The description of the figure is identical, except that there are no transitions controlled by lines 20 or 12 of the state table, as P1TRDY_ remains asserted (low) throughout the transfer. Rather, the read process continues to be determined by line 9 of the state table until LTN is asserted and state table line 10 applies, causing the next state to be Last Transfer (LTX).

Port1 DMA write transfers

Table 27 shows the state table for Port1 DMA write transfers, and Figure 47 shows a sequence diagram for a DMA write transfer. The table and the figure are best understood by considering the function of the various inputs, outputs, and states for write transfers.

Inputs

- ASEL_, TRDY_, LTN

  These inputs have the same definitions as shown page 113.

Outputs

- P1AD

  This is a bi-directional address/data bus. It has four possible states: Out-Address, Out-Data, Out-X (Don't Care), and Tri-state. In contrast to a DMA read, when the bus is changed from an address mode (outward) to a write data mode (outward), no intervening states are required.

- IRDY_

  When this output is asserted, the MXT3010EP is sourcing valid data. Thus, in Table 27, IRDY_ is asserted only when the next state is Out-Data.

- END

  The END_ output is asserted by the MXT3010EP during the Last Transfer state. This output can be used by any external logic that requires this information.

Although not shown in Table 27, the state machine also has a COMMSEL input. During DMA transfers, the COMMSEL signal is low for all states shown. Please see "Communication register I/O transfers" on page 133 for COMMSEL high.

States

- Address

  In contrast to a DMA read, the write state machine has only one state for Address. When a DMA cycle begins, and the MXT3010 EP samples both ASEL_ and TRDY_ as asserted (low), the MXT3010EP drives address information onto the P1AD bus; this is referred to as an *Address* state.

- Data Write

  During a Data Write, the MXT3010EP drives data onto the
  P1AD bus and an external device reads that data. Thus, the
  P1AD column in the state table shows *Out-Data*, and the
  IRDY_ column shows assertion (low) indicating that the
  MXT3010EP is sourcing valid data. There are three com-
  mon cases for what happens after a Data Write:

  - If ASEL_ remains de-asserted (high) and TRDY_
    remains asserted (low), the Data Write is followed by
    another Data Write.
  - If TRDY_ is de-asserted (high), the Data Write is
    followed by a Data Wait.
  - If LTN is asserted (high), the Data Write is followed by
    a Last Transfer.

  There are two other cases for what happens after a Data
  Write, but these are used less often than the three listed
  above.

  - If the states of ASEL_ and TRDY_ are switched to
    ASEL_ asserted (low) and TRDY_ de-asserted (high),
    the Data Write is followed by Tri-state (Data)[1].
  - If ASEL_ is asserted (low) and TRDY_ remains
    asserted (low), the Data Write is followed by an
    Address cycle.

- Data Wait

  During a Data Wait, the MXT3010EP drives data onto the
  P1AD bus, but the external device ignores that data. Thus,
  the P1AD column in the state table shows *Out-X*, and the
  IRDY_ column shows de-assertion (high) indicating that the
  external device should not read the data. There are three
  common cases for what happens after a Data Wait:

---

1. The state machine keeps track of several versions of the tri-state condition.
   For example, Tri-state (Data) refers to a tri-state condition entered from
   the Data Write state.

- If ASEL_ remains de-asserted (high) and TRDY_ remains de-asserted (high), the Data Wait is followed by another Data Wait.
- If TRDY_ is asserted (low) and LTN is de-asserted (low), the Data Wait is followed by a Data Write.
- If TRDY_ is asserted (low) and LTN is asserted (high), the Data Wait is followed by a Last Transfer.

There are two other cases for what happens after a Data Wait, but these are used less often than the three listed above.

- If the states of ASEL_ and TRDY_ are switched to ASEL_ asserted (low) and TRDY_ de-asserted (high), the Data Wait is followed by Tri-state (Data Wait) and all outputs are tri-stated.
- If ASEL_ is asserted (low) and TRDY_ remains asserted (low), the Data Wait is followed by an Address cycle.

• Tri-state

During a tri-state condition, all outputs are tri-state. This condition is always entered whenever ASEL_ is asserted (low) and TRDY_ is de-asserted (high). The state machine maintains separate versions of the tri-state condition depending upon the state from which the state machine entered the tri-state condition. The versions are Tri-state (Address), Tri-state (Data), Tri-state (Data Wait), Tri-state (Last Transfer), and Tri-state (Clean-up). As shown in Table 27, three of these states are identical, transitioning to Data Read or Last Transfer depending upon the state of the LTN input.

TABLE 27. State table for the Port1 DMA burst write state machine

Table Line	*Input Signals*			*Current State*	*Next State*	*Outputs in the Next State*		
	*ASEL_*	*TRDY_*	*LTN*			*P1AD*	*IRDY_*	*END_*
1	L	H	X	Any	Tri-state (current_state)	Tri-state		
2	L	L	X	Any	Address	Out-Addr	H	H
3	H	L	L	Address	Data Write	Out-Data	L	H
4	H	L	H	Address	Last Transfer	Out-Data	L	L
5	H	L	L	Data Write	Data Write	Out-Data	L	H
6	H	L	H	Data Write	Last Transfer	Out-Data	L	L
7	H	L	L	Data Wait	Data Write	Out-Data	L	H
8	H	L	H	Data Wait	Last Transfer	Out-Data	L	L
9	H	L	X	Last Transfer	Clean Up	Out-X	H	H
10	H	L	X	Clean Up	Idle[a]	Tri-state	H	H
11	H	L	L	Idle, Tri-state (Address, Data Write, or Data Wait)	Data Write	Out-Data	L	H
12	H	L	H	Tri-state (Address, Data Write, or Data Wait)	Last Transfer	Out-Data	L	L
13	H	L	X	Tri-state (Last Transfer, Clean Up)	Data Write	Out-X	H	H
14	H	H	X	Address, Data Write, Data Wait, Clean Up, Tri-state (Address, Data Write, Data Wait, Last Transfer, or Clean Up)	Data Wait	Out-X	H	H
15	H	H	X	Last Transfer	Clean Up	Out-X	H	H

a. The Idle state will be maintained indefinitely if there is no RQ_ assertion.

FIGURE 47.Port1 DMA Write transfer with a Wait state
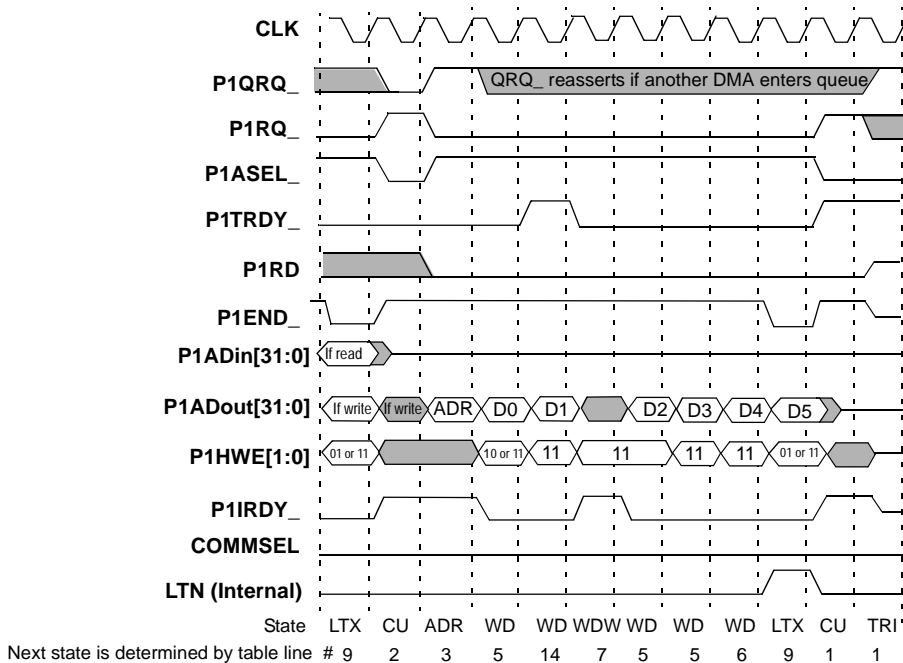
Figure 47 shows the Last Transfer (LTX) and Clean Up (CU) states of a previous DMA read or write transfer. During the Last Transfer state, the external logic that controls the ASEL_ and TRDY_ leads makes a decision as to whether it should:

1. Allow a COMM SEL transfer (having detected communication request)

2. Prepare for another DMA transfer (having detected P1QRQ_ asserted)

3. Relinquish the bus by entering a tri-state condition

4. Perform some other type of bus operation

Having decided on the appropriate course of action, the external control logic conditions ASEL_ and TRDY_ at the beginning of the Clean Up state so that the Port1 state machine will enter the desired state after the Clean Up state.

In the example shown in Figure 47, the external logic elects to perform a new DMA transfer, and drives both ASEL_ and TRDY_ to the asserted (low) state during the Clean Up state. With ASEL_ and TRDY_ low, the Clean Up state qualifies as *Any* state in the Table 27 state table. Thus, when the state machine samples the ASEL_ and TRDY_ leads, line 2 of the state table causes the next state to be Address (ADR).

During the Address state, the MXT3010EP puts address information onto the P1AD leads. IRDY_ and END_ are high. If, during the Address state, the external logic drives ASEL_ high while leaving TRDY_ low, the state machine samples that condition, and line 3 of the state table causes the next state to be Data Write.

During Data Write, the MXT3010EP places data on the P1AD leads, and the MXT3010EP asserts the IRDY_ lead low to indicate that it has done so. Since ASEL_ is still high and TRDY_ is low, the state machine samples that condition, and line 5 of the state table causes the next state to be a Data Write state.

The second Data Write in the above example is identical to the previous Data Write, except that the external logic has driven TRDY_ high. Since TRDY_ has gone high, line 14 of the state table causes the next state to be a Data Wait state. The use of a Data Wait state is optional. It is shown in this figure and subsequent figures only to illustrate the waveforms that occur during a Data Wait.

During Data Wait, the outgoing data should be ignored ("Out-X"), and the IRDY_ lead goes high to indicate that the MXT3010EP does not guarantee the data. In the example,

TRDY_ returns is returned to the low state, the state machine samples that condition, and line 7 of the state table causes the next state to be a Data Write state.

During Data Write, the MXT3010EP places data on the P1AD leads, and the MXT3010EP asserts the IRDY_ lead low to indicate that it has done so. Since ASEL_ is still high and TRDY_ is low, the state machine samples that condition, and line 5 of the state table causes the next state to be a Data Write state.

This Data Write is similar to the previous Data Writes. Since ASEL_ is still high and TRDY_ is low, line 5 of the state table causes the next state to be a Data Write state.

During this Data Write, the LTN signal (generated by the byte count logic within the MXT3010EP) is asserted. Therefore, line 6 of the state table causes the next state to be a Last Transfer (LTX).

Last Transfer is similar to a Data Write, except that the END_ output is asserted (low). Line 9 of the state table causes the next state to be a Clean Up (CU) state. During the Last Transfer state, the external controller decides what to do with the ASEL_ and TRDY_ inputs during the Clean Up state. The state that follows the Clean Up state depends upon that decision. In this example, the decision was to tri-state the bus. Thus, during Clean Up, the external controller has driven ASEL_ low and TRDY high. When the state machine samples that condition, line 1 of the state table causes the next state to be Tri-state (CleanUp).

FIGURE 48.Port1 DMA Write transfer without a Wait state
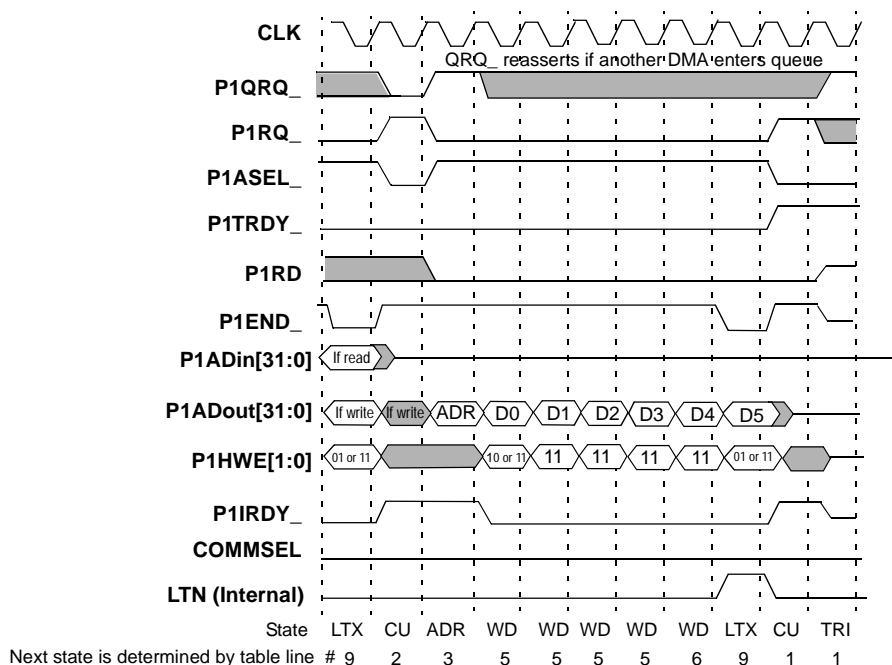
Figure 48 is similar to Figure 47, but without a wait state. The description of the figure is identical, except that there are no transitions controlled by lines 14 or 7 of the state table, as P1TRDY_ remains asserted (low) throughout the transfer. Rather, the write process continues to be determined by line 5 of the state table until LTN is asserted and state table line 6 applies, causing the next state to be Last Transfer (LTX).

Multiple Port1 Read and Write Transfers

Figure 45 and Figure 47 each show the conclusion of a DMA transfer followed by the read or write DMA transfer being described. In each case, the commencement of a DMA transfer depends upon the states of QRQ_, RQ_, ASEL_ and TRDY_ during the Clean Up phase of the preceding bus cycle. Thus, to

create timing diagrams representing an arbitrary sequence of Port1 reads and writes, photocopy Figure 49 and Figure 50 below, and cut them on the heavy lines shown. Paste them together to create the desired diagram.

FIGURE 49.Cut-and-Paste Version of Port1 Read
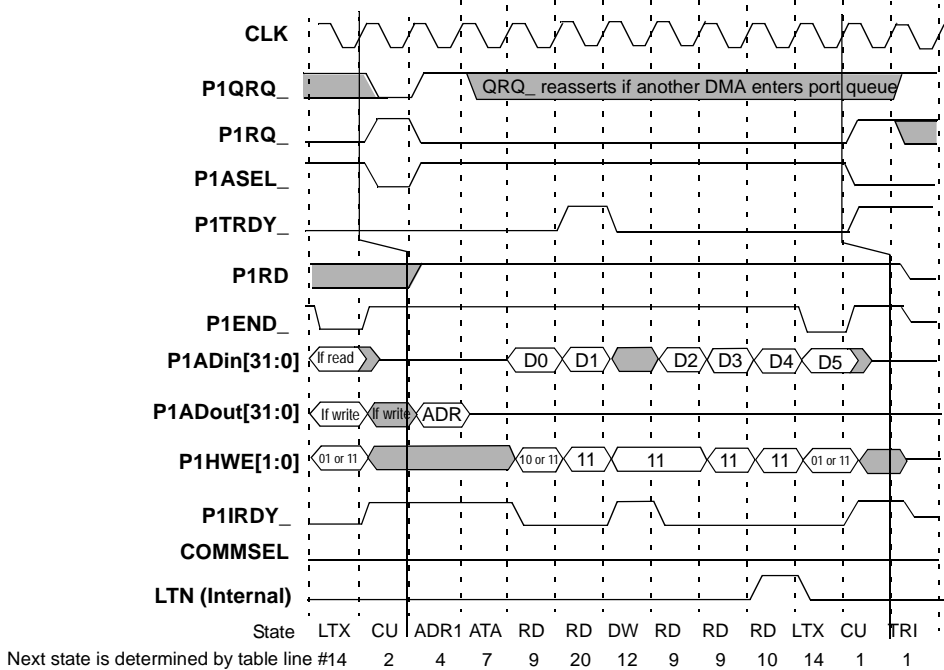
FIGURE 50. Cut-and-Paste Version of Port1 Write
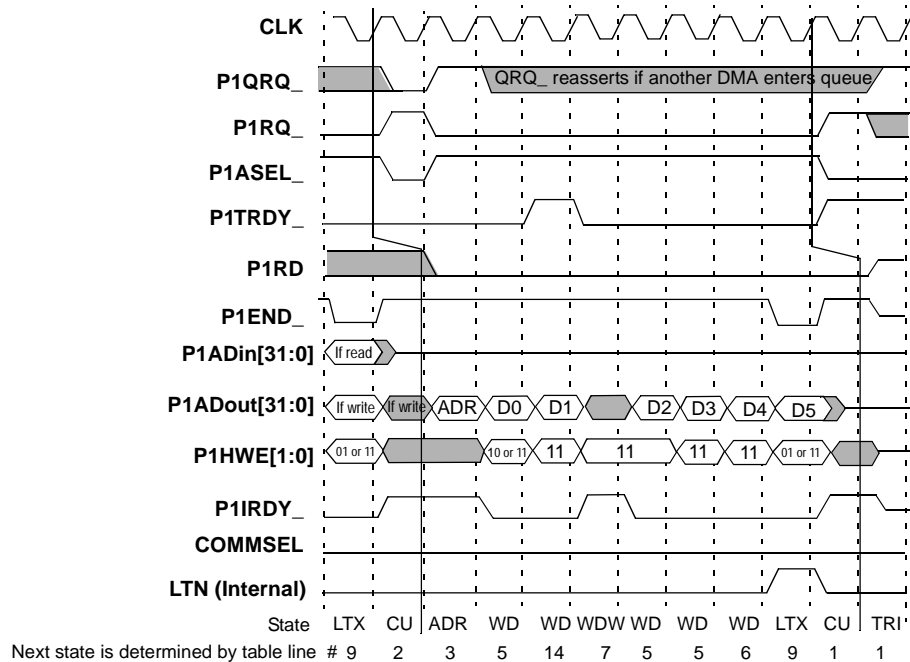
Communication register I/O transfers

In addition to monitoring bus request signals from the
MXT3010EP and other devices, arbitration logic external to the
MXT3010EP monitors external requests for Communication
Register I/O transfers. Upon deciding to start a Communication
Register I/O transfer, the external logic asserts ASEL_ (low) and
de-asserts TRDY_ (high) to bring the Port1 bus into a tri-state
condition. The external logic then asserts the COMMSEL input
of the MXT3010EP. The P1RD signal, driven by the external
device, determines whether the I/O transfer is a read or write.
Since the state tables for COMMSEL reads and COMMSEL
writes are so brief, Table 28 shows the combined state table for
both reads and writes. Figure 51 shows a sequence diagram for
a typical COMMIN write followed by a COMMOUT read.

TABLE 28. State table for Port1 communication I/O state machine

Table Line	ASEL_	TRDY_	COMMSEL	P1RD[a]	*Current State*	*Next State*	*Outputs in the Next State* P1AD
1	L	H	L	X	Any	Tri-state (current_state)	Tri-state
2	L	H	H	H	Tri-state (current_state)	Comm Out Read 1	Tri-state
3	L	H	H	H	Comm Out Read 1	Comm Out Read 2	Tri-state
4	L	H	H	H	Comm Out Read 2	Comm Out Data Valid	Data
5	L	H	H	H	Comm Out Data Valid	Comm Out Data Valid	Data
6	L	H	H	L	Tri-state (current_state)	Comm In Write	Tri-state[b]
7	L	H	H	L	Comm In Write	Comm In Write	Tri-state
8	L	H	L	L	Comm In Write	Comm In Data Strobe	Data from external device

a. During Communications I/O, P1RD is driven by an external device

b. During this and the next state, the MXT3010 tristates the bus. The external device drives data onto the bus.

FIGURE 51. COMMIN write[1] followed by COMMOUT read
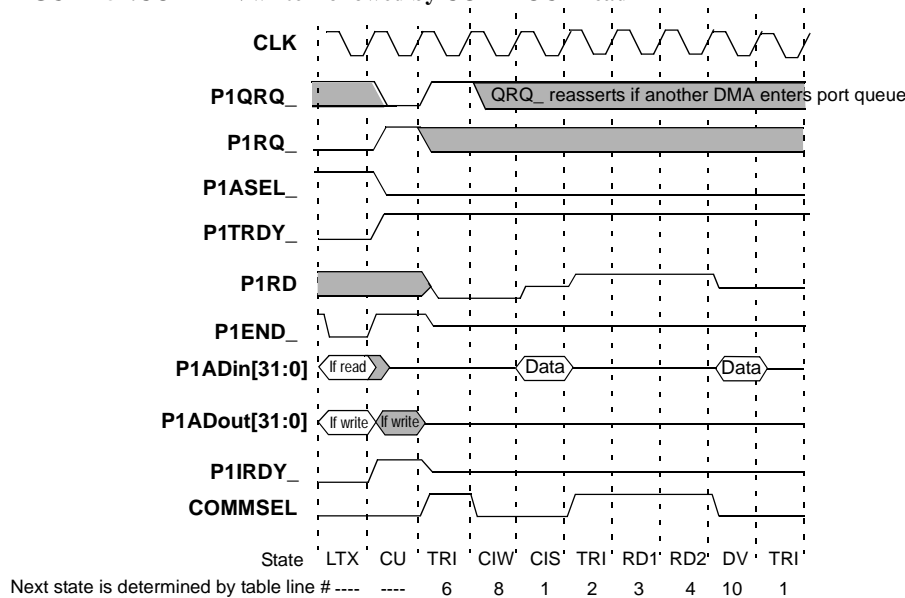
Figure 51 shows the Last Transfer (LTX) and Clean Up (CU) states of a previous DMA read or write transfer. During the Last Transfer state, the external logic that controls the ASEL_ and TRDY_ leads makes a decision as to whether it should:

1. Allow a COMM SEL transfer (having detected a communication request)

2. Prepare for another DMA transfer (having detected P1QRQ_ asserted)

3. Relinquish the bus by entering a tri-state condition

4. Perform some other type of bus operation

Having decided on the appropriate course of action, the external control logic conditions ASEL_ and TRDY_ at the beginning of the Clean Up state so that the Port1 state machine will enter the desired state after the Clean Up state.

---

1. In a COMMIN write, data is written from an external device into the MXT3010EP. In a COMMOUT read, data is read from the MXT3010EP by an external device.

In the example shown in Figure 51, the external logic elects to perform COMMIN and COMMOUT transfers. To do this, it drives ASEL_ low and TRDY_ high during the Clean Up state. Sampling ASEL_ low and TRDY_ high, the MXT3010EP places the P1 bus in the Tri-state condition (see line 1 of Table 26, Table 27, and Table 28).

During the Tri-state condition, the external logic asserts the COMMSEL input and drives P1RD to select a read or write transfer. Detecting the assertion of COMMSEL, the MXT3010EP prepares an internal data path for the read or write of R40/41, the Host Communication registers.

In Figure 51, the MXT3010EP samples the assertion of COM-MSEL high and P1RD low, and line 6 of the state table causes the next state to be Comm In Write. During the Comm In Write state, the external logic de-asserts COMMSEL while retaining P1RD low. The MXT3010EP samples these conditions and line 8 of the state table causes the next state to be Comm In Data Strobe.

During Comm In Data Strobe, data supplied by an external device is written into the 32-bit register formed by the concatenation of R40 and R41 within the MXT3010EP. Also during Comm In Data Strobe, the states of ASEL_ (low), TRDY_ (high), and COMMSEL (low) are such that line 1 of the state table causes the next state to be Tri-state.

During this Tri-state condition, the external logic asserts the COMMSEL input and drives P1RD high to select either a read transfer. In Figure 51, the MXT3010EP samples the assertion of COMMSEL high and P1RD high, and line 2 of the state table causes the next state to be Comm Out Read 1. If the COMMSEL and P1RD leads are maintained in their high states during Comm Out Read 1, line 3 of the state table causes the next state to be Comm Out Read 2.

If the COMMSEL and P1RD leads are maintained in their high states during Comm Out Read 2, line 4 of the state table causes the next state to be Comm Out Data Valid. During Comm Out Data Valid, data supplied by the concatenation of R40 and R41 within the MXT3010EP is supplied to the external device.

If the external device can sample the data quickly during Comm Out Data Valid, the external logic can condition the states of ASEL_ (low), TRDY_ (high), and COMMSEL (low) such that line 1 of the state table causes the next state to be Tri-state. If the external device requires more time to sample the data, the external logic can condition the states of ASEL_ (low), TRDY_ (high), and COMMSEL (high) such that line 5 of the state table causes the next state to be an additional period of Comm Out Data Valid.

Port2 basic protocol

The Port2 interface supports two transfer mechanisms: MXT3010-initiated DMA burst mode transfers and non-burst transfers. Each command issued to the DMA command queue is tagged as either burst or non-burst, via rsa bit 7. If rsa[7] is 1, the transfer is a burst transfer. If rsa[7] is 0, the transfer is a non-burst transfer. Non-burst transfers can insert a programmable number of wait states.

Figure 52 and Table 29 illustrate the correspondence between rsa/rsb register values, the Port2 bus signals, and a logical half-word address for Port2 burst DMA transfers.

FIGURE 52.Diagram of Port2 burst DMA instruction bits
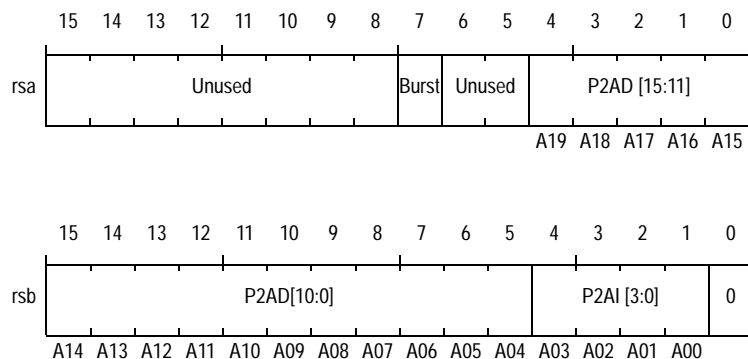
TABLE 29.   Port2 burst DMA instruction bit mapping

Reg	Bits	Function	Port2 Bus	Logical Halfword Bit
rsa	15:08	Not used	-	-
	07	Burst bit = 1	(selects mode)	-
	06:05	Not used	-	-
	04:00	Address	P2AD[15:11]	19:15
rsb	15:05	Address	P2AD[10:0]	14:04
	04:01	Address	P2AI[3:0]	3:0
	00	Discarded	-	-

The information in Table 29 can also be expressed as shown in Table 30.

TABLE 30.   Another view of Port2 burst DMA instruction bit mapping

Firmware Byte Address Bit	*MXT3010 Internal Register Bit*	*Memory Halfword Address Bit*	*MXT3010 Port2 Pin*
A00 (lsb)	RSB[0]	--	NC
A01	RSB[1]	A00 (lsb) HW	P2AI[0]
A02	RSB[2]	A01	P2AI[1]
A03	RSB[3]	A02	P2AI[2]
A04	RSB[4]	A03	P2AI[3]
A05	RSB[5]	A04	P2AD[0]
A06	RSB[6]	A05	P2AD[1]
A07	RSB[7]	A06	P2AD[2]
A08	RSB[8]	A07	P2AD[3]
A09	RSB[9]	A08	P2AD[4]
A10	RSB[10]	A09	P2AD[5]
A11	RSB[11]	A10	P2AD[6]
A12	RSB[12]	A11	P2AD[7]
A13	RSB[13]	A12	P2AD[8]
A14	RSB[14]	A13	P2AD[9]
A15	RSB[15]	A14	P2AD[10]
A16	RSA[0]	A15 (msb) HW	P2AD[11]
A17	RSA[1]	A16	P2AD[12]
A18	RSA[2]	A17	P2AD[13]
A19	RSA[3]	A18	P2AD[14]
A20	RSA[4]	A19	P2AD[15]

Since the Port2 burst DMA instruction bit mapping permits the use of 20-bit halfword addressing, one million (1M) 16-bit half-words can be addressed.

Figure 53 and Table 31 illustrate the correspondence between rsa/rsb register values, the Port2 bus signals, and a logical half-word address for Port2 non-burst DMA transfers.

FIGURE 53.Diagram of Port2 non-burst DMA instruction bits
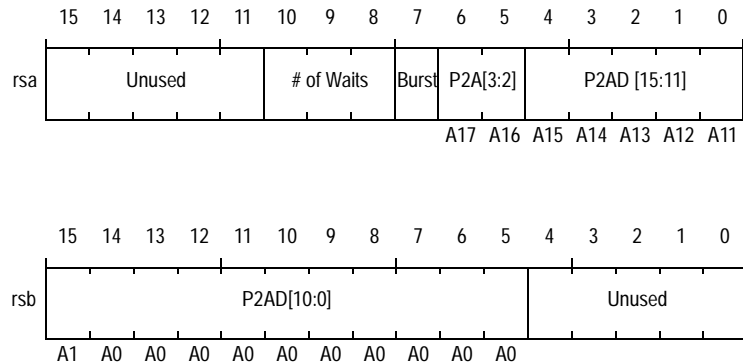
TABLE 31. Port2 non-burst DMA instruction bit mapping

Reg	Bits	Function	Port2 Bus	Logical Halfword Bit
rsa	15:11	Not used	-	-
	10:08	#waits [2:0]	(selects number of wait states)	-
	07	Burst bit = 0	(selects mode)	-
	06:05	Address	P2AI[3:2]	17:16
	04:00	Address	P2AD[15:11]	15:11
rsb	15:05	Address	P2AD[10:0]	10:0
	04:00	Discarded	-	-

The information in Table 31 can also be expressed as shown in Table 32.

TABLE 32. Another view of Port2 non-burst DMA instruction bit mapping

Firmware Byte Address Bit	*MXT3010 Internal Register Bit*	*Memory Halfword Address Bit*	*MXT3010 Port2 Pin*
A00 (lsb)	RSB[0]	--	NC
A01	RSB[1]	--	NC
A02	RSB[2]	--	NC
A03	RSB[3]	--	NC
A04	RSB[4]	--	NC
A05	RSB[5]	A00 (lsb) HW	P2AD[0]
A06	RSB[6]	A01	P2AD[1]
A07	RSB[7]	A02	P2AD[2]
A08	RSB[8]	A03	P2AD[3]
A09	RSB[9]	A04	P2AD[4]
A10	RSB[10]	A05	P2AD[5]
A11	RSB[11]	A06	P2AD[6]
A12	RSB[12]	A07	P2AD[7]
A13	RSB[13]	A08	P2AD[8]
A14	RSB[14]	A09	P2AD[9]
A15	RSB[15]	A10	P2AD[10]
A16	RSA[0]	A11	P2AD[11]
A17	RSA[1]	A12	P2AD[12]
A18	RSA[2]	A13	P2AD[13]
A19	RSA[3]	A14	P2AD[14]
A20	RSA[4]	A15 (msb) HW	P2AD[15]
A21	RSA[5]	A16	P2AI[2]
A22	RSA[6]	A17	P2AI[3]

Since the Port2 non-burst DMA instruction bit mapping permits the use of 18-bit halfword addressing, 256K 16-bit halfwords can be addressed.

Multi-function AI pins (P2AI[3:0])

In burst mode, P2AI [3:0] provide an address index consisting of the lower four bits of an address (see Table 29). In non-burst mode, P2AI[3:2] provide the most significant address bits (see Table 31). Also in non-burst mode, P2AI [1] represents P2RD_, and P2AI[0] represents Address Latch Enable. These signals can be used to provide a glueless interface to non-burst devices.

Port2 control signals

Table 33 describes the signals that control Port2 transfers

TABLE 33. Signals to control Port2 transfers

Signal	*Purpose*
P2QRQ_	When the Port2 state machine detects the presence of a command in the queue stage of the Port2 DMA command queue, the state machine asserts this signal to an external device; this provides advance indication that P2RQ_ will soon be asserted.
P2RQ_	When the Port2 state machine is in the Idle state and detects the presence of a command in the active stage of the Port2 DMA command queue, the state machine asserts this signal to an external device. The external device responds by manipulating P2ASEL_ and P2TRDY_ to control a DMA transfer.
P2TRDY_	This signal is an input to the MXT3010 and is driven by an external device. The external device uses this signal to insert wait states. The external device can also use this signal in conjunction with P2ASEL_ to deselect (tri-state) the Port2 DMA engine.
P2IRDY_	During DMA write data cycles, the MXT3010 asserts P2IRDY while it is sourcing valid data on P2AD[15:0]. During DMA read data operations, the MXT3010 asserts P2IRDY_ if it is able to sample P2AD[15:0] on the next rising edge of clock.
P2ASEL_	This signal is an input to the MXT3010 and is driven by an external device. The external device uses this signal to select between address and data cycles. The external device can also use this signal in conjunction with P2TRDY_ to deselect (tri-state) the Port2 DMA engine.
P2QBRST	This signal is an output driven by the MXT3010. The MXT3010 uses this signal to indicate the transfer mode, such as burst or non-burst, of the active command.
P2RD	During a DMA transfer, this signal is an output driven by the MXT3010. This signal indicates whether the transfer is a read (1) or a write (0) transfer.
P2END_	This signal indicates the last cycle of a DMA operation.
P2AD[15:0]	This is a multiplexed, bi-directional 16-bit bus. Data is read into and out of the MXT3010 during DMA transfers.
LTN	This is an internal signal indicating that the Last Transfer will occur Next (LTN).

The Port2 control state machine

Port2 DMA transfers originate and terminate as discussed in
"General information concerning DMA transfers" on page 113.

Port2 DMA burst-mode read transfers

Table 34shows the state table for Port2 DMA burst-mode read
transfers and Figure 56 shows a sequence diagram for a Port2
DMA burst-mode read transfer. Table 34 is identical to Table 26,
"State table for the Port1 DMA burst read state machine," on
page 118. The inputs, outputs, and states are the same as those
described in "Inputs" on page 113, "Outputs" on page 114, and
"States" on page 114, with four exceptions:

1. All signal names bear a P2 prefix instead of P1.
2. The P2AD bus is 16 bits; the P1AD bus is 32 bits (and has
   HalfWord Enable signals).
3. Only Port1 has a COMMSEL input.
4. Only Port2 has a P2QBRST output.

TABLE 34.   State table for the Port2 DMA burst-mode read state machine

Table Line	ASEL_	TRDY_	LTN	Current State	Next State	P2AD	IRDY_	END_
	Input Signals					**Outputs in the Next State**		
1	L	H	X	Any	Tri-state (current_state)	Tri-state		
2	L	L	X	Any pre Auto-turnaround	Address 1	Out-Addr	H	H
3	L	L	X	Any post Auto-turnaround	Address 2	Out-Addr	H	H
4	H	L	X	Address 1	Auto-turnaround	Tri-state	H	H
5	H	L	L	Address 2	Data Read	In-Data	L	H
6	H	L	H	Address 2	Last Transfer	In-Data	L	L
7	H	L	L	Auto-turnaround	Data Read	In-Data	L	H
8	H	L	H	Auto-turnaround	Last Transfer	In-Data	L	L
9	H	L	L	Data Read	Data Read	In-Data	L	H
10	H	L	H	Data Read	Last Transfer	In-Data	L	L
11	H	L	X	Auto-turnaround Wait	Auto-turnaround	Tri-state	H	H
12	H	L	L	Data Wait	Data Read	In-Data	L	H
13	H	L	H	Data Wait	Last Transfer	In-Data	L	L
14	H	L	X	Last Transfer	Clean Up	In-X	H	H
15	H	L	X	Clean Up	Data Read	In-X	H	H
16	H	L	X	Tri-state (Address 1)	Auto-turnaround	Tri-state	H	H
17	H	L	L	Tri-state (Address 2, Auto-turn-around, Data Read, or Data Wait)	Data Read	In-Data	L	H
18	H	L	H	Tri-state (Address 2, Auto-turn-around, or Data Wait)	Last Transfer	In-Data	L	L
19	H	L	X	Tri-state (Last Transfer, Clean Up)	Data Read	In-X	H	H
20	H	L	X	Tri-state (Auto-turnaround Wait)	Auto-turnaround	Tri-state	H	H
21	H	H	X	Address 1, Tri-state (Address 1, Auto-turnaround Wait)	Auto-turnaround Wait	In-X	H	H
22	H	H	X	Address 2, Auto-turnaround, Data Read, Data Wait, Clean Up, Tri-state (Auto-turnaround, Address2, Data, Data Wait, Last Transfer, or Clean Up)	Data Wait	In-X	H	H
23	H	H	X	Last Transfer	Clean Up	In-X	H	H

A sequence diagram for a typical DMA burst-mode read transfer using the Port2 read state table (Table 34) is shown in Figure 54. This diagram includes a wait state.

FIGURE 54.Port2 DMA burst-mode Read transfer with a Wait state
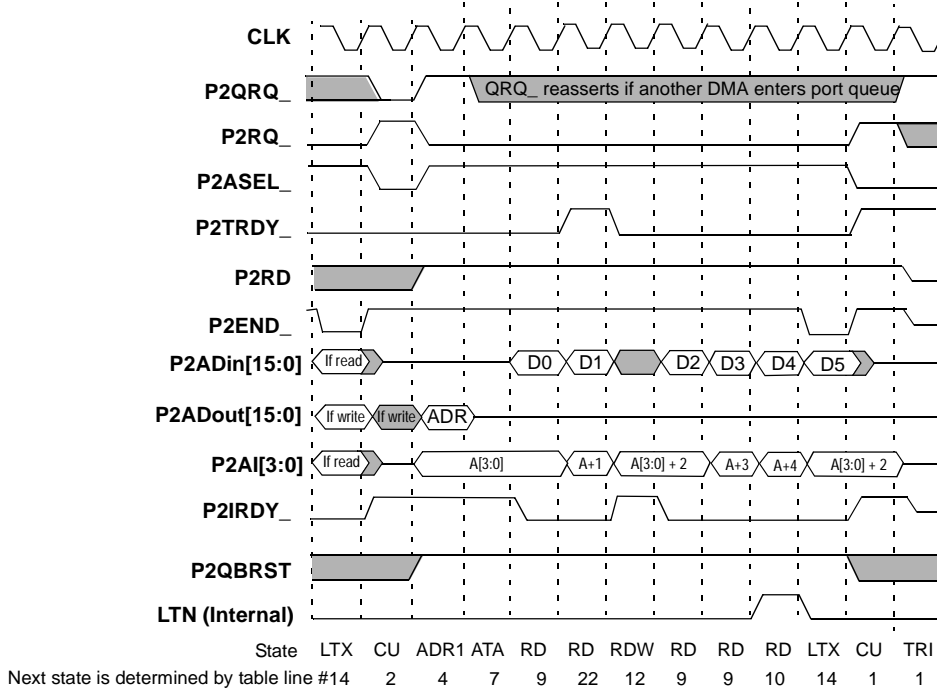
Figure 54 is identical to Figure 45 on page 119, with the following exceptions:

1. All signal names bear a P2 prefix instead of P1.

2. The P2AD bus is only16 bits.

3. There are no HalfWord Enable signals, but there are P2AI [3:0] signals.

4. Port2 has a P2QBRST output and has no COMMSEL input.

The sequence of states is identical to that shown in conjunction with Figure 45 on page 119, and the same explanatory text applies.

A second sequence diagram for a typical DMA burst-mode read transfer using the Port2 read state table (Table 34) is shown in Figure 55. This diagram does not include a wait state.

FIGURE 55. Port2 DMA burst-mode Read transfer without a Wait state
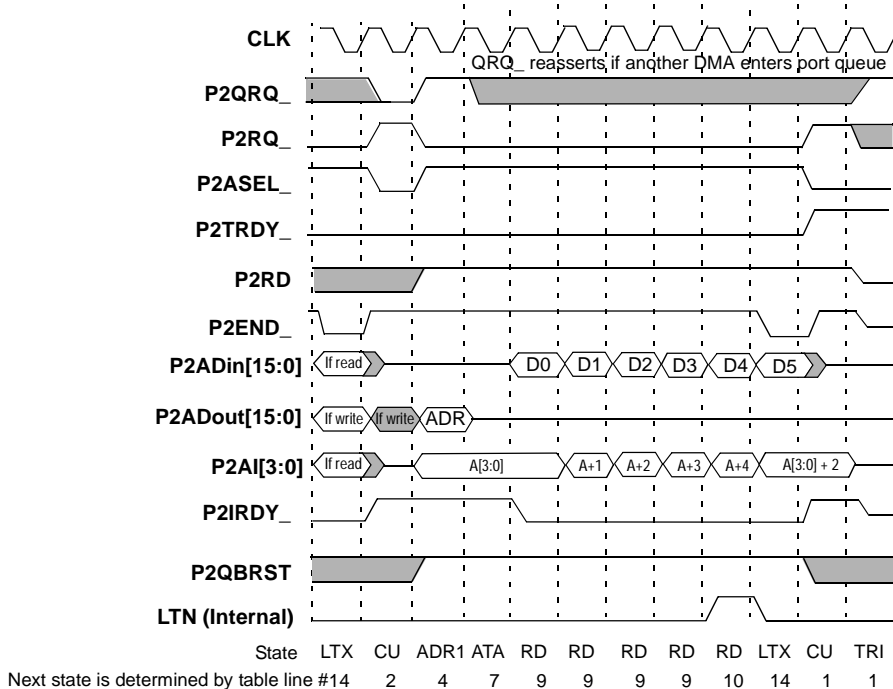
Figure 55 is identical to Figure 46 on page 122, with the following exceptions:

1. All signal names bear a P2 prefix instead of P1.

2. The P2AD bus is only16 bits.

3. There are no HalfWord Enable signals, but there are P2AI [3:0] signals.

4. Port2 has a P2QBRST output and has no COMMSEL input.

The sequence of states is identical to that shown in conjunction with Figure 46 on page 122, and the same explanatory text applies.

Port2 DMA burst-mode write transfers

Table 35 shows the state table for Port2 DMA burst-mode write transfers and Figure 56 shows a sequence diagram for a Port2 DMA burst-mode write transfer. Table 35 is identical to Table 27 on page 126. The inputs, outputs, and states are the same as those described in "Inputs" on page 113, "Outputs" on page 114, and "States" on page 114, with four exceptions:

1. All signal names bear a P2 prefix instead of P1.

2. The P2AD bus is 16 bits; the P1AD bus is 32 bits (and has HalfWord Enable signals).

3. Only Port1 has a COMMSEL input.

4. Only Port2 has a P2QBRST output.

TABLE 35. State table for the Port2 DMA burst write state machine

Table Line	Input Signals			Current State	Next State	Outputs in the Next State		
	*ASEL_*	*TRDY_*	*LTN*			*P1AD*	*IRDY_*	*END_*
1	L	H	X	Any	Tri-state (current_state)	Tri-state		
2	L	L	X	Any	Address	Out-Addr	H	H
3	H	L	L	Address	Data Write	Out-Data	L	H
4	H	L	H	Address	Last Transfer	Out-Data	L	L
5	H	L	L	Data Write	Data Write	Out-Data	L	H
6	H	L	H	Data Write	Last Transfer	Out-Data	L	L
7	H	L	L	Data Wait	Data Write	Out-Data	L	H
8	H	L	H	Data Wait	Last Transfer	Out-Data	L	L
9	H	L	X	Last Transfer	Clean Up	Out-X	H	H
10	H	L	X	Clean Up	Idle[a]	Tri-state	H	H
11	H	L	L	Idle, Tri-state (Address, Data Write, or Data Wait)	Data Write	Out-Data	L	H
12	H	L	H	Tri-state (Address, Data Write, or Data Wait)	Last Transfer	Out-Data	L	L
13	H	L	X	Tri-state (Last Transfer, Clean Up)	Data Write	Out-X	H	H
14	H	H	X	Address, Data Write, Data Wait, Clean Up, Tri-state (Address, Data Write, Data Wait, Last Transfer, or Clean Up)	Data Wait	Out-X	H	H
15	H	H	X	Last Transfer	Clean Up	Out-X	H	H

a. The Idle state will be maintained indefinitely if there is no RQ_ assertion.

A sequence diagram for a typical DMA burst-mode write transfer using Table 35 is shown in Figure 56. This diagram includes a wait state.

FIGURE 56.Port2 DMA burst-mode write transfer with a Wait state
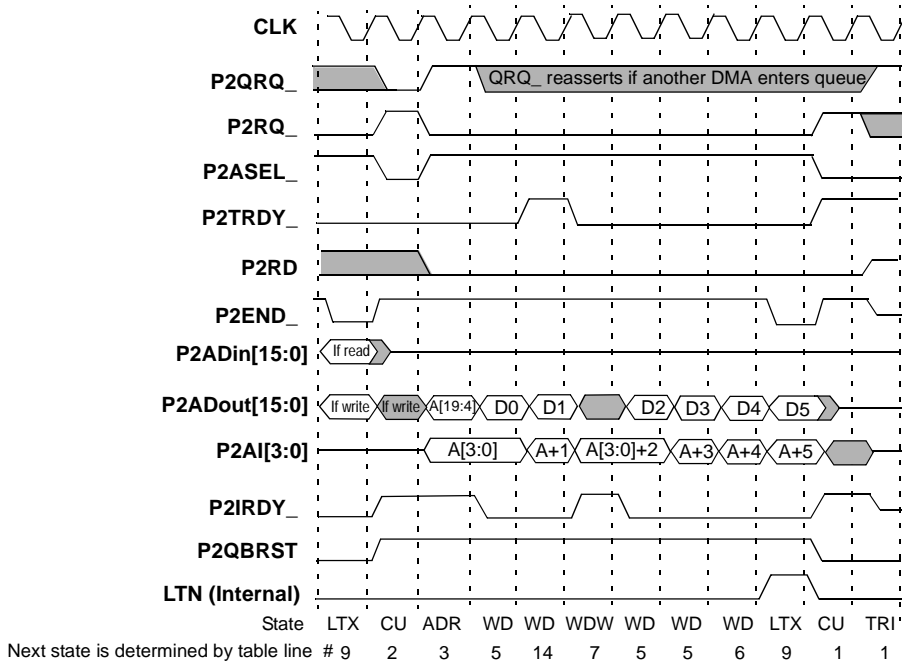
The sequence of states in Figure 56 is the same as that for Figure 47, "Port1 DMA Write transfer with a Wait state," on page 127, and the same explanatory text applies.

A sequence diagram for a typical DMA burst-mode write transfer using Table 35 is shown in Figure 56. This diagram does not include a wait state.

FIGURE 57.Port2 DMA burst-mode write transfer without a Wait state
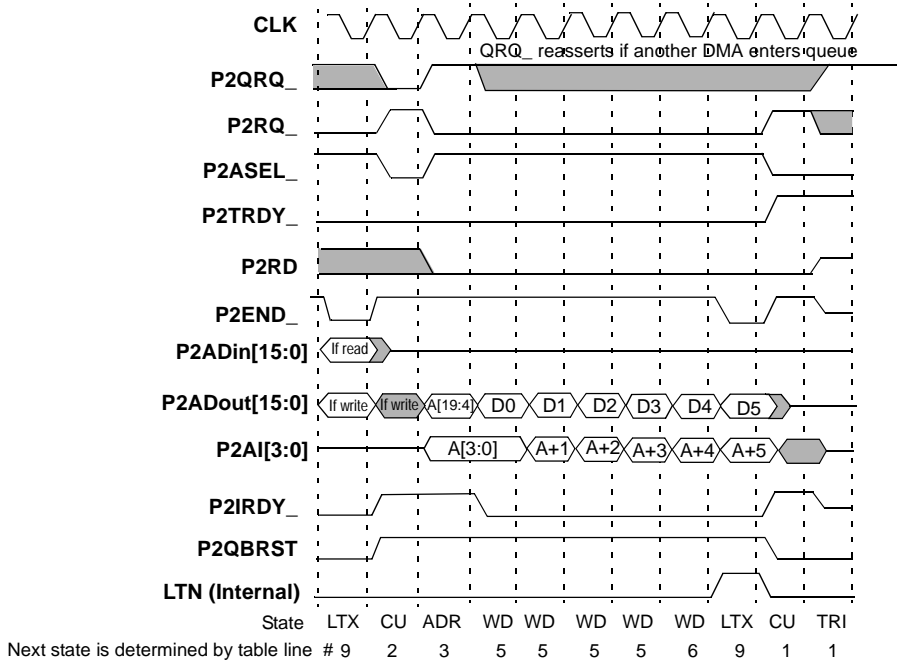
The sequence of states in Figure 57 is the same as that for Figure 48, "Port1 DMA Write transfer without a Wait state," on page 130, and the same explanatory text applies.

Port2 DMA non-burst-mode read transfers

Table 36shows the state table for Port2 DMA non-burst-mode read transfers and Figure 59 shows a sequence diagram for a Port2 DMA non-burst-mode read transfer.

TABLE 36. State table for the Port2 DMA non-burst-mode read state machine

Table Line	ASEL_	TRDY_	LTN[a]	Current State	Next State	P2AD	IRDY_	P2AI[3:2]	P2RD_	P2ALE_	END_
	Input Signals					*Outputs in the Next State*					
1	L	H	X	Any	Tri-state (current_state)	Tri-state					
2	L	L	X	Any	Address1	Out-Addr	H	V[b]	L	H	H
3	H	L	X	Address1	Address2	Out-Addr	L	V	L	H	H
4	H	L	X	Address2	Address Hold	Out-Addr	L	V	L	L	H
5	H	L	X	Address Hold	Data Wait	In-X	L	V	L	L	H
6	H	L	L	Data Wait	Data Wait	In-X	L	V	L	L	H
7	H	L	H	Data Wait	Last Transfer	In-Data	L	V	L	L	L
8	H	L	X	Last Transfer	Clean Up	In-X	H	V	L	L	H
9	H	L	X	Clean Up	Idle[c]	Tri-state	H	V	L	L	H
10	H	H	X	Data Wait	Data Wait	In-X	H	X	X	X	H

a. For non-burst-mode operation, the LTN (Last Transfer Next) signal is asserted when the programmable wait-timer expires. See "#waits [2:0]" in Table 31, "Port2 non-burst DMA instruction bit mapping," on page 139.

b. The P2AI[3:2] outputs have valid (V) address information on them throughout all states marked "V". These outputs can be decoded to form four chip selects if desired.

c. The Idle state will be maintained indefinitely if there is no RQ_ assertion.

A sequence diagram for a DMA non-burst-mode read transfer using Table 36 is shown in Figure 58.

FIGURE 58.Port2 DMA non-burst-mode Read transfer.
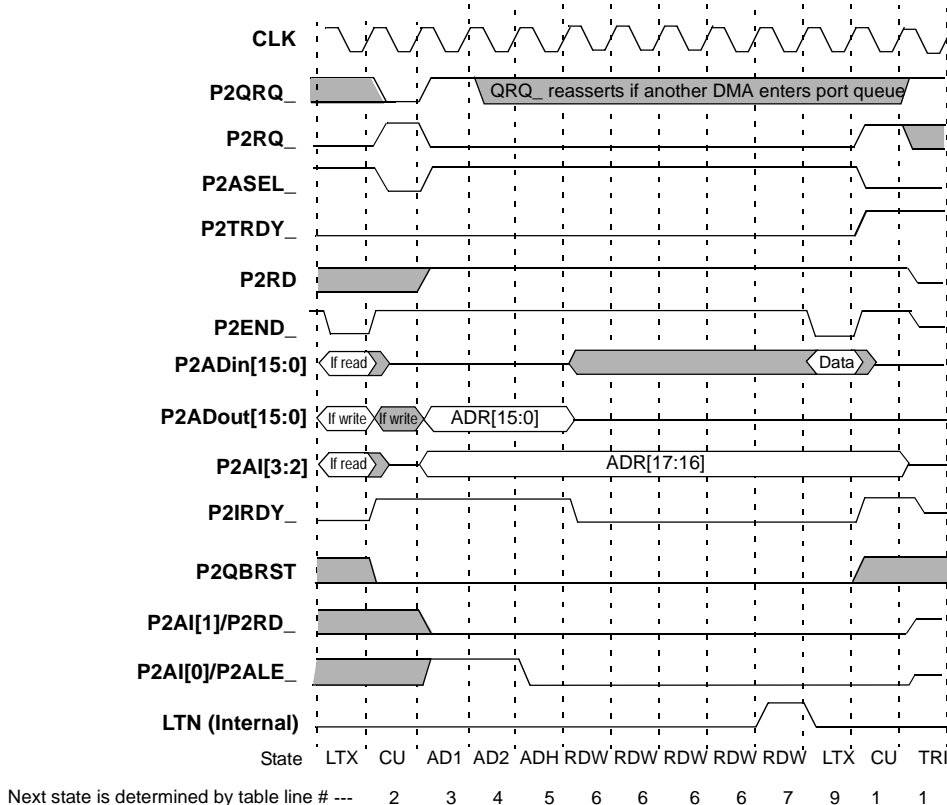
Note:During a Port2 Non-Burst DMA Read, an external device places data on the P2AD leads. The Port2 DMA Read command can specify, via bits [10:8] of the rsa register, the number of wait states that occur before the MXT3010EP samples the data. In the example shown above, 5 wait states have been inserted.

Figure 58 shows the Last Transfer (LTX) and Clean Up (CU) states of a previous DMA read or write transfer. During the Last Transfer state, the external logic that controls the ASEL_ and TRDY_ leads makes a decision as to whether it should:

1. Prepare for another DMA transfer (having detected P2QRQ_ asserted)
2. Relinquish the bus by entering a tri-state condition
3. Perform some other type of bus operation

Having decided on the appropriate course of action, the external control logic conditions ASEL_ and TRDY_ at the beginning of the Clean Up state so that the Port2 state machine will enter the desired state after the Clean Up state.

In this example, a non-burst read transfer is performed. During the Clean Up state, both ASEL_ and TRDY_ are low, and line 2 of the state table indicates that the next state is Address 1 (AD1). During Address 1, the ASEL_ lead is driven high, and line 3 of the state table indicates that the next state is Address 2 (AD2).

In the Address 2 state, the ASEL_ lead is still high and the TRDY_ lead is still low. Line 4 of the state table indicates that the next state is Address Hold (ADH). At this time, the P2AI[0] output, functioning as Address Latch Enable (ALE_) transitions from high to low, performing the address latching function characteristic of asynchronous, multiplexed busses.

During Address 1, Address 2, and Address Hold, the P2AD[15:0] leads carried the lowest order 16 bits of the desired address. P2AI[3:2]carried bits A[17:16] during the three address states and also carry those bits throughout the DMA transfer. Thus, they can used as chip selects if desired. P2AI[1] creates an inverted version of P2RD (P2RD_) to provide a glueless interface on the P2 bus.

During the Address Hold state, the ASEL_ lead is still high and the TRDY_ lead is still low, and line 5 of the state table indicates that the next state is Data Wait. As indicated by lines 6 and 7 of the state table, the Data Wait condition persists until the wait

timer (set by rsa[10:08]) expires. Expiration of the wait timer asserts LTN, and line 7 of the state table indicates the next state is Last Transfer (LTX).

As with all of the other DMA transfer types discussed, Last Transfer is followed by Clean Up; during Last Transfer the external arbiter selects conditions for ASEL_ and TRDY_ that determine the bus activity after the Clean Up state. In Figure 58, ASEL_ is low and TRDY_ is high during the Clean Up state, and line 1 of the state table indicates the next state is Tri-state.

It is also possible that ASEL_ could be retained high and TRDY_ could be retained low. In that case, the bus would enter an Idle state during which the data leads would be tri-state, but the control leads would still be in their previous state.

Port2 DMA non-burst-mode write transfers

Table 37shows the state table for Port2 DMA burst-mode write transfers and Figure 60 shows a sequence diagram for a Port2 DMA burst-mode read transfer.

TABLE 37. State table for the Port2 DMA non-burst-mode write state machine

Table Line	Input Signals ASEL_	TRDY_	LTN[a]	Current State	Next State	Outputs in the Next State P2AD	IRDY_	P2AI[3:2]	P2RD_	P2ALE_	END_
1	L	H	X	Any	Tri-state (current_state)	Tri-state					
2	L	L	X	Any	Address1	Out-Addr	H	V[b]	H	H	H
3	H	L	X	Address1	Address2	Out-Addr	L	V	H	H	H
4	H	L	X	Address2	Address Hold	Out-Addr	L	V	H	L	H
5	H	L	X	Address Hold	Data Wait	Out-Data	L	V	H	L	H
6	H	L	L	Data Wait	Data Wait	Out-Data	L	V	H	L	H
7	H	L	H	Data Wait	Last Transfer	Out-Data	L	V	H	L	L
8	H	L	X	Last Transfer	Clean Up	Out-X	H	V	H	L	H
9	H	L	X	Clean Up	Idle[c]	Tri-state	H	V	H	L	H
10	H	H	X	Data Wait	Data Wait	Out-Data	H	X	X	X	H

a. For non-burst-mode operation, the LTN (Last Transfer Next) signal is asserted when the programmable wait-timer expires. See "#waits [2:0]" in Table 31, "Port2 non-burst DMA instruction bit mapping," on page 139.

b. The P2AI[3:2] outputs have valid (V) address information on them throughout all states marked "V". These outputs can be decoded to form four chip selects if desired.

c. The Idle state will be maintained indefinitely if there is no RQ_ assertion.

A sequence diagram for a DMA non-burst-mode read transfer using Table 37 is shown in Figure 59.

FIGURE 59.Port2 DMA non-burst-mode Write transfer.

State	LTX	CU	AD1	AD2	ADH	WDW	WDW	WDW	WDW	WDW	LTX	CU	TRI
Next state is determined by table line # ---	2	3	4	5	6	6	6	6	7	9	1	1	
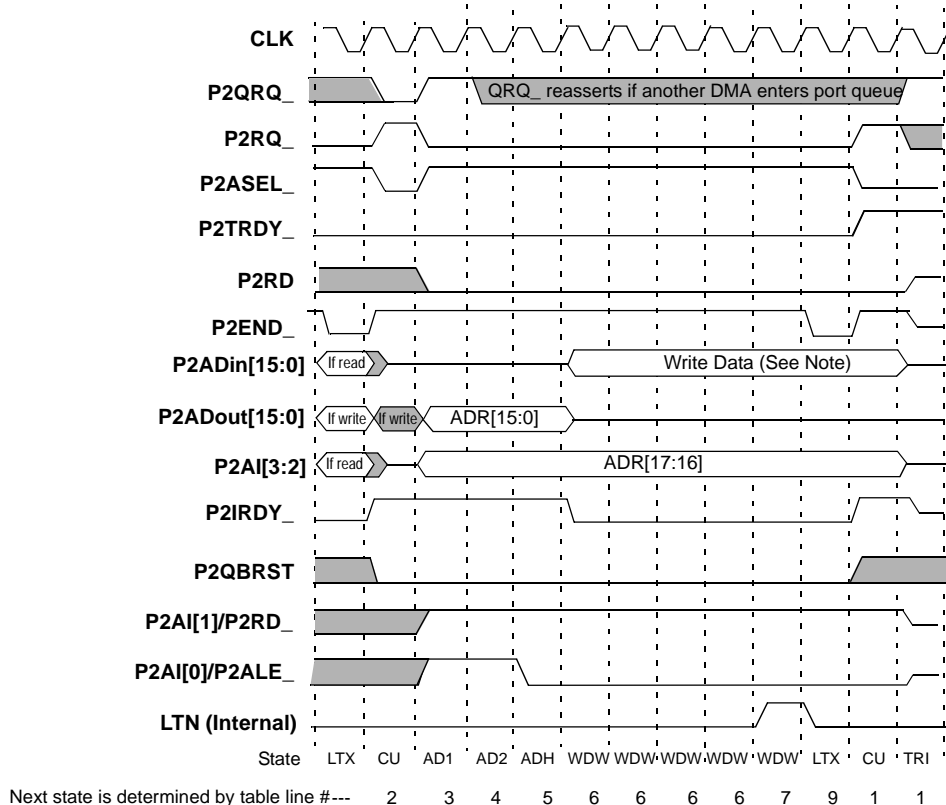
Note:During a Port2 Non-Burst DMA Write, the MXT3010EP places data on the P2AD leads for at least one clock cycle. The Port2 DMA Write command can specify, via bits [10:8] of the rsa register, the number of additional cycles (wait states) during which the MXT3010 holds the data on the bus. In the example shown above, 5 wait states have been inserted in addition to the minimum data assertion period of one clock cycle.

The sequence of states shown in Figure 59 is exactly the same as that shown in Figure 58, except that this is a write. The same description applies, substituting writes for reads as necessary.

Additional Port1 and Port2 Design Information

Arbitrating access to Port1

System configurations utilizing the MXT3010 often have a Host processor installed on Port1. This allows the Host processor to access the MXT3010 Communication I/O registers and to access the SRAM. For example, consider the system shown in Figure 60.

FIGURE 60.System example for Port1 bus.
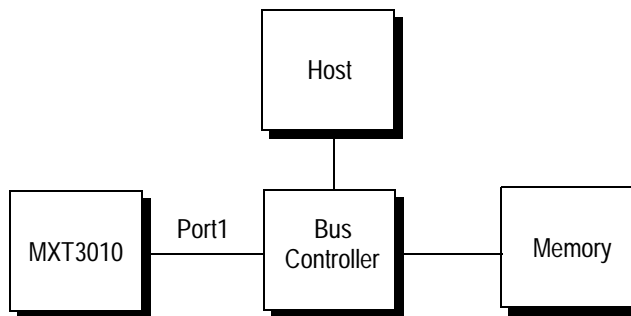
The functions of the Bus Controller are as follows:

1. In response to P1QRQ_ and P1RQ_,grant the MXT3010 access to the Memory, manipulating P1ASEL_ and P1TRDY_ to step the MXT3010 through read (P1RD high) or write (P1RD low) DMA transfers as described in Figure 45 and Figure 47 respectively.

2. In response to bus request signals from the Host, grant the Host access to the Memory or to the MXT3010 Communications I/O register, performing a read or write transfer as requested by the Host.

3. An existing DMA transfer should not be interrupted. The maximum MXT3010/Memory transfer is 255 bytes (64 bus data cycles). It is recommended that the maximum Host/Memory transfer also be 64 bus data cycles.

4. The Bus Controller may also include a Port1 to PCI bus adapter if desired. Also, the Bus Controller may also include Memory interface logic.

Simplified Port2 interfaces

When the MXT3010 is the only Port2 master, no arbitration function is required. The following simplified interfaces can be implemented:

A single master burst-mode interface

Logic to manipulate P2ASEL_ and P2TRDY_ can be built into the slave device attached to Port2. In this configuration, P2TRDY_ may be tied low if generation of Data Wait states is not required. When there is no bus activity, P2ASEL_ should also be held low. Once the slave device samples RQ_ as low, it samples the DMA address from P2AD and P2AI and drives P2ASEL_ high on the same or any subsequent clock edge.

Once P2 ASEL_ is high, the subsequent cycles follow the Table 34 or Table 35 sequences as determined by the state of RD (read/write). As with multi-master bus configurations, P2END_ low delineates the last halfword transfer. P2ASEL_ should be driven low when P2END_ is sampled low. P2ASEL_ must be held low until the next DMA.

A non-burst-only interface

P2ASEL_ can be tied high and P2TRDY_ can be tied low. Once RQ_ is low, the state machine begins operation and the slave device receives the DMA address from P2AD and P2AI. The slave device samples the address on the falling edge of P2AI[0]/P2ALE_. Subsequent bus cycles follow the Table 36 or Table 37 sequences as determined by the state of RD (read/write).

Bus driving, turnaround, and bus parking

The port interfaces can operate in a shared bus environment. In such an environment, the following port interface signals are shared between all devices on their respective busses:

Port 1	Port 2
P1AD[31:0]	P2AD[15:0]
P1END_	P2END_
P1RD	P2RD
P1IRDY_	P2IRDY_
P1HWE[1:0]	

To prevent bus contention on shared port interface signals, port interface controllers should create a tri-state cycle between the times the MXT3010EP and another device drive the bus. In addition, to prevent the bus from floating indefinitely, port inter-face controllers must ensure the bus is driven when there is no device performing transfers on the bus. This can be done, for example, by placing the MXT3010EP into address mode. The MXT3010 enters address mode on the rising edge of clock when ASEL_ is low, TRDY_ is low, and (for Port1) COMMSEL is low.

Speeding up transfers

Many of the timing diagrams for Port1 and Port2 interfaces show PxASEL_ switching one clock period after PxRQ_ is asserted to select the data phase. If the MXT3010 is being used in a system that does not need to tri-state the bus, PxASEL_ can be negated at the same time PxRQ_ is asserted if PxTRDY_ is asserted (bus parked). This speeds up Port cycles by one clock period. This process is described in more detail in "Port2 bus parking" on page 158.

Port2 bus parking

While the following text describes bus parking on Port2, Port1 bus parking operates similarly.

When the MXT3010 is the only Port2 Master, the device may be parked on the bus. Parking minimizes bus handshaking overhead. While the MXT3010 is parked, it actively drives the P2AD pins.

In a bus parking configuration, P2TRDY_ may be tied low. During periods of no bus activity, P2ASEL_ should also be held low. Once the P2 Slave device samples P2RQ_ low, it samples the DMA address from P2AD and P2AI and drives P2ASEL_ high on the same or any subsequent clock edge.

Once ASEL_ is deasserted, the MXT3010 drives valid data on subsequent cycles. As with non-parked bus configurations, P2END_ low delineates the last halfword transfer. P2ASEL_ should be driven low when P2END_ is sampled low. P2ASEL_

Data Alignment

The MXT3010 can begin Port1 reads on odd byte boundaries and can begin Port1 writes on odd halfword boundaries. The reads always appear (on the bus) to be word reads where the internal Port1 hardware shifts the data appropriately for the byte address. Although A1 and A0 represent the byte address, they can be, and usually are, ignored by external hardware. On writes, the P1HWE1 and P1HWE0 signals determine which half of the word is being written on 32-bit boundaries, halfword enabled. The MXT3010 takes care of any shifting/swapping.

DMA transfers in burst mode cause one or more data cycles on the bus. Each cycle can transfer four bytes (Port1) or two bytes (Port2). On reads, the MXT3010 can start at an odd byte (or in the case of Port1, an odd halfword) where it internally determines which bytes are important and performs lane switching (shifting). On writes, the MXT3010 can only write on halfword boundaries and an even number of bytes due to the halfword-oriented write enables. The last byte written for an odd byte trans-

fer may be ignored if the transfer desired an odd byte size, but the last byte transfer will be written. Thus, the number of data cycles for four bytes may be one or two bus data cycles depending on the byte alignment.

Transfer complete

A DMA transfer can conclude for either of two reasons:

- The byte count (BC/#) has reached zero

- The P1ABORT_ signal has been asserted (Port1 only)

Byte Count zero

Standard end
timing

For both Port1 and Port2, END_ is asserted during the data cycle that presents the final data on the bus. The DMA cycle concludes with a "cleanup" cycle.

FIGURE 61.DMA Read transfer with standard END_ signal
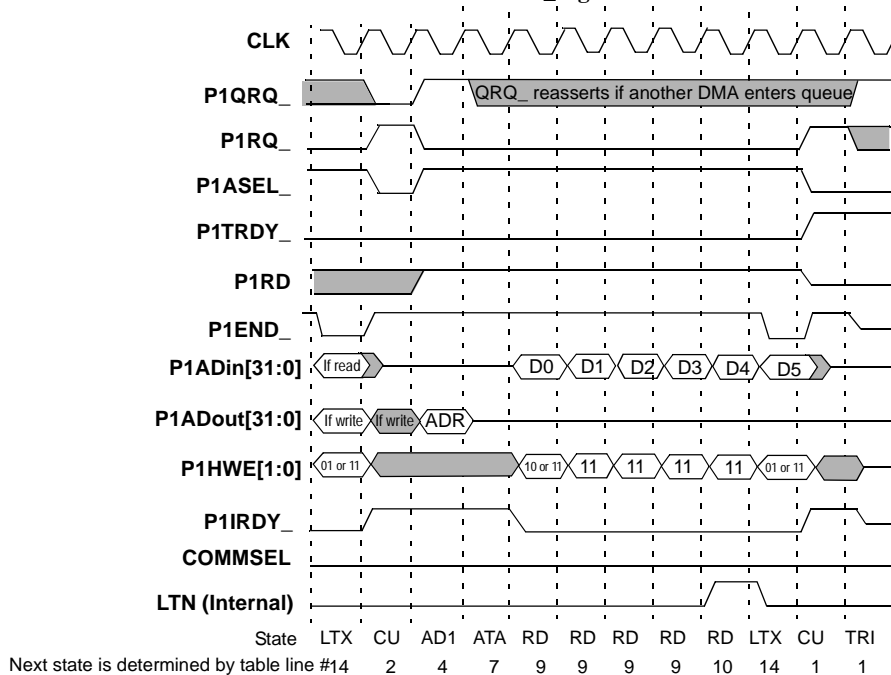
Early end option   Mode bits (bit 6 and bit 7) in the Mode Configuration register
                      (R42) enable an early end option for each port. When enabled,
                      the End signal asserts concurrent with the request for the next-
                      to-last data cycle. External circuitry must qualify this signal with
                      the appropriate control signals (ASEL_ and TRDY_) to deter-
                      mine that a data cycle is present on the bus. This ensures that the
                      external controller recognizes the actual end condition and not
                      that the current clock cycle is a wait state.

FIGURE 62.DMA Read transfer with Early END

State	LTX	CU	AD1	ATA	RD	RD	RD	RD	RD	LTX	CU	TRI
Next state is determined by table line #14	2	4	7	9	9	9	9*	10	14	1	1	

*While line 9 of the state table indicates the END_ output is high in the next state, enabling the Early End option
 allows the LTN signal to assert P1END_ immediately.

Note: External logic must ensure that ASEL_ is high and TRDY_ is low
      when END_ is asserted (low). If TRDY_ is high (shown by dashed
      lines), a Wait state is indicated and the last data cycle of the DMA
      transfer cycle is extended beyond the length indicated. For normal end
      conditions this is unimportant, but if a design is relying upon an "early
      end", this condition is important.

External DMA cycle abort (P1ABORT_)

The MXT3010 has an input signal (P1ABORT_) that permits an external device to indicate an early termination of a DMA read operation from Port1 memory. During a DMA Read operation on Port1, assertion of the P1ABORT_ signal terminates the read with the data in the following cycle. For example, asserting P1ABORT_ during the fifth data phase of a DMA burst terminates the operation after the sixth data phase has completed. The action of P1ABORT_ is similar to that of the internal signal LTN, except that when a transfer is terminated by P1ABORT_, no P1END_ assertion occurs.

FIGURE 63.DMA Read transfer terminated by P1ABORT_

State	LTX	CU	AD1	ATA	RD	RD	RD	RD	RD	LTX*	CU	TR
Next state is determined by table line #	14	2	4	7	9	9	9	9	10*	14*	1	1
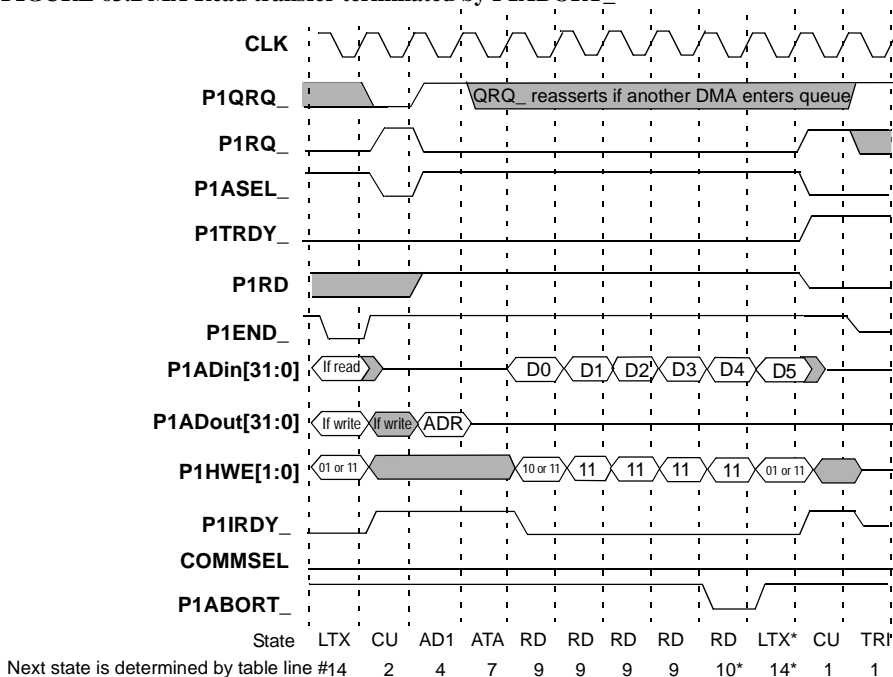
*The state table does not show the effects of P1ABORT_. The effects are equivalent to LTN, which is shown in the state table (with the line numbers cited here), with the exception that no END_ assertion occurs.

During a DMA Write operation on Port1, assertion of the P1ABORT_ signal terminates the write of the data in the following cycle.

Endian-ness

Within modern computer systems, there are two ways of addressing a multi-byte data value such as (hex) ABCD:

FIGURE 64.Most Significant Byte is the Lowest Address ("Big-endian")

Data:	A	B	C	D
Address:	0	1	2	3

FIGURE 65. Least Significant Byte is the Lowest Address ("Little-endian")

Data:	A	B	C	D
Address:	3	2	1	0

The mapping in Figure 64 stores the most significant byte in the lowest numeric byte address. The mapping in Figure 65 stores the least significant byte in the lowest numeric byte address; These methods are commonly referred to as "big-endian" and "little-endian" respectively.

If a processor that uses big-endian or little-endian addressing accesses the data shown Figure 64 and Figure 65 on a word basis, the entire 32-bit quantity ABCD is accessed, and no problems result. However, processsors that use big-endian addressing receive different results than those using little-endian addressing when making word or byte accesses. See Table 38.

TABLE 38.   Comparison of Big-endian and Little-endian Read Operations

Access	*Big-Endian Result*	*Little-Endian Result*
32-bit	ABCD	ABCD
16-bit xxx0	AB	CD
16-bit xxx2	CD	AB
byte xxx0	A	D
byte xxx1	B	C
byte xxx2	C	B
byte xxx3	D	A

A convenient method of dealing with this problem is to use the swapping instructions available in little-endian processors in combination with a hardware byte-swapper. A byte-swapper, implemented in hardware, in shown in Figure 66.

FIGURE 66.Hardware Byte-swapping Circuit
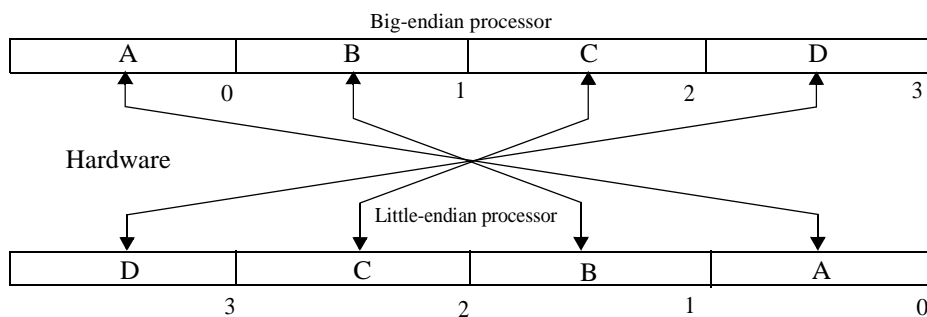
Figure 67 shows what happens when the hardware byte-swapper is used in conjunction with a software instruction that swaps the bytes on a word basis within the little-endian processor.

FIGURE 67.Word Access
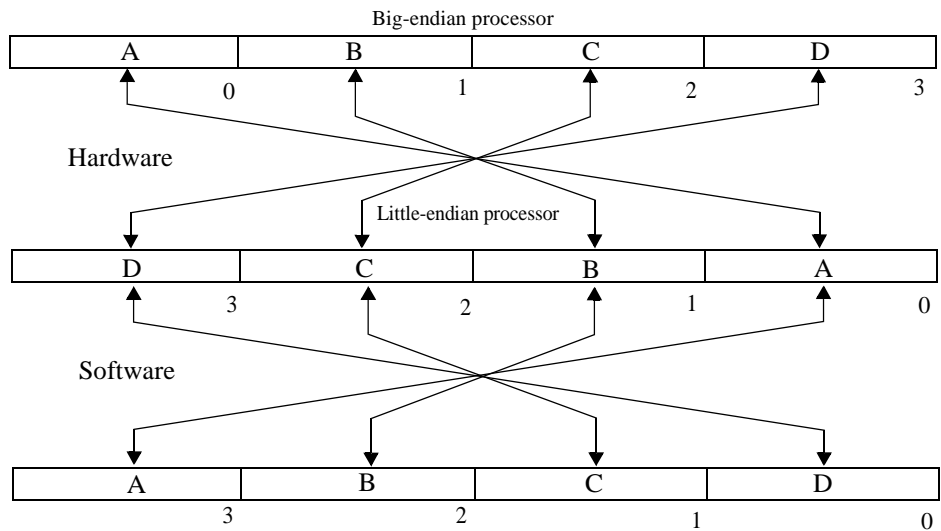
The combination of hardware and software shown in Figure 67 produces the same result as shown in Table 38 on page 165, the first line of which (in an expanded form) is reproduced in Table 39.

TABLE 39.   Accesses With Hardware and Software Swaps, 32-bit

Access	*Big-Endian Result*	*Little-Endian Result*	*Result after H/W-S/W Swaps*
32-bit	ABCD	ABCD	ABCD

At first glance, this appears to be a waste of hardware and software. However, the results for halfword (16-bit) and byte (8-bit) operations are more interesting.

FIGURE 68.16-bit xxx0 Access
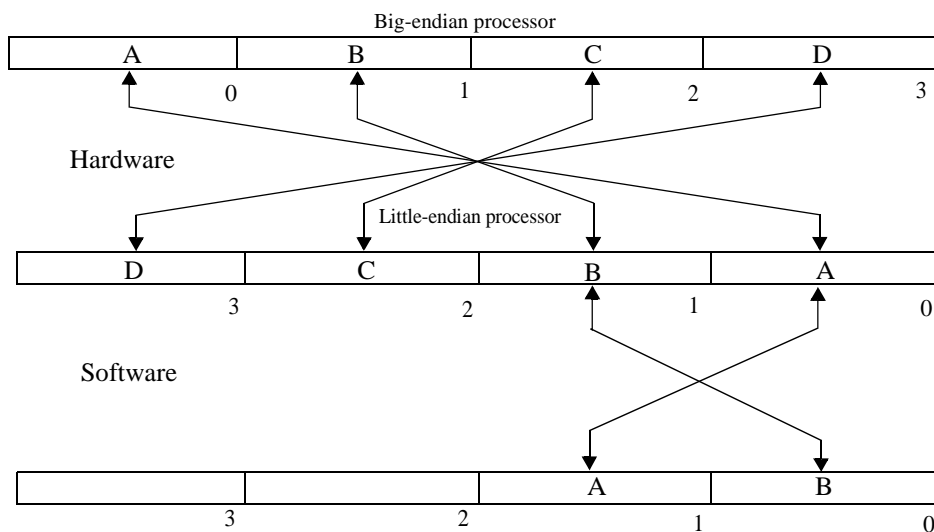
FIGURE 69. 16-bit xxx2 Access
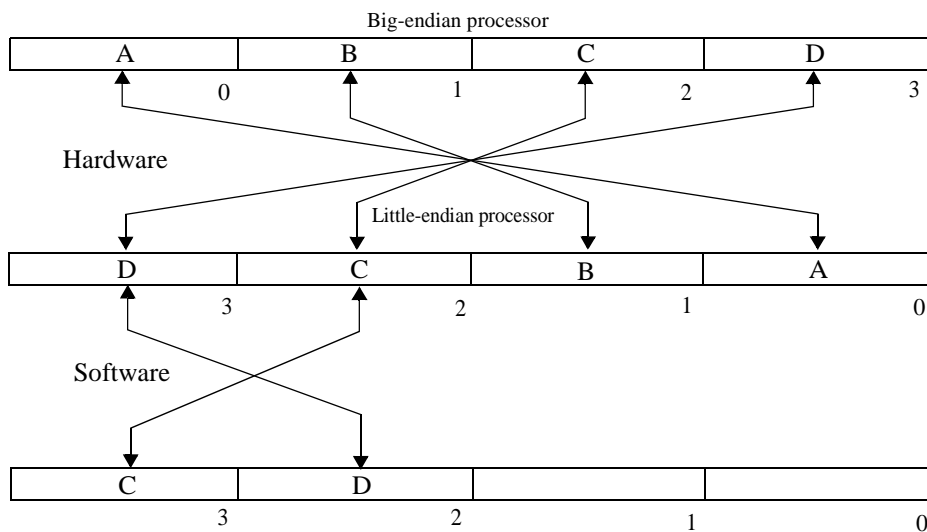
The combinations of hardware and software shown in Figures 68 and 69 produce Table 40.

TABLE 40. Accesses With Hardware and Software Swaps, 32-bit and 16-bit

Access	*Big-Endian Result*	*Little-Endian Result Per Table 38*	*Result after H/W-S/W Swaps*
32-bit	ABCD	ABCD	ABCD
16-bit xxx0	AB	CD	AB
16-bit xxx2	CD	AB	CD

FIGURE 70.Byte Access
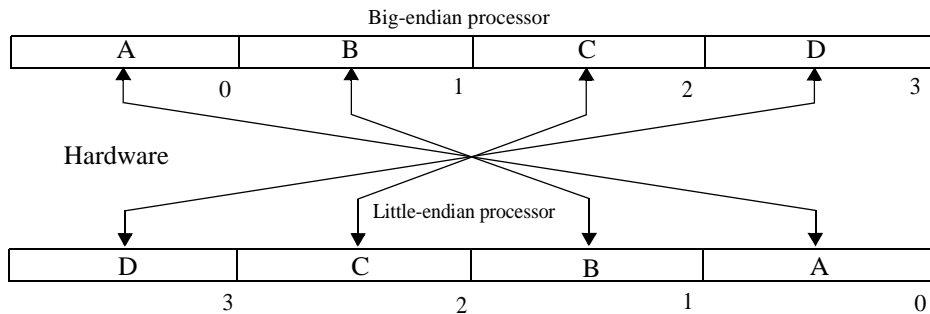
The combination of hardware and software shown in Figure 70 produces Table 41.

TABLE 41. Accesses With Hardware and Software Swaps, 32-bit, 16-bit, and 8-bit

Access	*Big-Endian Result*	*Little-Endian Result (Per Table 38)*	*Result after H/W-S/W Swaps*
32-bit	ABCD	ABCD	ABCD
16-bit xxx0	AB	CD	AB
16-bit xxx2	CD	AB	CD
byte xxx0	A	D	A
byte xxx1	B	C	B
byte xxx2	C	B	C
byte xxx3	D	A	D

As indicated in Table 41, the combination of the hardware byte-swapper and byte swapping instructions within the little-endian processor allow the little-endian processor to access information in the big-endian system and receive consistent results.

See "Endian Implementation in P1MemMaker" on page 171 and "Endian Implementation in P2MemMaker" on page 174 for examples of endian treatment in reference designs.

Port1 and Port2 Reference Designs

P1MemMaker

The MXT3010EP Port1 interface requires a memory controller function to support bus arbitration, bus selection, bus driving, bus turnaround, and bus holding operations. In Maker's MXT3025 evaluation Board and similar designs, Maker uses *P1MemMaker*, a device that is a integrated memory system controller, integrated PCI interface, COMMIN/COMMOUT Register, and MXT3010EP Port1 interface. It performs the following functions:

• Memory System Controller

  The Memory System Controller (MSC) provides the bus arbitration and selection functions for the PCI or Port1 access for tranfers to shared memory. It controls up to 4 Mbytes of shared DRAM. The typical memory system is organized as 1Mx32 and implemented with two (2) 1Mx16 EDO DRAMs. The memory system is usually mapped into the PCI memory space and mapped into the lower 4 Mbytes of the MXT3010EP Port1 address space. The MSC supports full speed burst transfers of up to 256 bytes to the memory system, but transfers must not cross 4-Kbyte boundaries. Also, the MSC controls the resetting and boot loading of the MXT3010EP through a 128 Kbyte boot PROM.

- PCI Interface

  The PCI bus interface is a 32-bit, 33 Mhz PCI Version 2.1 implementation supporting the PCI configuration registers, a slave-only interface, and no support for initiating transfers from the host processor to shared memory or the MXT3010EP device.

- COMMIN/COMMOUT Register

  The P1MemMaker also controls communications between the MXT3010EP COMMIN/COMMOUT register and the PCI host. The CINBUSY, COUTRDY, and COMMSEL signals are connected to the P1MemMaker. Typically the PCI host communicates to code running in the MXT3010EP via commands passed through the MXT3010EP's COMMIN register. The code running in the MXT3010EP communicates to the PCI host via commands passed through the MXT3010EP's COMMOUT register, writing to command responses and to indication queues. The COMMIN/COMMOUT register (32-bits) thus provides the two-way communications. Details of CINBUSY and COUTRDY operation are provided in Chapter 8 of the *MXT3010 Reference Manual.*

- MXT3010EP Port1 interface

  The Port1 Controller is a 32-bit multiplexed address and data bus operating at 50 Mhz supporting MXT3010EP accesses to shared memory.

Maker's P1MemMaker reference design is available through the WEB under the "Hardware Development Tools" page and provides the following verilog files:

- arbiter.v

  This module defines the memory arbiter.

- dp.v

  This module defines the data paths.

- dram_cntrl.v

  This module defines the DRAM controller.

- p1orca.v

  This module defines the top level of P1MemMaker

- p1ctrl.v

  This module defines the Port 1 A/B controller.

- pci_be.v

  This module defines the PCI back end controller with CSR.

Endian
Implementation in
P1MemMaker

Several Maker products utilize the Port1 MemMaker FPGA to allow the MXT3010 and a PCI bus to share a Port1 memory. Within Port1 MemMaker, the address and data information on the time-multiplexed Port1 and PCI busses are registered, and the data leads are transposed as shown in Figure 72. *No lead transpositions are performed on the address information.*

FIGURE 71.The Port1 MemMaker FPGA
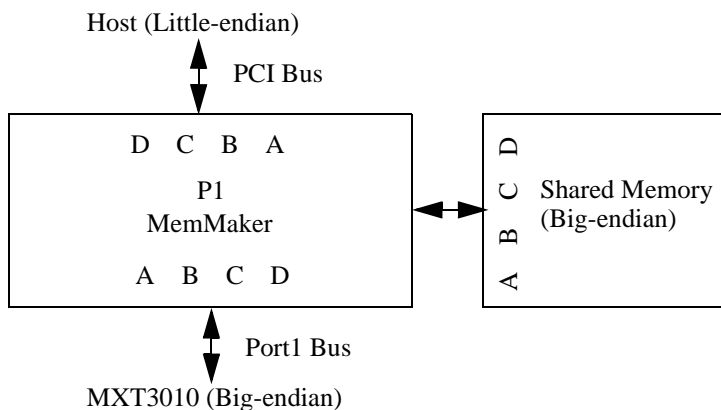
FIGURE 72.Data Path Connections - Shared Memory to PCI
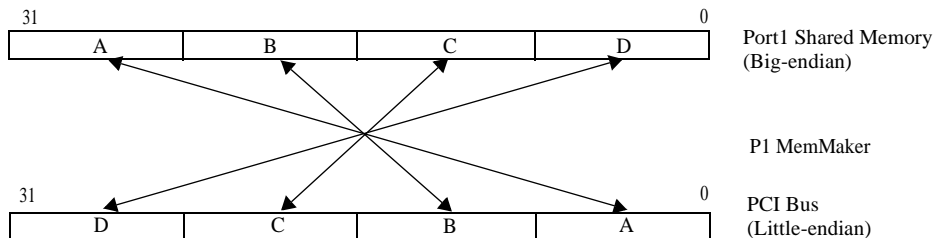
FIGURE 73.Data Path Connections - Shared Memory to MXT3010
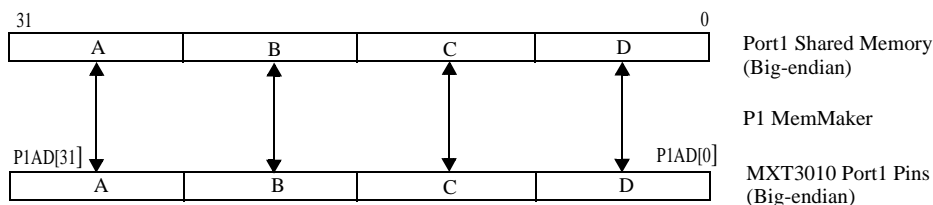
P2MemMaker

The MXT3010EP Port2 interface requires a memory controller function to support bus arbitration, bus selection, bus driving, bus turnaround, and bus holding operations. In Maker's MXT3025 evaluation Board and similar designs, Maker uses *P2MemMaker*, a device that is a integrated memory system controller, a PCI interface, and an MXT3010EP Port2 interface. It performs the following functions:

- Memory System Controller

  The Memory System Controller (MSC) provides the bus arbitration and selection functions for the PCI or Port2 access for tranfers to shared memory. It controls up to 2 Mbytes of shared SRAM. The typical memory system is organized as two (2) 64kx16 SRAMs. The memory system is typically mapped into the PCI memory space and mapped

into the lower 256Kbytes of the MXT3010EP Port2 address space. The MSC supports full speed burst tranfers up to 256 bytes to the memory system, but transfers must not cross 4-Kbyte boundaries. The MSC also controls transfers to the non-burst memory space.

- PCI Interface

    The PCI Bus interface is a 32-bit, 33 Mhz PCI Version 2.1 Implementation supporting the PCI configuration registers, a slave only interface, and no support for initiating transfers from the host processor to shared memory or the MXT3010EP device.

- Port2 Interface

    The Port2 Controller is a 16-bit multiplied Address and Data bus operating at 50 Mhz supporting MXT3010EP accesses to shared memory.

Maker's P2MemMaker reference design is available through the WEB under the "Hardware Development Tools" page and provides the following verilog files:

- mem_cntrl.v

    This module defines the SRAM controller.

- p2_arbiter.v

    This module defines the memory arbiter.

- p2_dp.v

    This module defines the data paths.

- p2_ORCA.v

- p2_pci_be.v

    This module defines the PCI back end controller.

- p2ctrl.v

This module defines the Port 2 Rx/Tx controller.

Endian
Implementation in
P2MemMaker

Several Maker products utilize the Port2 MemMaker FPGA to allow the MXT3010 and a PCI bus to share a Port2 memory. Within the Port2 MemMaker, the address and data information on the time-multiplexed Port2 and PCI busses are registered, and the data leads are transposed as shown in Figure 75. *No lead transpositions are performed on the address information.*

FIGURE 74.The Port2 MemMaker FPGA
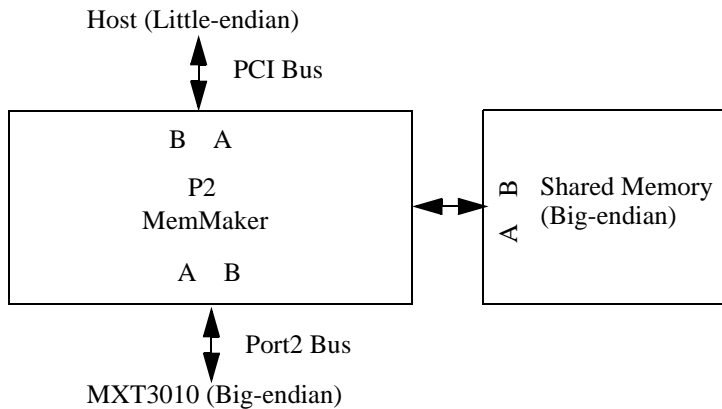
FIGURE 75.Data Path Connections - Shared Memory to PCI
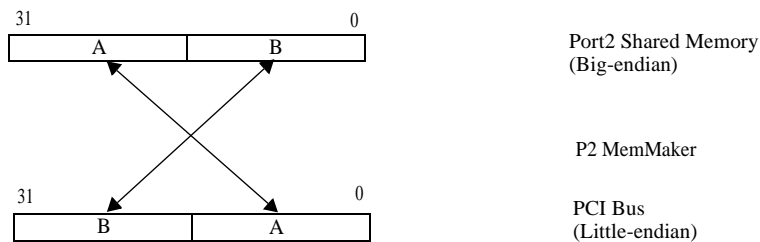
FIGURE 76.Data Path Connections - Shared Memory to MXT3010
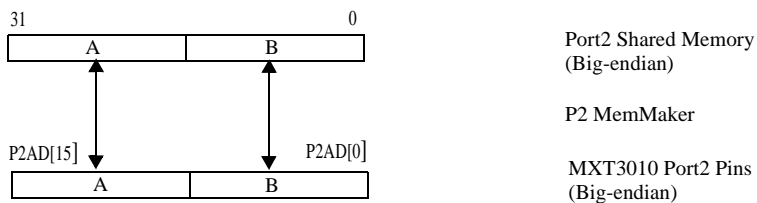
Communications
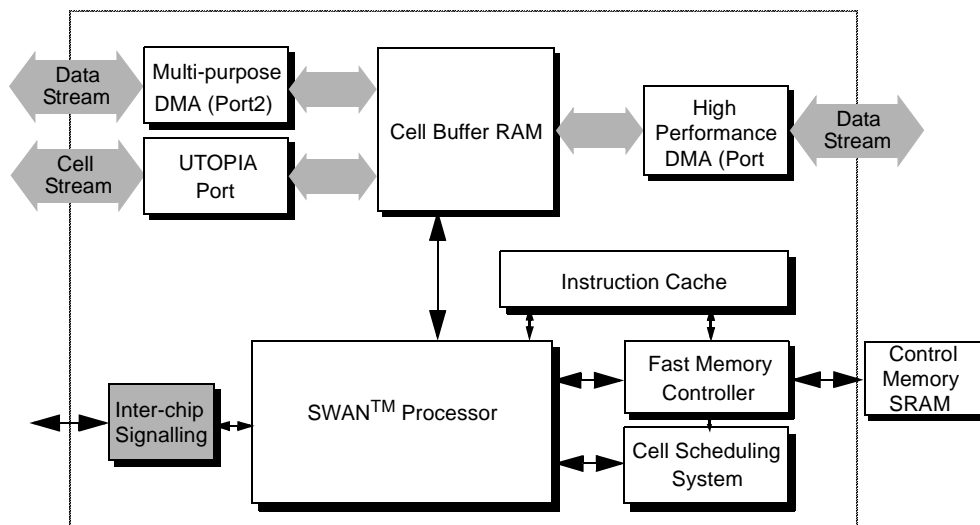
Host/MXT3010 communications include the COMMIN/COM-
MOUT register and the eight pins the MXT3010 assigns for inter-
chip communications. This chapter describes the communications
functions of the COMMIN/COMMOUT register and inter-chip
signalling pins.

The COMMIN/COMMOUT register

The MXT3010 device implements a two-way communications channel with the host processor. The communication channel consists of a 32-bit COMMIN/COMMOUT register implemented as a set of two 16-bit registers, R40 [31:16] and R41 [15:0]. Accessing the COMMIN/COMMOUT register via Port1, the host processor uses the register as a COMMIN register to write information (commands/status/addresses) into the device, and as a COMMOUT register to read information from the device.

Register	*Function*
COMMIN	Host to MXT3010 communications
COMMOUT	MXT3010 to host communications

CIN_BUSY and COUT_RDY

The MXT3010 device implements two output signals that allow both the host and the SWAN processor to determine the state of the COMMIN/COMMOUT register set. Definitions for these signals are provided in Table 42, and timing for these signals is shown in Figure 77 on page 180.

TABLE 42. Definitions of CIN_BUSY and COUT_RDY

Signal	*Function*	
CIN_BUSY	The CIN_BUSY signal is used for host to MXT3010 communications (R40/R41 used as COMMIN).	
	1	The host has written information into R40/R41 that has not yet been read by the SWAN processor.
	0	The SWAN has read the information in R40.
COUT_RDY	The COUT_RDY signal is used for MXT3010 to host communications (R40/R41 used as COMMOUT).	
	1	The SWAN processor has written information into R40 that has not yet been read by the host.
	0	The host has read the information in R40/R41.

As shown above, when the host processor writes to COMMIN (R40/R41), CIN_BUSY is asserted until the SWAN processor reads R40. Before writing the next word into the register R40/R41, the host must be sure that the SWAN processor has processed the previous word. The host does this by testing the state of CIN_BUSY. The state of CIN_BUSY is accessible to the SWAN as ESS6.

When the SWAN processor writes to COMMOUT _HIGH (R40), the COUT_RDY output is asserted until the host reads the COMMOUT register. The COUT_RDY output is accessible to the SWAN processor as ESS7. Therefore, the SWAN can check that the host has read COMMOUT by testing ESS7.

Restrictions on CIN_BSY and COUT_RDY

Since reads and writes to R40 affect the CIN_BUSY and COUT_RDY signals, the SWAN program should read or write R41 before reading or writing R40 when performing 32-bit communication. No such restriction applies to the host, as it uses 32-bit transfers that access R40 and R41 simultaneously. Since R40 controls the flags, COMMOUT_LOW (R41) can be used during debugging to pass data without affecting the flags.

The CIN_BSY flag is cleared when a nullified instruction accesses the COMMIN_HIGH register (R40). Therefore, do not place an instruction that accesses COMMIN_HIGH in a slot that may be nullified. For more information on the nullify operator, see "The Nullify operator" on page 265.

FIGURE 77.Timing of CIN_BUSY and COUT_RDY

CIW = Comm In Write; CIS = Comm In Data Strobe; C01,2 = Comm Out Read1,2; CODV = Comm Out Data Valid

DMA transfer   COMMIN Write   COMMOUT Read
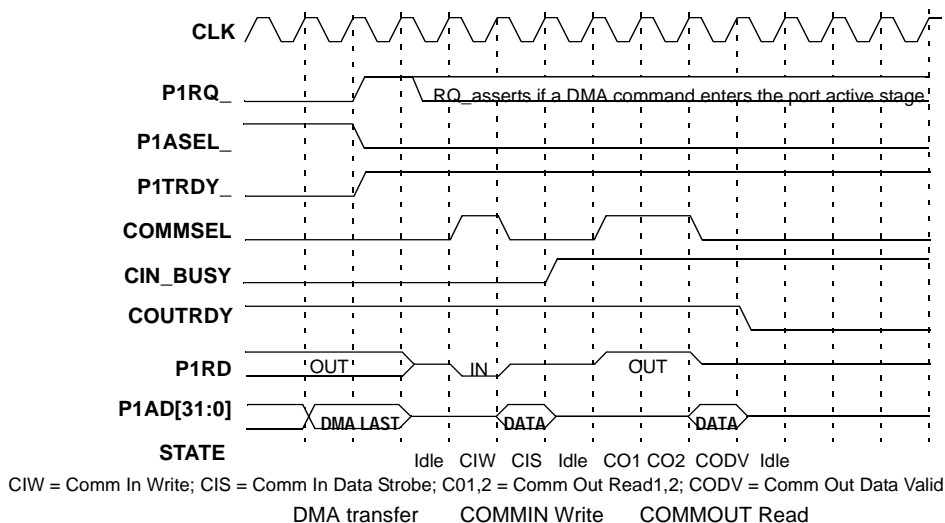
Interchip communications

Besides the COMMIN/COMMOUT register, the MXT3010 dedicates eight pins for interchip communication: four input pins, ICSI_[D:A], and four output pins, ICSO_[D:A].

The ICSI pins

The input pins are listed in Table 43.

TABLE 43.   ICSI pins

Pin	*I/O*	*Connected to*
ICSI_A	Input	Sparse Event/ICS register (R57) bit 12 (read)
		External State Signals (ESS) register (R42) bit 0
ICSI_B	Input	Sparse Event/ICS register (R57) bit 13 (read)
		External State Signals (ESS) register (R42) bit 1
ICSI_C	Input	Sparse Event/ICS register (R57) bit 14 (read)
ICSI_D	Input	Sparse Event/ICS register (R57) bit 15 (read)

The appearances of ICSI_A and ICSI_B in the ESS register are synchronized to the MXT3010 input clock, but are not latched.

In contrast, the appearances of these bits in the Sparse Events/
ICS register (R57) include both a masking feature and latching.

The appearances of ICSI[D:A] in the Sparse Events/ICS register
(R57) are enabled by masking conditions in the System register
(R63). If enabled, each input is sampled by the clock. If the sig-
nal is asserted for two successive clock cycles, this condition is
latched until cleared by the SWAN processor.

The ICSO pins   The output pins are listed in Table 44.

TABLE 44. ICSO pins

Pin	*I/O*	*Connected to*
ICSO_A	Output	Sparse Event/ICS register (R57) bit 12 (set/clear)
ICSO_B	Output	Sparse Event/ICS register (R57) bit 13 (set/clear)
ICSO_C	Output	Sparse Event/ICS register (R57) bit 14 (set/clear)
ICSO_D	Output	Sparse Event/ICS register (R57) bit 15 (set/clear)

The output pins are sourced from four Sparse Event/ICS register
(R57) outputs. The SWAN processor can change the state of any
one of the output pins by changing the state of the Sparse Event
register output bit associated with the pin.

Enabling the   The SWAN processor reads configuration information from
ICSO pins   ICSO_(D:A) during reset. To ensure that the SWAN processor
                      does not drive these pins at reset, the pins are reset in input
                      mode. The SWAN processor senses configuration information
                      from them as it exits reset. To use these pins as outputs, the soft-
                      ware must enable these pins by setting the EN bit in the System
                      register (R63).

For more information on the Sparse Event/ICS register, see
"R57-read Sparse Event/ICS register" on page 213. For more
information on the System register, see "R63 The System regis-
ter" on page 221.

Section 2   Register and Instruction Reference

This section includes register descriptions and the SWAN instruction set.

Registers

The register descriptions are organized by location, starting with the register file in locations R(31:0) and continuing with registers R32 through R63. A table is provided that includes the register location, name, size, and whether it's a read or write register.

Each register description includes the register location, bit map, description, reset value, bit definitions, and notes.

Table 45 lists the registers.

TABLE 45. Hardware registers

Location	*Name*	*Read/Write*
R32	General Purpose - 0000	R/W
R33	General Purpose - FFFF	R/W
R34	General Purpose - FF00	R/W
R35	General Purpose - 0040	R/W
R36-Write	The Bit Bucket	W
R37	General Purpose	R/W
R38	General Purpose	R/W
R39	General Purpose	R/W
R40	COMMOUT/COMMIN(31:16)	R/W
R41	COMMOUT/COMMIN(15:0)	R/W
R42-Read	ESS register	R
R42-Write	Mode Configuration register	Set/Clear
R43-Read	Fast Memory Bit Swap register	R
R43-Write	UTOPIA TX Control FIFO register	W
R44	CRC32PRX (15:0)	R/W
R45	CRC32PRX (31:16)	R/W
R46	CRC32PRY (15:0)	R/W
R47	CRC32PRY (31:16)	R/W
R48	rla Address register	R/W
R49	rla Address register	R/W
R50	rla Address register	R/W
R51	rla Address register	R/W
R52	Alternate Byte Count /ID register	R/W
R53	Instruction Base Address register	R/W
R54	Programmable Interval Timer (PIT0)	R/W
R55	Programmable Interval Timer (PIT1)	R/W
R56	The Fast Memory Data register	R/W
R57-Read	Sparse Event/ICS register	R
R57-Write	Sparse Event/ICS register	Set/Clear
R58	Fast Memory Shadow register	R/W
R59	Branch register	R/W
R60	CSS Configuration register	R/W
R61-Read	Scheduled Address register	R
R62	UTOPIA Configuration register	R/W
R63	System register	R/W

Instructions

The SWAN instruction set is organized functionally. The instructions are described in alphabetical order within each functional area. Also included in this section is a list of the functional groups and an alphabetical list of the instructions.

The specific instruction reference includes the instruction's full name, mnemonic, the layout of the 32-bit instruction word, format, purpose, description, fields, restrictions and any information specific to a functional area.

The functional groups of the instructions are:

- ALU instructions
- Branch instructions
- Cell Scheduling instructions
- DMA instructions
- Load and Store internal RAM and Fast Memory instructions

Table 46 lists the instructions alphabetically.

TABLE 46. Alphabetical list of instructions

Instruction Mnemonic	Instruction	Functional Group	Page
ADD	Add Registers	ALU	234
ADDI	Add Register and Immediate	ALU	235
AND	And Registers	ALU	236
ANDI	And Register and Immediate	ALU	237
BF	Branch Fast Memory First Word Shadow Register	Branch	270
BFL	Branch Fast Memory First Word Shadow Register and Link	Branch	271
BI	Branch Immediate	Branch	272
BIL	Branch Immediate and Link	Branch	273
BR	Branch Register	Branch	274
BRL	Branch Register and Link	Branch	275
CMP	Compare two Registers	ALU	238
CMPI	Compare Register and Immediate	ALU	239
CMPP	Compare two Registers with Previous	ALU	240
CMPPI	Compare Register and Immediate with Previous	ALU	241
DMA1R, DMA1W, DMA2R, DMA2W	DMA Operations	DMA	289, 290, 291, 292
FLS	Find Last Set	ALU	242
LIMD	Load Immediate	ALU	243
LD	Load Register	Load and Store Internal RAM	321
LDD	Load Double Register	Load and Store Internal RAM	322
LMFM	Load Multiple from Fast Memory	Load and Store Fast Memory	308
MAX	Maximum of two Registers	ALU	244
MAXI	Maximum of Register and Immediate	ALU	245
MIN	Minimum of two Registers	ALU	246

Instruction Mnemonic	Instruction	Functional Group	Page
MINI	Minimum of Register and Immediate	ALU	247
OR	Or Registers	ALU	248
ORI	Or Register and Immediate	ALU	249
POPC	Service Schedule	Cell Scheduling	278
PUSHC	Schedule	Cell Scheduling	280
SFT	Shift Right or Left based on Signed Shift Amount	ALU	250
SFTA	Shift Right Arithmetic	ALU	251
SFTAI	Shift Right Arithmetic Immediate	ALU	252
SFTC	Shift Right Circular	ALU	253
SFTCI	Shift Circular Immediate	ALU	254
SFTRI/ SFTLI	Shift Right or Left Immediate	ALU	255
SHFM	Store Halfword to Fast Memory	Load and Store Fast Memory Instructions	311
SRH	Store Register Halfword	Load and Store Fast Memory Instructions	312
ST	Store Register	Load and Store Internal RAM	323
STD	Store Double Register	Load and Store Internal RAM	324
SUB	Subtract Registers	ALU	256
SUBI	Subtract Register and Immediate	ALU	257
XOR	Exclusive-or Registers	ALU	258
XORI	Exclusive-or Register and Immediate	ALU	259

Instruction description notations

The following table lists the abbreviations used in the SWAN processor, describes them briefly, and indicates the functional instruction group(s) within which that abbreviation is used.

TABLE 47. Abbreviations used in SWAN instructions

Abbreviation	Description	Usage
rsa	Source register, software or hardware	ALU, DMA
rsb	Source register, software	ALU, DMA
rd	Destination register, software or hardware	ALU, Load/Store
abc	ALU branch condition IFO[a]	ALU
UM	Automatic update memory IFO	ALU
MODx	Modulo arithmetic IFO	ALU
AE	Always execute IFO	ALU
usi	Unsigned immediate value	ALU
si	Sign-extended 10-bit immediate value	ALU
usa	Unsigned shift amount	ALU
tcsa	Two's-complement shift amount	ALU
li	Long immediate value	ALU
ESS#	External State Signals register bit position	Branch
s	State of ESS bit for comparison	Branch
C	Conditional execution operator	Branch
cso	Counter system operation IFO	Branch
wadr	Target word address	Branch
rla	Load address register	Load/Store
IDX/#	Load address index	Load/Store
LNK	Linking IFO	Load/Store
#HW	Halfword count	Load/Store
BC/#	Byte count	DMA
CRCX, CRCY	CRC generation control	DMA
POD	DMA post-operation directive	DMA

a.   IFO = Instruction Field Option

CHAPTER 9   *Registers*

This chapter describes the registers associated with the SWAN processor.

Register types

The two types of registers in the SWAN processor are general-purpose and control/status. The general-purpose registers are classified as *software registers* because their usage and content is firmware dependent. The registers that control functions and provide status information are classified as *hardware registers*.

Software registers

The SWAN processor has 32 general-purpose software registers, R0-R31, each 16-bits wide. The software registers have no mandatory implicit hardware or software usage conventions. However, restrictions apply when software registers are used with the Load Multiple Fast Memory (LMFM) instruction. The specified

register is restricted based on the use of the Link instruction field option and the length of the transfer. For further information, see "General information for Load and Store Fast Memory instructions" on page 294.

Hardware registers

The SWAN processor has 32 control and status hardware registers, R32-R63. In certain cases the MXT3010 mode configuration affects the register function. For more information on modes, see "R42-write Mode Configuration register" on page 201.

Specifying registers in SWAN instructions

Most of the SWAN instructions include register read or write operations. In those instructions, fields are provided to specify which registers are used. The fields are identified by abbreviations that indicate whether the register is used as a source or as a destination, and whether any restrictions apply to that register. The following table lists the field abbreviations used, their descriptions, and the permitted registers for that field.

TABLE 48.   Field abbreviations

Abbreviation	Description	Permitted Registers	Instruction Type
rsa	Source register, software or hardware	R0-R63	ALU, DMA
rsb	Source register, software	R0-R31	ALU, DMA
rd	Destination register, software or hardware	R0-R63	ALU, Load/Store
rla	Load address register	R48-R51 GA, GB, GC, GD	Load/Store

Initializing software and hardware registers

The software registers R0-R31 are unchanged by device initialization and therefore are indeterminate at power up. Initialization software should clear these registers before use. The hardware register descriptions (R32-R63) indicate which registers are unchanged by device initialization and which are initialized to specific values.

TABLE 49. Hardware registers

Location	*Name*	*Read/Write*
R32	General Purpose - 0000	R/W
R33	General Purpose - FFFF	R/W
R34	General Purpose - FF00	R/W
R35	General Purpose - 0040	R/W
R36-write	The Bit Bucket	W
R37	General Purpose	R/W
R38	General Purpose	R/W
R39	General Purpose	R/W
R40	COMMOUT/COMMIN(31:16)	R/W
R41	COMMOUT/COMMIN(15:0)	R/W
R42-read	ESS register	R
R42-write	Mode Configuration register	Set/Clear
R43-read	Fast Memory Bit Swap register	R
R43-write	UTOPIA TX Control FIFO register	W
R44	CRC32PRX (15:0)	R/W
R45	CRC32PRX (31:16)	R/W
R46	CRC32PRY (15:0)	R/W
R47	CRC32PRY (31:16)	R/W
R48	rla Address register	R/W
R49	rla Address register	R/W
R50	rla Address register	R/W
R51	rla Address register	R/W
R52	Alternate Byte Count /ID register	R/W
R53	Instruction Base Address register	R/W

TABLE 49. Hardware registers

Location	*Name*	*Read/Write*
R54	Programmable Interval Timer (PIT0)	R/W
R55	Programmable Interval Timer (PIT1)	R/W
R56	The Fast Memory Data register	R/W
R57-read	Sparse Event/ICS register	R
R57-write	Sparse Event/ICS register	Set/Clear
R58	Fast Memory Shadow register	R/W
R59	Branch register	R/W
R60	CSS Configuration register	R/W
R61-read	Scheduled Address register	R
R62	UTOPIA Configuration register	R/W
R63	System register	R/W

R32   *General Purpose - 0000*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
General Purpose register															

Description: This is a general purpose read/write register that is initialized to 0x0000. This register is also used during HEC generation (see "HEC generation and check circuit" on page 25.)

Reset value: 0x0000

Bit definitions: N/A

Note: Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R33   *General Purpose - FFFF*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
							General Purpose register								
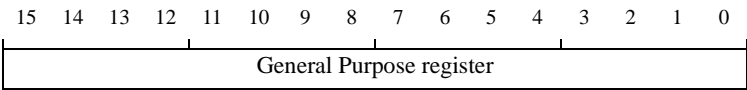
Description: This is a general purpose read/write register that is initialized to 0xFFFF. This register is also used during HEC generation (see "HEC generation and check circuit" on page 25.)

Reset value: 0xFFFF

Bit definitions: N/A

Note: Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R34   *General Purpose - FF00*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
General Purpose register
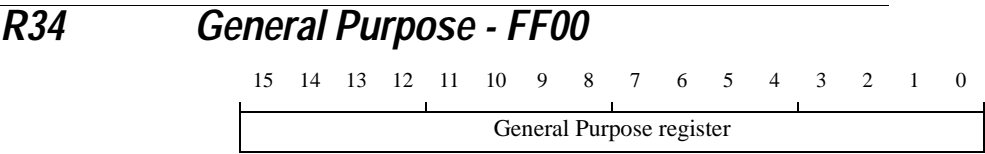
Description: This is a general purpose read/write register that is initialized to FF00.

Reset value: 0xFF00

Bit definitions: N/A

Note: Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R35　　　*General Purpose - 0040*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
General Purpose register															
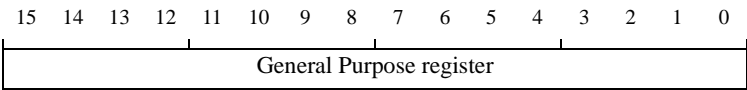
Description:　　　This is a general purpose read/write register that is initialized to 0040.

Reset value:　　　0x0040

Bit definitions:　　N/A

Note:　　　Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R36-write *Bit Bucket register*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit Bucket register															
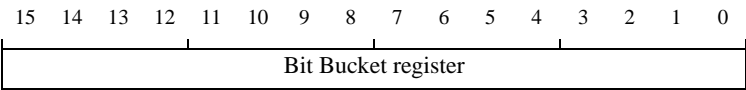
Description: The Bit Bucket register provides the SWAN processor with a location that can be written without any possibility of functional side effects. Information written to R36 is discarded. Thus, software can specify R36 as a destination and discard the results of any operation. R36 is used to emulate a no-op, as well as to implement testing pseudo-ops, such as TSET and TCLR.

Reset value: N/A

Bit definitions: N/A

Notes: The Bit Bucket register should not be read or otherwise specified as a source register. Because of the special treatment of this register location, a read operation can stall the SWAN processor indefinitely.

Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R37-R39   *General Purpose registers*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
General Purpose register																
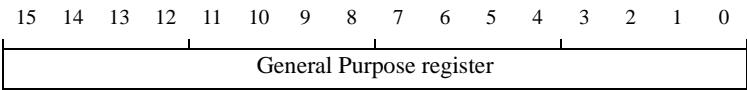
Description: Registers R37, R38, and R39 are 16-bit read/write general purpose registers. They are unchanged by device initialization, and therefore the contents are indeterminate at power-up.

Reset value: Indeterminate

Bit definitions: N/A

Note: Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R40-R41   *Host Communication registers*

R40 COMMIN_HIGH/COMMOUT_HIGH

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
COMMIN/COMMOUT bits [31:16]															
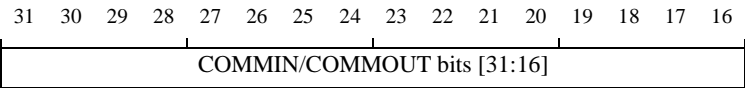
R41 COMMIN_LOW/COMMOUT_LOW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COMMIN/COMMOUT bits [15:0]															

Description: The Host Communication registers provide a 32-bit data transfer path between the SWAN processor and the external host processor. These registers, combined with their associated status flags and pins, form a bi-directional command and response mechanism for host communications.

Reset value: Indeterminate

Bit definitions: N/A

Notes:

1. When the SWAN processor reads location R40, CIN_BUSY (ESS6) is cleared. When the SWAN processor writes location R40, COUT_RDY (ESS7) is set. Since reads or writes to R40 affect the flags, the programmer should read or write R41 before reading or writing R40 to perform 32-bit communications with a host processor.

2. For more information on the Host Communications registers operation, see CHAPTER 8 "Communications" on page 177.

3. Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R42-read   *External State Signals (ESS) register*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

See "Bit Definitions" below

Description:

The SWAN processor can examine the state of certain internal conditions and pins by examining the External State Signals register.

Reset value:

See "Initializing the Mode Configuration register" on page 408.

Bit definitions:

Bit	Read Definition	Bit	Read Definition
15	Unconditional branch[a]	7	COUT_RDY[b]
14	DMA2 queue stage busy	6	CIN_BUSY[c]
13	DMA1 queue stage busy	5	CSS operation in progress
12	DMA2 out or queue stage busy	4	Assigned Cell flag register
11	DMA1 out or queue stage busy	3	RXBUSY Counter ≥ 4
10	TXFULL Counter = full	2	TXFULL Counter ≤ 2
9	RXBUSY Counter = /0	1	ICSI_B[a]
8	Sparse Event register bit OR[d]	0	ICSI_A[e]

a. ESS15 is hardwired to the asserted state. If a Branch instruction does not contain a specified value for the ESS field, the assembler codes that field as 1111, and an unconditional branch is taken.

b. After the SWAN processor writes the COMMOUT register, there is a 5-6 instruction delay before the COUT_RDY bit is set.

c. After the SWAN processor reads the COMMIN register, there is a 5-6 instruction delay before the CIN_BUSY bit is cleared.

d. When an external event occurs that is being monitored by the Sparse Events register, there is a 3-4 instruction delay between the external event and the Sparse Events OR indication of that event.

e. When an external event occurs that is being monitored by an ICSI bit, there is a 3-4 instruction delay between the external event and the ICSI indication of that event.

Note:

Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R42-write   Mode Configuration register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							SF	Set	Reserved			Target Bit Selector			

Description:
The Mode Configuration register includes provision for mode 0 and mode 1 operations. The SWAN processor does not write to the Mode Configuration register directly. Instead, it writes a control byte to the Mode Configuration register to set and clear certain bits. If bit 7 of the control byte is 0, the target bit is cleared (unless it is triggered simultaneously). If bit 7 is 1, the target bit is set.

Reset value:
See "Initializing the Mode Configuration register" on page 408.

Bit definitions:

Bit	*Name*	*Function*
15:9	Reserved	Programs should write zeroes to these bits.
8	Special Features	This bit enables special features in R43-write and R43-read. (See page 204 and page 205) 0 Special features disabled (normal operation) 1 Special features enabled
7	Set	The state of this bit determines whether the bit selected by the Target Bit Selector is set or cleared 0 The target bit is cleared 1 The target bit is set
6:4	Reserved	Programs should write zeroes to these bits.
3:0	Target Bit Selector	These bits select which bit is set or cleared.

R42 Bit	*Target Bit Selected*	*Bit State and Function*
0	0000	HEC Control
		0 HEC is generated and inserted
		1 HEC is omitted
1	0001	Cell Length Control
		0 52 byte cells
		1 56 byte cells

R42 Bit	*Target Bit Selected*	*Bit State and Function*
2	0010	Programs should write zeroes to these bits.
3	0011	Programs should write zeroes to these bits.
4	0100	Fast Memory Mode Control
		0 Fast Memory is in Mode 0
		1 Fast Memory is in Mode 1
5	0101	DMA Plus Control
		0 DMA Plus disabled
		1 DMA Plus performs automatic rla
6	0110	Port1 Operation Control
		0 Port1 normal operation
		1 Port1 Early End enabled
7	0111	Port2 Operation Control
		0 Port2 normal operation
		1 Port2 Early End enabled
8	1000	Reserved
		0 Reserved
		1 Reserved for PLL test mode
9	1001	R32 Control
		0 R32 in normal operation
		1 HEC8 circuit enabled on R32
10	1010	R55 Control
		0 PIT1 is disabled; R55 is a 16-bit R/W register
		1 PIT1 is enabled; R55 operates as a timer
11	1011	R54 Control
		0 PIT0 is disabled; R54 is a 16-bit R/W register
		1 PIT0 is enabled; R54 operates as a timer
13,12	1100, 1101	Programs should write zeroes to these bits.
14,15	1110, 1111	Reserved

Notes: Register access rules apply. See Table 4 on page 24.

Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R43-read   Fast Memory Bit Swap register (R42w[8]=0)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Fast Memory Bit Swap register (Note bit order)																
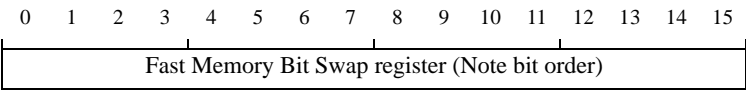
Description   When bit [8] of R42-write is zero (0), this register contains the same data as the Fast Memory Byte register (R56) with the bit order reversed.

Reset value: Indeterminate

Bit definitions: N/A

Notes: R43 can be used to implement a Find First Set instruction by loading a value into R56 and applying the Find Last Set (FLS) instruction (page 242) to R43.

Register access rules apply. See Table 4 on page 24.

Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R43-read *Special Features register (R42w[8]=1)*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		Port2 DMAs			Reserved			URX Count				UTX Count			

Description   When bit [8] of R42-write is one (1), this register implements special configuration features.

Reset value: Indeterminate

Bit definitions:

Bit	Name	Function
15:14	Reserved	Programs should write zeroes to these bits.
13:11	Port2 DMAs	Port2 DMAs completed. This three-bit counter is incremented on the last transfer of a DMA2 operation. It is decremented in software by a branch instruction with the following syntax: bi$label DMA count.
10:8	Reserved	Programs should write zeroes to these bits.
7:4	URX Count	This four-bit counter can read either the current state of the UTOPIA Receiver's Busy or Full Counter, depending upon R43-write [9].
3:0	UTX Count	This four-bit counter can read either the current state of the UTOPIA Transmitter's Busy or Full Counter, depending upon R43-write [8].

Notes: Register access rules apply. See Table 4 on page 24.

Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R43-write *UTOPIA Control FIFO register*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		Configuration Options						Rsv	I	CG			TXPHY		

Description:

Data written to the UTOPIA Control FIFO register provides certain characteristics for cells scheduled for transmission from Cell Buffer RAM through the UTOPIA port. The data written to this register is stored in a FIFO-like internal memory until an actual cell transmission by the UTOPIA port controller removes it. Up to eight control entries can be stored in the FIFO.

The upper byte of this register implements special configuration features controlled by bit [8] of "R42-write Mode Configuration register" on page 201.

Reset value:

Indeterminate

Bit definitions
(Lower byte):

Bit	*Name*	*Function*
7	Reserved	Programs should write zero to this bit.
6	I	Insert unassigned cell
5	CG	Generate and insert a CRC10 for this cell
4:0	TXPHY	Select the address of the target PHY in a multi-PHY system

Bit definitions (Upper byte):	If R42w [8] = 0, these bits are reserved, and programs should write zeroes to them. If R42w [8] = 1, these definitions apply:

Bit	Name	Function
15:14	Reserved	Programs should write zeroes to theses bits
13	PIT1 Sel	PIT1 External Select Pin 0 = URX_CTRL4 1 = ICSI_C
12	PIT0 Sel	PIT0 External Select Pin 0 = UTX_CTRL4 1 = ICSI_D
11	PIT1 Mode	PIT1 External Mode 0 = Disabled 1 = PIT1 is clocked by the rising edge of the external bit selected by bit [13]
10	PIT0 Mode	PIT0 External Mode 0 = Disabled 1 = PIT0 is clocked by the rising edge of the external bit selected by bit [12]
9	URX Count Select	UTOPIA Receiver Count Select 0 = URX Count in R43read is Receiver Busy 1 = URX Count in R43read is Receiver Full
8	UTX Count Select	UTOPIA Receiver Count Select 0 = UTX Count in R43read is Transmitter Busy 1 = UTX Count in R43read is Transmitter Full

Notes:

1. Software should write the control entry into the UTOPIA Control FIFO before incrementing TXBUSY. For more information on UTOPIA port operation, see CHAPTER 6 "The UTOPIA port" on page 69.

2. The FIFO is a hardware-managed 8-deep circular list. Entries can be re-used without writing new data.

3. CRC10 overwrites the last ten bits of the cell with the computed CRC value.

4. When bit I is set, the MXT3010 hardware stuffs an unassigned cell into the UTOPIA Control Byte FIFO without accessing the Cell Buffer RAM.

5. Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R44-R47 *CRC32PRX and CRC32PRY registers*

R44 CRC32PRX [15:0], R46 CRC32PRY [15:0]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CRC Partial Result registers (bits [15:0])															
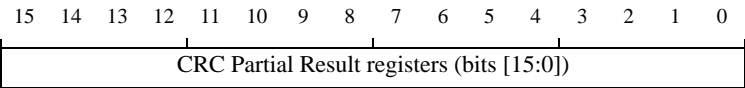
R45 CRC32PRX [31:16], R47 CRC32PRY [31:16]]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CRC Partial Result registers (bits [31:16])															

Description: These registers contain the partial results of CRC32 calculations. The CRCX and CRCY bits in the DMA instruction or the X and Y bits in the Alternate Byte Count/ID register (R52) determine which, if any, register set is used. Use of CRCX/CRCY control or X/Y control depends on whether the DMA instruction contains a BC/# instruction field option.

Reset value: 0x0000

Bit definitions: N/A

Notes: If R44-R47 are not used for CRC calculations, they can be used as general purpose registers. When used as general purpose registers, register access rules apply. See Table 3 on page 23.

Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R48-R51　*Local Address registers (rla)*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				S	G	MEM									

Description:

The Local Address registers provide four hardware registers that are used as address registers by local (internal) memory loads and stores.

Reset value:

Indeterminate

Bit definitions:

Bit	Name	Function
15:12	Reserved	Read, and should be written, as zeroes.
11	S	Scoreboard/Cell Buffer selection
		0　　Cell Buffer RAM access
		1　　Scoreboard access

For Cell Buffer RAM access, the following bit definitions apply:

10	G	Cell Buffer RAM Address Method
		0　　Linear Address
		1　　Gather Address
9:0	MEM	These 10 bits provide byte addressing for the 512 16-bit halfwords in the Cell Buffer RAM.

For Scoreboard access, the following bit definitions apply.

10:0	MEM	These 11 bits provide byte addressing for the 512 32-bit words in the Scoreboard.

Notes:

1. The MXT3010 implements four fixed value registers that can also be used as address registers. These rla constants are GA, GB, GC, and GD that are fixed at 0x400, 0x420, 0x440, and 0x460, respectively.

2. For more information on the Load and Store instructions, see "General information for Load and Store internal RAM instructions" on page 314.

3. Register access rules apply. See Table 3 on page 23.

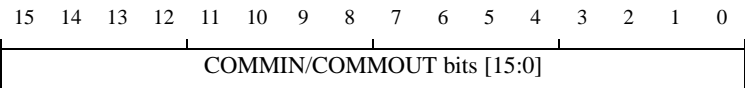4. Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R52   *Alternate Byte Count/ID register*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REV			Reserved			X	Y	Count							

Description: Software can use the Alternate Byte Count/ID register to pro-
vide DMA instruction information to either the Port1 or Port2
interface when executing a DMA instruction. DMA operations
use the contents of R52 if a DMA instruction is executed without
a BC/# instruction field option.

Reset value: 0x0000

Bit definitions:

Bits	*Name*	*Function*
15:13	REV	Device ID field. On reads, returns a value of 000xb for MXT3010 revision A, 001xb for MXT3010 revision B/C.
12:10	Reserved	Read, and should be written, as zeroes.
9	X	CRCX bit If set, a CRC32 partial result is generated based on CRC32PRX register's initial value and the result is deposited into CRC32PRX
8	Y	CRCY bit If set, a CRC32 Partial Result is generated based on CRC32PRY register's initial value and the result is deposited into CRC32PRY
7:0	Count	DMA Byte Count 00 = Zero byte operation FF = 255 byte operation

Note: Restrictions apply to the use of LD, LDD instructions with this
register. See "Register access rules" on page 22.

R53 *Instruction Base Address register*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved									Boot	NC	Segment ID				

Description:

The SWAN processor supports an instruction space of 128K 32-bit instructions, organized as 32 segments of 4K words each. The lowest order 5 bits of this register provide the ability to select any of the 32 segments for user code.

Reset value:

The Reset value is initially 0x0040, and then is dependent upon the starting address of the bootstrap loader.

Bit definitions:

Bits	*Name*	*Function*
15:7	Reserved	Read, and should be written, as zeroes.
6	Boot	This flag is set by the SWAN processor during its power-up initialization routine, and disables the execution of user instructions until the boot process is finished. This bit is cleared by the SWAN processor at the conclusion of power-up initialization.
5	NC	Non-Cached. Instructions executed while this bit is set will not be cached.
4:0	Segment ID	These bits are used as Fast Memory Address bits [18:14] during instruction fetches from the Fast Memory, and thus select which of 32 segments of 4K words will be addressed.

Note:

Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R54-R55   *Programmable Interval Timer registers*

R54 PIT0 [15:0], R55 PIT1 [15:0]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				Programmable Interval Timer registers											
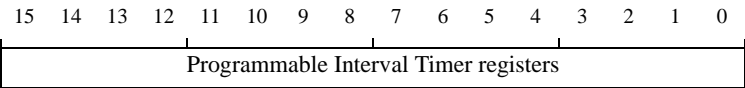
Description: The MXT3010 contains two 16-bit programmable interval timers (PITs), PIT0, and PIT1.

The present values of the PITs are mapped into the Programmable Interval Timer registers R54 (PIT0) and R55 (PIT1). The SWAN processor can read the present value of a PIT by specifying either R54 or R55 in an ALU operation.

Firmware sets an initial value by writing into R54 (PIT0) or R55 (PIT1). When firmware writes an initialization value, that value is immediately transferred into the PIT. PIT0 decrements by one on each rising edge of the external clock. PIT1 decrements by one on each rising edge of the CPU clock, which operates at twice the frequency of the external clock. When PIT0 times out, Bit 4 of the Sparse Events register (R57) is set. When PIT1 times out, Bit 5 of the Sparse Events register (R57) is set. Upon time-out of a PIT, it is automatically reloaded with its count initialization value and the count down process begins anew.

Reset value: 0x0000

Bit definitions: N/A

Notes: Firmware enables/disables a PIT from counting, timing out, and setting its bit in the Sparse Event register via enable bits in "R42-write Mode Configuration register" on page 201.

Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R56 *Fast Memory Data register*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						Fast Memory Data register									
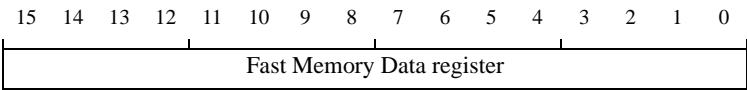
Description: The Store Halfword to Fast Memory (SHFM) instruction writes the contents of R56, the Fast Memory Data register, into the Fast Memory location specified by registers rsa and rsb. The contents of R56 are first entered into the Fast Memory Controller's write buffer before being written out to memory.

Reset value: Indeterminate

Bit definitions: N/A

Notes: An SHFM can immediately follow an instruction that modifies R56.

Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R57-read   Sparse Event/ICS register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								See "Bit Definitions" below							

Description:

This register records events that occur infrequently. Hardware performs a logical OR operation on bits (5:0) of this register and provides the result in ESS8. Hardware clears bits [9:6]when the condition causing the bit to be set is no longer true. Other bits (marked R/W) are cleared by the software via the "R57-write Sparse Event/ICS register (Set/Clear)" on page 214.

Reset value: N/A

Bit definitions:

Bits	Description	R/W
15:12	ICSO_(D:A)[a] These bits control the ICSO_(D:A) pins if the ICSO Output Enable bit of the System register is set. The SWAN can set and clear these bits to signal external devices.	R/W
11	ICSO_A Select: 0 = ICSO_A_SEL; 1 = TX_IDLE_SOC	R/W
10	ICSO_B_SEL: 0 = ICSO_B; 1 = STALL_DLY.	R/W
9	TXBUSY state indicator	R
8	RXFULL state indicator	R
7	CRC32X Error Indicator from Port1 Test only at the completion of a DMA operation.	R
6	CRC32Y Error Indicator from Port1 Test only at the completion of a DMA operation.	R
5	PIT1 Time Out Set when PIT1 counts down to 0.	R/W
4	PIT0 Time Out Set when PIT0 counts down to 0.	R/W
3:0	ICSI_(D:A)[b] Set if corresponding MXT3010 input is set and corresponding SER enable bit is set in the System register.	R/W

a. When the SWAN processor changes the state an ICSO bit, there is a 3-4 instruction delay before the new state appears at the ICSO pin.

b. When an external event occurs, there is a 3-4 instruction delay between the external event and the ICSI indication of that event.

c. LD, LDD restrictions apply. See "Register access rules" on page 22.

R57-write *Sparse Event/ICS register (Set/Clear)*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								Set	Reserved			Target Bit Selector			

Description: The SWAN processor does not write to the Sparse Event register directly. Instead, it writes a control byte to the Sparse Event register to set and clear certain bits. If bit 7 of the control byte is 0, the target bit is cleared (unless held set by hardware conditions). If bit 7 is 1, the target bit is set (unless held clear by hardware conditions).

Reset value: N/A

Bit definitions:

Bit	*Name*	*Function*
15:8	Reserved	Programs should write zeroes to these bits.
7	Set	The state of this bit determines whether the bit selected by the Target Bit Selector is set or cleared 0 The target bit is cleared 1 The target bit is set
6:4	Reserved	Programs should write zeroes to these bits.
3:0	Target Bit Selector	These bits select which bit is set or cleared. The target bits are listed in "R57-read Sparse Event/ICS register" on page 213.

Examples:

Bit	*To Set, Write*	*To Clear, Write*	*Bit*	*To Set, Write*	*To Clear, Write*
0	0x80	0x00	10	0x8A	0x0A
1	0x81	0x01	11	0x8B	0x0B
2	0x82	0x02	12	0x8C	0x0C
3	0x83	0x03	13	0x8D	0x0D
4	0x84	0x04	14	0x8E	0x0E
5	0x85	0x05	15	0x8F	0x0F

Notes:
1. Bits [9:6] are read only.
2. Register access rules apply. See Table 3 on page 23.
3. Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R58 *Fast Memory Shadow register*
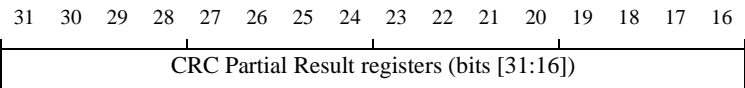
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				Branch Target Field											

Description: The Fast Memory Shadow register is automatically loaded with the first 16-bit word returned from Fast Memory during a read operation (LMFM instruction) that specifies the Link (LNK) instruction field option. The Branch Fast Memory instructions, BF and BFL, use the contents of this register as the target address of the branch operation.

Reset value: Indeterminate

Bit definitions: The Branch Target Field specifies the absolute word address within the current code segment (4096 words) at which execution is to continue when using the Branch Fast Memory instructions, BF and BFL. The reserved bits (15:12) are read, and should be written, as zeroes.

Notes:
1. Software can read and write the Fast Memory Shadow register in the same fashion as the Branch register (R59). To avoid accessing a stale value, separate the BF or BFL instruction from a preceding write to R58 by at least one instruction. See Table 3 on page 23

2. Software can use BF/BFL to gain fast access to a service address contained in the first halfword of a Channel Descriptor. Execution of a BF/BFL following execution of an LMFM instruction with LNK causes a CPU stall until the first halfword is read from memory. When the first halfword is returned, the stall condition terminates. Software can avoid a stall by separating the LMFM from the BF/BFL by at least five instructions.

3. Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R59　　*Branch register*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				Branch Target Field											

Description:　　　　The Branch register instructions, BR and BRL, use the content of this register as the target address of the branch operation. The address in R59 represents an absolute address to branch to within the active segment.

Reset value:　　　　Indeterminate

Bit definitions:　　The Branch Target Field specifies the absolute word address within the current code segment (4096 words) at which execution is to continue when using the Branch Register instructions, BR and BRL. The reserved bits (15:12) are read, and should be written, as zeroes.

Notes:　　　　　　Software can read and write the Branch register. To avoid accessing a stale value, separate the BR or BRL instruction from a preceding write to R59 by at least one instruction. See Table 3 on page 23

Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R60 *The Cell Scheduling System (CSS) Configuration register*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E	CR	SZ		CID				Reserved							

Description: The CSS Configuration register indicates the base address in memory of the Connection ID table. It also indicates the size of the Scoreboard to be used.

Reset value: 0x00FF

Bit definitions:

Bits	*Name*	*Description*
15	E	CSS error flag
14	CR	0 = No CCS Reset 1 = CSS Reset
13:12	SZ	Scoreboard Section Size 00 = 2,048 bits/entries per section; up to 8 sections 01 = 4,096 bits/entries per section; up to 4 sections 10 = 8,192 bits/entries per section; up to 2 sections 11 = 16,384 bits/entries per section; 1 section
11:8	CID	Connection ID Table Base Address: Used as FADRS(18:15) on Connection ID Table accesses for PUSHC and POPC.
7:0	Reserved	Reserved. The bits are undefined on reads and should be written as zeroes.

Notes:
1. Software must initialize the CSS Configuration register before using the Cell Scheduling System.
2. Register access rules apply. See Table 4 on page 24.
3. Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R61-read   *Scheduled Address register*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				Connection ID Table Address											

Description:

At the completion of a PUSHC operation (as indicated by the clearing of ESS5 in R42), the SWAN processor can read the selected Connection ID Table address (FADRS [18:1]). The number presented in this register is the 14-bit halfword address offset (FADRS [14:1]) within the table, and the table base address (FADRS [18:15]) is obtained from the CID bits in "R60 The Cell Scheduling System (CSS) Configuration register" on page 217.

Reset value:

Indeterminate

Bit definitions:

Bits 13:0 are automatically loaded with the Connection ID Table address selected by the Cell Scheduling System at the completion of a scheduled write operation, PUSHC and PUSHF. Bits 15 and 14 are reserved. They should be ignored.

Notes:

For more information on the Cell Scheduling System (CSS), see "The Cell Scheduling System" on page 27. For more information on the PUSHC operation, see "PUSHC Schedule" on page 280.

Software must not check this register until an outstanding PUSHC/PUSHF is complete. See "Scheduling" on page 32.

Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

R62 *The UTOPIA Configuration register*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
See "Bit Definitions" below															

Description: The UTOPIA Configuration register determines the operating characteristics of the UTOPIA port. In addition to the R62 register, two target bits ([0] and [1]) from R42 (the Mode Configuration register) are used to program the UTOPIA port.

Reset value: 0x0000

Bit definitions:

Bits	Description
15:14	Number of Physical PHY devices present

This value tells the UTOPIA Port Receiver the number of physical PHY devices present. This in turn determines the number of RXCLAV/TXCLAV and RXENB_/TXENB_ signals that should be used. Please see the UTOPIA port chapter in Section 1 and Table 50 below.

   00   Reserved
   01   1-PHY mode
   10   2-PHY mode
   11   Reserved

13:9   UTOPIA Port Most Significant PHY Address

   .   The UTOPIA Port Receiver polls PHY devices searching for an RXCLAV by incrementing the polled address according to the UTOPIA Level 2 specification. The UTOPIA Port Transmitter knows that it has reached the last address and should begin at zero again when it reaches this address. For examples of the use of these bits, see Figure 37 on page 89 and Figure 38 on page 90.

8   UTOPIA Port Data Bus Width

   0   16 Bits Wide
   1   8 Bits Wide
        Direction is determined by which device is not in Reset Mode. (See "Selecting transmit or receive mode" on page 72.)

Bits	Description
7	UTOPIA Port Operational / Output Clock Frequency Selection
	0 TXCLK and RXCLK operate at 1/2 of internal CLK frequency.
	1 TXCLK and RXCLK operate at 1/4 of internal CLK frequency. Note: 1/2 the internal CLK frequency is on the FN pin.
6:4	Transmit Cell Buffer Size in the Cell Buffer RAM
	001 Transmitter Buffer Size in the Cell Buffer RAM = 2 cells
	010 Transmitter Buffer Size in the Cell Buffer RAM = 3 cells
	011 Transmitter Buffer Size in the Cell Buffer RAM = 4 cells
	100 Transmitter Buffer Size in the Cell Buffer RAM = 5 cells
	101 Transmitter Buffer Size in the Cell Buffer RAM = 6 cells
	110 Transmitter Buffer Size in the Cell Buffer RAM = 7 cells
	111 Transmitter Buffer Size in the Cell Buffer RAM = 8 cells
3:1	Receive Cell Buffer Size in the Cell Buffer RAM
	000 UTOPIA Port Receiver in Reset Mode. All Rx outputs are tristated. This includes RXDATA (a bidirectional signal), but does not include RXCLK. All inputs are pulled to their inactive states by the MXT3010.
	001 Receiver Buffer Size in the Cell Buffer RAM = 2 cells
	010 Receiver Buffer Size in the Cell Buffer RAM = 3 cells
	011 Receiver Buffer Size in the Cell Buffer RAM = 4 cells
	100 Receiver Buffer Size in the Cell Buffer RAM = 5 cells
	101 Receiver Buffer Size in the Cell Buffer RAM = 6 cells
	110 Receiver Buffer Size in the Cell Buffer RAM = 7 cells
	111 Receiver Buffer Size in the Cell Buffer RAM = 8 cells
0	UTOPIA Receiver Reduction Mode Enable Bit
	0 Reduction Function Disabled (ATM Header bytes [2:3] written into the Cell Buffer RAM unchanged)
	1 Reduction Function Enabled (ATM header bytes [2:3] written into the Cell Buffer RAM after reduction function performed according to Reduction Mask Setting selected by R63[6:0]).

TABLE 50. Signal utilization for 1-PHY and 2-PHY modes

Mode	TX/RX CLAV	TX/RX ENB	ADRS
1 PHY	TX/RX_CLAV	TX/RX_ENB_	TX/RX CTRL [3:0]
2 PHY			
PHY 0	TX/RX_CLAV	TX/RX_ENB_	TX/RX CTRL [1:0]
PHY 1	TX/RX CTRL [3]	TX/RX CTRL [2]	TX/RX CTRL [1:0]

Note: See "Register access rules" on page 22.

R63   *The System register*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OD	OC	BS		IMASK				EN	VPI/VCI						

Description:

The System register determines the operating characteristics of the MXT3010.

Reset value:

0x0000

Bit definitions:

Bits	Name	Function	
15:14	OD, OC		Reflect the state of ICSO_(D:C) at reset removal.
13:12	BS	Boot Source	These bits indicate the source that was used to boot the MXT3010
		00	Via Fast Memory (used only in simulation)
		01	Via Port 1
		10	Via Port 2
		11	Via COMMIN register
11:8	IMASK	ICSI_(D:A) Sparse Event register enable	
		0	Bit in Sparse Event register is not set when ICSI_x is high.
		1	Bit in Sparse Event register is set when ICSI_x is high.
7	EN	ICSO_(D:A) Output Enable	
		0	Outputs Tristates (default at reset)
		1	Outputs Actively Driving
			Note: The MXT3010 reads configuration information from ICSO_(D:A) during reset. To ensure that the MXT3010 does not drive these pins at reset, the pins are reset in input mode. The MXT3010 senses configuration information from them as it exits reset. Software can enable these pins as outputs by setting this bit to one.

Bits	*Name*	*Function*	
6:0	VPI/VCI	UTOPIA Receiver Reduction Mask	
		Setting	Value written into ATM Header lower halfword in CBR
		0000001	{0,0,0,0,0,0, vpi(0), vci(7:0), clp}
		0000011	{0,0,0,0,0, vpi(1:0), vci(7:0), clp}
		0000111	{0,0,0,0, vpi(2:0), vci(7:0), clp}
		0001111	{0,0,0, vpi(3:0), vci(7:0), clp}
		0000010	{0,0,0,0,0, vpi(0), vci(8:0), clp}
		0000110	{0,0,0,0, vpi(1:0), vci(8:0), clp}
		0001110	{0,0,0, vpi(2:0), vci(8:0), clp}
		0011110	{0,0, vpi(3:0), vci(8:0), clp}
		0000100	{0,0,0,0, vpi(0), vci(9:0), clp}
		0001100	{0,0,0, vpi(1:0), vci(9:0), clp}
		0011100	{0,0, vpi(2:0), vci(9:0), clp}
		0111100	{0, vpi(3:0), vci(9:0), clp}
		0001000	{0,0,0, vpi(0), vci(10:0), clp}
		0011000	{0,0, vpi(1:0), vci(10:0), clp}
		0111000	{0, vpi(2:0), vci(10:0), clp}
		1111000	{vpi(3:0), vci(10:0), clp}
		0010000	{0,0,vpi(0), vci(11:0), clp}
		0110000	{0,vpi(1:0), vci(11:0), clp}
		1110000	{vpi(2:0), vci(11:0), clp}
		0100000	{0,vpi(0), vci(12:0), clp}
		1100000	{vpi(1:0), vci(12:0), clp}
		1000000	{vpi(0), vci(13:0), clp}
		0000000	{vci(14:0), clp}

Note: Register access rules apply. See Table 4 on page 24.

Restrictions apply to the use of LD, LDD instructions with this register. See "Register access rules" on page 22.

CHAPTER 10 *Arithmetic Logic Unit Instructions*

The arithmetic and logical instructions of the SWAN processor manipulate data contained in the register set.

Addressing modes

Two addressing modes are supported for arithmetic and logical instructions: triadic register and immediate.

Triadic register

Triadic register addressing mode uses three fields in the instruction to specify two source registers (rsa and rsb) and a destination register (rd). The rsa and rd registers might be any of the software registers (R0-R31) or any of the hardware registers (R32-R63). The rsb register can only be one of the software registers.

FIGURE 78. Triadic register operation
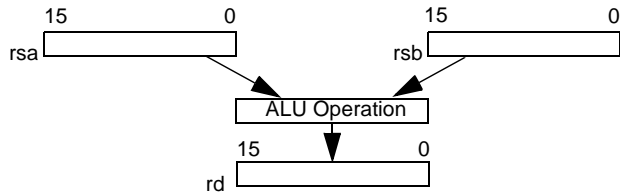
FIGURE 79. Triadic instruction format

- - - Instruction specific fields
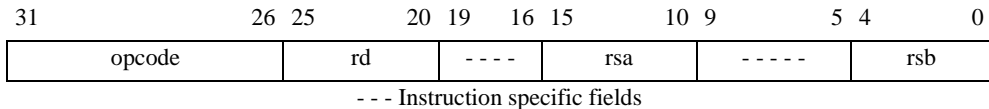
Immediate

Immediate addressing mode uses two bit fields in the instruction to specify one source register (rsa) and a destination register (rd). The second operand is provided as an immediate value in the instruction word.

The width of the immediate value and its format (signed, unsigned) is instruction dependent. Arithmetic instructions (ADDI, SUBI) use a 6-bit unsigned immediate value. Logical instructions (ANDI, CMPI, CMPPI, MAXI, MINI, ORI, XORI) use a 10-bit sign-extended immediate value. Shift instructions (SFTAI, SFTCI, SFTRI, SFTLI) use instruction-specific formats similar to the 6-bit immediate field used in arithmetic operations.

FIGURE 80.Immediate 10-bit instruction format

31		26	25		20	19	16	15		10	9		0
	opcode			rd		- - - -			rsa			10-bit signed immediate	

- - - Instruction specific fields

FIGURE 81.Immediate 6-bit instruction format

31		26	25		20	19	16	15		10	9	6	5		0
	opcode			rd		- - - -			rsa		- - - -		6-bit unsigned im.		

- - - Instruction specific fields
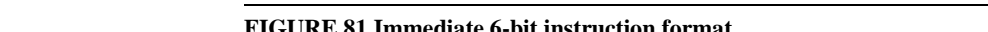
Overflow flag

Signed arithmetic is supported by an Overflow flag. During addition operations, the Overflow flag is set when both source operands have the same sign, and the sign of the result is different. During subtraction operations, the Overflow flag is set when the signs of the source operands differ, and the sign of the result matches the sign of the second operand.

Instructions that use this flag

Only add and subtract operations affect the Overflow flag. None of the other ALU instructions can change the state of this flag, nor can add and subtract operations that specify the use of mod-

ulo arithmetic. Other ALU instructions can test the state of this flag that resulted from the last arithmetic operation by using the Branch No Overflow ALU branching option.

Instruction options

Certain options are common to many of the ALU instructions. These options include modulo arithmetic, automatic memory updating, and ALU branching.

Modulo arithmetic

The ALU in the SWAN processor supports modulo arithmetic. With modulo arithmetic, the operation of an ALU instruction is constrained to the number of bit positions specified in the instruction word. Bits outside the specified operation width are not affected. Source bits from rsa are simply copied to the corresponding destination bits in rd.

Modulo arithmetic can be specified for any field width, from one bit to fifteen bits. The width of the desired modulo arithmetic is specified in ALU instructions with an instruction field option. Full 16-bit operation is the default width for ALU instructions where no modulo arithmetic instruction field option is specified. Table 51 on page 227 lists the modulo arithmetic options.

TABLE 51. Modulo arithmetic options

IFO	Width	rd	IFO	Width	rd
MOD2	1	rsa[15:1] | alu[0]	MOD512	9	rsa[15:9] | alu[8:0]
MOD4	2	rsa[15:2] | alu[1:0]	MOD1K	10	rsa[15:10] | alu[9:0]
MOD8	3	rsa[15:3] | alu[2:0]	MOD2K	11	rsa[15:11] | alu[10:0]
MOD16	4	rsa[15:4] | alu[3:0]	MOD4K	12	rsa[15:12] | alu[11:0]
MOD32	5	rsa[15:5] | alu[4:0]	MOD8K	13	rsa[15:13] | alu[12:0]
MOD64	6	rsa[15:6] | alu[5:0]	MOD16K	14	rsa[15:14] | alu[13:0]
MOD128	7	rsa[15:7] | alu[6:0]	MOD32K	15	rsa[15] | alu[14:0]
MOD256	8	rsa[15:8] | alu[7:0]	blank	16	alu[15:0]

A modulo arithmetic operation does not affect the ALU flag registers or Overflow.

Using modulo arithmetic and branch conditions

The Branch On Zero and the Branch On Non-Zero ALU branch conditions are evaluated based on the bits within the modulo field only. This allows one to test, for example, for the occurrence of a boundary crossing by a memory pointer that has a non-zero base address.

Modulo arithmetic example

In the following example, two Load Immediate (LIMD) instructions are used to load the hex numbers 1234 and 1111 into registers r0 and r1 respectively. A modulo 16 addition of these two registers is then performed, and the result is placed in r2. Note that due to the modulo 16 addition, only bits [3:0] have been affected by the addition process.

Address	Instruction	Result
0x0000	LIMD r0, 0x1234	r0 <- 0x1234
0x0001	LIMD r1, 0x1111	r1 <- 0x1111
0x0002	ADD r0, r1, r2 MOD16	r2 <- 0x1235

Automatic memory updates

When the automatic memory update feature is enabled, the Fast Memory controller writes the results of the ALU operation back into the linked Fast Memory location associated with the destination register, rd. This feature eliminates the need for separate store instructions to write results back to memory, thus saving machine cycles and reducing latency.

The Update Memory field (UM)

All ALU instructions except the compare instructions (CMP, CMPI, CMPP, CMPPI) can specify the automatic memory update option by including the letters UM in the command line.

IFO	*Result*
UM	Cause automatic memory update
blank	No memory update

For complete details on the operation of the automatic memory update feature, see "Memory update protocol" on page 49 and "Linking (the LNK bit)" on page 299.

ALU branching

The ALU in the SWAN processor provides all ALU instructions with an integrated conditional branching capability. In one instruction executing in a single machine cycle, the program can modify a register, test the results of that operation, and use the results to affect program flow.

The target address of an ALU branch operation is fixed in hardware at four instructions past the ALU instruction. If the specified condition code evaluates as true, the SWAN executes the instruction immediately following the ALU instruction (the

"committed slot") and then branches to the target address. If the specified condition code evaluates as false, the SWAN continues with sequential program flow.

| | | Instruction Sequence | |
| | | Branch Condition True | Branch Condition False |
Address	Contents		
N	ALU Instruction with branch	Executed	Executed
N+1	Committed slot, always executed	Executed	Executed
N+2	Instruction executed if branch condition not met	Skipped	Executed (Note 1)
N+3	Instruction executed if branch condition not met	Skipped	Executed (Note 1)
N+4	Branch target instruction	Executed	Executed (Note 1)
N+5	Sequential flow continues	Executed	Executed (Note 1)

Note 1: See "Example" on page 231 and "The Always Execute field (AE)" on page 231 for further details on program flow when the branch condition evaluates as false.

The SWAN core is optimized to take the ALU branch. Where the results of an ALU branch operation can be predicted, the programmer should write the code such that branches are taken more often than not.

The ALU Branch Condition field (abc)

The abc instruction field option (IFO) specifies the ALU branch condition to be tested during an ALU instruction. The absence of an abc IFO results in normal sequential program flow.

TABLE 52. ALU Branch Conditions for all instructions except Compare and Min/Max instructions

IFO	*Condition (branch if…)*	*IFO*	*Condition (branch if…)*
Blank	No branch	BLZ	Less-than zero
BGEZ	Greater-than or equal zero	BNZ	Not equal zero
BZ	Equal zero	BNO	No overflow flag set
BLEZ	Less-than or equal zero		

TABLE 53. ALU Branch Conditions for Compare and Min/Max instructions

IFO	*Condition (branch if…)*	*IFO*	*Condition (branch if…)*
Blank	No branch	BALEB	rsa < or = rsb
BAGB	rsa > rsb	BALB	rsa < rsb
BAGEB	rsa > or = rsb	BANEB	rsa not equal to rsb
BAEB	rsa = rsb		

Example	Consider the following program:

Address	Instruction
N-2	LIMD R2, 0
N-1	LIMD R3, 0
N	ADD r0, r1 BNZ
N+1	(Note 1)
N+2	BIL $SERVICE_CMD1
N+3	(Note 1)
N+4	LIMD R4, 0
N+5	BIL $SERVICE_CMD2

Notes: 1. Locations N+1 and N+3 are the committed slots for the ADD and BIL instructions respectively. Any instruction can be placed here except a Branch instruction or any instruction that could generate a branch (for example, an instruction containing a non-blank abc field).

2. The LIMD instructions are shown only for convenience in discussing possible rearrangement of the program. See "The Always Execute field (AE)" on page 231.

In this program, the ADD instruction result is predicted to evaluate to non-zero more often than zero. Thus, a majority of times through the ADD instruction, the SWAN executes the instruction at the committed slot (N+1), skips over the instructions at locations N+2 and N+3, and goes directly to the LIMD R4, 0 instruction at location N+4. If the ADD instruction result is zero, the SWAN executes the instruction at the committed slot (N+1) and then executes the BIL instruction at location N+2 and the instruction in its committed slot (N+3).

The Always Execute field (AE)

In the previous example, the ADD instruction result was predicted to evaluate to non-zero more often than zero. The resulting code sequence (N, N+1, N+4, N+5) executes with maximum efficiency. If the result of the ADD instruction evaluates to zero however, the SWAN processor instruction pipeline still fetches the instructions at N+4 and N+5, but discards the instructions before they are executed. Discarding these instructions (and

fetching the instructions at N+2 and N+3 instead) causes the equivalent of a two-cycle pipeline stall for the SWAN. The resulting code sequence is N, N+1, stall, stall, N+2, N+3.

To improve throughput, the MXT3010 provides an "Always Execute" option. When this option is enabled (by setting the AE bit in the branch instruction to 1), the SWAN processor always executes the instructions gathered by the instruction pipeline, rather than discarding them when the branch is not taken. If the AE option is specified, the instructions at N+4 and N+5 are always executed following the instruction in the committed slot (N+1) of the ALU branch. Thus, when using the AE option, the instructions placed at N+4 and N+5 must be instructions which the programmer wants executed regardless of whether the branch is taken.

To demonstrate use of the AE option, the previous example is presented again, but with these two minor changes:

- The ADD instruction now has its Always Execute (AE) bit set.

- The LIMD instructions for R2 and R3 have been moved down into positions N+4 and N+5 respectively.

The changes are shown in italics for emphasis.

Address	Instruction
N	ADD r0, r1, BNZ, *AE*
N+1	(See Note 1 in "Example" on page 231)
N+2	BIL $SERVICE_CMD1
N+3	(See Note 1 in "Example" on page 231)
N+4	*LIMD R2, 0*
N+5	*LIMD R3, 0*
N+6	BIL $SERVICE_CMD2
N+7	LIMD R4, 0

With the Always Execute option enabled, and the ALU branch condition code evaluated as true, the branch is taken normally. The sequence is: N, N+1, N+4, N+5, which is the same as it was without the Always Execute option enabled.

With the Always Execute option enabled, and the ALU branch condition code evaluated as false, the branch is not taken, but the instructions fetched by the pipeline process are executed rather than being discarded, and no stalls occur. The sequence is: N, N+1, N+4, N+5, N+2, N+3.

In summary, by using the Always Execute option and using the fourth and fifth locations beyond the branch instruction for instructions that are needed regardless of the branch results, the programmer can enjoy the performance advantages of the SWAN instruction pipeline without paying a performance penalty when the branch is not taken.

Committed slot restrictions

The ALU branch committed slot (N+1 in the example above) should not contain another ALU branch instruction nor a conditional branch instruction. ALU instructions without branch options and unconditional branch instructions can be placed in the committed slot.

If the Always Execute instruction field option is specified, the instruction at the branch target address (N+4 in the example above) and its following instruction should not contain another ALU branch instruction nor a conditional Branch instruction. ALU instructions without branch options and unconditional Branch instructions can be placed in these slots.

ADD   *Add Registers*

31 30 29 28 27 26	25 24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9 8 7 6	5	4 3 2 1 0
1 0 0 0 0 0	rd	U M	abc	rsa	MODx	A E	rsb

Format	ADD (rsa, rsb) rd [MODx] [abc] [AE] [UM]
Purpose	• To add two registers together using modulo arithmetic.
	• To alter program flow based on the result of the ALU operation (abc field) and to update a linked location in Fast Memory with the operation result (UM field).
Description	The ADD instruction adds the contents of register rsa to the contents of register rsb, placing the result in register rd.
Flags	If the source operands have the same sign, and the sign of the result is different, the Overflow flag is set. Operations specifying the modulo arithmetic option do not affect this flag.
Fields	A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229
AE	"The Always Execute field (AE)" on page 231
UM	"The Update Memory field (UM)" on page 228

ADDI   *Add Register and Immediate*

31 30 29 28 27 26	25 24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9 8 7 6	5 4 3 2 1 0
1 0 0 1 0 0	rd	U M	abc	rsa	MODx	usi

Format   ADDI (rsa, usi) rd   [MODx] [abc] [UM]

Purpose   • To add a register and a zero extended 6-bit immediate
                  together using modulo arithmetic.

                • To alter program flow based on the result of the ALU oper-
                  ation (abc field) and to update a linked location in Fast
                  Memory with the operation result (UM field).

Description   The 6-bit unsigned immediate (usi) is zero extended and added
                to the contents of register rsa using modulo arithmetic. The
                result is placed in register rd.

Flags   If the source operands have the same sign, and the sign of the
                result is different, the Overflow flag is set. Operations specify-
                ing the modulo arithmetic option do not affect this flag.

Fields   A summary of all fields for ALU instructions appears on
                page 413. Detailed descriptions for each field appear in the sec-
                tions cited in the following table.

Field	*For Further Information, See*
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229
UM	"The Update Memory field (UM)" on page 228

AND *And Registers*

31 30 29 28 27 26	25 24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9 8 7 6	5	4 3 2 1 0
1 1 0 0 0 0	rd	U M	abc	rsa	MODx	A E	rsb

Format   AND (rsa, rsb) rd [MODx] [abc] [AE] [UM]

Purpose   • To perform a Boolean AND function on two registers using modulo arithmetic.

                   • To alter program flow based on the result of the ALU operation (abc field) and to update a linked location in Fast Memory with the operation result (UM field).

Description   The contents of rsa are AND'ed together with contents of rsb using modulo arithmetic. The result is placed in register rd.

Flags   The Overflow flag is not affected by this operation.

Fields   A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229
AE	"The Always Execute field (AE)" on page 231
UM	"The Update Memory field (UM)" on page 228

ANDI   *And Register and Immediate*

31 30 29 28 27 26	25 24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0
1 1 0 1 0 0	rd	U M	abc	rsa	si
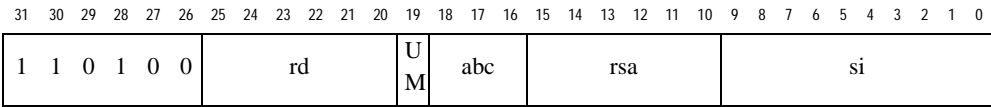
Format   ANDI (rsa, si) rd [abc] [UM]

Purpose   • To perform a Boolean AND function on a register and an immediate.

                  • To alter program flow based on the result of the ALU operation (abc field) and to update a linked location in Fast Memory with the operation result (UM field).

Description   The 10-bit immediate operand (si) is sign extended and AND'ed with the contents of register rsa, bit for bit. The result is placed in register rd.

Flags   The Overflow flag is not affected by this operation.

Fields   A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
abc	"The ALU Branch Condition field (abc)" on page 229
UM	"The Update Memory field (UM)" on page 228

ALU
Instructions

CMP *Compare Two Registers*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	0	0	0	0	0	0	0		abc				rsa				1	1	1	1	A E		rsb			

Format CMP (rsa, rsb) [abc] [AE]

Purpose
- To compare the contents of two registers.
- To alter program flow based on the result of the ALU operation (abc field).

Description The contents of register rsa are compared to the contents of register rsb. Both registers are treated as unsigned integers. The result can be used to alter program flow.

The results of the CMP instruction are produced without regard for previous compare results.

Flags The Overflow flag is not affected by this operation.

Fields A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
abc	"The ALU Branch Condition field (abc)" on page 229 Note: This instruction uses Table 53 rather than Table 52
AE	"The Always Execute field (AE)" on page 231

CMPI   *Compare Register and Immediate*

31	30	29	28	27	26	25	24	23	22	21	20	19	18 17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0
1	1	1	1	1	0	0	0	0	0	0	0	0	abc	rsa	usi

ALU Instructions

Format   CMPI (rsa, usi) [abc]

Purpose
- To compare the contents of a register and a 10-bit sign extended immediate.

- To alter program flow based on the result of the ALU operation (abc field).

Description   The 10-bit unsigned immediate (usi) has the value of bit 9 extended through bits [31:10] and is compared to the contents of the rsa register. Both operands are treated as unsigned integers.

Flags   The Overflow flag is not affected by this operation.

Fields   A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
abc	"The ALU Branch Condition field (abc)" on page 229 Note: This instruction uses Table 53 rather than Table 52

CMPP *Compare Two Registers with Previous*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	1	0	0	0	0	0	0		abc				rsa				1	1	1	1	A E		rsb			

Format   CMPP (rsa, rsb) [abc] [AE]

Purpose   • To compare the contents of two registers.

                       • To alter program flow based on the result of the ALU operation (abc field).

Description   The contents of register rsa are compared to the contents of register rsb. Both registers are treated as unsigned integers. The result can be used to alter program flow.

The Compare with Previous instruction can be used to accomplish the compare operation on integers that are larger than 16 bits. With this instruction, the compare should begin with the most significant halfwords and the simple CMP instruction. Only use the abc IFO with the last CMPP. For example, to determine whether a 32-bit number stored in [R16, R17] is equal to a 32-bit number stored in [R18,R19]:

```
CMP r16, r18
CMPP r17, r19, BAEB
```

Flags   The Overflow flag is not affected by this operation.

Fields   A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
abc	"The ALU Branch Condition field (abc)" on page 229 Note: This instruction uses Table 53 rather than Table 52
AE	"The Always Execute field (AE)" on page 231

CMPPI *Compare Register and Immediate with Previous*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	1	0	0	0	0	0	0		abc				rsa								usi					

ALU Instructions

Format	CMPPI (rsa, usi) [abc]
Purpose	• To compare the contents of a register and a 10-bit sign extended immediate
	• To alter program flow based on the result of the ALU operation (abc field).
Description	The 10-bit unsigned immediate (usi) has the value of bit 9 extended through bits [31:10] and is compared to the contents of the rsa register. Both operands are treated as unsigned integers.

This instruction can be used to accomplish the compare operation on integers that are larger than 16 bits. With this instruction, the compare should begin with the most significant halfwords and the simple CMPI instruction. Only use the abc IFO with the CMPPI. For example, to determine whether a 32-bit number stored in (R16, R17) is equal to 0x012301FF:

```
CMPI r16, 0x0123
CMPPI r17, 0x01FF BAEB
```

Since usi is a 10-bit field, 1FF is the largest number that can be used there, unless extension of bit 9 through bits [15:10] is desired.

Flags	The Overflow flag is not affected by this operation.
Fields	A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
abc	"The ALU Branch Condition field (abc)" on page 229
	Note: This instruction uses Table 53 rather than Table 52

FLS *Find Last Set*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

| 1 | 1 | 1 | 0 | 1 | 1 | | rd | | | | U M | | abc | | | | rsa | | | | | 1 | 1 | 1 | 1 | A E | 0 | 0 | 0 | 0 | 0 |

Format FLS rsa, rd [abc] [AE] [UM]

Purpose
- To determine the bit position of the MSB bit of a 16-bit halfword that is set to one.

- To alter program flow based on the result of the ALU operation (abc field).

Description The contents of register rsa are examined. The bit position of the most significant bit that is set is placed into rd. If bit 0 is the last bit set, the value 0x0000 is placed into rd. If bit position 15 is the last bit set, the value 0x000F is written into rd. If no bit is set, the value 0x8000 is placed into rd.

Flags The Overflow flag is not affected by this operation.

Fields A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
abc	"The ALU Branch Condition field (abc)" on page 229
AE	"The Always Execute field (AE)" on page 231
UM	"The Update Memory field (UM)" on page 228

Note A "Find First Set" instruction can be implemented by loading the halfword to be examined into "R56 Fast Memory Data register" on page 212 and performing an FLS instruction specifying "R43-read Fast Memory Bit Swap register (R42w[8]=0)" on page 203 as rsa.

LIMD   *Load Immediate*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1			rd				U M	0	0	0								li								

Format   LIMD rd, li [UM]

Purpose   • To initialize a 16-bit register in a single instruction.

              • To update a linked location in Fast Memory with the opera-
                tion result (UM field).

Description   The contents of register rd are loaded with the 16-bit long imme-
              diate (li) value.

Flags   The Overflow flag is not affected by this operation.

Fields   A summary of all fields for ALU instructions appears on
              page 413. Detailed descriptions for each field appear in the sec-
              tions cited in the following table.

Field	*For Further Information, See*
UM	"The Update Memory field (UM)" on page 228

ALU
Instructions

MAX *Maximum of Two Registers*

31 30 29 28 27 26	25 24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9 8 7 6	5	4 3 2 1 0
1 1 1 0 0 0	rd	U M	abc	rsa	MODx	A E	rsb

Format MAX (rsa, rsb) rd [MODx] [abc] [AE] [UM]

Purpose
- To choose the maximum of two registers.

- To alter program flow based on the result of the ALU oper-
  ation (abc field) and to update a linked location in Fast
  Memory with the operation result (UM field).

Description The contents of register rsa are compared to the contents of reg-
ister rsb. Both registers are treated as unsigned integers. The
maximum of the two registers is placed into rd.

Flags The Overflow flag is not affected by this operation.

Fields A summary of all fields for ALU instructions appears on
page 413. Detailed descriptions for each field appear in the sec-
tions cited in the following table.

Field	*For Further Information, See*
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229 Note: This instruction uses Table 53 rather than Table 52
AE	"The Always Execute field (AE)" on page 231
UM	"The Update Memory field (UM)" on page 228

MAXI *Maximum of Register and Immediate*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0			rd				U M		abc			rsa								usi						

Format   MAXI (rsa, usi) rd [abc] [UM]

Purpose   • To choose the maximum of a register and an immediate
                 value.

               • To alter program flow based on the result of the ALU oper-
                 ation (abc field) and to update a linked location in Fast
                 Memory with the operation result (UM field).

Description   The contents of register rsa are compared to the 10-bit unsigned
               immediate (usi) after bit 9 of usi has been extended through bits
               [31:10]. Both values are treated as unsigned integers. The max-
               imum of the two registers is placed into rd.

Flags   The Overflow flag is not affected by this operation.

Fields   A summary of all fields for ALU instructions appears on
               page 413. Detailed descriptions for each field appear in the sec-
               tions cited in the following table.

Field	*For Further Information, See*
abc	"The ALU Branch Condition field (abc)" on page 229
	Note: This instruction uses Table 53 rather than Table 52
UM	"The Update Memory field (UM)" on page 228

ALU
Instructions

MIN   *Minimum of Two Registers*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1			rd				U M		abc				rsa						MODx		A E			rsb		

Format

MIN (rsa, rsb) rd   [MODx] [abc] [AE] [UM]

Purpose

- To choose the minimum of two registers.

- To alter program flow based on the result of the ALU operation (abc field) and to update a linked location in Fast Memory with the operation result (UM field).

Description

The contents of register rsa are compared to the contents of register rsb. Both registers are treated as unsigned integers. The minimum of the two registers is placed into rd.

Flags

The Overflow flag is not affected by this operation.

Fields

A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229 Note: This instruction uses Table 53 rather than Table 52
AE	"The Always Execute field (AE)" on page 231
UM	"The Update Memory field (UM)" on page 228

MINI   *Minimum of Register and Immediate*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1			rd				U M		abc			rsa							usi							

ALU Instructions

Format   MINI (rsa, usi) rd [abc] [UM]

Purpose

- To choose the minimum of a register and an immediate value.

- To alter program flow based on the result of the ALU operation (abc field) and to update a linked location in Fast Memory with the operation result (UM field).

Description   The contents of register rsa are compared to the 10-bit unsigned immediate (usi) after bit 9 of usi has been extended through bits [31:10]. Both values are treated as unsigned integers. The minimum of the two registers is placed into rd.

Flags   The Overflow flag is not affected by this operation.
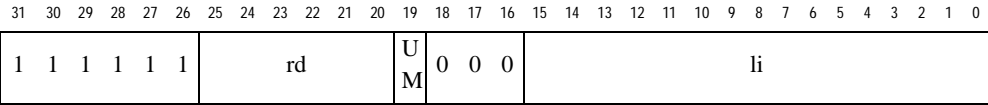
Fields   A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
abc	"The ALU Branch Condition field (abc)" on page 229 Note: This instruction uses Table 53 rather than Table 52
UM	"The Update Memory field (UM)" on page 228

OR   *Or Registers*

31 30 29 28 27 26	25 24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9 8 7 6	5	4 3 2 1 0
1 1 0 0 0 1	rd	U M	abc	rsa	MODx	A E	rsb

Format   OR (rsa, rsb) rd [MODx] [abc] [AE] [UM]

Purpose
- To perform a Boolean OR function on two registers using modulo arithmetic.

- To alter program flow based on the result of the ALU operation (abc field) and to update a linked location in Fast Memory with the operation result (UM field).

Description   The contents of rsa and rsb are OR'ed together using modulo arithmetic. The result is placed in register rd.

Flags   The Overflow flag is not affected by this operation.

Fields   A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229
AE	"The Always Execute field (AE)" on page 231
UM	"The Update Memory field (UM)" on page 228

ORI   *Or Register and Immediate*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

1	1	0	1	0	1	rd	U M	abc	rsa	si

ALU Instructions

Format   ORI (rsa, si) rd [abc] [UM]

Purpose
- To perform a Boolean OR function on a register and an immediate.

- To alter program flow based on the result of the ALU operation (abc field) and to update a linked location in Fast Memory with the operation result (UM field).

Description   The 10-bit immediate operand is sign extended and OR'ed with the contents of register rsa, bit for bit.

Flags   The Overflow flag is not affected by this operation.

Fields   A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
abc	"The ALU Branch Condition field (abc)" on page 229
UM	"The Update Memory field (UM)" on page 228

SFT   *Shift Signed Amount*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1				rd			U M		abc				rsa					MODx			0		rsb			

Format	SFT (rsa, rsb) rd [MODx] [abc] [UM]

Purpose

- To shift a register to the right or left based on the sign and magnitude of the shift amount contained in a second register.

- To alter program flow based on the result of the ALU operation (abc field) and to update a linked location in Fast Memory with the operation result (UM field).

Description

The contents of register rsa shift to the right or left based on the contents of register bits rsb(4:0) using modulo arithmetic. The result is placed in register rd. On a right shift, high-order bits are zero-filled. On a left shift, low-order bits are zero-filled.

The value of rsb(4:0) is interpreted as a signed shift amount. A negative number (bit 4=1) causes a shift to the right. A positive number (bit 4=0) causes a shift to the left. If the number is negative, the shift amount to the right is represented in two's complement form. See "Shift amount chart for SFT, SFTLI, and SFTRI" on page 414.

Flags

The Overflow flag is not affected by this operation.

Fields

A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	For Further Information, See
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229
AE	"The Always Execute field (AE)" on page 231
UM	"The Update Memory field (UM)" on page 228

SFTA *Shift Right Arithmetic*

31	30	29	28	27	26	25	24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9 8 7 6	5	4 3 2 1 0
1	0	1	0	1	1		rd	U M	abc	rsa	MODx	0	rsb

Format   SFTA (rsa, rsb) rd [MODx] [abc] [UM]

Purpose   • To shift a register to the right in an arithmetic fashion based on the shift amount contained in a second register.

                • To alter program flow based on the result of the ALU operation (abc field) and to update a linked location in Fast Memory with the operation result (UM field).
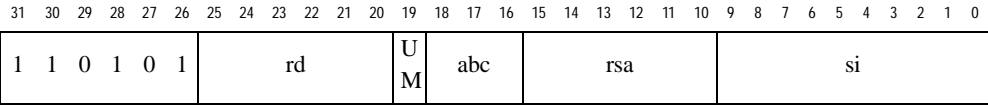
Description   The contents of register rsa shift to the right by the number of bit positions specified in bits rsb(3:0). See "Shift amount chart for SFTA" on page 415. The original value of bit rsa(15) is copied into all MSBs made vacant by the shift operation, thus accomplishing a sign extension/arithmetic shift. The result is placed in register rd.

Flags   The Overflow flag is not affected by this operation.

Fields   A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229
UM	"The Update Memory field (UM)" on page 228

ALU
Instructions

SFTAI   *Shift Right Arithmetic Immediate*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1			rd				U M		abc				rsa					MODx			0	1		usa		

Format

SFTAI (rsa, usa) rd [MODx] [abc] [UM]

Purpose

- To shift a register to the right in an arithmetic fashion based on the shift amount in an immediate value.

- To alter program flow based on the result of the ALU operation (abc field) and to update a linked location in Fast Memory with the operation result (UM field).

Description

The contents of register rsa shift to the right by the number of bit positions as specified in the usa field. See "Shift amount chart for SFTAI" on page 415. The original value of rsa(15) is copied into all MSBs made vacant by the shift operation, thus accomplishing a sign extension/arithmetic shift. The result is placed in register rd.

Flags

The Overflow flag is not affected by this operation.

Fields

A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	For Further Information, See
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229
UM	"The Update Memory field (UM)" on page 228

SFTC *Shift Left Circular*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0			rd				U M		abc			rsa					MODx				A E		rsb			

ALU Instructions

Format   SFTC (rsa, rsb) rd [MODx] [abc] [UM]

Purpose   • To shift a register to the left in a circular fashion based on
                      the shift amount contained in a second register.

                    • To alter program flow based on the result of the ALU oper-
                      ation (abc field) and to update a linked location in Fast
                      Memory with the operation result (UM field).

Description   The contents of register rsa shift to the left in a circular fashion
                    based on the value in register rsb(3:0). See "Shift amount chart
                    for SFTC and SFTCI" on page 414. Bits shifted out of bit posi-
                    tion 15 are shifted into bit position 0.

Flags   The Overflow flag is not affected by this operation.

Fields   A summary of all fields for ALU instructions appears on
                    page 413. Detailed descriptions for each field appear in the sec-
                    tions cited in the following table.

Field	*For Further Information, See*
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229
AE	"The Always Execute field (AE)" on page 231
UM	"The Update Memory field (UM)" on page 228

SFTCI   *Shift Circular Immediate*

31 30 29 28 27 26	25 24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9 8 7 6	5	4	3 2 1 0
1 0 1 1 1 0	rd	U M	abc	rsa	MODx	A E	1	usa

Format   SFTCI (rsa, usa) rd [MODx] [abc] [UM]

Purpose
- To shift a register to the left in a circular fashion based on an immediate shift amount.
- To alter program flow based on the result of the ALU operation (abc field) and to update a linked location in Fast Memory with the operation result (UM field).

Description   The contents of register rsa shift to the left in a circular fashion based on the value in the usa field, using modulo arithmetic. See "Shift amount chart for SFTC and SFTCI" on page 414. For example, if MOD16 is specified, bits rd(15:4) are taken from bits rsa(15:4) while bit rd(3:0), the modulo field, is taken from the Arithmetic Logic Unit result. Bits shifted out of bit position 15 are shifted into bit position 0.

Flags   The Overflow flag is not affected by this operation.

Fields   A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229
AE	"The Always Execute field (AE)" on page 231
UM	"The Update Memory field (UM)" on page 228

SFTRI/SFTLI   Shift Right or Left Immediate

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

| 1 | 0 | 1 | 1 | 0 | 1 | rd | U M | abc | rsa | MODx | A E | tcsa |

<div style="float:right">ALU Instructions</div>

Format   SFTRI (rsa, usa) rd [MODx] [abc] [UM]

                  SFTLI (rsa, usa) rd [MODx] [abc] [UM]

Purpose   • To shift the contents of register rsa to the right (SFTRI) or
                    to the left (SFTLI) by the amount specified in the unsigned
                    shift amount (usa). The assembler converts the unsigned
                    shift amount provided in the command line into a two's
                    complement shift amount (tcsa) for compatibility with the
                    hardware.

                  • To alter program flow based on the result of the ALU oper-
                    ation (abc field) and to update a linked location in Fast
                    Memory with the operation result (UM field).

Description   The contents of register rsa shift to the right or left based on the
                  contents of the usa field. See "Shift amount chart for SFT,
                  SFTLI, and SFTRI" on page 414. Bits made vacant by the shift
                  operation are filled with 0's.

Flags   The Overflow flag is not affected by this operation.

Fields   A summary of all fields for ALU instructions appears on
                  page 413. Detailed descriptions for each field appear in the sec-
                  tions cited in the following table.

Field	*For Further Information, See*
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229
AE	"The Always Execute field (AE)" on page 231
UM	"The Update Memory field (UM)" on page 228

SUB   *Subtract Registers*

31	30	29	28	27	26	25	24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9 8 7 6	5	4 3 2 1 0
1	0	0	0	1	0		rd	U M	abc	rsa	MODx	A E	rsb

Format

SUB (rsa, rsb) rd [MODx] [abc] [AE] [UM]

Purpose

- To subtract one register from another using modulo arithmetic.

- To alter program flow based on the result of the ALU operation (abc field) and to update a linked location in Fast Memory with the operation result (UM field).

Description

The contents of register rsb are subtracted from the contents of register rsa. The result is placed in register rd.

Flags

The Overflow flag is set when the signs of the source operands differ, and the sign of the result matches the sign of the second operand. Operations specifying the modulo arithmetic option do not affect this flag.

Fields

A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229
AE	"The Always Execute field (AE)" on page 231
UM	"The Update Memory field (UM)" on page 228

SUBI *Subtract Register and Immediate*

31 30 29 28 27 26	25 24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9 8 7 6	5 4 3 2 1 0
1 0 0 1 1 0	rd	U M	abc	rsa	MODx	usi

Format   SUBI (rsa, usi) rd [MODx] [abc] [UM]

Purpose
- To subtract an immediate value from a register using modulo arithmetic.

- To alter program flow based on the result of the ALU operation (abc field) and to update a linked location in Fast Memory with the operation result (UM field).

Description   The 6-bit unsigned immediate (usi) operand is zero extended and subtracted from the contents of register rsa, using modulo arithmetic. The result is placed in register rd.

Flags   The Overflow flag is set when the signs of the source operands differ, and the sign of the result matches the sign of the second operand. Operations specifying the modulo arithmetic option do not affect this flag.

Fields   A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229
UM	"The Update Memory field (UM)" on page 228

XOR *XOR Registers*

31	30	29	28	27	26	25	24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9 8 7 6	5	4 3 2 1 0
1	1	0	0	1	0		rd	U M	abc	rsa	MODx	A E	rsb

Format   XOR (rsa, rsb) rd [MODx] [abc] [AE] [UM]

Purpose   • To perform a Boolean exclusive-OR function on two regis-
                ters using modulo arithmetic.

                • To alter program flow based on the result of the ALU oper-
                ation (abc field) and to update a linked location in Fast
                Memory with the operation result (UM field).

Description   The contents of register rsa are logically XOR'ed with the con-
                tents of register rsb, bit for bit using modulo arithmetic. The
                result is placed in register rd. For example, if MOD16 is speci-
                fied, bits rd(15:4) are taken from bits rsa(15:4) while bit rd(3:0),
                the modulo field, is taken from the Arithmetic Logic Unit result.

Flags   The Overflow flag is not affected by this operation.

Fields   A summary of all fields for ALU instructions appears on
                page 413. Detailed descriptions for each field appear in the sec-
                tions cited in the following table.

Field	*For Further Information, See*
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229
AE	"The Always Execute field (AE)" on page 231
UM	"The Update Memory field (UM)" on page 228

XORI   *XOR Register and Immediate*

31 30 29 28 27 26	25 24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0
1 1 0 1 1 0	rd	U M	abc	rsa	si

Format XORI (rsa, si) rd [abc] [UM]

Purpose
- To perform a Boolean XOR function on a register and an immediate operand.

- To alter program flow based on the result of the ALU operation (abc field) and to update a linked location in Fast Memory with the operation result (UM field).

Description The 10-bit immediate operand (si) is sign extended and XOR'ed with the contents of register rsa, bit for bit. The results are placed in register rd.

Flags The Overflow flag is not affected by this operation.

Fields A summary of all fields for ALU instructions appears on page 413. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
MODx	"Modulo arithmetic" on page 226
abc	"The ALU Branch Condition field (abc)" on page 229
UM	"The Update Memory field (UM)" on page 228

CHAPTER 11 *Branch Instructions*

This chapter describes the suite of Branch instructions provided in the MXT3010. Branch instructions are one of two methods provided in the MXT3010 for altering the sequential execution stream of the SWAN processor. The other method uses branch condition fields in the ALU instructions to modify instruction execution flow. For details on ALU branching, see "Arithmetic Logic Unit Instructions" on page 223.

The first part of this chapter presents information common to all branch instructions. This information includes target address, condition codes, committed slot execution, subroutine linking, and counter system operations. Following the general branch information is a list of specific branch instructions, organized by name. For each branch instruction, there is a description, its mnemonic, purpose, and any information specific to that instruction.

General Branch instruction information

Introduction

A simplified version of the basic MXT3010 Branch instruction format is shown below:

FIGURE 82.Branch instruction format (simplified)

31 30 29 28 27 26	25 24 23 22	21	20	19 18 17 16 15 14 13 12	11 10 9 8 7 6 5 4 3 2 1 0
Op Code	ESS#	s	C	cso	-

Basic Branch instructions allow the programmer to specify conditional branching decisions which will alter the instruction execution sequence based on the state of the MXT3010's internal subsystems and certain external subsystems. The point to be tested is specified by the ESS field, and the state (1 or 0) which will cause a branch is specified by the s-bit. Branch instructions can also be used to manipulate the UTOPIA port's control counters via the counter system operation (cso) field.

Target address

The branch *target address* is the address at which execution continues if the specified branch condition is satisfied. The full branch target address within Fast Memory is formed from the Segment ID in the Instruction Base Address register (R53) and the branch target field.

FIGURE 83.Target address format in Fast Memory

18 17 16 15 14	13 12 11 10 9 8 7 6 5 4 3 2	1	0
Segment ID	Branch Target Field	0	0

Segment ID   The SWAN processor supports an instruction space of 128K
                    instructions. This 128K instruction space consists of 32 seg-
                    ments of 4K instructions each. A typical user code set fits within
                    one segment. The current segment ID (5 bits to represent one of
                    32 segments) is changed by writing a new segment number to
                    the Instruction Base Address register (R53).

Target Field   The branch *target field* is a 12-bit field that specifies the absolute
                    word address within the current code segment (4096 words) at
                    which execution is to continue. The branch target field may be
                    specified in one of three manners, as indicated in the following
                    table:

TABLE 54. Methods of specifying the Branch target field

Method	*Instructions Using This Method*
As bits [11:0] of the instruction	"BI Branch Immediate" on page 272 and "BIL Branch Immediate and Link" on page 273
As bits [11:0] of the Fast Memory Shadow register (R58). (Note 1)	"BF Branch Fast Memory Shadow Register" on page 270 and "BFL Branch Fast Memory Shadow Register and Link" on page 271
As bits [11:0] of the Branch register (R59)	"BR Branch Register" on page 274 and "BRL Branch Register and Link" on page 275

Note 1: The Fast Memory shadow register is loaded with the first halfword
        returned from memory during a Fast Memory read operation that
        specifies the LNK Instruction Field Option. The target field in R58
        represents an absolute address to branch to within the active segment.

Condition code (ESS Field)

When using a conditional branch, the programmer can specify
which bit in the External State Signals register (R42) is tested to
determine the outcome of the branch instruction. If no condition
code is specified, the assembler codes the ESS field as 1111, the
unconditional branch.

TABLE 55.   External State Signals register (R42) bits

ESS	Condition	ESS	Condition
ESS0	ICSI_A	ESS8	Sparse event register, bit OR
ESS1	ICSI_B	ESS9	RXBUSY counter > 0
ESS2	TXFULL counter ≤ 2	ESS10	TXFULL counter = full
ESS3	RXBUSY counter ≥ 4	ESS11	DMA1 Output or Queue stage busy
ESS4	Assigned Cell Flag	ESS12	DMA2 Output or Queue stage busy
ESS5	CSS operation in progress	ESS13	DMA1 Queue stage busy
ESS6	CIN_BSY	ESS14	DMA2 Queue stage busy
ESS7	COUT_RDY	blank	Unconditional Branch

The logical state identifier (S-Bit)

In addition to specifying the condition code tested via the ESS field, the programmer uses the logical state identifier, S, to indicate which state of the specified condition code results in the branch being taken.

TABLE 56.   Use of the S-bit

S	Branch Result
0	Branch is taken if condition = 0
1	Branch is taken if condition = 1

If the programmer knows that a bit will usually be asserted or will usually be de-asserted, he or she can optimize software for the expected branch condition by carefully selecting the logical state that will cause the branch to be taken.

Committed slot instructions

Execution

The SWAN processor implements a delayed branching technique in order to prevent pipeline delays during branch operations. When the SWAN processor encounters a branch

instruction, the instruction immediately following the branch, referred to as the *committed slot instruction*, is always fetched and entered into the execution pipeline. The programmer can control whether the committed slot instruction is executed or not by specifying options in the branch instruction.

The Conditional operator (C-bit)

If a conditional branch operation is specified, and the tested condition code is satisfied (branch is taken), the committed slot instruction is executed, and the instruction at the branch target address follows the committed slot instruction.

If a conditional branch operation is specified, and the tested condition code is not satisfied (branch is not taken), the execution of the committed slot instruction is determined by the presence of the Conditional operator, C, in the branch instruction. If the Conditional operator is absent, the committed slot instruction is executed. If the Conditional operator is present, the committed slot instruction is not executed, that is, the committed slot becomes "conditional" as well. In essence, this operator makes the committed slot instruction part of the targeted branch code, rather than part of the locally sequenced code.

The Nullify operator

If no ESS condition code is specified, the assembler codes the ESS field as 1111 (the unconditional branch) and codes the S-bit as zero (0). In this case, the execution of the committed slot instruction is determined by the presence of the Nullify operator, N, in the branch instruction. The absence or presence of the Nullify operator codes the Conditional operator (C-bit) as absent or present respectively. If the Nullify/Conditional operator is absent, the committed slot instruction is executed. If the Nullify/Conditional operator is present, the committed slot instruction is not executed. Table 57 on page 266 summarizes the use of the conditional and nullify operators.

TABLE 57. Use of the Conditional and Nullify operators

Type of Branch	*Condition Code Satisfied?*	*Applicable Operator*	*Operator Existence*	*Committed Slot Instruction Executed?*
Conditional	Yes	None	N/A	Yes
Conditional	No	Conditional	Absent	Yes
Conditional	No	Conditional	Present	No
Unconditional	N/A	Nullify	Absent	Yes
Unconditional	N/A	Nullify	Present	No

Committed slot restrictions for Branch instructions

The committed slot instruction of a branch should not be another branch unless the Nullify operator is specified with the first branch. In addition, the committed slot instruction of a branch should not contain an ALU instruction with an abc field.

Examples

TABLE 58. Example - conditional branch, condition satisfied

Address	*Instruction*	*Flow*	*Description*
0010	ADD r0,r1,r2	0010	
0011	BI 0x045 ESS1/1	0011	Branch to 0x045 if condition ESS1 = 1, assume success
0012	ADD r3,r4,r5	0012	Committed slot instruction is executed
0013	ADD r6,r7,r8		
…	…		
0045	ADD r9,r10,r11	0045	Execution continues at branch target address

TABLE 59. Example - conditional branch, condition not met

Address	Instruction	Flow	Description
0010	ADD r0,r1,r2	0010	
0011	BI 0x045 ESS1/1	0011	Branch to 0x045 if condition ESS1 = 1, assume failure
0012	ADD r3,r4,r5	0012	Committed slot instruction is executed
0013	ADD r6,r7,r8	0013	No branch occurs, sequential execution continues
…	…	…	
0045	ADD r9,r10,r11		

TABLE 60. Example - unconditional branch

Address	Instruction	Flow	Description
0010	ADD r0,r1,r2	0010	
0011	BI 0x045	0011	Branch to 0x045, no condition code specified
0012	ADD r3,r4,r5	0012	Committed slot instruction is executed
0013	ADD r6,r7,r8		
…	…		
0045	ADD r9,r10,r11	0045	Execution continues at branch target address

TABLE 61. Example - conditional operator, conditional branch, condition satisfied

Address	Instruction	Flow	Description
0010	ADD r0,r1,r2	0010	
0011	BI 0x045 ESS1/1/C	0011	Branch to 0x045 if condition ESS1 = 1, assume success
0012	ADD r3,r4,r5	0012	Committed slot instruction is executed
0013	ADD r6,r7,r8		
…	…		
0045	ADD r9,r10,r11	0045	Sequential execution continues at branch target address

TABLE 62.   Example - conditional operator, conditional branch, condition not satisfied

Address	Instruction	Flow	Description
0010	ADD r0,r1,r2	0010	
0011	BI 0x045 ESS1/1/ C	0011	Branch to 0x045 if condition ESS1 = 1, assume failure
0012	ADD r3,r4,r5		Committed slot is nullified due to C operator
0013	ADD r6,r7,r8	0013	Branch not taken, sequential execution continues
…	…	…	
0045	ADD r9,r10,r11		

Subroutine linking

The Branch Fast Memory (BF), Branch Immediate (BI), and Branch Register (BR) instructions are each available with a return address linking option. If the linking form of the branch instruction is specified (BFL, BIL, and BRL instructions), the address of the instruction immediately following the branch's committed slot is saved in the Branch register (R59). To return from the subroutine at a later time, the SWAN processor can execute a Branch Register (BR) instruction that returns the flow of execution to continue from the point where the linked branch occurred.

The Branch register (R59) is only written if the branch is taken, and is always written with the branch address instruction plus two, even if the branch was an unconditional branch with the nullify operator.

Restrictions for BFL,BIL, BR, BRL, and the Branch register (R59)

Whenever the Branch register (R59) is modified, whether by a load instruction directed to that register or by a branch instruction with linking, the modified value is not immediately available for use. Thus, any instruction which follows the modification must have at least one intervening instruction (after the modifier) to avoid using a stale Branch register value.

TABLE 63. Example - Branch with link, and return

Address	Instruction	Flow	Description
0010	ADD r0,r1,r2	0010	
0011	BIL 0x045 ESS1/1	0011	Branch to 0x045 if condition ESS1 = 1, assume success
0012	ADD r3,r4,r5	0012	Committed slot is executed, 0x013=>R59
0013	ADD r6,r7,r8		
…	…		
0045	ADD r9,r10,r11	0045	R59=0x013
0046	BR N	0046	Branch register specified return to saved address
0047	FOO		Committed slot not executed due to N operator
		0013	Sequential execution returns to saved link address

Counter system operation

Branch instructions are used to implement all counter system operations (CSO). These operations are used to increment and decrement the UTOPIA port control counters TXBUSY, TXFULL, RXBUSY, and RXFULL. A CSO can be specified as an optional operator on any branch instruction.

If a conditional branch instruction is executed, any CSO specified is unconditional. That is, the counter manipulation is performed without regard to whether the condition code is satisfied and the branch is taken.

TABLE 64. The CSO field

CSO	Hex / Binary Value	Operation
DRXBUSY	E0 / 1110 0000	Decrement RXBUSY counter
DRXFULL	E1 / 1110 0001	Decrement RXFULL counter
ITXBUSY	C2 / 1100 0010	Increment TXBUSY counter
ITXFULL	C3 / 1100 0011	Increment TXFULL counter

BF   *Branch Fast Memory Shadow Register*

31 30 29 28 27 26	25 24 23 22 21	20	19 18 17 16 15 14 13 12	11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 1 0 0	ESS#	s	C	cso	0 0 0 0 0 0 0 0 0 0 0 0

Note: the bit layout row reads — `0 0 0 1 0 0` (bits 31–26), `ESS#` (bits 25–22), `s` (bit 21), `C` (bit 20), `cso` (bits 19–12), `0 0 0 0 0 0 0 0 0 0 0 0` (bits 11–0).

Format   BF [ESS#/(0|1)/[C]] [cso] [N]

Purpose

- To allow for changes in program flow using conditional branching that tests the MXT3010's external state signals, and to increment and decrement UTOPIA control counters

- To provide a service routine address as the first word in a channel descriptor, and then branch to this service address.

Description   Based on the result of a specified condition code, this instruction can modify the SWAN processor's sequential flow resulting in a branch to the target address in the Fast Memory Shadow register.

Subsequent BF instructions will stall the SWAN processor until the first word is read from Fast Memory and copied into the Fast Memory Shadow (FMSR) register. If an LMFM instruction is executed, the FMSR is loaded only if the LMFM specified the LNK IFO with a non-zero halfword field. After the FMSR has been loaded by the LMFM, software can read and write the FMSR and use it as a second Branch register.

Fields   A summary of all fields for Branch instructions appears on page 416. Detailed descriptions for each field appear in the sections cited in the following table

Field	*For Further Information, See*
ESS#	"External State Signals register (R42) bits" on page 264
s	"The logical state identifier (S-Bit)" on page 264
C	"The Conditional operator (C-bit)" on page 265
cso	"Counter system operation" on page 269

Restrictions   See "Committed slot restrictions for Branch instructions" on page 266.

BFL *Branch Fast Memory Shadow Register and Link*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1		ESS#			s	C					cso					0	0	0	0	0	0	0	0	0	0	0

Format   BFL [ESS#/(0|1)/[C]] [cso] [N]

Purpose
- To allow for changes in program flow using conditional branching that tests the MXT3010's external state signals, and to increment and decrement UTOPIA control counters.

- To provide a service routine address as the first word in a channel descriptor, and then branch to this service address.

- To provide subroutine linking capability. (See "Subroutine linking" on page 268.)

Description   The BFL instruction is identical to the BF instruction, except that the address of the instruction immediately following the branch's committed slot is saved in the Branch register (R59). To return from the subroutine at a later time, the SWAN processor can execute a Branch Register (BR) instruction that returns the flow of execution to continue from the point where the linked branch occurred.

Fields   A summary of all fields for Branch instructions appears on page 416. Detailed descriptions for each field appear in the sections cited in the following table

Field	*For Further Information, See*
ESS#	"External State Signals register (R42) bits" on page 264
s	"The logical state identifier (S-Bit)" on page 264
C	"The Conditional operator (C-bit)" on page 265
cso	"Counter system operation" on page 269

Restrictions   See "Committed slot restrictions for Branch instructions" on page 266 and "Restrictions for BFL,BIL, BR, BRL, and the Branch register (R59)" on page 268.

Branch Instructions

BI *Branch Immediate*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0		ESS#			s	C				cso							word address (wadr)									

Format   BI wadr   [ESS#/(0|1)/[C]] [cso] [N]

Purpose   • To allow for changes in program flow using conditional
                        branching that tests the MXT3010's external state signals,
                        and to increment and decrement UTOPIA control counters.

Description   Based on the results of the specified condition code, the BI
                      instruction can modify the SWAN processor's sequential flow
                      resulting in a branch to the target address in the wadr field [11:0]
                      of the BI instruction.

Fields   A summary of all fields for Branch instructions appears on
                      page 416. Detailed descriptions for each field appear in the sec-
                      tions cited in the following table

Field	*For Further Information, See*
ESS#	"External State Signals register (R42) bits" on page 264
s	"The logical state identifier (S-Bit)" on page 264
C	"The Conditional operator (C-bit)" on page 265
cso	"Counter system operation" on page 269
wadr	"Target address" on page 262

Restrictions   See "Committed slot restrictions for Branch instructions" on
                      page 266.

BIL   *Branch Immediate and Link*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1		ESS#			s	C					cso									word address (wadr)							

Format
     BIL   wadr   [ESS#/(0|1)/[C]] [cso] [N]

Purpose
- To allow for changes in program flow using conditional branching that tests the MXT3010's external state signals, and to increment and decrement UTOPIA control counters.

- To provide subroutine linking capability. (See "Subroutine linking" on page 268.)

Description
The BIL instruction is identical to the BI instruction, except that the address of the instruction immediately following the branch's committed slot is saved in the Branch register (R59). To return from the subroutine at a later time, the SWAN processor can execute a Branch Register (BR) instruction that returns the flow of execution to continue from the point where the linked branch occurred.

Fields
A summary of all fields for Branch instructions appears on page 416. Detailed descriptions for each field appear in the sections cited in the following table

Field	*For Further Information, See*
ESS#	"External State Signals register (R42) bits" on page 264
s	"The logical state identifier (S-Bit)" on page 264
C	"The Conditional operator (C-bit)" on page 265
cso	"Counter system operation" on page 269
wadr	"Target address" on page 262

Restrictions
See "Committed slot restrictions for Branch instructions" on page 266 and "Restrictions for BFL,BIL, BR, BRL, and the Branch register (R59)" on page 268.

Branch Instructions

BR *Branch Register*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0		ESS#			s	C					cso					0	0	0	0	0	0	0	0	0	0	0	0

Format   BR [ESS#/(0|1)/[C]] [cso] [N]

Purpose
- To allow for changes in program flow using conditional branching that tests the MXT3010's external state signals, and to increment and decrement UTOPIA control counters.

- To branch to and return from subroutine operations.

Description   Based on the results of the specified condition code, the BR instruction can modify the SWAN processor's sequential flow resulting in a branch to the target address in the Branch register (R59).

Fields   A summary of all fields for Branch instructions appears on page 416. Detailed descriptions for each field appear in the sections cited in the following table

Field	*For Further Information, See*
ESS#	"External State Signals register (R42) bits" on page 264
s	"The logical state identifier (S-Bit)" on page 264
C	"The Conditional operator (C-bit)" on page 265
cso	"Counter system operation" on page 269
wadr	"Target address" on page 262

Restrictions   See "Committed slot restrictions for Branch instructions" on page 266 and "Restrictions for BFL,BIL, BR, BRL, and the Branch register (R59)" on page 268.

BRL *Branch Register and Link*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1		ESS#			s	C					cso					0	0	0	0	0	0	0	0	0	0	0	0

Format   BRL   [ESS#/(0|1)/[C]] [cso] [N]

Purpose
- To allow for changes in program flow using conditional branching that tests the MXT3010's external state signals, and to increment and decrement UTOPIA control counters.

- To branch to and return from subroutine operations.

- To provide subroutine linking capability. (See "Subroutine linking" on page 268.)

Description   The BRL instruction is identical to the BR instruction, except that the address of the instruction immediately following the branch's committed slot is saved in the Branch register (R59). To return from the subroutine at a later time, the SWAN processor can execute a Branch Register (BR) instruction that returns the flow of execution to continue from the point where the linked branch occurred.

Fields   A summary of all fields for Branch instructions appears on page 416. Detailed descriptions for each field appear in the sections cited in the following table

Field	*For Further Information, See*
ESS#	"External State Signals register (R42) bits" on page 264
s	"The logical state identifier (S-Bit)" on page 264
C	"The Conditional operator (C-bit)" on page 265
cso	"Counter system operation" on page 269
wadr	"Target address" on page 262

Restrictions   See "Committed slot restrictions for Branch instructions" on page 266 and "Restrictions for BFL,BIL, BR, BRL, and the Branch register (R59)" on page 268.

Branch
Instructions

CHAPTER 12 *Cell Scheduling Instructions*

This chapter describes the Cell Scheduling instructions, POPC, POPF, PUSHC, and PUSHF. Each command reference page includes the instruction name, its mnemonic, format, purpose, descriptions, fields, and restrictions.

Cell Scheduling System target address

All Cell Scheduling instructions utilize the rsb field to specify a register that contains a 14-bit target address for the Cell Scheduling operation. The target address specifies a location within the Connection ID table, and via logic within the MXT3010, a corresponding bit position in the Scoreboard. The complete Fast Memory halfword address (FADRS [19:1]) used to access the Connection ID table is formed using FADRS [19] hardwired to zero (0), the base address information from bits [11:8] of the Cell Scheduling System Configuration register(R60) as FADRS [18:15] and the target address from rsb [13:0] as FADRS [14:1]. See Table 6 on page 44.

POPC   *Service Schedule*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1			rd				0		000				000000				0	0	0	0	0			rsb		

Format   POPC rd @rsb

Purpose   To identify the Connection ID associated with a specified loca-
                    tion in the Cell Scheduling System Scoreboard and to determine
                    whether a connection is scheduled for servicing at that location.

Description   The target address specified by register rsb is translated into a
                    Cell Scheduling System Scoreboard bit position. The state of
                    that bit is copied into the Assigned Cell flag (see below), and the
                    bit location is cleared. In addition, the Connection ID table is
                    accessed in Fast Memory, and the associated Connection ID is
                    read into the destination register, rd.

Assigned Cell flag The Assigned Cell flag output is connected to ESS4 (R42). The
                    SWAN processor can test to see if a connection was scheduled
                    to become active in the current cell slot time by testing ESS4. If
                    ESS4 is set to 1, then a connection was scheduled for the current
                    cell time and the processor uses the Connection ID returned
                    from the Connection ID table to access the Channel Descriptor
                    for the connection. If the ESS4 is set to 0, no cell is scheduled
                    for transmission at the cell current time, and the Connection ID
                    shown in rd is stale information and should be ignored. For more
                    information on the Cell Scheduling System, see CHAPTER 3
                    "The Cell Scheduling System" on page 27.

Restrictions   The instruction immediately following POPC must not access
                    the destination register, rd. If a subsequent instruction accesses
                    rd, the correct value is read, but a stall may occur. See "Register
                    access rules" on page 22.

                    The MXT3010 does not support hardware registers (R32-R63)
                    as the destination of a POPC instruction.

POPF   POP Fast

31	30	29	28	27	26	25 24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9	8	7	6	5	4 3 2 1 0
0	0	1	0	0	1	rd	0	001	000000	0	0	0	0	0	rsb

Format   POPF rd @rsb

Purpose   To manipulate the Cell Scheduling System. This instruction manipulates the internal Scoreboards without accessing the Connection ID Table in Fast Memory. POPF can improve the speed of scheduling algorithms that scan multiple Scoreboard entries before connecting. By eliminating unnecessary accesses to Fast Memory, memory read/write latencies are avoided.

Description   The target address specified by register rsb is translated into a Cell Scheduling System Scoreboard bit position. The state of that bit is copied into the Assigned Cell flag (see below), and the bit location is cleared. The Fast Memory is not accessed, and location rd is not modified.

Assigned Cell flag The Assigned Cell flag output is connected to ESS4 (R42). The SWAN processor can test to see if a connection was scheduled to become active in the current cell slot time by testing ESS4. If ESS4 is set to 1, then a connection was scheduled for the current cell time. If the ESS4 is set to 0, no cell is scheduled for transmission at the cell current time. For more information on the Cell Scheduling System, see CHAPTER 3 "The Cell Scheduling System" on page 27.

Restrictions   There must be at least three instructions between one POPF instruction and another POPF instruction.

CSS
Instructions

PUSHC   *Schedule*

31	30	29	28	27	26	25	24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9	8	7	6	5	4 3 2 1 0
0	0	1	0	0	0		000000	0	000	rsa	0	0	0	0	0	rsb

Format   PUSHC rsa @rsb

Purpose   To dispatch a scheduling request to the Cell Scheduling System.

Description   The target address specified by register rsb is translated into a
                  Cell Scheduling System Scoreboard bit position. The Cell
                  Scheduling System searches for the first available location in the
                  Scoreboard at or after that bit position and sets the bit for that
                  location to reserve it. It also writes the 16-bit user-defined Con-
                  nection ID from the rsa register into the Connection ID table
                  location corresponding to the reserved Scoreboard bit. The
                  address contained in rsb is the earliest that the connection can
                  become active.

                  During a PUSHC instruction, if the Scoreboard is full, the Cell
                  Scheduling System returns an error by setting bit 15 in the CSS
                  Configuration register (R60). For more information on the Cell
                  Scheduling System, see CHAPTER 3 "The Cell Scheduling Sys-
                  tem" on page 27.

Note   Execution of this instruction updates the Scheduled Address
                  register (R61).

PUSHF   *Push Fast*

31	30	29	28	27	26	25	24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9	8	7	6	5	4 3 2 1 0
0	0	1	0	0	0		000000	0	001	rsa	0	0	0	0	0	rsb

Format   PUSHF rsa @rsb

Purpose   To manipulate the Cell Scheduling System. This instruction
                manipulates the internal Scoreboards without accessing the
                Connection ID table in Fast Memory. PUSHF can improve the
                speed of scheduling algorithms that rely on reserved Scoreboard
                locations when fixing bandwidth connections. By eliminating
                unnecessary accesses to Fast Memory, memory read/write laten-
                cies are avoided.

Description   The target address specified by register rsb is translated into a
                Cell Scheduling System Scoreboard bit position. The Cell
                Scheduling System searches for the first available location in the
                Scoreboard at or after that bit position and sets the bit for that
                location to reserve it. No new Connection ID is written into the
                Connection ID table location corresponding to the reserved
                Scoreboard bit. Rather, the existing Connection ID at that loca-
                tion will be scheduled. The address contained in rsb is the earli-
                est that the connection can become active.

                During a PUSHF instruction, if the Scoreboard is full, the Cell
                Scheduling System returns an error by setting bit 15 in the CSS
                Configuration register (R60). For more information on the Cell
                Scheduling System, see CHAPTER 3 "The Cell Scheduling Sys-
                tem" on page 27.

Note   Execution of this instruction updates the Scheduled Address
                register (R61).

CHAPTER 13 *Direct Memory Access Instructions*

This chapter describes the Direct Memory Access (DMA) instructions, beginning with information common to all DMA instructions. This information includes op codes, byte counts, and control fields. Following the general information is a list of specific DMA instructions, organized by name. For each instruction, there is a description, its mnemonic, purpose, and any information specific to that instruction.

General DMA instruction information

Introduction

A simplified version of the basic MXT3010 DMA instruction format is shown below:

FIGURE 84.DMA instruction format (simplified)

31 30 29 28 27	26 25	24 23 22 21 20 19 18	17 16 15	14 13 12 11 10 9	8 7 6 5	4 3 2 1 0
Op Code	i	BC	rla	rsa	Control	rsb

Basic DMA instructions allow the programmer to add read or write DMA transfer requests to either the Port1 or Port2 command queue by selecting the appropriate Op Code. The length of the transfer and various control features are determined by the Byte Count (BC) Instruction Field Option and the Control field. In addition, a feature is provided which permits the Alternate Byte Count register (R52) to provide control of the transfer length and other features. The rla, rsa, and rsb fields identify the registers used in the transfer.

Op codes for DMA instructions

The following table applies:

TABLE 65.   Op codes for DMA instructions

Bits [31:29]	*Bits [28:27]*	*Description*
011	00	DMA Read, Port1 (DMA1R instruction)
011	01	DMA Write, Port1 (DMA1W instruction)
011	10	DMA Read, Port2 (DMA2R instruction)
011	11	DMA Write, Port2 (DMA2W instruction)

The RLA increment bit (i-bit)

The MXT3010 DMA instructions include an option that provides an automatic increment to the target rla register upon dispatch of the DMA instruction. The increment is 64 modulo 512 and saves the SWAN processor the code needed to advance the rla register to the next cell buffer in the Cell Buffer RAM following each DMA transfer. To make this option available, Target Bit 0101 ("DMA Plus Control") in the Mode Configuration Register (R42) must be set to 1. (See "R42-write Mode Configuration register" on page 201.)

Use of bit 26

The instruction used and the status of the DMA Plus Control affect how bit 26 is coded by the assembler. The possibilities are shown in the table.

TABLE 66. Use of Bit 26

Instruction Used	*DMA Plus Control*	*Bits [26]*	*Description*
DMA1R DMA1W DMA2R DMA2W	Disabled	x	This bit is available as the highest order bit of the byte count field
DMA1R DMA1W DMA2R DMA2W	Enabled	0	Do not increment the rla register
DMA1R+ DMA1W+ DMA2R+ DMA2W+	Enabled	1	Increment rla register upon completion of DMA operation

Timing considerations for accessing rla

If the instruction immediately following a DMA operation with rla increment accesses the rla register, it will see the non-incremented value. If the second following instruction accesses the rla register, it will see the incremented or the non-incremented

value depending on the correlation of pipeline stalls and the DMA commitment. After the second following instruction, all further instructions will see the incremented rla value.

TABLE 67. Timing chart for accessing rla after a DMA

Instruction	*rla value*
DMA instruction	non-incremented value
Instruction following the DMA	non-incremented value
Second instruction following the DMA	indeterminate
Third instruction following the DMA	incremented value
Subsequent instructions following the DMA	incremented value

Note: The information in this table differs from that in Table 3 on page 23 and "Avoiding stale rla values" on page 315 because those refer to simple read/write operation, whereas this table refers to DMA operation.

The Byte Count instruction field option (BC)

The Byte Count field indicates the length[1] of the DMA transfer in accordance with the following table:

TABLE 68. Use of the BC field

Without DMA Plus Enabled		*With DMA Plus Enabled*	
Bits [26:19]	*Description*	*Bits [25:19]*	*Description*
0	Transfer 0 bytes.	0	Transfer 0 bytes.
1	Transfer 1 byte	1	Transfer 1 byte
2	Transfer 2 bytes	2	Transfer 2 bytes
3	Transfer 3 bytes	3	Transfer 3 bytes
-	-	-	-
127	Transfer 127 bytes	127	Transfer 127 bytes
-	-	Transfers larger than 127 bytes are not available when the DMA Plus Control is enabled.	
255	Transfer 255 bytes		
Transfers larger than 255 bytes are not available.			

1. See "Use of odd BC values" on page 287.

The "Use Alternate Byte Count Register (R52)" Feature

If the programmer does not specify the BC/# Instruction Field Option, the length of the transfer and the CRC treatment will be controlled by the Alternate Byte Count/ID register (R52) rather than by the BC field, CRCX, and CRCY bits in the instruction.

Use of odd BC values

The following restrictions apply to DMA operations using odd BC values:

*   DMA1R using BC = odd# transfers BC bytes

*   DMA1W, DMA2R, DMA2W using BC = odd# transfers BC-1 bytes.

The Port1 bus supports byte operations only on read operations. The Port2 bus does not support byte operations at all, and will always round down the BC field.

The Control instruction field option

Bits [9:5] of each DMA instruction are the Control field, which has the following format:

FIGURE 85.Control field format)

9	8	7	6	5
IBI	CRCX	CRCY	POD	ST

Note:The CRCX, CRCY, and ST bits apply only to the DMA1R, DMA1R+, DMA1W, and DMA1W+ instructions.

The bit definitions for the Control byte are given in the following table:

TABLE 69. Use of the Control byte

Bit	Name	Function
9	IBI	The Instruction Byte count Indicator is an internal flag used by the MXT3010. If the programmer has specified a BC/# value, the MXT3010 sets IBI and uses the BC/# and CRCX/CRCY values to control the transfer. If the programmer has not specified a BC/# value, the MXT3010 clears IBI and uses the values in R52 to control the transfer.
8	CRCX	If clear, CRC32 Partial Result registers are not modified. If set, a CRC32 Partial Result is generated based on CRC32PRX register's value and the result is deposited into CRC32PRX (R44/R45).
7	CRCY	If clear, CRC32 Partial Result registers are not modified. If set, a CRC32 Partial Result is generated based on CRC32PRY register's value and the result is deposited into CRC32PRY (R46/R47)
6	POD	If clear, no UTOPIA Port Post Operative Directive (POD) is performed. If set, TXBUSY is incremented upon the completion of DMA reads, and RXFULL is decremented upon completion of DMA writes.
5	ST	If clear, the DMA is performed in normal fashion. If set, a "Silent Transfer" is performed. In a Silent Transfer, a DMA is performed which includes CRC calculation, but does not require data from the host or other host intervention.

DMA1R   Direct Memory Operation - Port1 Read
DMA1R+   Direct Memory+ Operation - Port1 Read

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0				BC						rla				rsa						Control					rsb		

Formats	DMA1R rsa/rsb, rla [BC/#][CRC {X,Y}][POD][ST]
	DMA1R+ rsa/rsb, rla [BC/#][CRC {X,Y}][POD][ST]
Purpose	To initiate direct memory read operations on Port1
Description	Execution of this instruction causes a DMA read operation to be written into the Port1 DMA command queue.
Fields	The register selected by the rla field contains the Cell Buffer RAM address. See "Register load address (rla field)" on page 314. The rsa and rsb fields determine the Port1 memory address as shown in Table 24 on page 111. A summary of all fields for DMA instructions appears on page 417. Detailed descriptions for each field appear in the sections cited in the following table.

Field	For Further Information, See
i	"The RLA increment bit (i-bit)" on page 285
BC	"The Byte Count instruction field option (BC)" on page 286
Control	"The Control instruction field option" on page 287

Notes:	For use of bit 26, see "Use of bit 26" on page 285.
	To make the DMA1R+ instruction available, Target Bit 5 ("DMA Plus Control") in the Mode Configuration Register (R42) must be set to 1.
	For timing considerations concerning accesses to the rla register, see "Timing considerations for accessing rla" on page 285.

DMA
Instructions

DMA1W   Direct Memory Operation - Port1 Write
DMA1W+ Direct Memory+ Operation - Port1 Write

31	30	29	28	27	26	25	24 23 22 21 20 19	18 17 16	15 14 13 12 11 10	9 8 7 6 5	4 3 2 1 0
0	1	1	0	1			BC	rla	rsa	Control	rsb

Format   DMA1W rsa/rsb, rla [BC/#][CRC {X,Y}][POD][ST]
                DMA1W rsa/rsb, rla [BC/#][CRC {X,Y}][POD][ST]

Purpose   To initiate direct memory write operations on Port1

Description   Execution of this instruction causes a DMA write operation to
                be written into the Port1 DMA command queue.

Fields   The register selected by the rla field contains the Cell Buffer
                RAM address. See "Register load address (rla field)" on
                page 314. The rsa and rsb fields determine the Port1 memory
                address as shown in Table 24 on page 111. A summary of all
                fields for DMA instructions appears on page 417. Detailed
                descriptions for each field appear in the sections cited in the fol-
                lowing table.

Field	*For Further Information, See*
i	"The RLA increment bit (i-bit)" on page 285
BC	"The Byte Count instruction field option (BC)" on page 286
Control	"The Control instruction field option" on page 287

Notes: For use of bit 26, see "Use of bit 26" on page 285.

                To make the DMA1W+ instruction available, Target Bit 5
                ("DMA Plus Control") in the Mode Configuration Register
                (R42) must be set to 1.

                For timing considerations concerning accesses to the rla register,
                see "Timing considerations for accessing rla" on page 285.

DMA2R   *Direct Memory Operation - Port2 Read*
DMA2R+   *Direct Memory+ Operation - Port2 Read*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0				BC						rla				rsa					Control				rsb			

Format	DMA2R rsa/rsb, rla [BC/#][POD]
	DMA2R+ rsa/rsb, rla [BC/#][POD]
Purpose	To initiate direct memory read operations on Port2
Description	Execution of this instruction causes a DMA read operation to be written into the Port2 DMA command queue.
Fields	The register selected by the rla field contains the Cell Buffer RAM address. See "Register load address (rla field)" on page 314. The rsa and rsb fields determine the Port2 memory address as shown in Table 29 on page 137 and Table 31 on page 139. A summary of all fields for DMA instructions appears on page 417. Detailed descriptions for each field appear in the sections cited in the following table.

Field	For Further Information, See
i	"The RLA increment bit (i-bit)" on page 285
BC	"The Byte Count instruction field option (BC)" on page 286
Control	"The Control instruction field option" on page 287

Notes:	For use of bit 26, see "Use of bit 26" on page 285.
	To make the DMA2R+ instruction available, Target Bit 5 ("DMA Plus Control") in the Mode Configuration Register (R42) must be set to 1.
	For timing considerations concerning accesses to the rla register, see "Timing considerations for accessing rla" on page 285.

DMA
Instructions

DMA2W   Direct Memory Operation - Port2 Write
DMA2W+   Direct Memory+ Operation - Port2 Write

31	30	29	28	27	26	25	24 23 22 21 20 19 18	17 16	15 14 13 12 11 10 9	8 7 6 5	4 3 2 1 0
0	1	1	1	1			BC	rla	rsa	Control	rsb

Format	DMA2W rsa/rsb, rla [BC/#][POD]
	DMA2W rsa/rsb, rla [BC/#][POD]
Purpose	To initiate direct memory write operations on Port2
Description	Execution of this instruction causes a DMA write operation to be written into the Port2 DMA command queue.
Fields	The register selected by the rla field contains the Cell Buffer RAM address. See "Register load address (rla field)" on page 314. The rsa and rsb fields determine the Port2 memory address as shown in Table 29 on page 137 and Table 31 on page 139. A summary of all fields for DMA instructions appears on page 417. Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
i	"The RLA increment bit (i-bit)" on page 285
BC	"The Byte Count instruction field option (BC)" on page 286
Control	"The Control instruction field option" on page 287

Notes:	For use of bit 26, see "Use of bit 26" on page 285.
	To make the DMA2W+ instruction available, Target Bit 5 ("DMA Plus Control") in the Mode Configuration Register (R42) must be set to 1.
	For timing considerations concerning accesses to the rla register, see "Timing considerations for accessing rla" on page 285.

CHAPTER 14 *Load and Store Fast Memory Instructions*

This chapter describes the Load and instructions for Fast Memory. Each command reference page includes the instruction name, its mnemonic, purpose, and any information specific to that instruction. The common information includes descriptions, fields, and notes.

General information for Load and Store Fast Memory instructions

Introduction

Simplified versions of the MXT3010 Load and Store instruction formats for Fast Memory operations are shown below.

TABLE 70. Load Fast Memory instruction format

31 30 29 28 27 26 25 24	23 22 21 20 19	18	17 16	15	14 13 12 11 10	9 8 7 6 5	4 3 2 1 0
Op Code	rd	LNK	00	Z	rsa	#HW	rsb

TABLE 71. Store Fast Memory instruction format

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15	14 13 12 11 10	9 8 7 6 5	4 3 2 1 0
Op Code	000000000	rsa	#HW	rsb

Loading

The software tables and data structures stored in Fast Memory are accessed by the SWAN processor through the LMFM (Load Multiple from Fast Memory) instruction. The SWAN processor uses the #HW field to specify the number of halfwords to be fetched and the rsa and rsb fields to specify the Fast Memory byte address at which the transfer will begin. In response to the LMFM instruction, the Fast Memory interface controller will write the halfwords returned from memory into the SWAN's register file beginning with register rd and continuing with rd+1, rd+2, etc. until the designated number of halfwords have been transferred. Thus, the LMFM instruction allows the SWAN processor to transfer up to 16 halfwords from the Fast Memory into the register file in a single instruction.

If the LNK instruction field option is specified, the fast memory interface controller links the loaded registers to the locations in Fast Memory from which their contents were read. ALU instructions which modify these registers can force the modifications to be written back to Fast Memory by specifying the UM (update memory) option. Thus, the UM function allows the SWAN pro-

cessor to update the data structure in Fast Memory without executing a dedicated Store instruction. In addition, use of the LNK option also causes the first halfword read from memory to be read into the Fast Memory Shadow Register (R58), where it can be used by BF/BFL instructions. (See "BF Branch Fast Memory Shadow Register" on page 270 and "BFL Branch Fast Memory Shadow Register and Link" on page 271.)

Storing
Fast Memory writes can be accomplished utilizing the memory update function described above or by utilizing the Store halfword to Fast Memory (SHFM) instruction. Execution of the SHFM instruction causes the Fast Memory interface controller to write the halfword contained in the Fast Memory Data register (R56) into the halfword addressed by the byte address contained in registers rsa and rsb. A more powerful store instruction, Store Register Halfword (SRH) is also available. Further details are provided in "SRH Store Register Halfword" on page 312.

Transfer size (the #HW field)

The #HW field specifies the number of 16-bit halfwords to load from, or store to, Fast Memory.

Restrictions
If the LNK instruction field option is enabled, there are some restrictions to the values that can be used in this field. See "Limitations on #HW when linking" on page 300. If the LNK instruction field option is not enabled, any 0-16 halfword transfer is permitted, but the programmer must ensure that a multiple halfword entity is aligned on a 4-byte boundary.

Fast Memory address (the rsa and rsb fields)

Bits [3:0] of the register specified in the rsa field contain bits [19:16] of the Fast Memory Byte Address at which transfers begin. Bits [15:0] of the register specified in the rsb field contain bits [15:0] of the Fast Memory Byte Address at which transfers begin. This information is summarized in the following table:

TABLE 72. Use of the rsa and rsb fields

Field	*Register Bits Used*	*Function*
rsa	[3:0]	Fast Memory Address (FADRS) [19:16]
rsb	[15:0]	Fast Memory Address (FADRS) [15:0]

Address masking (the Z-bit)

A masking option, the Z-bit, provides improved access for aligned data structures. When set, this bit causes the least significant bits of the indicated rsb register to be masked out during the Fast Memory accesses, effectively forcing the transfer to start on an aligned structure boundary. When the Z-bit is clear, no masking is done.

The number of bits masked to zero is determined by the choice of destination register rd, as shown in the following table.

TABLE 73. Use of the Z-bit

Z-bit	*Function*	
0	Data read from Fast Memory at rsa [3:0]	rsb [15:0]	
1	Data read from Fast Memory at rsa [3:0]	rsb [15:n+1]	0[n:0]	
	rd = 16	masks rsb [4:0]
	rd = 24	masks rsb [3:0]
	rd = 28	masks rsb [2:0]
	rd = 30	masks rsb [1:0]
	rd = 31	masks rsb [0]

The Z-bit option permits an optimization to SWAN code for accessing the Connection ID (CID)Table wherein CIDs may be stored in such a manner that the retrieved CID value can be used for both high and low Fast Memory address as rsa and rsb. This enhancement can save several instructions in the critical PUSH/ POP code segments. Two examples are given to clarify the concept.

Z-bit usage example 1

Assume it is desired to access Fast Memory location 0x50000.

18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
101	0000	0000	0000	0000
rsa	rsb			

Normally, this would require that rsa contain 0x0005 and rsb contain 0x0000. However, performing:

LMFM rd @ rsa/rsa 16HW Z

will produce the following address:

18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
101	0000	0000	0000	0101
rsa	rsa			

Use of the Z-bit option causes masking of [4:0], producing:

18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
101	0000	0000	0000	0000
rsa	rsa with [4:0] masked			

This is the desired address. Thus, this is a simplified example of "CIDs may be stored in such a manner that the retrieved CID value can be used for both high and low Fast Memory address as rsa and rsb."

Z-bit usage example 2

Channel descriptors are organized in Fast Memory in 32-byte aligned data structures starting at 0x20000 and ending at 0x7FFE0. Due to the 32-byte alignment, bits [4:0] of the first

entry in a channel descriptor are always zero. Thus, sixteen bits [20:5] uniquely define the first entry of each channel descriptor. The MXT3010 constructs a Connection ID (CID) for each descriptor by using bits [15:5] from the descriptor address as CID bits [15:5] and using bits [20:16] from the descriptor address as CID bits [4:0]. This creation of a unique 16-bit Connection ID is important for use with the POPC instruction.

When a POPC instruction is used to determine whether a connection has been scheduled at a particular scoreboard location, and a connection has been scheduled, the Connection ID will be returned in the rd register specified in the POPC instruction.

A typical channel descriptor address, the Connection ID created from it, and the results of a POPC instruction are shown in Figure 86.

FIGURE 86.Z-bit usage example

Channel Descriptor Address in Fast Memory
Example: 0x3CAE0

| 0000 | 0000 | 0000 | 0011 | 1100 | 1010 | 1110 | 0000 |

MXT3010 hardware creates the
Connection ID by concatenating
address [15:5]|[20:16]

| 1100 | 1010 | 1110 | 0011 |

POPC rd@rsb
This instruction loads rd with the
Connection ID from the table

| 1100 | 1010 | 111 0 | 0011 |

mask

0 0000

LMFM r16@rd/rd 16HW Z
This instruction uses rd for both rsa and rsb
but Z option zeroes the lower five bits

| xxxx | xxxx | xxx0 | 0011 | 1100 | 1010 | 1110 | 0000 |
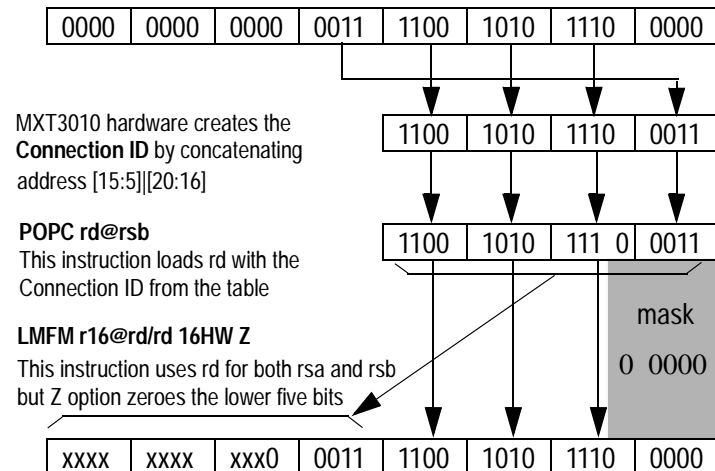
When an LMFM instruction is issued with the Z option enabled, the contents of rd can be used for both the rsa and rsb registers. As shown in the figure, specifying rd for both rsa and rsb, in conjunction with the masking action caused by the Z option, re-creates the original channel descriptor address.

Destination register (the rd field)

The rd field specifies the destination register for the initial half-word transfer. Subsequent halfwords will be transferred to rd+1, rd+2, etc. The register specified in the rd field can be any register, subject to the restrictions in "Choice of rd register" on page 300 and "Limitations on #HW when linking" on page 300

Linking (the LNK bit)

If this bit is set, the linking option is enabled. As indicated above, if the LNK option is enabled, the fast memory interface controller links the loaded registers to the locations in Fast Memory from which their contents were read. ALU instructions which modify these registers can force the modifications to be written back to Fast Memory by specifying the UM (update memory) option. In addition, use of the LNK option also causes the first halfword read from memory to be read into the Fast Memory Shadow Register (R58), where it can be used by BF/BFL instructions.

Example of LNK usage

Two very simplified channel descriptors are shown in Figure 87. In this example, P1_HI and P1_LO form a pointer to where the next received cell on that VC should go, and CRC_HI and CRC_LO are the accumulated CRC. Upon arrival of a cell for this particular VC (VC0 for example), the program performs a Load Multiple Fast Memory (LMFM) instruction, loading four halfwords into four locations starting at rd= R28. If the LNK

instruction field option is specified in the LMFM instruction, and the contents of R29 are subsequently changed using an instruction with an Update Memory (UM) option, the value of P1_LO will be changed. This provides a convenient way to update the pointer in response to arrival of a new cell.

FIGURE 87.Simplified Channel Descriptors

Address	*Contents*	*Loaded to*	*VC*
xxx00	P1_HI	R28	
xxx02	P1_LO	R29	0
xxx04	CRC_HI	R30	
xxx06	CRC_LO	R31	
xxx08	P1_HI		
xxx0A	P1_LO		1
xxx0C	CRC_HI		
xxx0E	CRC_LO		

Choice of rd register

While the linking option can be used with any register designated as the destination register rd, registers R16, R24, R28, R30, and R31 are most commonly used, as the MXT3010 logic is optimized for memory updates using these registers. When a hardware register (R32-R63) is used as the destination of an LMFM instruction, loading takes an additional cycle compared to loading a software register (R0-R31).

Limitations on #HW when linking

The choice of rd has an effect on the number of subsequent locations that can be linked, and hence places a limit on the size of the transfer (#HW). The following table applies:

TABLE 74. Limits on #HW when linking to rd

rd	*Permissible Values of #HW*
R16	16 or less
R24	8 or less
R28	4 or less
R30	2 or less
R31	1 or less (Note 1)

Note 1: 0 HW is a legal transfer size. An LMFM operation specifying 0 HW can establish a link without actually loading information from memory.

As an aid to understanding the Update Memory feature when used in conjunction with the LMFM instruction, and to understand the data structure alignments required to make best use of this feature, the following section provides a detailed explanation of the logic used by the MXT3010 to accomplish the memory update function.

Generation of UM addresses

In order to do a memory update to a linked location in the channel descriptor table[1] in Fast Memory, the MXT3010 logic needs to know the location of the desired channel descriptor within the table, and the offset of the location to be updated within the desired channel descriptor.

The least significant four bits of the destination register number (in binary) are inverted and saved in hardware as a mask, "lfm_adrs_mask [3:0]". In addition to the mask, the linked address is also saved in hardware as "lfm_linked_adrs [19:1]". Masking is performed on the linked address to ensure that it follows the memory alignment requirements shown in Table 75 on page 304. The masking equation is as follows:

lfm_linked_adrs [19:1] = {rsa [3:0], rsb [15:5], (rsb [4:1] & rd [3:0])}

When a subsequent instruction with the UM option enabled is performed, the MXT3010 computes the address linked to the register "exe_reg_dest[5:0]" as follows:

lfm_write_adrs[19:0] ={[lfm_linked_adrs [19:5], (lfm_linked_adrs[4:1]|(exe_reg_dest[3:0]&lfm_adrs_mask)),0 }

UM update example

Let us assume that a channel descriptor has been stored in Fast Memory beginning at location 0x08010, and let us further assume that an LMFM instruction has been issued to transfer

---

1. While this section specifically describes the channel descriptor table, the principles involved apply to a linked location in any table in Fast Memory.

four halfwords, with rd = R24. Example code to set up this situation is:

```
LIMD rsa, 0x000O
LIMD rsb, 0x8010
LMFM R24, rsa/rsb 6HW LNK
```

Presented as a figure, the result is:

FIGURE 88.Channel Descriptor for LMFM and UM example

Address	Contents	Loaded to
8010	STATUS_A	R24
8012	STATUS_B	R25
8014	P1_HI	R26
8016	P1_LO	R27
8018	CRC_HI	R28
801A	CRC_LO	R29

Bits [19:0] for 0x8010 are 0000 1000 0000 0001 0000, so "lfm_linked_adrs [19:1]" is 0000 1000 0000 0001 000-. The destination register (rd) is R24, which in binary is 11000. The least significant four bits (1000) invert to be 0111 or 0x0007, which is stored as "lfm_adrs_mask [3:0]". With these numbers saved in hardware, the MXT3010 is ready for subsequent instructions that manipulate any of these registers and specify the Update memory (UM) option. An example of such an instruction is the following:

```
ADDI R27, 1, R27 UM
```

In response to this instruction, the MXT3010 must not only increment the value in R27, it must also update the corresponding Fast Memory location with the new incremented value.

The register that was updated is referred to in the hardware as "exe_reg_dest[5:0]". For R27, the binary value [5:0] is: 011011. Bits [3:0] are 1011. The equation to be solved is:

lfm_write_adrs[19:0} = [lfm_linked_adrs [19:05], (lfm_linked_adrs[4:1]|(exe_reg_dest[3:0]&lfm_adrs_mask)),0 }

The information known is:

lfm_linked_address [19:0]	0000 1000 0000 0001 0000
lfm_linked_address [19:1]	0000 1000 0000 0001 000
lfm_linked_address [19:5]	0000 1000 0000 000
lfm_linked_address [4:1]	1000
exe_reg_dest[3:0]	1011
lfm_adrs_mask	0111

And-ing exe_reg_dest[3:0] with lfm_adrs_mask gives 0011; or-ing that with lfm_linked_address [4:1] gives 1011; concatenating that result with lfm_linked_address [19:5] and concatenating an LSB of 0 gives 0000 1000 0000 0001 0110 = 0x8016. Reference to Figure 88 on page 302 will confirm that the Fast Memory location to updated when R27 is updated is indeed at address 0x8016.

Memory alignment requirements

The various and-ing, or-ing, inverting, and masking functions performed by the MXT3010 hardware to correctly generate addresses for the Update Memory function place requirements on the alignment of data structures constructed in Fast Memory. Specifically, to use the LMFM instruction with the LNK option enabled, the following memory alignment requirements are recommended:

TABLE 75.   Memory alignment requirements

rd	FADRS [4:0]
R16	00000
R24	X0000
R28	XX000
R30	XXX00
R31	XXXX0

Memory
alignment
example

If the LMFM instruction is to be used with the LNK option
enabled, and the size of the transfer (#HW) is to be greater than
8 half-words, rd must be R16 (see "Limits on #HW when linking
to rd" on page 300). If R16 is used, and subsequent use of the
UM feature is desired, the data structure being copied from Fast
Memory should be aligned to a 32-byte boundary.

Instructions for accelerating CRC operations

The Store Register Halfword (SRH) instruction greatly accelerates the handling of partial CRC results during AAL5 packet segmentation or reassembly. Because DMA operations function independently of SWAN code execution once they have been started, firmware is able to start processing the next channel descriptor in parallel with the DMA transfer (and CRC accumulation) of the previous channel as soon as the DMA operation has been committed for that previous channel. This parallelism provides processing time to the SWAN that might otherwise be wasted waiting for the transfer to complete. However, it is still necessary to save the results of the partial CRC accumulation at the conclusion of a DMA transfer. These partial results must be saved in what is now the previously serviced channel descriptor.

If the SRH instruction is not used, the SWAN processor must save the address of the channel descriptor in which the partial results were to be stored, recover that address upon completion of the DMA operation, and finally store the partial results at the appropriate offset within the channel descriptor. This saving and recovering process requires seven instructions per cell time in each direction. Hence, use of the SRH instruction is highly recommended, as it eliminates the need for saving and recovering the partial CRC address information.

At the time that a DMA read or write operation with CRCX or CRCY indicated is initiated to Port 1, the MXT3010 automatically stores the address contained in the internal FAST Memory Link Address register into a temporary holding register. There are two holding registers – one for CRCX operations and one for CRCY operations. Typically the FAST Memory Link Address register (within the MXT3010 logic) will have the current Channel Descriptor address. Thus, as a DMA1R with CRC or a DMA1W with CRC is executed, the address of the current Channel Descriptor is automatically set aside.

Upon completion of the DMA transfer, the SRH instruction is used to write the contents of the partial CRC registers (R44/ R45 or R46/ R47) to FAST Memory using the address contained in either the CRCX holding register or the CRCY holding register as the base address for the transfer. An offset can be specified with the SRH instruction, allowing the partial results to be placed at the appropriate field within the Channel Descriptor.

The SRH instruction is based on the Store Halfword to Fast Memory (SHFM) instruction, and the SHFM instruction is now a valid subset of the more flexible SRH instruction. In addition to the rsa and rsb fields found in the SHFM instruction, the SRH instruction has three special fields:

Alternate address (the adr field)

The adr field (bits [20:19] of the SRH instruction) specifies the location from which the Fast Memory address is obtained. The following table applies:

TABLE 76. Use of the adr field

Bits [20:19]	*Function (Target Fast Memory Address is:)*
00	rsa/rsb (same as SHFM instruction)
01	rsa/rsb with lsbs field substituted for address bits [4:0]
10	CRCX holding register with lsbs field appended
11	CRCY holding register with lsbs field appended

The valid entries for this field are CRCX and CRCY. If neither is specified, the assembler codes bits [20:19] as 00 if no lsbs field is specified, or as 01 if an lsbs field is specified. The lsbs field is described on page 307.

Hardware register (reg field)

The reg field selects one of eight hardware registers that can be written to Fast Memory. This is in contrast to the more limited SHFM, where only the contents of the Fast Memory Data register (R56) can be written to Fast Memory. The following table applies:

TABLE 77. Use of the reg field

Bits [20:19]	Function (*Register to be Written to Fast Memory*)
000	Register R56 (same as SHFM instruction)
001	Register R37
010	Register R38
011	Register R39
100	Register R44
101	Register R45
110	Register R46
111	Register R47

Least significant bits (the lsbs field)

In adr mode 00 (SHFM compatibility mode), this field is unused. In the other adr modes (01,10,11), lsbs contains the five least significant bits of the target Fast Memory address. This is not an index field; rather, it is a bit substitution field.

LMFM   *Load Multiple from Fast Memory*

31	30	29	28	27	26	25 24 23 22 21 20	19	18 17 16		15 14 13 12 11 10	9 8 7 6 5	4	3 2 1 0
0	1	0	1	0	1	rd	LNK	00	Z	rsa	#HW	0	rsb

Format: LMFM rd @rsa/rsb #HW [LNK]

Purpose
- To initiate a burst transfer of data from Fast Memory directly into the SWAN processor's register file.

- To automatically link the data structure and the registers to reflect register modifications back to memory through the Update Memory options with ALU instructions that modify the loaded registers.

Description   With the LMFM instruction, the Fast Memory interface controller can initiate a block fetch operation to transfer #HW half-words from Fast Memory directly into the CPU's register file. The transfer begins at the address specified in registers rsa and rsb.

Restrictions   When an LMFM instruction is executed, a sequential update to registers rd, rd+1, and subsequent registers, takes place. As a result, instructions following an LMFM must not access registers that are in the process of being updated. Table 78 shows the register updating process for the following LMFM instruction:

LMFM r2 r15/r16 4HW

In the example, move instructions are shown as typical instructions that might be used to access registers R2, R3, R4, and R5.

TABLE 78.   Restrictions on access to rd registers after LMFM

| Address | Instruction | Status at time of instruction execution | | | |
		rd (R2)	rd+1 (R3)	rd+2 (R4)	rd+3 (R5)
0000	LMFM r2 r16/r17 4HW	Changing	Undefined	Undefined	Undefined
0004	MV r2 r17	New[a]	Changing	Undefined	Undefined
0008	MV r3 r18	New	New[b]	Changing	Undefined
000C	MV r4 r19	New	New[b]	New[b]	Changing
0010	MV r5 r20	New	New[b]	New[b]	New[b]

a. The MXT3010EP stalls for four internal clock cycles before executing the MV r2 r17 instruction to ensure that register rd has valid data. If desired, four instructions that do not access r2 through r5 can be inserted between the LMFM and the MV r2 r17 instruction.

b. Availability of new data at this time requires that access to rd has occurred since the LMFM.

For registers R0:R15, the programmer must follow the sequential order shown or undefined results will occur. For example, attempting to access register rd+1 immediately after the LMFM will produce erroneous results.

For registers R16:R31, a register control scoreboarding system, implemented in hardware, protects registers rd+1 and beyond. This system introduces stalls if the access restrictions are not followed. Since registers R16:R31 are intended for the manipulation of channel descriptors, the register control scoreboarding system simplifies the programming model.

Without LNK, any 0-16 halfword transfer is legal, but make sure the burst transfer does not cross a 32-byte boundary.

Stalls   Hardware interlocks stall the CPU if it tries to access a register the Fast Memory Interface Controller is changing. The CPU remains stalled until the Fast Memory Interface Controller writes a new value into the register.

Fast Memory
Instructions

Fields

Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
#HW	"Transfer size (the #HW field)" on page 295
rsa	"Fast Memory address (the rsa and rsb fields)" on page 296
rsb	"Fast Memory address (the rsa and rsb fields)" on page 296
Z-bit	"Address masking (the Z-bit)" on page 296
rd	"Destination register (the rd field)" on page 299
LNK-bit	"Linking (the LNK bit)" on page 299

SHFM   *Store Halfword to Fast Memory*

31	30	29	28	27	26	25	24 23 22 21 20 19 18 17 16	15 14 13 12 11 10	9 8 7 6 5	4 3 2 1 0
0	1	0	1	1	1		000000000	rsa	00000	rsb

Format   SHFM @rsa/rsb

Purpose   • To store a halfword to Fast Memory.

Description   SHFM causes the Fast Memory interface controller to write the halfword contained in the Fast Memory Data register(R56) into the halfword addressed by the byte address contained in registers rsa and rsb.

The Fast Memory interface controller writes the halfword into a write buffer first so the SWAN processor can continue executing.

Stalls   A write buffer full stall occurs if the four deep write buffer is full when the SHFM instruction is executed.

Fields   Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
#HW	"Transfer size (the #HW field)" on page 295
rsa	"Fast Memory address (the rsa and rsb fields)" on page 296
rsb	"Fast Memory address (the rsa and rsb fields)" on page 296

Fast Memory
Instructions

SRH   *Store Register Halfword*

31	30	29	28	27	26	25	24 23 22 21	20 19	18 17 16	15 14 13 12 11 10	9 8 7 6 5	4 3 2 1 0
0	1	0	1	1	1		00000	adr	reg	rsa	lsbs	rsb

Format: SRH @reg [CRCXADR, CRCYADR, rsa/rsb] [lsbs/#]

Purpose:
- To store a halfword to Fast Memory, using any of several registers as a direct source.

- To greatly accelerates the handling of partial CRC results during AAL5 packet segmentation or reassembly.

Description: In typical use, upon completion of the DMA transfer, the SRH instruction is used to write the contents of the partial CRC registers (R44/ R45 or R46/ R47) to FAST Memory using the address contained in either the CRCX holding register or the CRCY holding register as the base address for the transfer. An offset can be specified with the SRH instruction, allowing the partial results to be placed at the appropriate field within the Channel Descriptor.

Fields   Detailed descriptions for each field appear in the sections cited in the following table.

Field	For Further Information, See
adr	"Alternate address (the adr field)" on page 306
reg	"Hardware register (reg field)" on page 307
rsa	"Fast Memory address (the rsa and rsb fields)" on page 296
lsbs	"Least significant bits (the lsbs field)" on page 307
rsb	"Fast Memory address (the rsa and rsb fields)" on page 296

Note   By setting adr=00 and reg=000, the SRH instruction becomes the original SHFM instruction.

CHAPTER 15 *Load and Store Internal RAM Instructions*

This chapter describes the Load and Store instructions for internal RAM, beginning with information common to all Load and Store instructions. Following the general information is a list of specific Load and Store instructions, organized by name. For each instruction, there is a description, its mnemonic, purpose, and any information specific to that instruction.

General information for Load and Store internal RAM instructions

Introduction

Simplified versions of the MXT3010 Load and Store instruction formats for internal RAM operations are shown below.

TABLE 79. Load internal RAM instruction format

31 30 29 28 27 26 25	24 23 22 21 20	19	18 17 16	15 14 13 12	11 10 9 8	7 6 5	4 3 2 1 0
Op Code	rd	0	rla	0000	Swap	IDX	00000

TABLE 80. Store internal RAM instruction format

31 30 29 28 27 26 25	24 23 22 21 20	19	18 17 16	15 14 13 12 11 10	9 8 7 6 5	4 3 2 1 0
Op Code	0000	Swap 0	rla	rsa	IDX	00000

The Load and Store internal RAM instructions move data between the SWAN register set and memory internal to the MXT3010. The internal memories addressable by these instructions are the Cell Buffer RAM and the Cell Scheduling System Scoreboard. The load and store (LD, ST) instructions move one 16-bit halfword between a specified register and a target halfword address. The load and store double (LDD, STD) instructions move two 16-bit halfwords between two consecutive registers and two consecutive target addresses. Load and Store instructions that swap bytes and/or half-words are also available.

Register load address (rla field)

Choices for the rla register

Four hardware registers and four fixed value registers can be specified as the rla register. The hardware registers are R48, R49, R50, and R51. The fixed value registers are GA, GB, GC, and GD. The compiler codes the choice of rla into the 3-bit rla field. The G registers point to different 64-byte blocks in gather space (page 317). In many instances, this allows software to

access gather space without modifying one of the hardware registers.

TABLE 81.   Use of the rla field

rla value	Register selected	Register content
000	R48	Variable
001	R49	Variable
010	R50	Variable
011	R51	Variable
100	GA	0x0400
101	GB	0x0420
110	GC	0x0440
111	GD	0x0460

Avoiding stale rla values

To prevent a stale value of the rla register from being used to generate the internal RAM address, separate a load or store instruction that uses R48, R49, R50, or R51 from a preceding instruction that modifies the register by at least one instruction. This intervening instruction cannot be an LD or LDD instruction to a hardware register - see "Register access rules" on page 22.

The index field (IDX)

This field can be used to index into a table from a base address.

Using IDX to calculate the target address

The target address is formed from the content of the specified rla register and an immediate index value contained in the index field of the instruction. The index field, IDX, is exclusive-or-ed with the rla content; see Figure 89.

FIGURE 89.XOR operation between IDX and rla
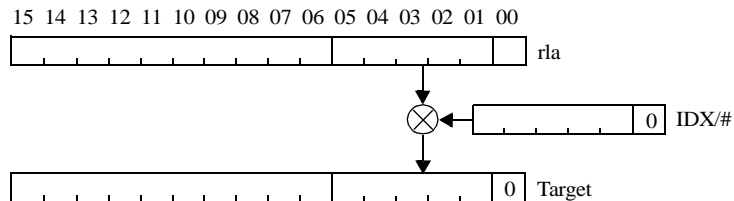
The upper bits [15:06] of the rla content are unchanged, and bit 0 is forced to zero. As a result of this XOR function, the load or store instruction can access any 16-bit halfword within the 64-byte block addressed by the rla register by changing the value in the IDX field and leaving the contents of the rla register unchanged.

Note that although the index field is treated as a five-bit half-word index, the value used in the SWAN assembler (IDX/#) is always specified as a byte index and is transformed into a word by the assembler. The IDX/# field can take even values from IDX/0 to IDX/62. The assembler inserts the appropriate five-bit value into the instruction field.

Selecting the Cell Buffer RAM or the Scoreboard

The target address selects the 16-bit halfword for a load or store instruction, or the first of two consecutive 16-bit halfwords for a load or store double instruction. The two internal RAMs that can be accessed with these instructions are the Cell Buffer RAM and the Cell Scheduling System Scoreboard. Bit 11 of the rla register selects the RAM to be accessed.

Bit 11	Internal RAM selected
0	Cell Buffer RAM
1	Cell Scheduling System Scoreboard

Cell Buffer RAM accesses

Selecting an
access method

The Cell Buffer RAM can be accessed with linear or gather access methods. Bit 10 of the rla register selects the access method of the Cell Buffer RAM; it does not affect the access method for the Cell Scheduling System Scoreboard.

Bit 10	*Cell Buffer RAM method selected*
0	Linear
1	Gather

Linear method
accesses

In linear method accesses, the Cell Buffer RAM is treated as a simple contiguous memory 1024 bytes in length. Bits [9:1] of the target address select the 16-bit halfword within this space.

Gather method
accesses

Since the cells stored in the Cell Buffer RAM are 52 or 56 bytes in length, the last eight bytes of each 64-byte section of the Cell Buffer RAM are normally unused. In gather method accesses, the last eight bytes of each 64-byte section appear as a contiguous 128-byte block of memory. The first 16-bit halfword of this block is at address 0x0400 of the gather address method. The last 16-bit halfword is at address 0x047E. Figure 90 illustrates this addressing method. Thus, gather access recovers discontinuous regions of Cell Buffer RAM memory into one continuous address space. This is not additional space, but rather a method of making use of small pieces of existing space.

FIGURE 90.Gather method accesses
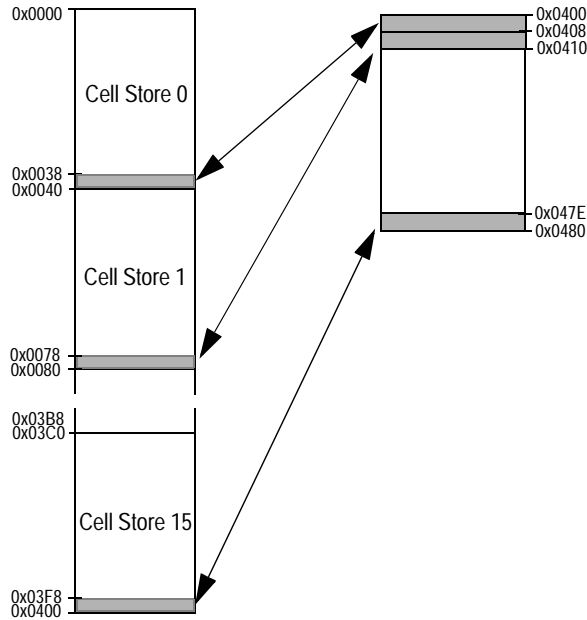
Cell Scheduling System Scoreboard accesses

The Cell Scheduling System Scoreboard is a 16Kbit memory accessed as 512, 32-bit words. With two exceptions, do not modify the Scoreboard content with these instructions. Rather, use the Cell Scheduling System instructions (PUSHC, POPC, PUSHF, POPF) to maintain coherency of the Scoreboard content with other Cell Scheduling System mechanisms. The two exceptions when load (LD, LDD) and store (STD[1]) instructions can be used are:

1. When initially clearing the scoreboard

---

1. There is no support for 16-bit writes to the Scoreboard. Use only STD instructions when writing to the Scoreboard. Do not use ST.

2. When using portions the scoreboard space for applications other than call scheduling

Byte swap support

The load and store instructions provide a programmable function for swapping bytes in half-word and word data structures for systems with mixed big-endian and little-endian entities. The instructions perform byte swapping on either half-words (16-bit) or words (32-bit) as the data is read from Cell Buffer RAM memory (load) or written to Cell Buffer RAM memory (store).

The Swap field

Bits [11:10] of Load instructions and bits [21:20] of Store instructions provide a *Swap* field. Two swap byte and swap half-word functions can be asserted for any LD, LDD, ST, or STD instruction that accesses the Cell Buffer RAM. These functions are not defined for the internal Scoreboard memory. Although all combinations of swap byte and swap halfword functions are valid in the SWAN core, not all combinations are useful. The syntax and instruction fields are the same for the byte swapping instructions as for the ordinary load and store instructions.

The following tables list the most useful byte-swapping load and store instructions.

TABLE 82.   Byte-swapping Load instructions

Instruction	Bits [11:10]	Source addr	Dest addr	Source data	Dest data	Function
LD r0 GA	00	0x400	r0	aa55	aa55	Normal register load
LDSB r0 GA	01	0x400	r0	aa55	55aa	Byte swap 16-bit operand and load register
LDD r0 GA	00	0x400 0x402	r0 r1	aa55 bb66	aa55 bb66	Normal double register load
LDDSBH r0 GA	11	0x400 0x402	r1 r0	aa55 bb66	66bb 55aa	Byte swap 32-bit operand and load register
LDDSB r0 GA	01	0x400 0x402	r0 r1	aa55 bb66	55aa 66bb	Byte swap two 16-bit operands and load registers

TABLE 83.   Byte-swapping Store instructions

Instruction	Bits [21:20]	Source addr	Dest addr	Source data	Dest data	Function
ST r0 GA	00	r0	0x400	aa55	aa55	Normal register store
STSB r0 GA	01	r0	0x400	aa55	55aa	Byte swap 16-bit operand and register store
STD r0 GA	00	r0 r1	0x400 0x402	aa55 bb66	aa55 bb66	Normal double register store
STDSBH r0 GA	11	r1 r0	0x400 0x402	aa55 bb66	66bb 55aa	Byte swap 32-bit operand and store register
STDSB r0 GA	01	r0 r1	0x400 0x402	aa55 bb66	55aa 66bb	Byte swap two 16-bit operands and store registers

LD   *Load Register*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	1	0	0	0	0		rd		0	rla		0000		Swap	IDX		00000	

Format   LD rd   @rla [IDX/#]

Purpose   Use LD to read a 16-bit halfword from an internal memory.

Description   The content of register rla and the Index field are used to form a target source address in internal memory. The memory is read and register rd is loaded with the result.

Notes   IDX/# must be specified as a byte index value even though bit 0 is ignored.

                LD instructions which perform byte-swaps and/or half-word-swaps are also available. See "Byte swap support" on page 319.

                Restrictions apply to the use of LD instructions with hardware registers. See "Register access rules" on page 22.

Stalls   If the Cell Buffer RAM is unavailable due to concurrent Port1, UTOPIA Port, and Port2 Cell Buffer RAM accesses, the CPU stalls if it tries to access the destination register of the LD before its data is returned. The CPU can continue executing instructions as long as it does not try to access rd before rd is returned from the Cell Buffer RAM. To guarantee the best overall throughput, separate Cell Buffer RAM loads and instructions that access the loaded data by two or more instruction slots.

Fields   Detailed descriptions for each field appear in the sections cited in the following table.

Field	For Further Information, See
Swap	"The Swap field" on page 319
IDX	"The index field (IDX)" on page 315

Internal RAM
Instructions

LDD *Load Double Register*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1				rd			0		rla			0000			Swap			IDX					00000			

Format	LDD rd @rla [IDX/#]
Purpose	Use LDD to read two 16-bit halfwords from internal memory.
Description	The content of register rla and the Index field are used to form a target source address in internal memory. The memory is read and register rd is loaded with the result. Register rd+1 is also loaded with a 16-bit halfword read from internal memory. The internal memory address for this halfword is obtained by exclusive-or-ing 0x0002 with the calculated target address.
Notes	IDX/# must be specified as a byte index value even though bit 0 is ignored. Instructions that perform byte-swaps and/or half-word-swaps are also available. See "Byte swap support" on page 319. Restrictions apply to the use of LDD instructions with hardware registers. See "Register access rules" on page 22.
Stalls	If the Cell Buffer RAM is unavailable due to concurrent Port1, UTOPIA Port, and Port2 Cell Buffer RAM accesses, the CPU stalls if it tries to access the destination register of the LD before its data is returned. The CPU can continue executing instructions as long as it does not try to access rd before rd is returned from the Cell Buffer RAM. To guarantee the best overall throughput, separate Cell Buffer RAM loads and instructions that access the loaded data by two or more instruction slots.
Fields	Detailed descriptions for each field appear in the sections cited in the following table.

Field	For Further Information, See
Swap	"The Swap field" on page 319
IDX	"The index field (IDX)" on page 315

ST   *Store Register*

31	30	29	28	27	26	25 24 23 22	21 20	19	18 17 16	15 14 13 12 11 10	9 8 7 6 5	4 3 2 1 0
0	1	0	0	1	0	000000	Swap	0	rla	rsa	IDX	00000

Format	ST rsa @rla [IDX/#]
Purpose	Use ST to write a 16-bit halfword to internal memory.
Description	The content of register rla and the Index field are used to form a target address in internal memory. The content of register rsa is written to this memory location.
Notes	IDX/# must be specified as a byte index value even though bit 0 is ignored.
	ST instructions which perform byte-swaps and/or half-word-swaps are also available. See "Byte swap support" on page 319.
Stalls	All Cell Buffer RAM writes are written into a write buffer. If the Cell Buffer RAM is unavailable due to concurrent Port1, UTO-PIA port, and Port2 Cell Buffer RAM accesses, the CPU stalls if it tries to write to the Cell Buffer RAM while the write buffer is busy. The CPU can execute instructions as long as it does not try to write to the Cell Buffer RAM while the write buffer is busy. The write buffer can hold a single ST or STD instruction.
Fields	Detailed descriptions for each field appear in the sections cited in the following table.

Field	*For Further Information, See*
Swap	"The Swap field" on page 319
IDX	"The index field (IDX)" on page 315

Restrictions	Since this is a 16-bit instruction, it should not be used to access space in the Scoreboard RAM unless the program performs the save and restore operations necessary to access a 32-bit quantity. Use STD instead.

Internal RAM
Instructions

STD   *Store Double Register*

31	30	29	28	27	26	25	24 23 22	21 20	19	18 17 16	15 14 13 12 11 10	9 8 7 6 5	4 3 2 1 0
0	1	0	0	1	1		0000	Swap	0	rla	rsa	IDX	rsb

Format   STD rsa/rsb   @rla [IDX/#]

Purpose   Use STD to write two 16-bit halfwords into internal memory.

Description   The content of register rla and the Index field are used to form a
                target address in internal memory. The content of register rsa is
                written to this memory location. The content of register rsb is
                also written to internal memory. The memory address for this
                halfword is obtained by exclusive-or ring 0x0002 with the cal-
                culated target address.

Notes   IDX/# must be specified as a byte index value even though bit 0
                is ignored.

                Versions of this instruction which perform byte-swaps and/or
                half-word-swaps are also available. See "Byte swap support" on
                page 319.

Stalls   All Cell Buffer RAM writes are written into a write buffer. If the
                Cell Buffer RAM is unavailable due to concurrent Port1, UTO-
                PIA port, and Port2 Cell Buffer RAM accesses, the CPU stalls
                if it tries to write to the Cell Buffer RAM while the write buffer
                is busy. The CPU can execute instructions as long as it does not
                try to write to the Cell Buffer RAM while the write buffer is
                busy. The write buffer can hold a single ST or STD instruction.

Fields   Detailed descriptions for each field appear in the sections cited
                in the following table.

Field	*For Further Information, See*
Swap	"The Swap field" on page 319
IDX	"The index field (IDX)" on page 315

CHAPTER 16 *Swan Instruction Reference Examples*

This chapter provides examples for the following instructions:

- Add and Subtract

- Branch

- Logical

- Load and Store

- Shift

Also provided is a section of miscellaneous examples.

Add and Subtract examples

The Add and Subtract examples include:

- 16-bit arithmetic
- ALU branching
- Modulo arithmetic

Formats

The examples in this section use the ADD, ADDI, OR and SUB instructions, which have the following formats:

ADD   (rsa, rsb) rd   [MODx][abc][UM]

ADDI   (rsa, usi) rd   [MODx][abc][UM]

OR   (rsa, rsb) rd   [MODx][abc][UM]

SUB   (rsa, rsb) rd   [MODx][abc][UM]

16-bit arithmetic

```
ADDI (R12, 23) R4
```

The unsigned immediate, 23 decimal, is zero extended and added to the contents of R12. The result is placed into R4. The Overflow flag register is set if an overflow results from the operation.

Modulo arithmetic

```
ADD (R8, R9) R10 MOD64, UM
```

The contents of R8 are added to the contents of R9. The result is placed into R10. Since MOD64 arithmetic is specified, the results are a combination of the ALU result and R8 source bits. R10(15:6) are taken from R8(15:6) while R10(5:0) are taken from the ALU result bits(5:0). The final result of the operation updates Fast Memory because the update memory (UM) option is specified. The Overflow flag register remains unchanged when modulo arithmetic is used.

ALU branching

```
.loc 0x0000
ADDI  (R11, 0x001A) R18 MOD32, BZ
OR    (R0, R1) R2

      BI        $RECOVER
      NOP

OR    (R5, R6) R7
OR    (R5, R6) R7
```

The constant 0x001A is added to the contents of R11. Since MOD32 arithmetic is specified, the result is a combination of the ALU result and R11 source bits. R18(15:5) are taken from R11(15:5) while R18(4:0) are taken from the ALU result (4:0). If bits (4:0) of R18 are zero, the branch is taken, regardless of the state of R18(15:5). The branch results in program flow of 0x0000, 0x0001, 0x0004, 0x0005. If bits (4:0) of R18 are not zero, the branch is not taken and program flow proceeds as 0x0000, 0x0001, 0x0002, 0x0003. The instructions at 0x0004 and 0x0005 are fetched but not executed. The Overflow flag register remains unchanged when modulo arithmetic is used. If the always execute (AE) instruction field option (IFO) was included with the ADDI instruction, 0x0004 and 0x0005 are executed even if the branch condition is false.

Branch examples

The Branch examples include:

- Branching and the committed slot
- Branch with link
- Branch with counter control
- Branch with shadow address

Formats

The examples in this section use the ADD, BFL, BI, BIL, BRL, LIMD, and OR instructions, which have the following formats:

ADD   (rsa, rsb) rd   [MODx][abc][UM]

BFL [ESS#/(0|1)/[C]][(cso)][N]

BI wadr [ESS#/(0|1)/[C]][(cso)][N]

BIL wadr [ESS#/(0|1)/[C]][(cso)][N]

BRL [ESS#/(0|1)/[C]][(cso)][N]

LIMD rd, li [UM]

OR   (rsa, rsb) rd   [MODx][abc][UM]

Branching and the committed slot

```
BI    0x024A ESS10/1/C
ADD   R0, R1, R2
```

Branch to address 0x024A if External State Signal ESS10 is set to "1." Execute the committed slot instruction (ADD) only if the branch is taken (/C).

Branch with link

```
BIL   $SCHEDULE ESS10/1/C
ADD   R0, R1, R2
OR    R2, R3, R4
```

Branch to the subroutine $SCHEDULE if External State Signal
ESS10 is set to "1." Execute the committed slot instruction only
if the branch is taken. Save the return address (the address of the
OR instruction) in register R59, but only if the branch is taken.

Branch with counter control

```
BI    0x0333   ITXBUSY
ADD   R0, R1, R2
```

Branch to address 0x0333. Execute the committed slot instruc-
tion. The ITXBUSY operation increments the UTOPIA Port's
TXBUSY counter.

Branch with shadow address

```
BFL
ADD   R0, R1, R2
OR    R2, R3, R4
```

Branch to the address contained in the Fast Memory Shadow
Register, unconditionally following execution of the committed
slot instruction. Stall if a LMFM with the LNK IFO specified is
active or pending but has yet to return the first word to the First
Word Shadow Register. Save the return address (the address of
the OR instruction) in register R59.

```
BIL   $SERVICE  ESS1/0 DRXFULL
ADD   R0, R1, R2
OR    R2, R3, R4
```

Branch to the subroutine $SERVICE if External State Signal
ESS1 is set to "0." Execute the committed slot instruction
whether or not the branch is taken. Save the return address (the
address of the OR instruction) in register R59, but only if the
branch is taken. Decrement the UTOPIA Port's RXFULL
counter.

```
.loc 0x0000
LIMD  R59, 0x0010
LIMD  R59, 0x0020
BRL
NOP
```

```
            .loc 0x0004
            MV    R59, R1

            .loc 0x0010
            BRL
            MV    R59, R0
```

The first BRL branches to address 0x0010 because the second
LIMD has not updated R59 by the time the branch accesses R59.
Consecutive LIMD and BRL instructions must be separated by
one instruction for the modification to take effect in time. The
BRL at location 0x0010 causes R59 to be loaded with the return
address, which is location 0x0004. therefore, the BRL at loca-
tion 0x0010 branches to locations 0x0004. R0 contains 0x0004
because the second BRL has not updated R59 by the time the
MV is executed. R1 contains 0x0012 because R59 has been
updated by the BRL by the time the MV instruction at 0x0004 is
executed.

Load and Store Fast Memory examples

Formats

The examples in this section use the LMFM, SHFM, and SRH instructions, which have the following formats:

LMFM rd @ras/rsb #HW [LNK]

SHFM @ rsa/rsb

SRH @rsa/rsb [adr] [reg] [lsbs]

Loading from Fast Memory

```
LMFM R16 @R10/R11 16HW LNK
```

Sixteen 16-bit halfwords are copied from Fast Memory to registers R16 through R31. Bits [19:16] of the Fast Memory address come from R10 bits [3:0], and bits [15:0] come from R11 [15:0]. Links are created such that any change to one of these registers is subsequently replicated in the corresponding Fast Memory location if the change was made by an instruction utilizing the Update Memory (UM) option.

Storing into Fast Memory

```
SHFM @ R16/R17
```

The Fast Memory controller writes the halfword contained in the Fast Memory byte register (R56) into the halfword addressed by the byte address contained in registers R16 and R17.

```
SRH R44 CRCXADR LSBS/10
```

The Fast Memory controller writes the CRC partial result halfword contained in R44 into the halfword 10 bytes beyond the address contained in the CRCX address holding register.

Load and Store Internal RAM examples

Formats

The examples in this section use the LD, LDD, ST, and STD instructions, which have the following formats:

LD rd @rla [IDX/#]

LDD rd @rla [IDX/#]

ST rsa @rla [IDX/#]

STD rsa/rsb @rla [IDX/#]

Loading from
Internal RAM

```
LD R13 @R48 IDX/10
```

The contents of register R48 and the Index field (in this case 10 bytes) are used to form the source address in internal memory. R13 is loaded with the contents of the memory location pointed to by R48, offset by 10.

IDX/# must be specified as a byte index value even though bit 0 is ignored. Register rd must be a software register (R0-R31).

```
LDD R13   @R48 IDX/10
```

The contents of register R48 and the Index field (in this case 10 bytes) are used to form the source address in internal memory. The memory is read and register R13 is loaded with the result. Register R14 is also loaded with a 16-bit halfword read from internal memory because this is a Load *double* instruction. The internal memory address for this halfword is obtained by exclusive-or-ing 0x0002 with the calculated target address.

```
ST R10   @R49 IDX/20
```

The content of register R49 and the Index field (in this case 20 bytes) are used to form a target address in internal memory. The content of register R10 is written to this memory location.

```
STD R10/R11   @R49 IDX/20
```

Description   The content of register rla and the Index field are used to form a target address in internal memory. The content of register R10 is written to this memory location. The content of register R11 is also written to internal memory because this is a Store *Double* instruction. The memory address for this halfword is obtained by exclusive-or ring 0x0002 with the calculated target address.

Logical examples

Formats

The examples in this section use the AND, OR, and XORI instructions, which have the following formats:

AND (rsa, rsb) rd [MODx][abc][AE][UM]

OR (rsa, rsb) rd [MODx][abc][AE][UM]

XORI (rsa, si) rd [abc][UM]

Using the AND Instruction

```
AND   R8, R9, R16 UM
```

The contents if R8 are AND'ed with the contents of R9. The result is placed into R16. The result is also written back into the Fast Memory location linked to R16.

Using the OR instruction

```
OR   R12, R13, R4
```

The contents if R12 are OR'ed with the contents of R13. The result is placed into R4.

Using the XORI instruction

```
XORI R12, 0x007F, R4 BZ
```

The contents if R12 are XOR'ed with 0x007F. The result is placed into R4. If the result is zero, program control is passed to the instruction four instruction slots away. If the result is not zero, sequential program flow occurs although a stall penalty of two cycles is incurred due to the incorrect branch prediction.

Using the AND instruction with MOD and abc fields

```
AND   R8, R9, R16 MOD64 BZ
```

The contents if R8 are AND'ed with the contents of R9. The result, MOD64, is placed into R16. This implies that R8[15:5] is combined with the ALU result bits [5:0] and written into R16. More importantly, the conditional branch is then only based on bits[5:0] of the result rather than on the entire result.

Shift examples

The Shift examples include:

- Shift right
- Shift left

Formats

The examples in this section use the SFT, SFTA, SFTC, SFTCI, SFTLI, and SFTRI instructions, which have the following formats:

SFT (rsa, rsb) rd [MODx][abc][UM]

SFTA (rsa, rsb) rd [MODx][abc][UM]

SFTC (rsa, rsb) rd [MODx][abc][UM]

SFTCI (rsa, usa) rd [MODx][abc][UM]

SFTLI (rsa, usa) rd [MODx][abc][UM]

SFTRI (rsa, usa) rd [MODx][abc][UM]

General case

```
SFT   R8, R9, R10
```

The contents of R8 shifts to the either the right or the left, based on the value of R9[4:0]. The result is placed in R10. All vacated bit positions are filled with 0's. The Overflow flag registers is not modified.

Shift right

if R9[4:0] = **1**0011xb, R8 is shifted to the right by 13 positions.

Right shift amount calculation:

Absolute Value of 10011 is: 01100xb + 1 = 1101xb = 13

Shift left

if R9[4:0] = **0**0011xb, R8 is shifted to the left by three positions.

Circular shifts

```
SFTC R8, R9, R10
```

The contents of R8 shifts in a circular/rotational fashion to the left by the amount specified in R9[3:0]. The shift direction/sign bit R9[4] is ignored because the SFTC and SFTCI instructions shift only to the left. Bits shifted out of R8[15] are shifted into bit position 0, and so on. The result is placed into R10.

Arithmetic shifts   `SFTA R8, R9, R10`

The contents of R8 shifts to the right, based on the contents of bits [3:0] of R9. The shift direction/sign bit R9[4] is ignored because the SFTA and SFTAI instructions shift only to the right. The beginning value of R8[15], the sign bit, is copied into all vacated positions.

Immediate shifts   `SFTLI R16, 7, R17`

The contents of R16 shift to the left by seven positions, with all vacated bits are filled with 0's. To accomplish a left shift by seven positions, the assembler places 00111xb into the SSA field.

Shift Amount = 7 = 00111xb

The assembler places 00111xb into the SSA field

`SFTRI R16, 7, R17`

The contents of R16 shift to the right by seven positions, with all vacated bits are filled with 0's. However, to accomplish a right shift by seven positions, the assembler must place the two's complement representation into the SSA field. Therefore, the assembler converts seven into its two's complement value and places that value in the SSA field as follows:

Shift Amount = 7 = 00111xb

Two's complement representation is 11000xb + 1 = 11001xb.

11001xb is placed by the assembler into the SSA field

```
SFTCI R9, 7, R10
```

The contents of R8 shifts in a circular/rotational fashion to the left by the seven positions. Bits shifted out of bit position 15 are shifted into bit position 0, and so on. The result is placed into R10.

Miscellaneous examples

Formats

The examples in this section use the CMP, CMPP, FLS, LIMD, MAX, and MIN instructions, which have the following formats:

CMP (rsa, rsb) [abc][AE]

CMPP (rsa, rsb) [abc][AE]

FLS rd [abc][UM]

LIMD rd, li [UM]

MAX (rsa, rsb) rd [MODx][abc][AE][UM]

MIN (rsa, rsb) rd [MODx][abc][AE][UM]

Using CMP and CMPP

```
CMPP R6, R7, BAGB
```

The contents of R6 are compared to the contents of R7. The results from the previous compare (CMP<u>P</u>) are also factored into the A>B? decision. If the previous compare operation indicated A>B, the branch is taken regardless of the results of the present compare operation. If the previous compare operation indicated A < B, the branch is not taken regardless of the results of the present compare operation. If the previous compare operation indicated A=B, the decision to branch is made based on the results of the current compare operation.

```
CMP  R4, R5
```

The contents of R4 are compared to the contents of R5. Since no Branch Condition was specified, the results are logged for future use.

32-bit compare operation example

```
CMP   #RARRIVAL_TIME_HI, #REARLIEST_ALLOWD_HI
CMPP  #RARRIVAL_TIME_LO, #REARLIEST_ALLOWD_LO BAGEB
      BI        $DISCARD_CELL
      NOP
```

| 64-bit compare operation example | ```
CMP   #RARRIVAL_TIME_64, #REARLIEST_ALLOWD_64
CMPP  #RARRIVAL_TIME_48, #REARLIEST_ALLOWD_48
CMPP  #RARRIVAL_TIME_32, #REARLIEST_ALLOWD_32
CMPP  #RARRIVAL_TIME_16, #REARLIEST_ALLOWD_16 BAGEB
``` |
|---|---|

Using FLS

```
FLS R9, R10, BGEZ
```

The 2^e (exponential) position of the last bit set in R9 is written into R10. For example, if bit 15 of R9 is set (the MSB), FLS writes 0x000F into R10. In the example above, the state of bits 14:0 does not affect the result. If bit0 of R9 is the only bit set, 0x0000 is written into R10. If no bit is set, 0x8000 is written into R10 allowing for a test of a negative result to determine whether or not a bit was set. The test is performed by the BGEZ, which branches only if the result is greater or equal to zero.

Using LIMD

```
LIMD R8, 0x67FC
```

Load Register R8 with the 16-bit value 0x67FC.

Using MAX and MIN

```
MAX   R8, R9, R17
```

For the purpose of the MAX and MIN instructions, R8 and R9 are treated as unsigned numbers. The maximum of R8 and R9 is placed into R17.

```
MIN   R8, R9, R17, UM
```

For the purpose of the MAX and MIN instructions, R8 and R9 are treated as unsigned numbers. The minimum of R8 and R9 is placed into R17 and is written back into the memory location linked to R17 by a previous LMFM instruction.

Section 3   Signal Descriptions and Electrical Characteristics

This section of the manual describes the signal descriptions and electrical characteristics of the MXT3010. The chapters included in this chapter are:

- Timing

- Pin information

- Electrical parameters

- Mechanical and thermal characteristics

Timing

MXT3010EP timing - general information

Definition of switching levels

FIGURE 91.Switching level voltages

The following switching level information has been used in the generation of the MXT3010EP device timing.

- For a low-to-high transition, a signal is considered to no longer be low when it reaches 0.8 V and is considered to be high upon reaching 2.0 V.

- For a high-to-low transition, a signal is considered to no longer be high when it reaches 2.0 V and is considered to be low upon reaching 0.8 V.

Input clock details

FIGURE 92.Input clock waveform (pin FN)

TABLE 84.   Input clock timing parameters

	100MHz		
Parameter	*Min*	*Max*	*Description*
TC(FN)	19.98	(1)	Input clock period
TH(FN)	.4 TC	.6 TC	Input clock high duration
TL(FN)	.4 TC	.6 TC	Input clock low duration
TR(FN)	-	1.5	Input clock rise time (2)
TF(FN)	-	1.5	Input clock fall time (2)

1. With the exception of the PLL circuit, the MXT3010 is a fully static design and can operate with $1/TC(FN) = 0$. The device is characterized for operation approaching 0 Hz, but is not tested under this condition.

2. In order to maintain low jitter, pay close attention to the input clock edge rate. One primary component of jitter occurs only during the input clock state transition. To reduce this jitter component, Maker recommends that the FN pin be driven directly from the output of a part designed for clock tree distribution. Maker's reference design uses an FCT3807 device from IDT. Other designs that require a clock driver with an integrated PLL use the CDC586 clock driver from Texas Instruments.

MXT3010EP Fast Memory interface timing

This section includes a Fast Memory timing table and abbreviated timing diagrams that show only enough signals to identify all of the timing parameters. For a more complete explanation of the signalling used in various transfers, see "Fast Memory sequence diagrams" on page 56.

These notes relate to Fast Memory timing issues:

- During cycles in which Fast Memory is IDLE, the MXT3010EP sources the CPU's Instruction fetch address onto FADRS(17:2) so that one can view the SWAN processor's instruction execution flow on a logic analyzer. The MXT3010EP performs Fast Memory reads during these cycles but discards the data read from the SRAMs.

- In a dual bank system, the bank accessed by the Fast Memory controller during IDLE cycles is determined by the ICACHE address. If the ICACHE address maps to bank 0, bank 0 is read during IDLE cycles. If it maps to bank 1, bank 1 is read during IDLE cycles.

- All address and control lines should be series terminated.

- At the boundary of the chip, the MXT3010EP guarantees that FOE0_ is de-asserted before asserting FOE1_, and similarly that FOE1_ is de-asserted before asserting FOE0_.

TABLE 85. Fast Memory timing for the Maker MXT3010EP

	100 MHz		Fast Memory timing (in nanoseconds)	
Par	*Min*	*Max*	*Pins*	*Description*
T_1		17.0	FADRS[17:2]	Clk to address output valid
T_2	11.3		FADRS[17:2]	Hold time provided by MXT3010
T_3	3.8		FDAT[31:0]	Input setup time to rising clk
T_4	1.0		FDAT[31:0]	Input hold time from rising clock
T_5		17.0	FCS0_, FCS1_, FWE[0:3]_	Clk to output valid
T_6	11.3		FCS0_, FCS1_, FWE[0:3]_	Hold time provided by MXT3010
T_7	1.3	2.5	FOE0_, FOE1_	CLK to FOE
T_8	7.0	10.0	FOE0_, FOE1_	CLK to FOEx_
T_9	5.8	15.0	FDAT[31:0]	CLK to output in low Z state
T_{10}	1.8	8.0	FDAT[31:0]	CLK to output in high Z state
T_{11}		8.5[a]	FDAT[31:0]	CLK to FDAT[31:0] output valid
T_{12}	1.8		FDAT[31:0]	Hold time provided by MXT3010

a. For the first word of data, T_9 is the critical timing parameter.

FIGURE 93.Timing for Fast Memory reads

FIGURE 94.Timing for Fast Memory writes

MXT3010EP UTOPIA interface timing

This section includes a UTOPIA timing table and abbreviated timing diagrams that show only enough signals to identify all of the timing parameters. For a more complete explanation of the signalling used in various transfers, see "UTOPIA port sequence diagrams" on page 94.

All timing shown in Table 86 is relative to either RX_CLK or TX_CLK as shown in Figure 97 or Figure 98 respectively. The relationship between the MXT3010EP input clock (FN) and a half-speed RX_CLK/TX_CLK is shown in Figure 95 (also see Figure 26 on page 73). The relationship between the MXT3010EP input clock (FN) and a quarter-speed RX_CLK/TX_CLK is shown in Figure 96 (also see Figure 27 on page 73). The values of T_9 and T_{10} are shown in Table 87.

FIGURE 95.FN and half-speed RX_CLK/TX_CLK

FIGURE 96.FN and quarter-speed RX_CLK/TX_CLK

TABLE 86.   UTOPIA timing for Maker MXT3010EP

Par	100 MHz Min	100 MHz Max	UTOPIA timing (in nanoseconds) Pins	Description
T_1		6.0	TXSOC, TXENB_, RXENB_, TXCTRL, RXCTRL	TX_CLK to output valid
T_2	1.3		TXSOC, TXENB_, RXENB_, TXCTRL, RXCTRL	Hold time provided by MXT3010
T_3		7.0	TXDATA[7:0]	TX_CLK to output valid
T_4	1.3		TXDATA[7:0]	Hold time provided by MXT3010
T_5	4.0		RXSOC, RXCLAV	Input setup time to TX_CLK/ RX_CLK
	4.0		TXCLAV	Input setup time to TX_CLK/ RX_CLK
T_6	1.3		RXSOC, RXCLAV	Input hold time from TX_CLK/ RX_CLK
	1.3		TXCLAV	Input hold time from TX_CLK/ RX_CLK
T_7	4.0		RXDATA[7:0]	Input setup time to RX_CLK
T_8	1.3		RXDATA[7:0]	Input hold time from RX_CLK

Notes:1. Adrs/Chip Selects/Write Enables are driving at 100 MHz edge corresponding with falling edge of 50 MHz chip clock.

2. All maximum timing is specified with 30 pF loads (Adrs/Ctrl), 25 pF (Data). All minimum timing is specified with 5 pF loads.

3. A circuit stretches the minimum time-on time of the data on read followed by write cycles.

4. Currently the FOE of one bank is guaranteed to be de-asserted before the second bank is asserted. This is not actually required since the RAMs are designed to allow back-to-back bank operation.

TABLE 87. Delay of UTOPIA clocks relative to MXT3010EP internal clock (CLK)

| *Par* | *100 MHz* | | *UTOPIA TIMING* | |
	Min	*Max*	*Pins*	*Description*
T_9	1.3	5.5	RX_CLK/TX_CLK	Rise time (r-min/r-max)
T_{10}	1.3	6.0	RX_CLK/TX_CLK	Fall time (f-min/f-max)

Notes: 1. The hold time for data dn control is reduced on the MXT3010EP at the expense of the setup time. This was done to allow an easier interface to PHY devices when guaranteeing hold time.

2. Multi-PHY designs must ensure that no bus fight exists on the CLAV lines.

3. All maximum timing is specified with 15 pF loads. All minimum timing is specified with 5 pF loads.

FIGURE 97. UTOPIA port receive timing

FIGURE 98.UTOPIA port transmit timing

MXT3010EP Port1 timing

This section includes a Port1 timing table and abbreviated timing diagrams that show only enough signals to identify all of the timing parameters. For a more complete explanation of the signalling used in various transfers, see "Port1 basic protocol" on page 110.

These notes relate to Port1 timing issues:

- If the external controller requires the MXT3010EP to drive the P1AD(31:0) and P1 control buses when no other master owns the bus, the external controller should select address cycles.

- For exact timing numbers for CIN_BSY and COUT_RDY assertion and deassertion, see "MXT3010EP miscellaneous control signal timing" on page 359.

- Add Wait States during COMMIN register writes and COMMOUT register reads by extending COMMSEL and P1RD for one or more additional cycles (beyond those shown). For COMMIN register writes, COMMIN data MXT3010EP samples the final write clock cycle (the cycle in which COMMSEL is sampled low at the end of the cycle). For COMMOUT register reads, MXT3010EP sources the contents of the COMMOUT register throughout the extended cycle.

TABLE 88.   Port1 timing table

Par	100 MHz Min	Max	Port1 read and write timing (in nanoseconds) Pins	Description
T_1		7.0	P1QRQ_, P1RQ_, P1RD, P1END_, P1IRDY_, P1HWE[0:1] P1AD[31:0]	CLK to output valid
T_2	1.3		P1QRQ_, P1RQ_, P1RD, P1END_, P1IRDY_, P1HWE[0:1], P1AD[31:0]	Hold time provided by MXT3010
T_3	7.0		P1TRDY_, P1ASEL_	Input setup time to rising clock
T_4	1.0		P1TRDY_, P1ASEL_	Input hold time from rising clock
T_5	1.3	7.5	P1RD, P1END_, P1HWE[0:1], P1IRDY_	Clock to output in low Z state
T_6	1.3	10.5	P1RD, P1END_, P1HWE[0:1], P1IRDY_	Clock to output in high Z state
T_7	1.3	7.3	P1AD[31:0]	Clock to output in low Z state
T_8	1.3	10.5	P1AD[31:0]	Clock to output in high Z state
T_9	1.3	8.8	P1AD[31:0]	Clock to P1AD(31:0) valid
T_{10}	4.0		P1AD[31:0]	Input setup time to rising clock
T_{11}	1.0		P1AD[31:0]	Input hold time from rising clock
T_{12}	6.0		COMMSEL, P1RD	Input setup time to rising clock
T_{13}	1.0		COMMSEL, P1RD	Input hold time from rising clock

Note: All maximum timing is specified with 15 pF loads. All minimum timing is specified with 5 pF loads.

FIGURE 99.Port1 read timing

FIGURE 100.Port1 write timing

FIGURE 101.COMMIN register write, COMMOUT register read timing

MXT3010EP Port2 timing

This section includes a Port2 timing table and abbreviated timing diagrams that show only enough signals to identify all of the timing parameters. For a more complete explanation of the signalling used in various transfers, see "Port2 basic protocol" on page 137.

This note relates to Port2 timing issues:

- If the external controller requires the MXT3010EP to actively drive the P2AD(15:0) and P2 control buses when no other master owns the bus, it should do so by selecting address cycles.

TABLE 89. Port2 timing table

	100 MHz		Port2 read and write timing (in nanoseconds)	
Par	*Min*	*Max*	*Pins*	*Description*
T_1		8.0	P2QRQ_ P2QBRST, P2RQ_, P2RD, P2END_, P2IRDY_ P2AD[15:0]	CLK to output valid
T_2	1.3		P2QRQ_, P2QBRST, P2RQ_, P2RD, P2END_, P2IRDY_, P2AD[15:0]	Hold time provided by MXT3010
T_3	8.0		P2TRDY_, P2ASEL_	Input setup time to rising clock
T_4	1.0		P2TRDY_, P2ASEL_	Input hold time from rising clock
T_5	2.0	8.0	P2RD, P2END_, P2AI[3:0], P2IRDY_	Clock to output in low Z state
T_6	2.0	11.0	P2RD P2END_, P2AI[3:0], P2IRDY_	Clock to output in high Z state
T_7	1.3	8.0	P2AD[15:0]	Clock to output in low Z state
T_8	2.0	10.0	P2AD[15:0]	Clock to output in high Z state
T_9	1.3	8.0	P2AD[15:0]	Clock to P2AD(15:0) valid
T_{10}	4.0		P2AD[15:0]	Input setup time to rising clock
T_{11}	1.0		P2AD[15:0]	Input hold time from rising clock

Note: All maximum timing is specified with 15 pF loads. All minimum timing is specified with 5 pF loads.

FIGURE 102.Port2 read timing

FIGURE 103.Port2 write timing

MXT3010EP miscellaneous control signal timing

This section includes a miscellaneous control signal timing table and a miscellaneous control signal timing diagram.

Note that the MXT3010EP drives CIN_BUSY within two clock cycles of a COMMIN Register write operation. Therefore, the host should wait at least two system clock cycles from the completion of a COMMIN register write before testing CIN_BUSY.

TABLE 90. Miscellaneous control signal timing

	100 MHz		*Misc. control signal timing (in nanoseconds)*	
Par	*Min*	*Max*	*Pins*	*Description*
T_1		8.0	ICSO_(D:A)	CLK to output valid
T_2	1.3		ICSO_(D:A)	Hold time provided by MXT3010
T_3	3.5		ICSI_(D:A)	Input setup time to rising clock
T_4	1.0		ICSI_(D:A)	Input hold time from rising clock
T_5		8.5	CINBUSY, COUTRDY	CLK to output valid
T_6	1.3		CINBUSY, COUTRDY	Hold time provided by MXT3010

FIGURE 104. Timing of CIN_BUSY and COUT_RDY

MXT3010EP Reset timing

This section includes a MXT3010EP reset timing table and diagram.

These notes relate to MXT3010EP reset timing issues:

1. When RESET_ is de-asserted, two events occur: a) the MXT3010EP fetches boot code from the designated port, and b) the MXT3010EP begins a cache initialization routine which takes 1028 input clock cycles.

2. The MXT3010EP does not actively drive ICSO_(D:A) during reset. Therefore, these pins float unless actively driven or pulled up or down externally. The values sensed on these pins at RESET_ removal provide configuration information to the MXT3010EP. For more information, see "Device Initialization" on page 401.

TABLE 91. MXT3010EP reset timing

Par	100 MHz		Reset timing (in nanoseconds)	
	Min	Max	Pins	Description
T_1	10,000 clock cycles		CLK, also called FN	Minimum number of clock cycles that reset must be held low for to allow internal PLL to lock.
T_2	13.0		RESET	Input setup time to rising CLK for removal of reset signal. Reset may be asserted asynchronously but must be deasserted synchronous to CLK.
T_{2a}	1.5		RESET	RESET hold time
T_3	5		ICSO_[D:A]	Input setup time to rising CLK. This setup requirement need only be met for the rising edge of CLK for which RESET is sampled high for the first time.
T_4	3		ICSO_[D:A]	Input hold time from rising CLK. This hold requirement need only be met for the rising edge of CLK following the rising edge of CLK for which RESET is sampled high for the first time.

FIGURE 105. MXT3010EP reset timing

As indicated in Figure 105, the de-assertion of RESET_ must be done within certain timing constraints. These timing constrains occur because the MXT3010EP samples the RESET_ pin with an internal clock that operates at twice the rate of the input clock. This is done to establish a phase relationship with the input clock. For example, if the input clock operates at 50 MHz, the MXT3010EP samples RESET_ with a 100 MHz internal clock. The system designer must meet the timing requirements shown in Figure 106 and Table 92.

FIGURE 106.Reset trailing edge timing

TABLE 92.   MXT3010EP RESET_ timing parameters

Par	*100 MHz*		*Reset timing (in nanoseconds)*
	Min	*Max*	*Description*
T_1	20		Input clock period
T_2	10		Internal clock period
T_3	3		Setup to internal clock edge
T_4	1.5		Hold from internal clock edge
T_5	13		Setup to input clock edge (Note 1)

Notes: 1.Parameter T_5 in Table 92 is the same as parameter T_2 in Table 91.

2. Unless otherwise specified, all times in this table are relative to the input clock.

Given the 20ns period of the 50Mhz input clock, this leaves a 5.5ns (20-13-1.5=5.5) window in which RESET_ can be removed. Although care must be taken to meet these requirements, it can be routinely accomplished.

Several methods of achieving this timing are possible. Current ASIC technology meets this timing. FPGA implementations are more difficult, but possible. On the MXT3015 Evaluation Card, Maker uses a fast external flip-flop to constrain timing to within this window. Figure 107 shows this circuit.

FIGURE 107.Reset timing circuit

MXT3010EP Fast Memory interface operation

This example shows a Fast Memory connection using Samsung KM718B90 Synchronous SRAMs. Connections are shown for a single bank system using two SRAM.

MXT3010EP	*SRAM with bits (31:16) of Fast Data.*	*SRAM with bits (15:0) of Fast Data.*	*Comments*
FADRS(17:2)	A(15:0)	A(15:0)	The MXT3010EP provides a word address and individual byte write enable lines. For a 64Kx32 Control Memory, FADRS(17:2) are connected to A(15:0) of each of the SRAMS. For a 32Kx32 Control Memory, FADRS(17) is unconnected and FADRS(16:2) are connected to A(14:0) of each of the SRAMs.
FDAT(15:0)	N/C	I/O(15:0)	
FDAT(31:16)	I/O(15:0)	N/C	
Tied to GND through 10K Ohm resistors	I/O(17:16)	I/O(17:16)	If x18 devices are used, tie I/O(17:16) to ground through dedicated 10K Ohm resistors.
FCS0_	CS_, ADSC_	CS_, ADSC_	In a single bank system, FCS0_ is tied to the CS_ and ADSC_ of each of the SRAM devices.
FOE0_	OE_	OE_	In a single bank system, FOE0_ is tied to the output enable of each of the SRAM devices.
Tied to PWR	ADV_, ADSP_	ADV_, ADSP_	The MXT3010EP does not use these control signals.
FWE0_	UW_		FWE0_ controls byte FDAT(31:24)
FWE1_	LW_		FWE1_ controls byte FDAT(23:16)
FWE2_		UW_	FWE2_ controls byte FDAT(15:8)
FWE3_		LW_	FWE3_ controls byte FDAT(7:0)
FN (CLK)	K	K	Tied to system clock. This is the same clock that is connected to the MXT3010EP's clock input (FN).

MXT3010EP JTAG operation

For JTAG SCAN chain connection information contact Maker Communications.

The MXT3010EP provides a pin, TRI_, that places all output drivers in the high impedance state except RXCLK, TXCLK, and the outputs associated with the PLL.

CHAPTER 18 *Pin Information*

This chapter provides information on the MXT3010EP pinouts. The information includes pin diagrams, signal descriptions, and pin listings.

MXT3010EP pinout

Figure 108 provides a diagram of the MXT3010EP pinout.

FIGURE 108.MXT3010EP package/pin diagram

MXT3010EP signal descriptions

- Port1

- Port2

- UTOPIA port

- Fast Memory controller

- Inter-chip and communication registers

- Miscellaneous signals, such as clock, control and test

- Power and ground pins

TABLE 93. MXT3010EP Port1 signal descriptions

Pin #[a]	Symbol	I/O	Name	Description
179, 178, 174, 173, 171, 160, 170, 168, 157, 153, 166, 155, 164, 161, 147, 158, 163, 156, 148, 152, 144, 151, 138, 132, 146, 150, 143, 141, 142, 136, 140, 137	P1AD[31:0]	I/O	Port1 Address/Data [31:0]	This is a multiplexed, bi-directional 32-bit bus. Data is read into and out of the MXT3010EP during DMA and COMM operations; see "The Port1 and Port2 Interfaces" on page 97. and see "Communications" on page 177.
126	P1RQ_	O	Port1 Request	This signal indicates that commands are in the active stage of the Port1 DMA command queue.
131	P1QRQ_	O	Port1 DMA Queue Request	This signal indicates that commands are in the queue stage of the Port1 DMA command queue.
134	P1RD	I/O	Port1 Read / Write select	The MXT3010EP drives this signal during a DMA transfer, and the host drives the signal during a communication register transfer. In either case, this signal indicates a read (1) or a write (0) transfer.
130	P1END_	O	Port1 End	This signal indicates the last cycle of a DMA operation.
128 125	P1HWE1 P1HWE0	O O	Port1 Halfword Enable [1:0]	During data cycles, P1HWE[1:0] act as Half Word Enables. If P1HWE[0] is asserted, P1AD[31:16] should contain valid data. If P1HWE[1] is asserted, P1AD[15:0] should contain valid data.
127	P1IRDY_	O	Port1 Interface Initiator Ready	During DMA write data cycles, the MXT3010EP asserts P1IRDY_ while it is sourcing valid data on P1AD[31:0]. During DMA read data cycles, the MXT3010EP asserts P1IRDY_ if it can sample P1AD[31:0] on the next rising edge of clock.
135	P1ABRT_	I	Port1 Transfer Abort	The host drives this signal to abort burst DMA operation.
116	P1TRDY_	I	Port1 Target Ready	The host drives this signal and the host inserts wait states. With P1ASEL_, the host deselects (i.e., tri-state) MXT3010EPs.
112	P1ASEL_	I	Port1 Address Select	The host drives this signal and the host selects an address or data cycle. With P1TRDY_ the host deselects (tri-state) MXT3010EPs.

a. Pin numbers in this table, and in all subsequent tables, are listed in descending order (for example, P1AD31 to P1AD0).

TABLE 94. MXT3010EP Port2 signal descriptions

Pin #	Symbol	I/O	Name	Description
185, 187, 196, 192, 194, 197, 201, 190, 204, 206, 203, 195, 210, 212, 200, 208	P2AD [15:0]	I/O	Port2 Address/Data [15:0]	This is a multiplexed, bi-directional 16-bit bus. Data is read into and out of the MXT3010EP during DMA operations. For operational details see "The Port1 and Port2 Interfaces" on page 97.
198, 211, 216, 202	P2AI[3:0]	O	Port2 Address Index Bus	These signals are multi-purpose. In burst mode P2AI[3:0] represent an address index consisting of the lower four bits of an address. In non-burst mode P2AI[3:2] represent the two most significant address bits. P2AI[1] represents P2RD_. P2AI[0] represents Address Latch Enable.
186	P2RQ_	O	Port2 Request	This signal indicates that commands are in the active stage of the Port2 DMA command queue.
183	P2QRQ_	O	Port2 DMA Queue Request	This signal indicates that commands are in the queue stage of the Port2 DMA command queue.
191	P2RD	O	Port2 Read / Write Select	The MXT3010EP drives this signal during a DMA transfer. This signal indicates a write (0) transfer or a read (1) transfer.
188	P2END_	O	Port2 End	On DMA operations, the MXT3010EP asserts P2END_ to indicate the last cycle of the transfer.
182	P2QBRST	O	Port2 Burst	This signal indicates burst (1) or non-burst (0) transfer mode. For operational details see "The Port1 and Port2 Interfaces" on page 97.
184	P2IRDY_	O	Port2 Interface Initiator Ready	During DMA write data cycles, the MXT3010EP asserts P2IRDY_ while it is sourcing valid data on P2AD[15:0]. During DMA read data cycles, the MXT3010EP asserts P2IRDY_ if it can sample P2AD[15:0] on the next rising edge of clock.
172	P2TRDY_	I	Port2 Target Ready	The host drives this signal and inserts wait states. With P2ASEL_ the host can deselect [i.e., tri-state] MXT3010EPs.
176	P2ASEL_	I	Port2 Address Select	The host drives this signal and selects address or data cycle. With P2TRDY_ the host can deselect (tri-state) MXT3010EPs.

TABLE 95. UTOPIA port signal description

Pin #	*Symbol*	*I/O*	*Name*	*Description*
237, 234, 2, 233, 231, 230, 221, 218	TXDATA [7:0]	I/O	UTOPIA Transmit Data [7:0]	In 8-bit bi-directional mode, these pins send data to the PHY device. In 16-bit uni-directional mode, these pins are byte 0 or byte 1 of the bus. For operational details see "The UTOPIA port" on page 69.
239, 232, 238, 220	TXCTRL [3:0]	I/O	UTOPIA Transmit Control	These signals provide multi-PHY control information. For operational details see "The UTOPIA port" on page 69.
224	TXCLAV	I	Transmit Cell Available	This signal indicates to the MXT3010EP that the PHY is ready to accept a cell.
223	TXENB_	O	Transmit Enable	This signal indicates to the PHY that valid data is on the bus.
222	TXSOC	O	Transmit Start of Cell	This signal indicates to the PHY the start of a cell.
226	TX_CLK	O	Transmit Clock	The MXT3010EP provides the transmit clock.
17, 16, 21, 23, 12, 14, 30, 4	RXDATA [7:0]	I/O	UTOPIA Receive Data [7:0]	In 8-bit bi-directional mode, these pins are used to receive data from the PHY device. In 16-bit uni-directional mode, these pins are byte 0 or byte 1 of the bus. For operational details see "The UTOPIA port" on page 69.
26, 3, 24, 18	RXCTRL [3:0]	I/O	UTOPIA Receive Control	These signals provide multi -PHY control information. For operational details see "The UTOPIA port" on page 69.
20	RXCLAV	I	Receive Cell Available	This signal indicates to the MXT3010EP that the PHY has a cell ready to send to the MXT3010EP.
15	RXENB_	O	Receive Enable	This signal indicates to the PHY that the MXT3010EP is ready to receive data.
11	RXSOC	I	Receive Start of Cell	This signal indicates to the MXT3010EP the start of a cell.
8	RX_CLK	O	Receive Clock	The MXT3010EP provides the receive clock.

TABLE 96.   MXT3010EP Fast Memory controller signal description

Pin #	*Symbol*	*I/O*	*Name*	*Description*
80, 90, 98, 87, 91, 94, 93, 97, 95, 70, 66, 72, 74, 71, 76, 75	FADRS [17:2]	O	Fast Memory Address Bus [17:2]	Byte address.
44, 46, 50, 48, 51-54, 57-59, 56, 62, 67, 64, 68, 104, 102, 100, 103, 106, 108, 107, 110, 113, 111, 117, 114, 118, 119, 122, 124,	FDAT [31:0]	I/O	Fast Memory Data Bus [31:0]	Fast Memory data bus.
82 84	FCS1_ FCS0_	O	Fast Memory Control Signals [1:0]	These signals select the bank of SRAM addressed during Fast Memory operations. FCS0_ low = bank 1 FCS1_ low = bank 2 When operating in Mode 1, these Chip Select pins are used as Fast Memory Address lines 18 and 19. FCS1_ = FADRS[19] FCS0_ = FADRS[18] See Figure 16 on page 55.
86 78	FOE1_ FOE0_	O	Fast Memory Output Enable [1:0]	These signals enable bank 1 and bank 2 of SRAM. FOE0_ low = bank 1 FOE1_ low = bank 2
88 83 77 81	FWE3_ FWE2_ FWE1_ FWE0_	O	Fast Memory Write Enable [3:0]	These signals select the byte target during a Fast Memory write operation: FWE0_ low = byte 0 FDAT[31:24] FWE1_ low = byte 1 FDAT[23:16] FWE2_ low = byte 2 FDAT[15:8] FWE3_ low = byte 3 FDAT[7:0]

TABLE 97. MXT3010EP inter-chip and communication registers signal description

Pin #	Symbol	I/O	Name	Description
167	CINBUSY	O	COMMIN Busy	COMMIN Busy signals the status of the COMMIN Register. The MXT3010EP drives this pin high when the Host writes to the COMMIN Register. The MXT3010EP clears the signal when it reads the COMMIN Register. As long as CINBUSY is high, the COMMIN Register is full.
177	COUTRDY	O	COMMOUT Ready	This signal signals the status of the COMMOUT Register. The MXT3010EP asserts this signal (1) when it writes to the COMMOUT Register. When the host reads the COMMOUT register, the MXT3010EP clears the signal (0). As long as COUTRDY is 1, the COMMOUT Register is full.
162	COMMSEL	I	Comm Select	When the Comm Select signal is asserted (1) and the P1RD signal is low (0), the COMMIN register is a target of a write operation (Host to MXT3010EP). When the Comm Select signal is asserted (1) and the P1RD signal is high (1), the COMMOUT register is the source of a read operation (MXT3010EP to Host).
42 47 35 6	ICSI_D ICSI_C ICSI_B ICSI_A	I	ICS Input [D:A]	The MXT3010EP uses these signals to poll the state of external devices. They also control the Sparse Event Register.
31 236 28 22	ICSO_D ICSO_C ICSO_B ICSO_A	O	ICS Output [A:D]	These signals are used by the MXT3010EP to signal the state of the MXT3010EP to external devices. The SWAN processor sets the state of these signals by setting or clearing bits in the Sparse Event Register. These signals are also used during device initialization.

TABLE 98.   MXT3010EP miscellaneous clock, control, and test signal descriptions

Pin #	Symbol	I/O	Name	Description
40	FN	I	Input Clock	This signal provides the MXT3010EP device clock.
123	RESET_	I	Reset	Device reset.
217	OSC_EN	I	Oscillator Enable	Oscillator enable is used for testing of ring oscillator.
63	LSSD_TEST	I		This pin is used for scan test.
34	RANGE	I	Operation Range select	The RANGE pin affects the operating range of the PLL VCO: High = Output frequency 50-100 mHz Low = Output frequency 100-400 mHz This pin is customarily left floating, thus allowing the internal pull up to keep the pin in the high state.
38	BP_		Bypass Pin	This pin is used during production testing of the PLL.
213	TRI_	I	Tri-State Test	This signal places all of the tri-state and bi-directional I/Os into tri-state.
27	TCK	I	Test Clock	JTAG Test Clock input
33	TDI	I	Test Data In	JTAG Test Data input
227	TDO	O	Test Data Out	JTAG Test Data output
207	TMS	I	Test Mode Select	JTAG Test Mode Select input
215	TRS	I	Test Reset	JTAG Reset input
5, 10, 32, 101, 228	RESRVD	I/O	Reserved	These pins are reserved for future functionality and should be left floating.

TABLE 99. Power and ground pin descriptions

Pin #	Symbol	I/O	Name	Description
13, 29, 41, 49, 60, 65, 73, 89, 96, 109, 120, 133, 149, 169, 180, 193, 209, 229, 240	VDD	I	3.3 volt power supply	These pins each require a +3.3 VDC (± 5%) power supply input. They supply current to the 3.3-volt output buffers and the core logic of the device.
37	VAA		PLL power supply	This pin requires a +3.3 VDC (± 5%) power supply input. It is used to supply current to the PLL. Please refer to the section on the PLL for proper decoupling strategy.
1, 7, 9, 19, 25, 36, 39, 43, 45, 55, 61, 69, 79, 85, 92, 99, 105, 115, 121, 129, 139, 145, 154, 159, 165, 175, 181, 189, 199, 205, 214, 219, 225, 235	GND	-	Ground	These pins provide ground return paths for the various power supply inputs.

MXT3010EP JTAG/PLL pin termination

Table 100 indicates how test and reserved pins on the
MXT3010EP should be terminated for normal operation.

TABLE 100. MXT3010EP pin terminations

Pin Name	Pin #	Termination
BP_	38	External pull up (1 K ohms) to +3.3V
RANGE	34	External pull up (4.7K ohms) to +3.3V
OSC_EN_	217	
TRI_	213	
TCK	27	
TDI	33	
TMS	207	
LSSD_TEST	63	External pull down (120 ohms) to GND
TRS	215	
RESERVED	5, 10, 32, 101, 228	Leave floating

MXT3010EP pin listing

This section provides the pin listings for the MXT3010EP. Table 102 provides descriptions of the pin types listed in Table 101.

TABLE 101. MXT3010EP pin listing

Pin	Pin Label	Pad	Pin	Pin	Pin Label	Pad	Pin	Pin	Pin Label	Pad	Pin
1	GND			30	RXDATA1	IO2	I/O	59	FDAT21	IO4	I/O
2	TXDATA5	IO2	I/O	31	ICSO_D	IO4	I/O	60	VDD		
3	RXCTRL2[a]	IO3	OUT	32	RESRVD			61	GND		
4	RXDATA0	IO2	I/O	33	TDI	IN1	IN	62	FDAT19	IO4	I/O
5	RESRVD			34	RANGE	IO4	PLL	63	LSSD_TEST	IN2	IN
6	ICSI_A	IN1	IN	35	ICSI_B	IN1	IN	64	FDAT17	IO4	I/O
7	GND			36	GND			65	VDD		
8	RX_CLK	IO2	I/O	37	VAA			66	FADRS7	IO6	OUT
9	GND			38	BP_			67	FDAT18	IO4	I/O
10	RESRVD			39	GND			68	FDAT16	IO4	I/O
11	RXSOC	IO3	IN	40	FN			69	GND		
12	RXDATA3	IO2	I/O	41	VDD			70	FADRS8	IO6	OUT
13	VDD			42	ICSI_D	IN1	IN	71	FADRS4	IO6	OUT
14	RXDATA2	IO2	I/O	43	GND			72	FADRS6	IO6	OUT
15	RXENB	IO3	OUT	44	FDAT31	IO4	I/O	73	VDD		
16	RXDATA6	IO2	I/O	45	GND			74	FADRS5	IO6	OUT
17	RXDATA7	IO2	I/O	46	FDAT30	IO4	I/O	75	FADRS2	IO6	OUT
18	RXCTRL0	IO3	OUT	47	ICSI_C	IN1	IN	76	FADRS3	IO6	OUT
19	GND			48	FDAT28	IO4	I/O	77	FWE1_	IO6	OUT
20	RXCLAV	IO3	IN	49	VDD			78	FOE0_	IO6	OUT
21	RXDATA5	IO2	I/O	50	FDAT29	IO4	I/O	79	GND		
22	ICSO_A	IO4	I/O	51	FDAT27	IO4	I/O	80	FADRS17	IO6	OUT
23	RXDATA4	IO2	I/O	52	FDAT26	IO4	I/O	81	FWE0_	IO6	OUT
24	RXCTRL1	IO1	OUT	53	FDAT25	IO4	I/O	82	FCS1_	IO6	OUT
25	GND			54	FDAT24	IO4	I/O	83	FWE2_	IO6	OUT
26	RXCTRL3	IO2	OUT	55	GND			84	FCS0_	IO6	OUT
27	TCK	IN1	IN	56	FDAT20	IO4	I/O	85	GND		
28	ICSO_B	IO4	I/O	57	FDAT23	IO4	I/O	86	FOE1_	IO6	OUT
29	VDD			58	FDAT22	IO4	I/O	87	FADRS14	IO6	OUT

TABLE 101. MXT3010EP pin listing

88	FWE3_	IO6	OUT	126	P1RQ_	IO4	OUT	164	P1AD19	IO4	I/O
89	VDD			127	P1IRDY_	IO4	OUT	165	GND		
90	FADRS16	IO6	OUT	128	P1HWE1	IO4	OUT	166	P1AD21	IO4	I/O
91	FADRS13	IO6	OUT	129	GND			167	CINBUSY	IO5	OUT
92	GND			130	P1END_	IO4	OUT	168	P1AD24	IO4	I/O
93	FADRS11	IO6	OUT	131	P1QRQ_	IO4	OUT	169	VDD		
94	FADRS12	IO6	OUT	132	P1AD8	IO4	I/O	170	P1AD25	IO4	I/O
95	FADRS9	IO6	OUT	133	VDD			171	P1AD27	IO4	I/O
96	VDD			134	P1RD	IO4	I/O	172	P2TRDY_	IO2	IN
97	FADRS10	IO6	OUT	135	P1ABRT_	IO4	IN	173	P1AD28	IO4	I/O
98	FADRS15	IO6	OUT	136	P1AD2	IO4	I/O	174	P1AD29	IO4	I/O
99	GND			137	P1AD0	IO4	I/O	175	GND		
100	FDAT13	IO4	I/O	138	P1AD9	IO4	I/O	176	P2ASEL_	IO2	IN
101	RESRVD			139	GND			177	COUTRDY	IO5	OUT
102	FDAT14	IO4	I/O	140	P1AD1	IO4	I/O	178	P1AD30	IO4	I/O
103	FDAT12	IO4	I/O	141	P1AD4	IO4	I/O	179	P1AD31	IO4	I/O
104	FDAT15	IO4	I/O	142	P1AD3	IO4	I/O	180	VDD		
105	GND			143	P1AD5	IO4	I/O	181	GND		
106	FDAT11	IO4	I/O	144	P1AD11	IO4	I/O	182	P2QBRST	IO4	OUT
107	FDAT9	IO4	I/O	145	GND			183	P2QRQ_	IO4	OUT
108	FDAT10	IO4	I/O	146	P1AD7	IO4	I/O	184	P2IRDY_	IO4	OUT
109	VDD			147	P1AD17	IO4	I/O	185	P2AD15	IO4	I/O
110	FDAT8	IO4	I/O	148	P1AD13	IO4	I/O	186	P2RQ_	IO4	OUT
111	FDAT6	IO4	I/O	149	VDD			187	P2AD14	IO4	I/O
112	P1ASEL_	IO4	IN	150	P1AD6	IO4	I/O	188	P2END_	IO4	OUT
113	FDAT7	IO4	I/O	151	P1AD10	IO4	I/O	189	GND		
114	FDAT4	IO4	I/O	152	P1AD12	IO4	I/O	190	P2AD8	IO4	I/O
115	GND			153	P1AD22	IO4	I/O	191	P2RD	IO4	OUT
116	P1TRDY_	IO4	IN	154	GND			192	P2AD12	IO4	I/O
117	FDAT5	IO4	I/O	155	P1AD20	IO4	I/O	193	VDD		
118	FDAT3	IO4	I/O	156	P1AD14	IO4	I/O	194	P2AD11	IO4	I/O
119	FDAT2	IO4	I/O	157	P1AD23	IO4	I/O	195	P2AD4	IO4	I/O
120	VDD	IO4		158	P1AD16	IO4	I/O	196	P2AD13	IO4	I/O
121	GND			159	GND			197	P2AD10	IO4	I/O
122	FDAT1	IO4	I/O	160	P1AD26	IO4	I/O	198	P2AI3	IO4	OUT
123	RESET_	IN1	IN	161	P1AD18	IO4	I/O	199	GND		
124	FDAT0	IO4	I/O	162	COMMSEL	IO5	IN	200	P2AD1	IO4	I/O
125	P1HWE0	IO4	OUT	163	P1AD15	IO4	I/O	201	P2AD9	IO4	I/O

TABLE 101. MXT3010EP pin listing

202	P2AI0	IO4	OUT	215	TRS	IN2	IN	228	RESRVD		
203	P2AD5	IO4	I/O	216	P2AI1	IO4	OUT	229	VDD		
204	P2AD7	IO4	I/O	217	OSC_EN_	IN1	IN	230	TXDATA2	IO2	I/O
205	GND			218	TXDATA0	IO2	I/O	231	TXDATA3	IO2	I/O
206	P2AD6	IO4	I/O	219	GND			232	TXCTRL2	IO3	OUT
207	TMS	IN1	IN	220	TXCTRL0[b]	IO2	OUT	233	TXDATA4	IO2	I/O
208	P2AD0	IO4	I/O	221	TXDATA1	IO2	I/O	234	TXDATA6	IO2	I/O
209	VDD			222	TXSOC	IO2	OUT	235	GND		
210	P2AD3	IO4	I/O	223	TXENB_	IO2	OUT	236	ICSO_C	IO4	I/O
211	P2AI2	IO4	OUT	224	TXCLAV	IO3	IN	237	TXDATA7	IO2	I/O
212	P2AD2	IO4	I/O	225	GND			238	TXCTRL1	IO3	OUT
213	TRI_	IN1	IN	226	TX_CLK	IO2	OUT	239	TXCTRL3	IO3	OUT
214	GND			227	TDO	IO2	OUT	240	VDD		

a. The RXCTRL signals use differing pad types due to their varying use in multi-PHY configurations. RXCTRL [3:0] are IO2, IO3, IO1, and IO3 respectively.

b. The TXCTRL signals use differing pad types due to their varying use in multi-PHY configurations. TXCTRL [3:0] are IO3, IO3, IO3, and IO2 respectively.

I/O pad reference

The table below cross-maps an I/O pin to the actual CMOS5S
I/O pad. SPICE models for these devices can be obtained by
contacting ***support@maker.com.***

TABLE 102. I/O pad types

TYPE	*PAD*	*DESCRIPTION*
IO1	BT520PU_A_G	5V Tolerant Bi-direct buffer, A-slew, 20 ohm, 3-state IO with pullup resistor.
IO2	BT520PU_B_G	5V Tolerant Bi-direct buffer, B-slew, 20 ohm, 3-state IO with pullup resistor.
IO3	BT520PD_B_G	5V Tolerant Bi-direct buffer, B-slew, 20 ohm, 3-state IO with pulldown resistor.
IO4	BT520PU_C_G	5V Tolerant Bi-direct buffer, C-slew, 20 ohm, 3-state IO with pullup resistor.
IO5	BT520PD_C_G	5V Tolerant Bi-direct buffer, C-slew, 20 ohm, 3-state IO with pulldown resistor.
IO6	BT520_C_G	5V Tolerant Bi-direct buffer, C-slew, 20 Ohm, 3 state IO.
IN1	IT5PUT_G	5V Tolerant LVTTL Input, with internal pull up.
IN2	IT5PDT_G	5V Tolerant LVTTL Input, with internal pull down.

CHAPTER 19 *Electrical Parameters*

This chapter provides information about the electrical parameters of the MXT3010EP. The following topics are included:

- MXT3010EP Operating conditions and maximum ratings

- MXT3010EP Power sequencing

- MXT3010EP Phase Lock Loop (PLL) implementation

MXT3010EP maximum ratings and operating conditions

TABLE 103. Absolute maximum ratings (VSS = 0V)

Symbol	Parameter	Min	Max	Units
VDD	3.3 volt supply	-0.3	7.0	V
VIN	Input voltage	-0.3	7.0	V
TA	Operating free-air temperature range	0	See Note 2	°C
TSTG	Storage temperature range	-65	150	°C

Notes 1: Stresses beyond the "Absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only. Operation at conditions beyond the indicated "Recommended operating conditions" is not recommended and may adversely affect device reliability.

2: Refer to *Application Note 27, MXT3010EP Thermal Test Report*

TABLE 104.Recommended operating conditions

Symbol	Parameter	Min	Max	Units
VDD	3.3 volt supply	3.14	3.47	V
VIH	High-level input voltage	2		V
VIL	Low-level input voltage		0.8	V
IOH	High-level output current	0	11.4^1	mA
IOL	Low-level output current		11.4^1	mA
TJ	Operating junction temperature	0	110^2	°C

Notes 1: See "Adjustments to Idc" on page 388.

2. Refer to *Application Note 27, MXT3010EP Thermal Test Report*

DC electrical characteristics

TABLE 105. DC Electrical characteristics

Symbol	Parameter	Min	Max	Units
ICC	3.3 volt supply current (100 MHz)		970	mA
VOH	VDD = min, IOH = max	2.4		V
VOL	VDD = min, IOH = max		0.4	V
CIO		Typ	7	pF
RIO	I/O Output Impedance (nominal)		20	Ohms
Pd	Power Dissipation @3.3/3.47V			
	@ 66 MHz		1.9/2.1	W
	@ 80 Mhz		2.3/2.6	W
	@ 100 Mhz		2.9/3.2	W

AC electrical characteristics

I/O performance levels

With the exception of the TDO scan output, all MXT3010EP outputs utilize either a medium or fast speed I/O pad. The table below summarizes the slew rate of each pad at nominal process, 25°C and 3.3V supply. For more accurate analysis, SPICE models of the I/O pads are available.

Performance Level	Slew Rate (di/dt)	Driver Speed
A	30 mA/ns	Slow
B	60 mA/ns	Medium
C	100 mA/ns	Fast

MXT3010EP power sequencing

Overview

The MXT3010EP uses a single voltage, +3.3 VDC ±5%. Therefore, there is no need to follow multiple voltage power sequencing rules. There are, however, two concerns regarding the application of power to CMOS devices such as the MXT3010EP. Both concerns relate to current flowing from an I/O pin into the chip's VDD rail when the I/O pin of the device is powered, and VDD to the device is not present.

- Damage to I/O pad metal

  When current flows into the I/O pad of the unpowered chip, the current flows from the I/O pad to the ESD diodes and from there to the VDD pad. This metal is rather thin and can be damaged from a high instantaneous inrush current. In this case, the metal connection fuses. Another way the metal can be damaged is through electromigration. When electromigration occurs, the metal erodes due to a moderate current flowing for an extended period of time.

- Latch-up

  The second major concern during power sequencing is a condition known as latch-up. Latch-up is a destructive event that can be induced in CMOS devices. The term refers to the 'turning on' of the parasitic PNPN structure that exists as a normal part of the CMOS gate structure. The PNPN structure is 'connected' between VDD and ground and has positive feedback. As this structure begins to conduct current from VDD to ground, it is turned on harder, presenting a lower impedance. The lower impedance causes more current to flow, and the process reinforces itself until the device overheats and is destroyed.

Current can flow from the I/O pin to the VDD rail through the I/O pad's ESD structure. The existence and magnitude of the current (Ipad) generally depends on the I/O type and the electro-static discharge (ESD) protection device it contains. The ESD structure is a set of series diodes from the I/O pin to VDD. In the case of the BT520 I/O pad used in the MXT3010EP, the ESD structure consists of a chain of 5 series diodes.

The two problems outlined above are analyzed in greater detail, with specific application to the MXT3010EP, in the sections which follow.

Damage to I/O pad metal

To determine whether damage to the I/O pad metal will occur from fusing or metal migration, one must first analyze how much current will flow into the pad. When powered down, the large capacitance associated with VDD (chip, package, card, other card components) could be modeled as a short circuit to ground (GND). Thus, the maximum current from a pad through the ESD diode(s) to VDD would be determined by the voltage and output resistance of the source supplying the pad voltage, the number and forward voltage drops of the ESD diodes, and the series resistance of the diodes and interconnects. The following equation applies:

$Ipad = (Vsource-(N*0.7)) / (Rsource+2)$

> N = the number of diodes (1,3,5) between the pad and VDD
>   **$Vsource$** = source voltage applied to the pad (volts)
>   **$Rsource$** = output resistance of the Vsource supply (ohms)
>   **$Ipad$** = current into the pad (amps)

There are two limits to the acceptable pad current (Ipad). Both are associated with current required to cause failures in the Metal-1 (M1) wiring connecting from the pad to the ESD device

and from the ESD device to VDD. The first limit is called Ifuse. Currents above Ifuse may immediately destroy the metal layer 1 connections. All MXT3010EP I/O pads are 5V tolerant and can withstand an Ifuse current of 129ma. The second current limit is Idc. Currents below Idc can be safely applied for extended periods of time without causing M1 electromigration (wear-out) failures.

Adjustments to Idc

Idc is a strong function of both temperature and the number of power-on hours (POH) over which the current is applied All MXT3010EP I/O pads are rated for 11.4ma under the conditions of 110K POH and 100°C. Idc may be adjusted for other conditions using the following multipliers:

$$Idc(POH) = Idc(table\ value) * (110000/POH)**0.588$$

$$Idc(temp) = Idc(table\ value) * exp((5459/(temp+273)-14.64))$$

Sample calculation

Assume all 5V tolerant inputs are being driven prior to the MXT3010EP's VDD rail being powered. Assume further that VDD on the hot-plugged ASIC part comes up 1 second after its I/O's pad receives the signal net voltage. Finally, assume that this scenario occurs 10,000 times over the life of the product, in a system running at 100°C. A further assumption is made the signal driving the I/O pad has a 5V nominal supply voltage and has a 20ohm nominal output impedance. Thus, Vsource = 2.5 volts and Rsource = 20 ohms. Since the BT520* I/O pad uses 5 series ESD diodes from a pad to VDD, N=5. Calculating Ipad yields:

$$Ipad = (Vsource-(N*0.7)) / (Rsource+2)$$
$$= (5-(5*0.7)/(20+2)$$
$$= 0.0681$$
$$= 68.1\ mA$$

The calculated value Ipad is well below the 129 mA limit for
Ifuse, so pad damage from fusing does not occur. However, the
calculated value of Ipad is well above the 11.4 mA limit for Idc,
so this amount of current cannot be applied indefinitely without
affecting reliability.

Since we assumed that the Ipad current was being applied for 2.7
hours (10000 times for 1 seconds/time = 10000 seconds = 166
minutes = 2.7 hours) over the life of a product, Idc should be
adjusted for time:

Idc(POH) = Idc(Pad) * (110000/POH)**0.588

   Idc(2.7 hours) = 11.4 mA * (110000/2.7)**0.588
            = 11.4 * 513.7
            = 5856 mA

This number is nearly two orders of magnitude above Ipad, so
we could stop here and conclude that the described application
will not effect reliability. Clearly, most cases of this type will be
limited by Ifuse well before Idc, but both Ifuse and Idc limits
should be checked.

I/O pad latch-up

I/O pad latch-up occurs when free charge in the semiconductor
substrate gets to the wrong place. Troublesome amounts of free
charge can be introduced by very large currents flowing through
the ESD diodes or other paths. Latchup is generally prevented
by isolating and protecting the parasitic PNPN structure from
collecting free charge.

It is difficult or impossible to induce latch-up in the device dur-
ing power up. As the device is conducting current from the I/O
pin to the VDD rail, there may be some free charge introduced
into the substrate. There is no power applied to the device at this
point, so latch up cannot occur.

As the device VDD is applied, the current flowing through the ESD diodes is sharply reduced. This is because the forward voltage drop of the ESD diodes is about 3.5 volts, so for an I/O pad voltage of 5V, the ESD current drops to zero when the chip's VDD reaches 1.5V. Before the VDD level of the chip attains sufficient voltage to sustain latch-up, the ESD current has been neutralized due to the forward drop of the 5 series connected ESD diodes.

Additionally, the CMOS device is generally designed to withstand several hundred milliamperes of ESD diode current without having latchup problems. This level of current is never attained during this power up situation, further reinforcing the latch up protection.

MXT3010EP PLL considerations

Overview

The MXT3010EP has an internal Phase Lock Loop (PLL) which it uses to generate the on-chip clock. This PLL allows the on-chip clock tree delay to be neutralized, and optimum performance of the IC to be obtained. The on-chip PLL can be affected by external circuit noise, so careful circuit design must be employed to optimize the performance of the PLL.

Degradation of the PLL performance manifests itself as jitter. This jitter is measured as the timing variation of the chip's internal clock to a stable reference clock supplied to the chip on the FN pin (pin 40). The internal clock cannot be observed directly, but any jitter on the internal clock will show up as jitter on the UTOPIA transmit clock, TX_CLK (pin 226). Jitter will cause a

variation in the timing of the chip relative to the board clock. The timing variation will affect setup and hold timing and erode timing margins at the chip interface.

The following sections cover circuit design issues which affect the operation of the PLL. Key areas of interest are de-coupling, creating a quiet PLL VDD, and ensuring a good PCB layout of the PLL area. The following sections also discuss the use of reference clocks, which may have jitter, and a method to bypass the internal PLL of the MXT3010 in special applications.

VAA decoupling

The PLL has a separate power pin labeled VAA (pin 37). This pin must be supplied with a very stable voltage level and should be well decoupled. The current draw of this pin is very low, 2.5 mA nominal. The low current draw allows the voltage to be isolated from the 3.3V power plane with a resistor. The use of a resistor instead of an inductor provides very good isolation from lower frequency noise such as power supply switching noise. A ferrite bead or inductor will not introduce a DC voltage drop, but it will also not filter low frequency noise. Due to the low current draw, use of a resistor is the recommended solution. The VAA pin should also be bypassed with a combination of a 10_μF tantalum cap and a 0.01_μF ceramic cap as shown in Figure 109 on page 392.

If the VAA pin is supplied voltage from a linear regulator, the designer must ensure that enough current is being drawn to keep the regulator in regulation. The output of a linear regulator is essentially noise free.

FIGURE 109.Generating a quiet VAA

27 ohm

VDD Plane ——————/\/\/\——————•————————•———————— VAA (Pin 37)

10 µF   .01 µF

Locate close to
pin 37 of the
MXT3010EP

General decoupling

The MXT3010EP must be properly decoupled to ensure clean
PLL operation. The PLL is most sensitive to noise on the VDD
supply. VDD noise contains both low frequency and high fre-
quency components. Power supply switching noise or insuffi-
cient bulk decoupling causes low frequency VDD noise. The
switching of the digital logic drivers causes high frequency
noise. Both of these noise sources must be taken into account to
ensure optimum performance.

The MXT3010EP has nineteen 3.3V supply pins. There should
be nineteen high frequency decoupling caps on the 3.3V supply
surrounding the chip. Additionally, there should be a minimum
20µF of bulk decoupling on the supply voltage (VDD) nearby to
the chip. This can be a single 22µF tantalum capacitor, or pref-
erably a pair of 10µF tantalum capacitors.

In a switching power supply environment, it is beneficial to fil-
ter the switching noise. This can be accomplished by filtering
the MXT3010EP's VDD with a ferrite bead. The ferrite bead
works in conjunction with the bulk decoupling capacitors to
effectively filter the power supply switching noise. The ferrite
bead must be sized to handle the current draw of the entire chip.
An appropriate part is the FairRite 2743021446 surface mount
ferrite bead.

FIGURE 110. MXT3010EP decoupling capacitor location

Figure 110 shows the optimal location of the decoupling capacitors around the MXT3010EP. This diagram depicts the location of 0805 size 0.01μF capacitors under the chip pin pads on the bottom side of the board. The capacitors are located close to the associated power pins. The capacitor should share a common via with the power pin of the chip with a minimum length etch. The same should be done with the ground connections.

Reference clock jitter

The PLL of the MXT3010EP locks the internal chip clock to the reference clock supplied to the device. The PLL will not necessarily be able to track jitter which is on the reference clock. If there is significant jitter on the reference, and the chip clock does not track it, the jitter will cause a reduction in timing margin at the chip interface.

Jitter on the reference clock can be caused by power supply noise affecting components of the clock generation and distribution circuit. One potential source of jitter is power supply noise or poor decoupling of crystal oscillators. Noise on the oscillator power pin, whether from the board or self-induced, can convert to timing jitter at the oscillator output. Some devices are better

than others in this aspect of operation. To reduce this noise source, ensure that the oscillator is well decoupled according to the manufacturer's specifications.

The distribution of the reference clock can also introduce clock jitter. Designs that use dividers in the reference clock path must avoid the possibility of simultaneous switching jitter, which can occur in synchronous counters. PLL clock buffers can also be a source of jitter, as these devices are generally susceptible to power supply noise, and can convert this noise to timing jitter.

Circuit design goals

It is desirable to keep VDD noise as low as possible. The PLL performance may start to degrade for high frequency noise greater than 40mV p-p and low frequency noise greater than 20mV p-p. The low frequency noise is defined as the noise below 20 MHz. The high frequency noise is defined as the full bandwidth noise measurement minus the low frequency noise. To ensure accuracy, measurements should be performed with a coaxial probe terminated in 50 ohms.

The PLL is sensitive to the frequency of the noise on VDD. The above guidelines may be conservative depending upon the application. Low frequency noise in the 100kHz to 500kHz range is the most critical.

The recommended VAA decoupling should guarantee less than 2mV p-p noise on the VAA voltage.

The jitter on the reference clock should be kept to less than 500pS peak to peak. The PLL is sensitive to the frequency of this jitter and may track or filter this jitter based on the jitter frequency and the PLL bandwidth. If the PLL does not track the jitter closely, the board level timing will be affected.

CHAPTER 20 *Mechanical and Thermal Information*

This chapter provides information on the MXT3010EP mechanical and thermal properties.

MXT3010EP mechanical/thermal information

The MXT3010EP is packaged in a 240-pin thermally enhanced quad flat-pack.

FIGURE 111.MXT3010EP package/pin diagram - top view

FIGURE 112. MXT3010EP package/pin diagram - side view

TABLE 106. MXT3010EP package summary

Package			θjc (×C/W)	θja (×C/W)	
Package Type	*Body size (mm)*	*Lead pitch (mm)*		**Still Air*	**Air Flow*
MHS PQFP 240	32.0 x 32.0 x 3.0	0.5	2.0	20	14

* These numbers will vary depending on the board stack-up and orientation. All airflow numbers are quoted with 1m/sec of air flow over the device.

The MXT3010EP is a level 3 IAW IPC-SM-786A or JESD 22-A112 device. The MXT3010's safe floor life (out of bag) prior to solder reflow is 1 week at ≤ 30°C/60% RH.

APPENDIX A *Acronyms*

Acronym	Definition
AAL	ATM Adaptation Layer
ABR	Available Bit Rate
ACR	Available Cell Rate
ATM	Asynchronous Transfer Mode
CAM	Content Addressable Memory
CBR	Constant Bit Rate
CDV	Cell Delay Variation
CDVT	Cell Delay Variation Tolerance
CI	Congestion Indicator
CLP	Cell Loss Priority
CPCS	Common Part Convergence Sublayer
CPI	Common Part Identifier
CRC	Cyclic Redundancy Check
CSS	Cell Scheduling System
DMA	Direct Memory Access
E1	European 2.048 Mbps rate TDM system
EFCI	Explicit Forward Congestion Indicator
ESS	External State Signals
FIFO	First In First Out
GCRA	Generic Cell Rate Algorithm
GFC	General Flow Control

Acronym	Definition
HEC	Header Error Control
ICS	Interchip Communication System
IFO	Instruction Field Option
JT2	96-channel TDM system used by Japan Telephone
MIB	Management Information Base
MVIP	Multi-Vendor Integration Protocol™
OAM	Operations and Management
PCR	Peak Cell Rate
PDU	Physical Data Unit
PHY	Physical Layer
PIT	Programmable Interval Timer
PTI	Payload Type Identifier
RAM	Random Access Memory
RM	Resource Management
RX	Receive
SAR	Segmentation and Reassembly
SCSA	Signal Computing System Architecture, ANSI standard
SDU	Service Data Unit
SHFM	Store Halfword to Fast Memory
SRAM	Static Random Access Memory
SRTS	Synchronous Residual Time Stamp
SWAN	Soft-Wired ATM Network
TDM	Time Division Multiplexing
T1	24-channel TDM system used in North America
TX	Transmit
UBR	Undefined Bit Rate
UDT	Unstructured Data Transfer
UU	User-to-User
VBR	Variable Bit Rate
VC	Virtual Channel
VCI	Virtual Channel Identifier
VP	Virtual Path
VPI	Virtual Path Identifier

APPENDIX B *Device Initialization*

This appendix describes the procedures for initializing and down-loading firmware to the MXT3010. The following information appears in this appendix:

- Initializing the MXT3010

- Downloading firmware

- Initializing the Mode Configuration register

Initializing the MXT3010EP

To initialize the MXT3010EP:

1. Assert RESET_ asynchronously to the input clock, FN.

2. Hold the RESET_ pin low for a period of time to allow the PLL to lock. For power-up, hold RESET_ as indicated in "Timing" on page 343. For reset during powered operation, hold RESET_ for 16 clock ticks.

3. Remove RESET_ synchronously with respect to the input clock. The reset state continues for 2056 clock ticks following the removal of RESET_. The MXT3010EP will not read or write the COMMIN/COMMOUT registers during this time, *nor will the CIN_BUSY or COUT_RDY flags function during this time.* Maker recommends that a host software timer be used between the removal of RESET_ and the beginning of boot download.

Downloading firmware

This section describes:

- How the system determines the boot path

- How the application program uses the output pins

- How the code set is structured

- How to boot the firmware

- Limitations on the size of boot code

How the system determines the boot path

Firmware structured as a single-user code set for the MXT3010 can be loaded through one of three paths:

- Through Port1, from a byte-wide device.

- Through Port2, from a byte-wide device.

- Through the COMMIN register.

The system signals the choice of boot path to the MXT3010 as the device exits reset mode. During reset, the MXT3010 places the ICSO_A and ICSO_B pins into tri-state mode. The MXT3010 senses the state of these pins as RESET_ is removed to determine the boot method. Each of these pins is pulled either high or low to signal the appropriate boot path to the MXT3010. During normal device operation, the ICSO_A and ICSO_B pins function as outputs.

TABLE 107. Selecting boot mode with ISCO_A and ICSO_B

ICSO_A	*ICSO_B*	*Boot MODE*
0	0	Reserved
0	1	Port1 Memory
1	0	Port2 Memory
1	1	COMMIN Register

How the application uses the output pins

After the MXT3010 initialization routine is completed, control of the device passes to the application program. The application program can then use the ICSO_A and ICSO_B pins by setting the appropriate bits in the system register.

How the code set is structured

The output of the SWAN Processor's assembler is one or more user-code sets. The user-code set includes four fields; see Table 108. The MXT3010 loads a single user-code set at device initialization. Support for loading multiple user-code sets comes from an intermediate boot loader routine.

TABLE 108. User code set's four fields

Field	Description	Size
1.	The starting word address (code address) in Fast Memory to which the user code set is to be stored.	2 bytes
2.	The number of half words in the user code set.	2 bytes
3.	The user code.	variable
4.	The checksum calculated by the host for code set containing all four fields.	2 bytes

As the code set is loaded, the MXT3010 computes a 16-bit checksum. This checksum is a running 16-bit sum of each half-word of the user code set. All carries are discarded and not used as part of the checksum routine. Upon completion of the transfer, the checksum is written to the COMMOUT register. The host must read the COMMOUT register to clear the COMMOUT Busy flag. The host can then compare the checksum to the checksum contained in the tail of the image block. The MXT3010 does not read the checksum field of the user-code set when loading a .ld file from Port1 or Port2 memory. The MXT3010 firmware *does* read the checksum field when downloading a .ubf file.

As the code set is loaded into Fast Memory, the MXT3010 stores the starting address location. At the completion of the loading operation, the MXT3010 branches to this location in Fast Memory to execute the program.

How to boot

This section describes how to boot from Port1, Port2, and the COMMIN register.

Booting from Port1

When the Port1 memory boot mode is selected, the MXT3010 reads the user-code set from a Port1-based memory device (RAM or ROM) beginning at location 0xFFF00000. To support byte-wide boot ROMs, the MXT3010 reads a single byte from each 32-bit word location. Therefore, for the host processor to copy the MXT3010's boot image into RAM located at 0xFFF00000, it must place a single byte of code into each 32-bit word location. Align bits (7:0) of the byte-wide boot device with PIAD (31:24) of the MXT3010.

Address	Byte 0	Byte 1	Byte 2	Byte 3
	31			0
0x0000		Not used	Not used	Not used
0x0004		Not used	Not used	Not used
0x0008		Not used	Not used	Not used
0x000C		Not used	Not used	Not used
0x00010		Not used	Not used	Not used
.		Not used	Not used	Not used
.		Not used	Not used	Not used
.		Not used	Not used	Not used

The MXT3010 issues single-word Port1 memory read operations until all of the fields shown in Table 108, except field #4, the checksum field, are read from Port1 memory. Once these fields are read and placed into the specified location in Fast Memory, the MXT3010 writes the result of its checksum operation into COMMOUT register[31:16], or the Port1 memory address location specified, and jumps to the first word of user code.

Booting from Port2

When the Port2 memory boot mode is selected, the MXT3010 reads the user-code set from a Port2-based memory device, such as a flash RAM or EEPROM device. The Port2 boot address of 0x0000 maps in non-burst space. Using this mapping, the MXT3010 can use a slow flash device as the initialization device. The MXT3010 inserts seven wait states for each Port2 read operation. To support byte-wide boot ROMs, the MXT3010 reads a single byte from each 16 bit word of Port2 memory.

Address	Byte 0	Byte 1
	15	0
0x0000		Not used
0x0002		Not used
0x0004		Not used
0x0006		Not used
0x0008		Not used
.		Not used
.		Not used
.		Not used

The MXT3010 issues Port2 memory read operations until all of the fields shown in Table 108, except field #4, the checksum field, are read from Port2 memory. Once these fields are read and placed into the specified location in Fast Memory, the MXT3010 writes the result of its checksum operation into the COMMOUT register[31:16], or the Port2 memory address location specified, and jumps to the first word of user code.

Booting from the COMMIN Register

A boot from the COMMIN register is performed 16 bits at a time. The host writes the block into the COMMIN register using COMM I/O. The first 16 bits are written from bits (31:16) of the first image block word, the second 16 bits are written from bits

(15:0) of the first image block word, and so on. The following diagram shows this process, arbitrarily assigning the letters A, B, C, and D to represent successive 16-bit quantities.

.

The host must write all of the fields shown in Table 108 on page 404, except field #4, the checksum field. Once these fields are read and placed into the specified location in Fast Memory, the MXT3010 writes the result of its checksum operation into the COMMOUT register[31:16], and jumps to the first word of user code.

Limitations on the size of boot code

Due to address calculation carry limitations, the MXT3010EP has restrictions on the maximum size of the code it can boot from the boot image. The restrictions depend upon the boot path used:

Boot Path	*Restriction*
Port1	4K instructions
Port2	512 instructions
COMMIN register	No restrictions

For those boot paths that have restrictions, specialized bootstrap code can be written. For example, using Port2, the code could load 512 instructions starting at a 512 word boundary. That code could include a secondary bootstrap program to perform address calculations and load the remainder of the application independent of code size restrictions.

Initializing the Mode Configuration register

The Mode Configuration Register(R42) is affected by three processes in the MXT3010. These processes proceed serially, starting at hardware reset.

1. Hardware reset

   Hardware reset initializes R42 to all zeros.

2. Operation at boot time

   The Mode Configuration register (R42) must be initialized at boot time since some system aspects of the MXT3010's operation are controlled by this register. At boot time as the executable image is loaded into the MXT3010, the first 32-bit word of the loaded image sets the least-significant eight mode bits in R42 with the indicated values. For an explanation of these bits, see "R42-write Mode Configuration register" on page 201.

   The format of this boot word is a bit-mapped 8-bit field. The state of each of the relevant bits indicates the desired state of the associated mode bit. Although writes to R42 can only set or clear one bit at a time, the boot word affects the state of bits [7:0] simultaneously. The micro-boot sequence parses the boot word and creates the individual R42 writes needed to affect each bit.

   The syntax for the assembler is:

```
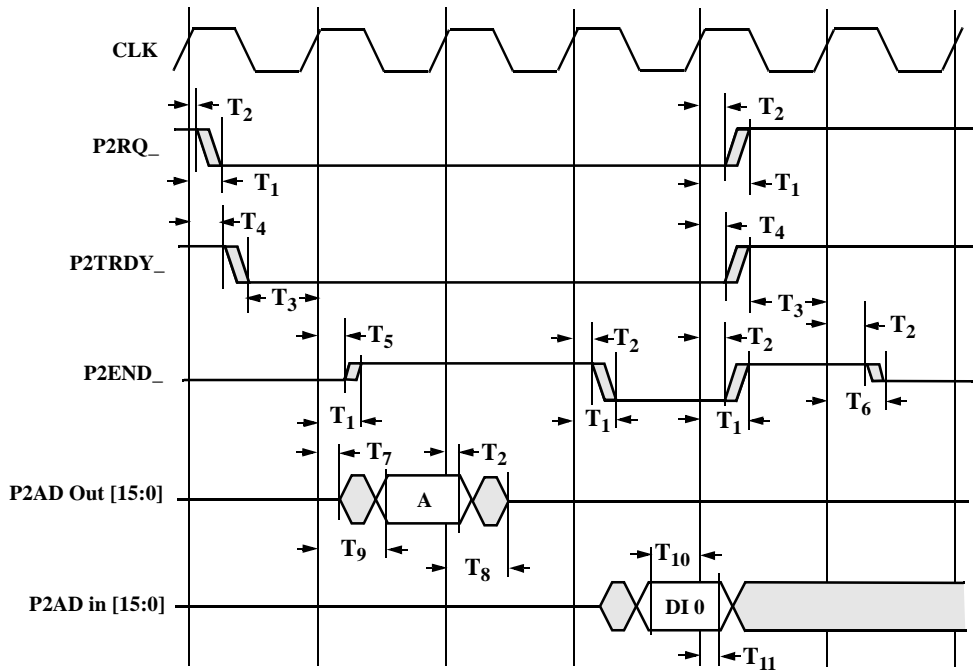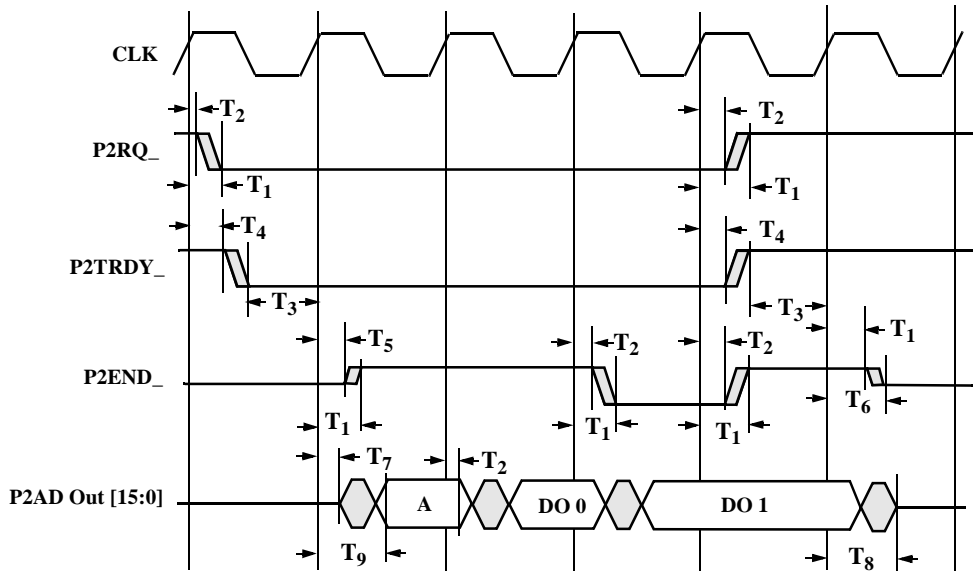#define #boot_value 0x000000zz;'zz is the
                                 ;desired value

....

LIMD    R36, #boot_value      ;'zz programmed
                              ;to R42
                              ;automatically by
                              ;microboot

....
```

The LIMD instruction must be the first instruction in the executable image. The register used for this operation must be the bit bucket (R36).

3. Firmware changes to register value

Although this register is automatically initialized, firmware can still change the value of this register through the set/clear operations.

Restrictions on starting addresses

All systems operating in Fast Memory mode 1 have restrictions on the values that can be used for bootstrap starting addresses. Systems operating in Fast Memory mode 1 should use starting addresses from the following table:

TABLE 109. Bootstrap starting addresses for Fast Memory mode 1

MXT3010EP - modulo 32K
0x00000
0x08000
0x10000

For applications that must run at a different starting address, these restrictions can be avoided by using a secondary bootstrap program.

APPENDIX C *Quick Reference*

This appendix contains duplicate copies of useful charts which appear elsewhere in this book, plus a summary of the MXT3010 SWAN processor instruction set.

Hardware register summary

TABLE 110. Hardware registers

Location	Name	Read/Write
R32	General Purpose - 0000	R/W
R33	General Purpose - FFFF	R/W
R34	General Purpose - FF00	R/W
R35	General Purpose - 0040	R/W
R36-Write	The Bit Bucket	W
R37	General Purpose	R/W
R38	General Purpose	R/W
R39	General Purpose	R/W
R40	COMMOUT/COMMIN(31:16)	R/W
R41	COMMOUT/COMMIN(15:0)	R/W
R42-Read	ESS register	R
R42-Write	Mode Configuration register	Set/Clear
R43-Read	Fast Memory Bit Swap register	R
R43-Write	UTOPIA TX Control FIFO register	W
R44	CRC32PRX (15:0)	R/W
R45	CRC32PRX (31:16)	R/W
R46	CRC32PRY (15:0)	R/W
R47	CRC32PRY (31:16)	R/W
R48	rla Address register	R/W
R49	rla Address register	R/W
R50	rla Address register	R/W
R51	rla Address register	R/W
R52	Alternate Byte Count /ID register	R/W
R53	Instruction Base Address register	R/W
R54	Programmable Interval Timer (PIT0)	R/W
R55	Programmable Interval Timer (PIT1)	R/W
R56	The Fast Memory Data register	R/W
R57-Read	Sparse Event/ICS register	R
R57-Write	Sparse Event/ICS register	Set/Clear
R58	Fast Memory Shadow register	R/W
R59	Branch register	R/W
R60	CSS Configuration register	R/W
R61-Read	Scheduled Address register	R
R62	UTOPIA Configuration register	R/W
R63	System register	R/W

ALU instruction field summary

TABLE 111. MODx fields

Value	Modulo Operation	Value	Modulo Operation
0000	MOD2	1000	MOD512
0001	MOD4	1001	MOD1K
0010	MOD8	1010	MOD2K
0011	MOD16	1011	MOD4K
0100	MOD32	1100	MOD8K
0101	MOD64	1101	MOD16K
0110	MOD128	1110	MOD32K
0111	MOD256	1111	Default

TABLE 112. abc fields

Value	Branch Code	Description
000	default	No branch
001	BGEZ	Branch if greater than or equal to zero
010	-----	------
011	BZ	Branch if zero
100	BLEZ	Branch if less than or equal to zero
101	BLZ	Branch if less than zero
110	BNZ	Branch if not zero
111	BNO	Branch if no overflow

TABLE 113. AE field

Value	Action
0	Conditional execution
1	Always execute target instructions

TABLE 114. UM field

Value	Action
0	Don't update memory
1	Update memory

Shift amount summary

TABLE 115. Shift amount chart for SFT, SFTLI, and SFTRI

SFT/SFTLI (4:0)	Shift Left by	SFT/SFTRI (4:0)	Shift Right by
00000	0	10000	16
00001	1	10001	15
00010	2	10010	14
00011	3	10011	13
00100	4	10100	12
00101	5	10101	11
00110	6	10110	10
00111	7	10111	9
01000	8	11000	8
01001	9	11001	7
01010	10	11010	6
01011	11	11011	5
01100	12	11100	4
01101	13	11101	3
01110	14	11110	2
01111	15	11111	1

TABLE 116. Shift amount chart for SFTC and SFTCI

(3:0)	Shift left circular by	(3:0)	Shift left circular by
0000	0	1000	8
0001	1	1001	9
0010	2	1010	10
0011	3	1011	11
0100	4	1100	12
0101	5	1101	13
0110	6	1110	14
0111	7	1111	15

TABLE 117. Shift amount chart for SFTA

(4:0)	Shift right arithmetic by	(4:0)	Shift right arithmetic by
00000	0	10000	16
00001	1	10001	15
00010	2	10010	14
00011	3	10011	13
00100	4	10100	12
00101	5	10101	11
00110	6	10110	10
00111	7	10111	9
01000	8	11000	8
01001	9	11001	7
01010	10	11010	6
01011	11	11011	5
01100	12	11100	4
01101	13	11101	3
01110	14	11110	2
01111	15	11111	1

TABLE 118. Shift amount chart for SFTAI

(3:0)	Shift right arithmetic by	(3:0)	Shift right arithmetic by
0000	0	1000	8
0001	1	1001	9
0010	2	1010	10
0011	3	1011	11
0100	4	1100	12
0101	5	1101	13
0110	6	1110	14
0111	7	1111	15

Branch instruction field summary

TABLE 119. The ESS field (condition codes)

ESS	Condition	ESS	Condition
ESS0	ICSI_A	ESS8	Sparse event register, bit OR
ESS1	ICSI_B	ESS9	RXBUSY counter > 0
ESS2	TXFULL counter ≤ 2	ESS10	TXFULL counter = full
ESS3	RXBUSY counter ≥ 4	ESS11	DMA1 Output or Queue stage busy
ESS4	Assigned Cell Flag	ESS12	DMA2 Output or Queue stage busy
ESS5	CSS operation in progress	ESS13	DMA1 Queue stage busy
ESS6	COMMIN_BSY	ESS14	DMA2 Queue stage busy
ESS7	COMMOUT_BSY	blank	Unconditional Branch

TABLE 120. The S-bit field

S	Branch Result
0	Branch is taken if condition = 0
1	Branch is taken if condition = 1

TABLE 121. The C-bit field

Type of Branch	Condition Code Satisfied?	Conditional Operator (C-bit)	Never Execute Operator	Committed Slot Instruction Executed?
Conditional	Yes	Note 1	Note 2	Yes
Conditional	No	Absent	Note 2	Yes
Conditional	No	Present	Note 2	No
Unconditional		Note 1	Absent	Yes
Unconditional		Note 1	Present	No

TABLE 122. The CSO field

CSO	Hex / Binary Value	Operation
DRXBUSY	E0 / 1110 0000	Decrement RXBUSY counter
DRXFULL	E1 / 1110 0001	Decrement RXFULL counter
ITXBUSY	C2 / 1100 0010	Increment TXBUSY counter
ITXFULL	C3 / 1100 0011	Increment TXFULL counter

DMA instruction field summary

TABLE 123. Use of the I-bit

Bits [26]	Description
0	Do not increment the rla register
1	Increment rla register upon completion of DMA operation

TABLE 124. Use of the BC field

DMA Instructions Bits [26:19]	DMA+ Instructions Bits [25:19]	Description[a]
0	0	Transfer 0 Bytes.
2	2	Transfer 2 bytes
4	4	Transfer 4 bytes
6	6	Transfer 6 bytes
-	-	-
126	126	Transfer 126 bytes
128	Not Available	Transfer 128 bytes
-	Not Available	-
254	Not Available	Transfer 254 bytes

a. See "Use of odd BC values" on page 287.

TABLE 125. Use of the Control byte

Bit	Name	Function
9	IBI	Internal flag. Not used by programmers.
8	CRCX	CRC32 Partial Result is generated based on CRC32PRX register's value and the result is deposited into CRC32PRX (R44/R45).
7	CRCY	If set, a CRC32 Partial Result is generated based on CRC32PRY register's value and the result is deposited into CRC32PRY (R46/R47)
6	POD	If set, TXBUSY is incremented upon the completion of DMA reads, and RXFULL is decremented upon completion of DMA writes.
5	ST	If set, a "Silent Transfer" is performed.

Instruction summary

Instruction	*Function & Format*	*Pg.*
DMA1R+	Direct memory operation - Port 1 read DMA1R+ rsa/rsb, rla [BC/#][CRC {X,Y}][POD] [ST]	page 289
DMA1W	Direct memory operation - Port 1 write DMA1W rsa/rsb, rla [BC/#][CRC {X,Y}][POD] [ST]	page 290
DMA1W+	Direct memory operation - Port 1write DMA1W+ rsa/rsb, rla [BC/#][CRC {X,Y}][POD] [ST]	page 290
DMA2R	Direct memory operation - Port 2 read DMA2R rsa/rsb, rla [BC/#][POD]	page 291
DMA2R+	Direct memory operation - Port 2 read DMA2R+ rsa/rsb, rla [BC/#][POD]	page 291
DMA2W	Direct memory operation - Port 2 write DMA2W rsa/rsb, rla [BC/#][POD]	page 292
DMA2W+	Direct memory operation - Port 2write DMA2W+ rsa/rsb, rla [BC/#][POD]	page 292
FLS	Find last set FLS (rsa,rsb) rd [abc][AE][UM]	page 242
LD	Load register LD rd @rla [IDX/#]	page 321
LDD	Load double register LDD rd @rla [IDX/#]	page 322
LIMD	Load immediate LIMD rd,li [UM]	page 243
LMFM	Load multiple from Fast Memory LMFM rd @rsa/rsb #HW [LNK]	page 308
MAX	Maximum of two registers MAX (rsa,rsb) rd [MODx][abc][AE][UM]	page 244
MAXI	Maximum of register and intermediate MAXI (rsa,si) rd [abc][UM]	page 245
MIN	Minimum of two registers MIN (rsa,rsb) rd [MODx][abc][AE][UM]	page 246
MINI	Minimum of register and intermediate MINI (rsa,si) rd [abc][UM]	page 247

Instruction	Function & Format	Pg.
OR	OR registers OR (rsa,rsb) rd [MODx][abc][AE][UM]	page 248
ORI	OR register and immediate ORI (rsa,si) rd [abc][UM]	page 249
POPC	Service schedule POPC rd@rsb	page 278
POPF	POP fast POPC rd@rsb	page 279
PUSHC	Schedule PUSHC rsa@rsb	page 280
PUSHF	PUSH Fast PUSHF rsa@rsb	page 281
SFT	Shift signed amount SFT (rsa,rsb) rd [MODx][abc][UM]	page 250
SFTA	Shift right arithmetic SFTA (rsa,rsb) rd [MODx][abc][UM]	page 251
SFTAI	Shift right arithmetic immediate SFTAI (rsa,usa) rd [MODx][abc][UM]	page 252
SFTC	Shift left circular SFTC (rsa,rsb) rd [MODx][abc][UM]	page 253
SFTCI	Shift circular immediate SFTCI (rsa,usa) rd [MODx][abc][UM]	page 254
SFTLI	Shift left immediate SFTLI (rsa,usa) rd [MODx][abc][UM]	page 255
SFTRI	Shift right immediate SFTRI (rsa,usa) rd [MODx][abc][UM]	page 255
SHFM	Store halfword to Fast Memory SHFM @rsa/rsb	page 311
SRH	Store register halfword SRH @rsa/rsb [adr][reg][lsbs]	page 312
ST	Store register ST rsa @rla [IDX/#]	page 323
STD	Store double register STD rsa/rsb @rla [IDX/#]	page 324

Instruction	Function & Format	Pg.
SUB	Subtract registers SUB (rsa,rsb) rd [MODx][abc][AE][UM]	page 256
SUBI	Subtract register and intermediate SUBI (rsa,usi) rd [MODx][abc][UM]	page 257
XOR	XOR registers XOR (rsa,rsb) rd [MODx][abc][AE][UM]	page 258
XORI	XOR register and intermediate XORI (rsa,usi) rd [abc][UM]	page 259

Index

RLA Increment bit (i-bit) 285
RLA increment option 107
RLA register
    Choices for rla register 208, 314
RXBUSY counter 79, 200
RXFULL
    Counter 81
    Decrementing 109
    State indicator 213

S
S bit 264
SAR PDU 62
Scheduled Address register (R61) 33, 218
Scoreboard 28
    Address bits 44
    Address generation 44
    Initialization 45
    Section size selection 217
    Sections 46
    Size 45
Scoreboard/Cell Buffer selection 208
Segment ID 262
Segment ID bits 210
Sequence diagrams
    CIN_BUSY and COUT_RDY 179
    Comm I/O transfer 133
    Fast Memory 56
    Port2 144, 147, 151, 155
    UTOPIA Port 94
SHFM instruction 50
SHFM instruction, use of R56 212
Signal descriptions 369
    Clock, control, and test signals 375
    Fast Memory controller 373
    Inter-chip and communication register 374
    Port1 370
    Port2 371
    Power and ground pins 376
    UTOPIA Port 372
Signed arithmetic 225
Silent transfers 105, 288
Sparse Event register bit OR 200
Sparse Event register enables 221
Sparse Event/ICS register (R57) 180, 213

SPICE models 381
SRH instruction 50
ST bit 288
ST option 105
Stalls
    Load Double Register 322
    Load Register 321
    Store Double Register 324
    Store Register 323
    With LMFM instruction 309
    With SHFM instruction 311
Subroutine linking 268
Sustained Cell Rate 35
SWAN instruction set 185
Swap field 319
System register (R63) 91, 221

T
Target address
    Branch instructions 262
    Cell scheduling 277
    Load and Store Internal RAM 315
Target field 263
Timing 343
    Control signals 359
    Definition of switching levels 343
    Fast Memory interface 345
    Port1 352
    Port2 356
    Reset 360
    UTOPIA interface 348
Timing restrictions
    LMFM instruction 309
Transfer complete
    Byte count zero
            Early end 162
            Standard end 161
    External abort (P1ABORT_) 163
Transmit Cell Buffer Size 220
TXBUSY
    Incrementing 109, 206
    State indicator 213
TXFULL Counter 200