Innovasic® Semiconductor
Extended Life Semiconductor Solutions

The fido1100® User Guide for the 32-Bit Real-Time Communications Controller

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CONVENTIONS

Arial Bold	Designates headings, figure captions, and table captions.
Blue	Designates hyperlinks (PDF copy only).
`Courier`	Designates code text.
Italics	Designates emphasis or caution related to nearby information. Italics is also used to designate variables, refer to related documents, and to differentiate terms from other common words (e.g., "If the *set* parameter is 0, then they will be cleared.").

NOMENCLATURE

ADC	Analog-to-Digital Converter.
CMU	Context Management Unit—Controls context switching based on context priorities.
DMA	Direct Memory Access—Independent means of transferring data without CPU intervention.
ISR	Interrupt Service Routine—Code that will be executed when an exception or interrupt is processed. Address of ISR is obtained from exception vector table for the executing context.
MPU	Memory Protection Unit—Provides access control to blocks of memory on a context basis.
PMU	Peripheral Management Unit—The fido1100 subsystem containing transmit and receive frame buffers.
RREM	Relocatable Rapid Execution Memory—Configurable internal instruction memory used to speed up execution of critical sections of applications.
SDRAM	Synchronous Dynamic RAM
SPIDER	Software Profiling and Integrated Debug Environment—The fido1100 hardware breakpoints, watchpoints, and trace features.
TCU	Timer Counter Unit.
UIC	Universal I/O Controller—Firmware controlled communication protocol engines.
URAM	User RAM—Internal 24 Kbyte block of RAM available for use by applications.
WDT	Watchdog Timer—Timer used to prevent runaway execution.

SAFETY INFORMATION

DANGER Certain applications using semiconductor products may involve potential risks of death, personal injury, or environmental damage. To minimize the risks associated with end-user applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

WARNING Innovasic Semiconductor, Inc. (Innovasic), products are not designed, warranted, or authorized for use in life-support devices or systems or in other critical applications. The inclusion of Innovasic products in such applications is fully at the customer’s risk.

WARNING Be aware of all hazards involved in handling electrical circuitry and be familiar with practices for preventing accidents that may cause personal injury or death.

CAUTION Electrostatic discharge (ESD) can destroy or damage integrated circuits and semiconductor devices. When working near or handling these components, ensure that proper ESD suppression measures are taken. Additionally, do not store or ship these components near strong electrostatic, electromagnetic, magnetic, or radioactive fields.

PREFACE

This document describes the capabilities, operation, and programming of the fido1100™ communications controller. The organization and content of this document are as follows:

- Chapter 3, Context Architecture—An introduction to the fido1100 multiple hardware contexts concept.
- Chapter 4, Core CPU—Covers the Execution Unit, register model and data types, exception processing, and details of the multiple hardware contexts operation.
- Chapter 5, Memory Management and Protection—Description of the Memory Protection Unit features of the fido1100 and the Relocatable Rapid Execution Memory. Introduces the programmable Chip Select registers. Also includes a Register Map quick reference of the fido1100, sorted by address.
- Chapter 6, External Bus Interface—Details of the programmable chip-select registers and the SDRAM Controller and its registers. Covers how external bus arbitration can be used to connect the fido1100 on a shared address/data bus.
- Chapter 7, Peripheral Management Unit—Details the operation and registers of the PMU subsystem in the fido1100. Covers the transmit and receive buffer functions of the PMU. Describes how to load a UIC with firmware. Details the two channels of DMA operation and how to use the DMA in conjunction with the PMU to offload the CPU for data transmit and receive functions. The MAC Filter section details the usage of the programmable MAC address filter table.
- Chapter 8, Internal Peripherals—Details the System Timer, Timer Counter Units, Watchdog Timer and Analog-to-Digital Converter in the fido1100.
- Chapter 9, Debug Features—Covers the debug, trace and breakpoint features of the fido1100. The JTAG debug interface, hardware breakpoints and watchpoints, and tracing are illustrated by examples.
- Chapter 10, Power Control—Describes how to control the power to the fido1100 internal peripherals and the LPSTOP and STOP instructions.
- Chapter 11, Access Controlled Registers—Details the fido1100 mechanism for limiting access to registers or bit fields in register to the Master Context only. Access control is used to protect sensitive fields or settings controlled by registers.
- Chapter 12, Register Map Reference—Presents all the fido1100 registers organized by function peripheral with descriptions of the bit fields in the registers.

1. Introduction to fido1100 Architecture

Innovasic Semiconductor's fido1100 is the first product in the fido™ family of real-time communications controllers. The fido communications controller architecture is uniquely optimized for solving memory bottlenecks, and is designed from the ground up for deterministic processing. Critical timing parameters, such as context switching and interrupt latency, are precisely predictable for real-time tasks. The fido1100 also incorporates the Universal I/O Controller (UIC™) that is configurable to support various communication protocols across multiple platforms. This flexibility relieves the designer of the task of searching product matrices to find the set of peripherals that most closely match the system interface needs. The Software Profiling and Integrated Debug EnviRonment (SPIDER™) has extensive real-time code debug capabilities without the burden of code instrumentation.

1.1 Features

The fido1100 communications controller's features include:

- A real-time 32-bit microcontroller
- CISC architecture optimized for real time
- CPU32+ (Motorola® 68000) instruction-set compatible
- Five hardware contexts, each with its own register set and interrupt vector table
- An 8- or 16-bit external bus interface with programmable chip selects
- 24 Kbytes of high-speed internal user SRAM
- 32 Kbytes of high-speed internal user-mappable Relocatable Rapid Execution Memory (RREM)
- A Memory Protection Unit
- An SDRAM controller
- Flat, contiguous memory space
- A non-aligned memory access
- A dedicated Peripheral Management Unit (PMU)
- Four Universal I/O Controllers (UICs) capable of supporting the following protocols:
  - GPIO
  - 10/100 Ethernet with flexible MAC Address Filtering schemes
  - EIA-232
  - CAN

- SPI
- I^2C™ Bus
- SMBus
- HDLC

- Two channels of full-featured direct memory access (DMA) with deterministic arbitration
- Two Timer/Counter Units (TCU)
- A Watchdog timer, system timer, and context timers
- JTAG emulation and debug interface
- Standard 208 PQFP, TQFP, and BGA packaging
- 3.3V operation with 5V-tolerant I/O
- Industrial temperature grade
- Software development supported by libraries and tools including UIC firmware for various interface protocols and formats, as well as a customized GNU tool set.

1.2 Architectural Overview

A diagram of the fido1100 communications controller is shown in Figure 1-1. Brief descriptions of the major architectural components are provided in the following subsections.

1.2.1 Core CPU

The fido1100 includes a 32-bit CPU featuring Complex-Instruction-Set-Computer (CISC) architecture optimized for real time. The Motorola®-CPU32-instruction-set compatible CPU contains the following subsystems:

- Execution Unit—A single unit that handles all instruction fetch and execution for all contexts.
- Context Management Unit—Handles context priorities and when a context switch is required.
- Exception Handling (Interrupts and Faults)—Determines priorities of interrupts and handles processing of interrupts (detailed in later sections).

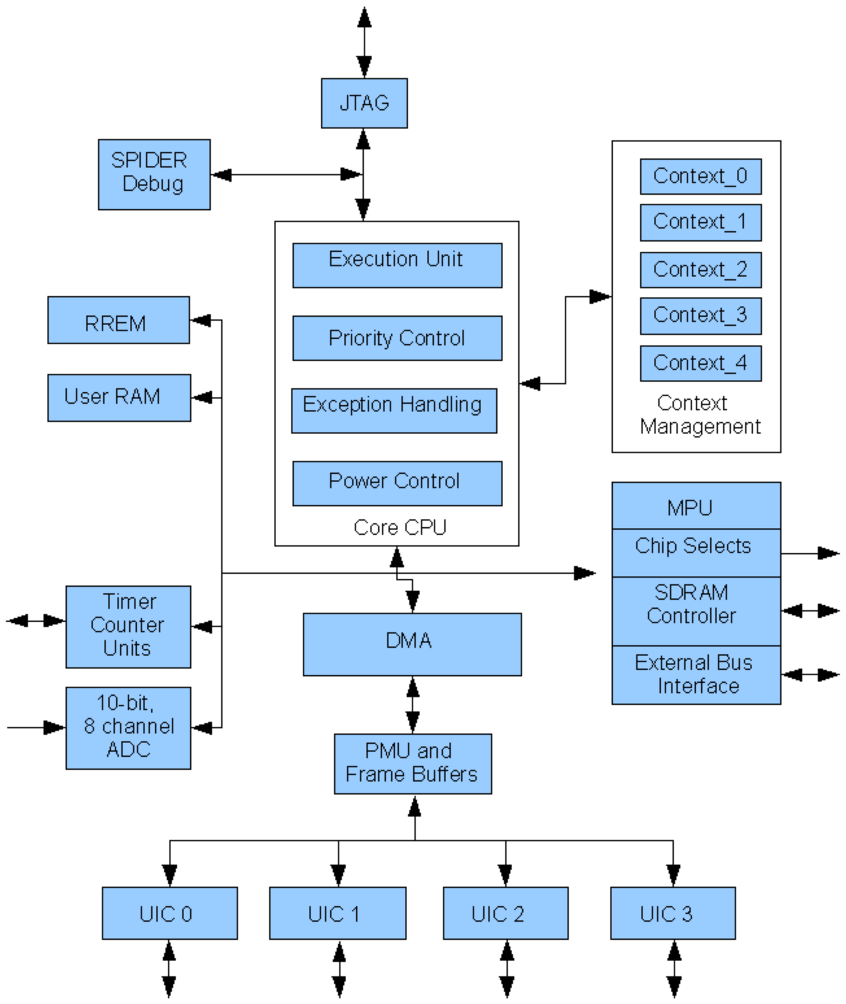


Figure 1-1. Diagram of the fido1100 Communications Controller

1.2.2 Memory Management

- Relocatable Rapid Execution Memory (RREM)—Internal 32-Kbyte memory that can be used as an instruction source for code that requires maximum execution speed.

- Memory Protection Unit (MPU)—Access-control method for 16 user-configurable blocks of internal or external memory on a context basis. A block of memory may be inaccessible, read only or read/write accessible to a selectable set of contexts. The MPU provides the space partitioning needed in deterministic, real-time systems.

1.2.3 External Bus Interface

The interface to all external memory. It handles memory interface timing and arbitration of external bus requests. The external bus interfaces provide all address, data, and control line to implement either an 8- or 16-bit microcontroller system bus. Additional details on the characteristics, operation, and timing of the external bus interfaces are provided in Chapter 6, External Bus Interface.

- Address/data bus
  - 31-bit address bus to access up to 2 Gbytes of memory space
  - 8- or 16-bit data bus
- Chip Selects—Eight programmable chip selects with programmable size, data width, and timing.
- SDRAM Controller—Supports 8- or 16-bit data interfaces to SDRAM.
- External Bus Arbitration—The fido1100 provides signals to allow it to operate in a multi-bus master environment.

1.2.4 PMU/UIC/CPU DMA

The PMU, UIC, and CPU DMA work together as a fast data transport scheme that requires minimal Core CPU overhead or intervention.

- Peripheral Management Unit (PMU)—A set of user-configurable buffers for data transmission and reception via the UICs.
- Universal Input/Output Controller (UIC)—Programmable protocol engine
- CPU DMA—Two independent channels of DMA for data transfer

1.2.5 Internal Peripherals

The fido1100 incorporates the following set of internal peripherals:

- Two Timer Counter Units (TCU)—Additional details on the characteristics, operation, and timing are provided in Section 8.3, Timer Counter Units.
- Analog-to-Digital Converter (ADC)—The characteristics, operation, and timing of the 10-bit, 8-channel ADC are provided in Section 8.4, Analog-to Digital Converter.
- Power Control—Internal peripherals can be put into a low-power consumption mode. Detailed information can be found in Chapter 10, Power Control.

1.2.6 JTAG/Debug

The JTAG Interface is used for controlling the SPIDER (see Chapter 9, Debug Features).

- Breakpoints—Eight hardware context-aware breakpoints that can be chained to set up if/then triggering conditions.
- Watchpoints—Eight hardware watchpoints.
- Trace—Follow program execution with trace buffers.
- Debug Control—Hardware single-step and context status control.
- Statistical Profiling—SPIDER provides statistical software profiling to identify critical pieces of code.

1.3 Programming Model

1.3.1 CPU32 Instruction Set Compatible

The fido1100 supports the CPU32 instruction set with some modifications, and has some new instructions (see Chapter 4, Core CPU, and *The fido1100 Instruction Set Reference Guide* for details). The fido1100 general instruction classes include machine functions for all of the following operations:

- Data movement
- Arithmetic operations
- Logical operations
- Shifts and rotates
- Bit manipulation
- Binary-Coded Decimal (BCD) arithmetic
- Program control
- System control
- Power control

1.3.2 Memory-Mapped Address Space

- Address and data space (31-bit address space and 32-bit data paths internally, 16-bit data path externally)

- Supports byte (8-bit), word (16-bit), long-word (32-bit), and quad-word (64-bit) operations
- Supports non-aligned accesses to memory
- Selectable little-endian or big-endian accesses to memory

2. Programmer Reference Overview

2.1 User Guide Structure

This User Guide presents procedures for designing and programming a system using the fido1100 communications controller. Detailed information about the fido1100 CPU, internal peripherals, context management, and exception processing is provided.

The guide is divided into chapters that present the following topics:

- Core CPU (Chapter 3 and Chapter 4)
  - Context Management
  - Execution Unit
  - Exception Handling (Interrupts and Faults)

- Memory Management (Chapter 5)
  - RREM (Relocatable Rapid Execution Memory)
  - MPU (Memory Protection Unit)

- External Bus Interface (Chapter 6)
  - Address/Data bus
  - Programmable Chip Selects
  - SDRAM Controller
  - External Bus Arbitration

- PMU/UIC/CPU DMA (Chapter 7)
  - PMU (Peripheral Management Unit)
  - UIC (Universal I/O Controller)
  - CPU DMA
  - MAC Filter

- Internal Peripherals (Chapter 8 and Chapter 10)
  - TCU (Timer Counter Units)
  - ADC (Analog-to-Digital Converter)
  - Power Control

- JTAG/Debug (Chapter 9)
  - Breakpoints
  - Watchpoints
  - Trace
  - Debug Control/Status Registers

2.2 Chapter Overview

- Chapter 3, Context Architecture—An introduction to the fido1100 multiple hardware contexts concept.
- Chapter 4, Core CPU—Covers the Execution Unit, register model and data types, exception processing, and details of the multiple hardware contexts operation.
- Chapter 5, Memory Management and Protection—Description of the Memory Protection Unit features of the fido1100 and the Relocatable Rapid Execution Memory. Introduces the programmable Chip Select registers. Also includes a Register Map quick reference of the fido1100, sorted by address.
- Chapter 6, External Bus Interface—Details of the programmable chip-select registers and the SDRAM Controller and its registers. Covers how external bus arbitration can be used to connect the fido1100 on a shared address/data bus.
- Chapter 7, Peripheral Management Unit—Details the operation and registers of the PMU subsystem in the fido1100. Covers the transmit and receive buffer functions of the PMU. Describes how to load a UIC with firmware. Details the two channels of DMA operation and how to use the DMA in conjunction with the PMU to offload the CPU for data transmit and receive functions. The MAC Filter section details the usage of the programmable MAC address filter table.
- Chapter 8, Internal Peripherals—Details the System Timer, Timer Counter Units, Watchdog Timer and Analog-to-Digital Converter in the fido1100.
- Chapter 9, Debug Features—Covers the debug, trace and breakpoint features of the fido1100. The JTAG debug interface, hardware breakpoints and watchpoints, and tracing are illustrated by examples.
- Chapter 10, Power Control—Describes how to control the power to the fido1100 internal peripherals and the LPSTOP and STOP instructions.
- Chapter 11, Access Controlled Registers—Details the fido1100 mechanism for limiting access to registers or bit fields in register to the Master Context only. Access control is used to protect sensitive fields or settings controlled by registers.
- Chapter 12, Register Map Reference—Presents all the fido1100 registers organized by function peripheral with descriptions of the bit fields in the registers.

3. Context Architecture

3.1 Introduction

One way to gain a better understanding of the fido context architecture is to view it from the perspective of a standard microprocessor— for example, a CPU32. A standard CPU32 can be viewed as a single hardware context, with a standard set of registers. Interrupts and other peripheral inputs are handled and processed as required by the microprocessor execution engine. Whenever an interrupt occurs, priority/masking willing, the currently executing task is stopped, stack operations occur, the interrupt is processed as required by the handler and the task execution resumes.

In a multi-task software application, any task switch will have the overhead of caching and de-caching registers for tasks to be saved/resumed. A faulting task will typically be suspended by an OS (if one exists) or can even halt a system, requiring a reset or restart to recover the system (see Figure 3-1).

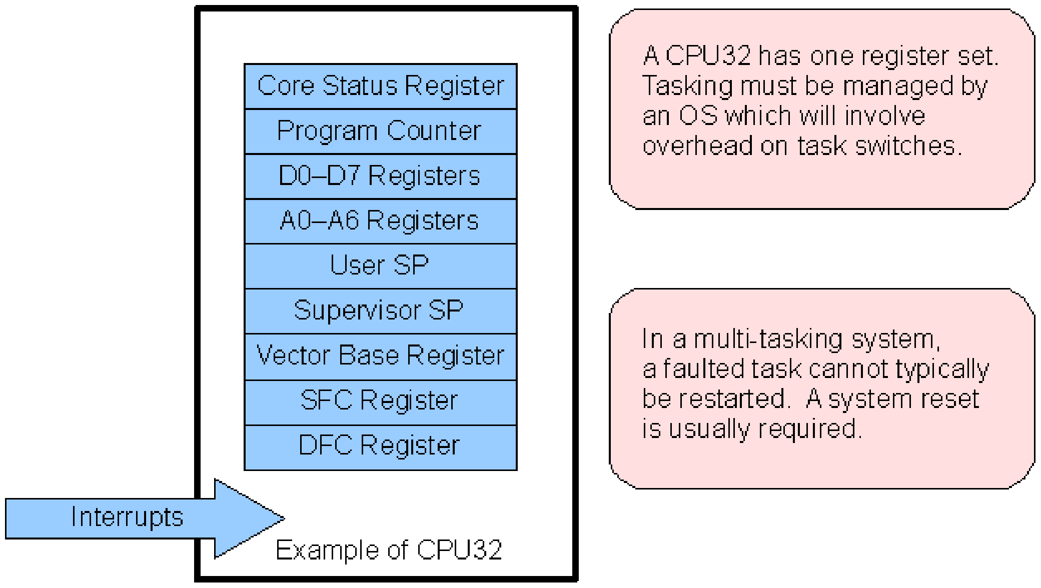

Core Status Register
Program Counter
D0–D7 Registers
A0–A6 Registers
User SP
Supervisor SP
Vector Base Register
SFC Register
DFC Register
Interrupts
Example of CPU32
A CPU32 has one register set. Tasking must be managed by an OS which will involve overhead on task switches.
In a multi-tasking system, a faulted task cannot typically be restarted. A system reset is usually required.

Figure 3-1. Standard Microprocessor, Single Context View

A discussion of the fido1100 architecture implementation follows.

The fido1100 implements five independent hardware contexts. Each has its own stack, register set, and additional registers for context control and operation. Each interrupt in the system is “associated” with a given context (details are provided in later chapters). Each context and each interrupt has a priority mechanism used by the hardware execution unit to facilitate operation.

With fido, a true multi-tasking system is possible with or without an OS, and task-switching times are greatly reduced and can be "deterministic" based on context type (see Section 4.8, Context Management). No caching of registers to memory is ever needed or used to switch contexts.

In fido, a faulting event is isolated to a given context, protecting the remaining system that can continue to run. A faulting task can be reset or quickly re-started if desired, which of course is application dependent (see Figure 3-2).

A discussion of the different context types and their modes of operation follows.

3.2 Context Types and Operational Modes

There are three types of fido contexts: Standard, Fast-Vectored, and Fast Single-Thread (FST). The context type determines both context behavior and switching time. Each context has an assigned priority, can have interrupts associated with it, and includes such features as time slicing. Although contexts can be completely isolated from one another, they can communicate and share resources. Interrupts are inextricably coupled to contexts in that any interrupt in the system is associated with only one context. Each context has three possible states, ready (RDY), not ready (NOT_RDY), and halted. The fido pioneers a new "sleep" instruction to manage these states. The sleep instruction can change the state of any context from RDY to NOT_RDY. This mechanism interrupts the CPU to allow another context to run (see Figure 3-3).

3.2.1 Standard Context

The standard context retains standard, single-context operational compatibility. If code is executing in a standard context and an interrupt associated with this context occurs, the interrupt is handled in the usual manner. The interrupt service routine address is fetched from the vector table and if the interrupt is of higher priority than the context, the current PC and status register are stacked then re-loaded from the vector table. The handler (which needs to PUSH/POP any registers it uses) is then called, executes, performs an interrupt return, the PC and status register are unstacked, and the code resumes where it left off.

3.2.2 Fast-Vectored Context

The fast context is really an interrupt handler that operates as a context. If an interrupt occurs that is associated with a Fast Context, the PC for that context is loaded from the vector table, and execution begins. No stack operations occur except those needed by the context itself. It runs to completion then sleeps itself. Since a Fast Context loads the PC with the ISR address from the vector table, any number of interrupts can be associated with a context of this type. The context switch time for a Fast Context is only the time it takes to load the PC and begin.

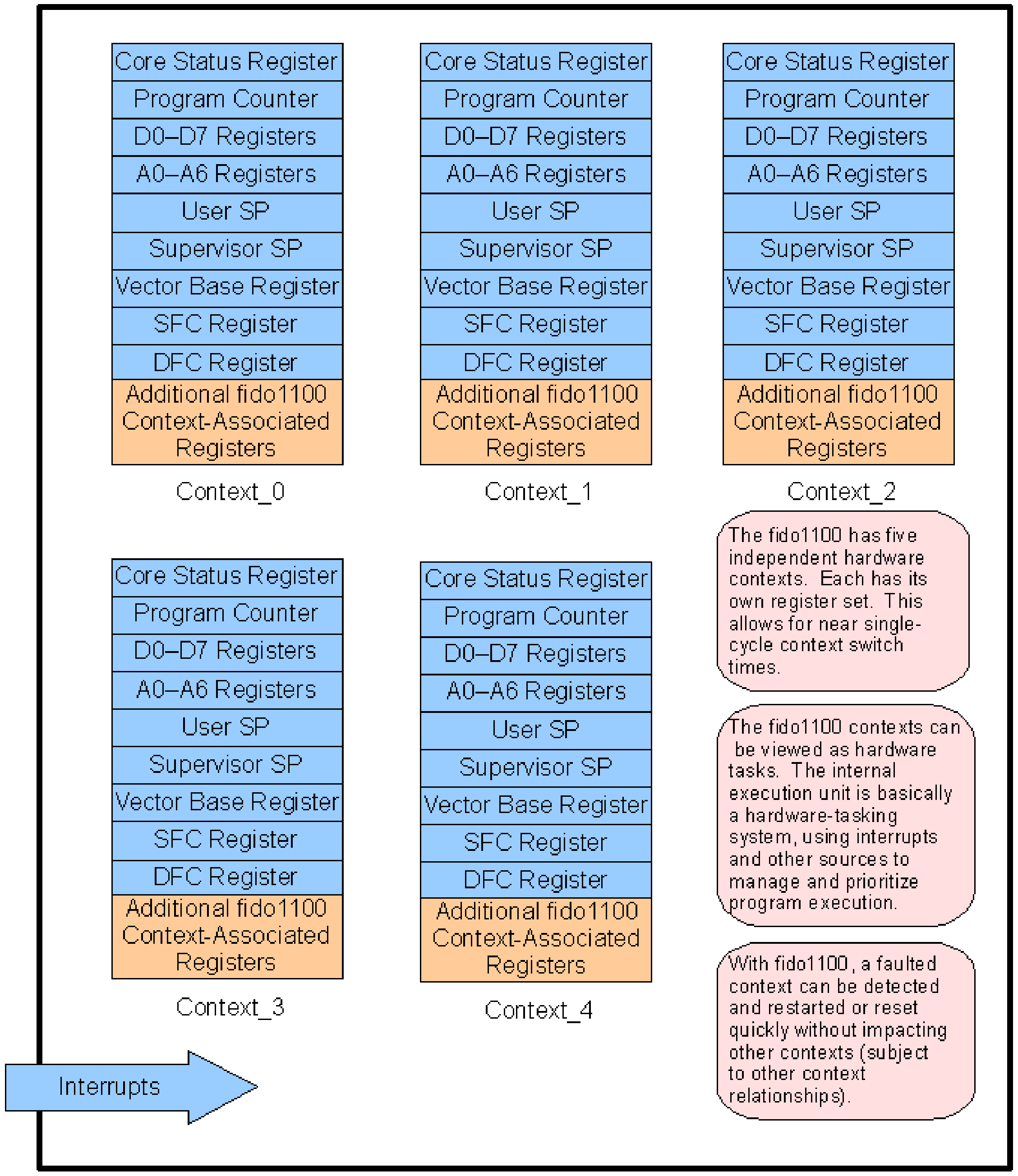

Core Status Register
Program Counter
D0–D7 Registers
A0–A6 Registers
User SP
Supervisor SP
Vector Base Register
SFC Register
DFC Register
Additional fido1100 Context-Associated Registers
Context_0
Context_1
Context_2
Context_3
Context_4
The fido1100 has five independent hardware contexts. Each has its own register set. This allows for near single-cycle context switch times.
The fido1100 contexts can be viewed as hardware tasks. The internal execution unit is basically a hardware-tasking system, using interrupts and other sources to manage and prioritize program execution.
With fido1100, a faulted context can be detected and restarted or reset quickly without impacting other contexts (subject to other context relationships).
Interrupts

Figure 3-2. The fido1100 Multi-Context View

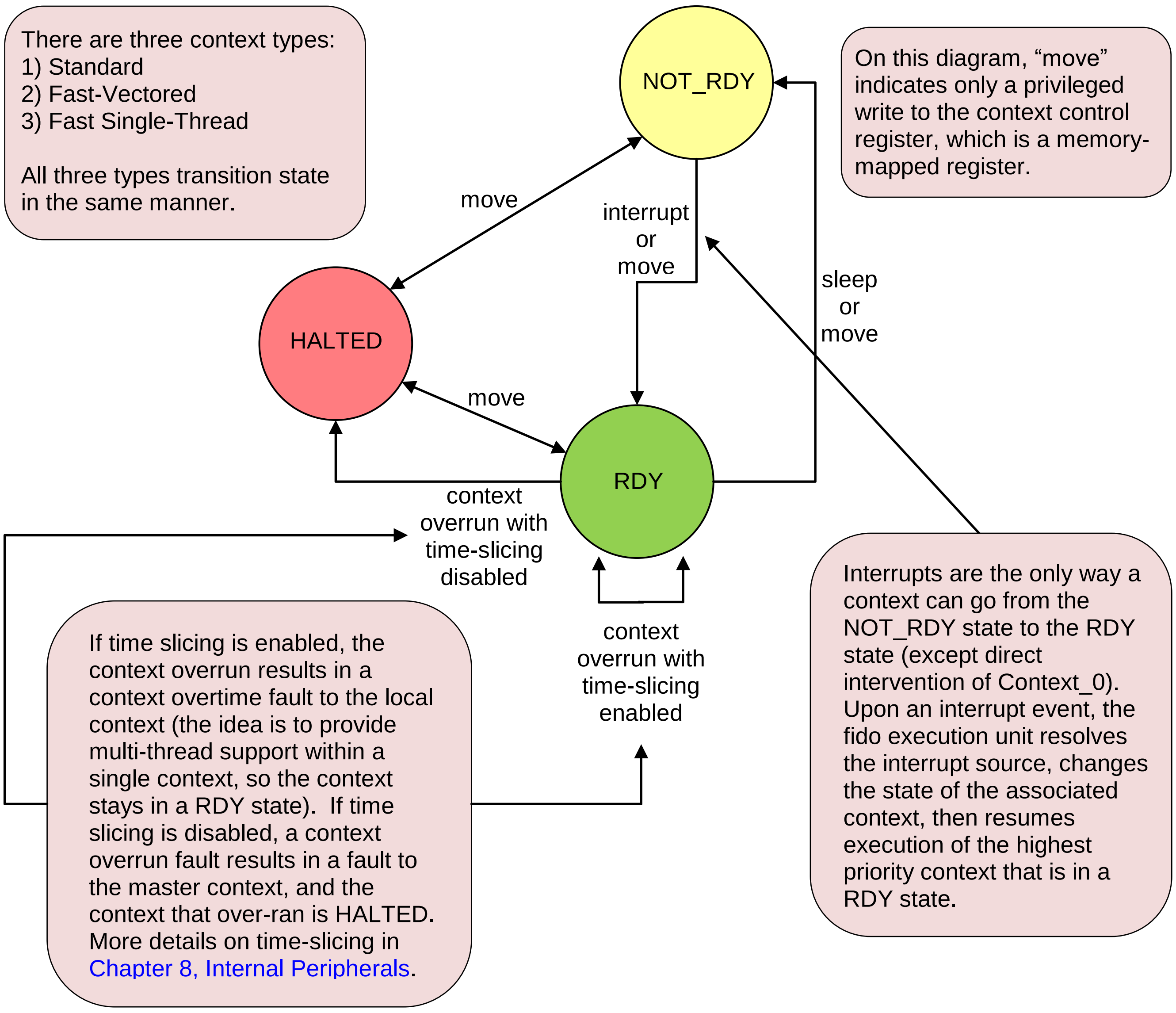

There are three context types:
1) Standard
2) Fast-Vectored
3) Fast Single-Thread
All three types transition state in the same manner.
NOT_RDY
On this diagram, "move" indicates only a privileged write to the context control register, which is a memory-mapped register.
move
interrupt or move
sleep or move
HALTED
move
RDY
context overrun with time-slicing disabled
context overrun with time-slicing enabled
If time slicing is enabled, the context overrun results in a context overtime fault to the local context (the idea is to provide multi-thread support within a single context, so the context stays in a RDY state). If time slicing is disabled, a context overrun fault results in a fault to the master context, and the context that over-ran is HALTED. More details on time-slicing in Chapter 8, Internal Peripherals.
Interrupts are the only way a context can go from the NOT_RDY state to the RDY state (except direct intervention of Context_0). Upon an interrupt event, the fido execution unit resolves the interrupt source, changes the state of the associated context, then resumes execution of the highest priority context that is in a RDY state.

Figure 3-3. Context Types and State Transitioning

3.2.3 Fast Single-Thread Context

The FST context is really a single thread that operates as a context. If an interrupt occurs that is associated with a FST context, execution begins immediately at the current PC for that context. No vector-table access or stacking occurs. The FST type of context must be coded to terminate using a sleep/branch combination to ensure the PC for the context is always where the context should begin execution the next time the interrupt occurs. The context switch time for a FST context is near single cycle. Applications for this type of context include situations where high priority, minimal response time I/O, or high frequency events must be handled in the system (see Figure 3-4).

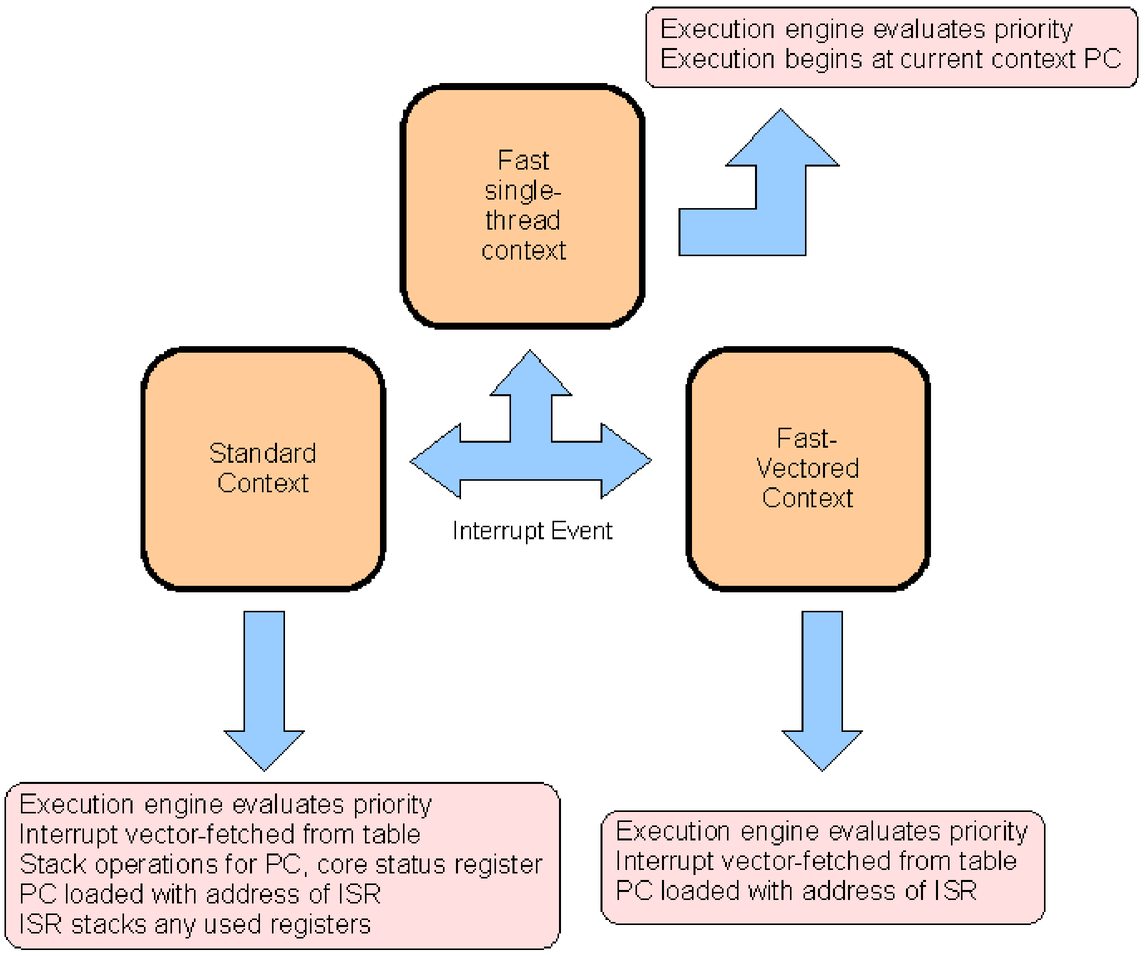

Execution engine evaluates priority
Execution begins at current context PC
Fast single-thread context
Standard Context
Fast-Vectored Context
Interrupt Event
Execution engine evaluates priority
Interrupt vector-fetched from table
Stack operations for PC, core status register
PC loaded with address of ISR
ISR stacks any used registers
Execution engine evaluates priority
Interrupt vector-fetched from table
PC loaded with address of ISR

Figure 3-4. Context Interrupt Overhead

3.3 Master Context

Context_0 is the Master Context and it holds special powers. By default, it is the highest priority context regardless of priority settings for other contexts. Only code running in Context_0 can write to every fido register, including the context control register of every context. By default, the fido powers up by running in Context_0. A recommended practice is to house all initialization code here, set up all other contexts according to application needs, then sleep Context_0 and use it to handle exceptions from all other contexts.

Many applications can run entirely in Context_0, which is perfectly acceptable. After Context_0, the priorities of the other contexts are assignable through registers. However, if both Context_1 and Context_4 are assigned a priority of six for example, Context_1 will have the higher priority.

Many fido registers have access-controlled bit fields (discussed in later chapters). Only Context_0 can write to access-controlled bit fields, and only while in supervisor mode. This restriction prevents program corruption—a safety feature that keeps order (and limits chaos) in this multi-threaded, real-time embedded application running on a very flexible machine.

3.4 Summary

There are no other real rules or restrictions for context usage, understanding the chip's flexibility and how it works is the key to implementing the application.

- Any or all of the five fido contexts can be used by an application.
- Unused contexts are disabled and have no affect on system operation.
- In review, contexts have three states (HALTED, RDY, and NOT_RDY). A HALTED state will not run (that is, the hardware execution engine will bypass it for scheduling consideration) without master Context_0 writing to the context control register or JTAG intervention.
- When a context is done (temporarily), the SLEEP instruction is used to transition from a RDY to the NOT_RDY state.
- All interrupts have an association to a context and a priority relative to that context.
- If a context is set up as a Fast Single-Thread context, any interrupt associated with that context will cause the context to execute the code indicated by the context's PC.
- For a Fast context, any interrupt associated with that context will immediately cause the context to execute the Interrupt Service Routine defined by the context's vector exception table.
- Certain registers and the access-controlled field in all registers are writable by only the Master Context, Context_0.

Chapters herein will discuss the use of context timers, time-slicing, memory protection, claim registers, and the use of internal peripherals and other features of fido. These are closely related to context management, and are typically set up by the Master Context during initialization based on application needs. See Section 4.8, Context Management, for details.

4. Core CPU

4.1 Overview

The core CPU is the main computer in the chip. The fido1100 core is based on the CPU32 architecture, and is compatible with the CPU32 instruction set. Some exceptions will be listed in this chapter. All contexts share the same Execution Unit, which is the portion of the Core CPU that fetches, decodes, and executes instructions. However, as introduced in Chapter 3, Content Architecture, each of the five hardware contexts in the fido1100 has its own register set. This unique feature of fido allows it to operate as five machines in one where the application is concerned.

The following features of the fido1100 are discussed in this section:

- Address and Data Space (32-bit address space and 32-bit data paths)
  - Byte, word (16-bit), long-word (32-bit), and quad-word (64-bit) operations
- Register Model
- User/Supervisor Space
- Instruction Set Summary
- Interrupts, Faults, and Exceptions
  - External Interrupts
  - Interrupt Priorities and Control
  - Interrupt, Fault, and Exception Handling
    - Short Format Stack Frames
    - Instruction Error Stack Frame
    - Fault and Exception Handling
- Reset Processing
- Context Management

Other internal features of the fido1100 (the PMU, Internal SRAM, the RREM, Memory Protection Unit [MPU], SDRAM controller, and External Bus Interface) are discussed in subsequent chapters.

4.2 Address and Data Space

The fido1100 internal address and data bus are 32-bits wide. It supports the following data types:

- Data types
  - BCD—4-bit representation (supported by certain instructions)
  - Byte—8-bit signed and unsigned representations
  - Word—16-bit signed and unsigned representations
  - Long-word—32-bit signed and unsigned representations
  - Quad-word—64-bit signed and unsigned representations (supported by certain instructions, requires a register pair to contain data)

The fido1100 supports an extensive set of addressing modes for source and destination operands of an instruction.

- Memory addressing modes
  - Dn—Data Register
  - An—Address Register
  - (An)—Address Register Indirect
  - (An)+—Address Register Indirect with post-increment
  - -(An)—Address Register indirect with pre-decrement
  - (d16,An)—Address Register Indirect with displacement
  - (d8,An,Xn.Size*Scale)—address Register indirect with displacement and scaling
  - (bd,An,Xn.Size*Scale)—address Register indirect with displacement and scaling
  - (d16,PC)—Program Counter relative with displacement
  - (d8,PC,Xn.Size*Scale)—Program Counter relative with displacement and scaling
  - (bd,PC,Xn.Size*Scale)—Program Counter relative with displacement and scaling
  - (xxx).W—memory pointer
  - (xxx).L—memory pointer
  - #xxx—immediate

An important fido1100 feature to be remembered is that the most significant bit (Bit [31]) of the address is used for Endian Mode Control. This limits the fido1100 to two Gbytes of space as opposed to four Gbytes of space. See Chapter 5, Memory Management and Protection, for details on Endian Mode Control.

The external address bus of the fido1100 is 31 bits and the external data bus is 16 bits. The external interface of the fido1100 is discussed in Chapter 6, External Bus Interface.

4.3 Register Model

In the fido1100, each hardware context has its own set of registers. These registers are writable by only the context to which they belong, and by the master context. The register set summary is as follows:

- Eight 32-bit User Data Registers (D0-D7)
- Seven 32-bit Address Registers (A0-A6)
- Two 32-bit Stack Pointers (A7 and A7')
  - The two stack pointers are indexed by supervisor status (A7' stack pointer for supervisor mode, and A7 for user mode)
- One 32-bit Program Counter
- One 16-bit Status Register (SR)
- One 32-bit Vector Base Register (VBR)
- Two 3-bit Alternate Function Registers (SFC, DFC)

For the internal address of these registers, refer to the Memory and Register Group Address Map Table in Chapter 5.

The fido1100 architecture uses the big-endian format (i.e., the most significant bytes are stored at the lowest addresses). For internal data representation, formats are used as described in the following sections.

4.3.1 Data Register Operands

- Each data register is 32 bits wide.
- Byte operands occupy the low-order 8 bits, word operands occupy the low-order 16 bits, and long-word operands occupy the full 32 bits.
- When a data register is used as either a source or destination operand (byte or word), only the low-order byte or word is changed; the high-order portion is not used and is left unchanged.
- Operands in data registers can represent a single byte, a 16-bit word, a 32-bit long word, or a 64-bit quad word.
  - Quad-word data can consist of any two data registers without restrictions on order or pairing.
  - Quad words result from 32-by-32 multiply or divide operations.
  - There are no explicit instructions for manipulating quad-word data types.
- For signed operations, standard 2’s complement notation is used.
- Instructions requiring BCD operands use the low-order byte as two packed BCD digits.

4.3.2 Address Register Operands

- Each address register is 32 bits wide.
- Address registers cannot be used for byte-sized operands.
- Either the low-order word or the entire 32 bits are used as a source operand. When used as a word-sized source operand, the 16-bit word is sign-extended to 32 bits before being used.
- As a destination operand, the entire register is affected regardless of operation size.

4.3.3 Operands in Memory

- Memory is byte addressable.
- Multi-byte data is stored in big-endian format.
- Non-aligned access support:
  – Accessing word- or long-word-sized operands does not require that the operand be word aligned or long-word aligned. Accessing this type of operand on an odd address will not result in an address exception.

4.4 User/Supervisor Space

Context applications can be run in either User Mode or Supervisor Mode. Privileged instructions can be executed only in Supervisor mode. This mode control is set via the Status Register (SR).

The SR contains condition codes, the interrupt priority mask, the supervisor/user state, and trace enable. Only the Condition codes (lower 8 bits) are manipulated by user-mode instructions. When in supervisor mode, the entire register can be manipulated. Each context has its own copy of the status register. The SR is defined in Section 4.6.3, Interrupt Priorities and Control.

Separate stacks (as defined by A7 and A7') are used in User and Supervisor mode.

4.5 Instruction Set Summary

The fido1100 supports the CPU32 instruction set with some modifications, and has some new instructions. The fido1100 general instruction classes include machine functions for all of the following operations:

- Data movement
- Arithmetic operations
- Logical operations
- Shifts and rotates

- Bit manipulation
- Binary-Coded Decimal (BCD) arithmetic
- Program control
- System control
- Power Control

Following is a discussion of each instruction class. The complete instruction set is provided in *The fido1100 Instruction Set Reference Guide*.

4.5.1 Data Movement Instructions

The MOVE instruction is the basic means of transferring and storing address and data. MOVE instructions transfer byte, word, and long-word operands from memory to memory, memory to register, register to memory, and register to register. Address movement instructions (MOVE or MOVEA) transfer word and long-word operands and ensure that only valid address manipulations are executed.

In addition to the general MOVE instructions, there are several special data movement instructions: move multiple registers (MOVEM), move peripheral data (MOVEP), move quickly (MOVEQ), exchange registers (EXG), load effective address (LEA), push effective address (PEA), link stack (LINK), and unlink stack (UNLK).

4.5.2 Integer Arithmetic Operations

The arithmetic operations include the four basic operations of add (ADD), subtract (SUB), multiply (MUL), and divide (DIV) as well as arithmetic compare (CMP, CMPM, CMP2), clear (CLR), and negate (NEG). *The fido1100 Instruction Set Reference Guide* includes ADD, CMP, and SUB instructions for both address and data operations with all operand sizes valid for data operations. Address operands consist of 16 or 32 bits. The clear and negate instructions apply to all sizes of data operands.

Signed and unsigned MUL and DIV instructions include:

- Word multiply to produce a long-word product

- Long-word multiply to produce a long-word or quad-word product

- Division of a long-word dividend by a word divisor (word quotient and word remainder)

- Division of a long-word or quad-word dividend by a long-word divisor (long-word quotient and long-word remainder)

A set of extended instructions provides multi-precision and mixed-size arithmetic. These instructions are add extended (ADDX), subtract extended (SUBX), sign extend (EXT), and negate binary with extend (NEGX).

4.5.3 Logic Instructions

The logical operation instructions (AND, OR, EOR, and NOT) perform logical operations with all sizes of integer data operands. A similar set of immediate instructions (ANDI, ORI, and EORI) provide these logical operations with all sizes of immediate data. The TST instruction arithmetically compares the operand with zero, placing the result in the condition code register.

4.5.4 Shift and Rotate Instructions

The arithmetic shift instructions, ASR and ASL, and logical shift instructions, LSR and LSL, provide shift operations in both directions. The ROR, ROL, ROXR, and ROXL instructions perform rotate (circular shift) operations, with and without the extend bit. All shift and rotate operations can be performed on either registers or memory.

Register shift and rotate operations shift all operand sizes. The shift count may be specified in the instruction operation word (to shift from one to eight places) or in a register (modulo 64-shift count).

Memory shift and rotate operations shift word-length operands only one bit position. The SWAP instruction exchanges the 16-bit halves of a register. Performance of shift/rotate instructions is enhanced so that use of the ROR and ROL instructions with a shift count of eight allows fast byte swapping.

4.5.5 Bit Manipulation Instructions

Bit manipulation operations are accomplished using the following instructions: bit test (BTST), bit test and set (BSET), bit test and clear (BCLR), and bit test and change (BCHG). All bit manipulation operations can be performed on either registers or memory. The bit number is specified as immediate data or in a data register. Register operands are 32 bits long. Memory operands are 8 bits long.

4.5.6 Binary-Coded Decimal Instructions

Five instructions support operations on Binary-Coded Decimal (BCD) numbers. The arithmetic operations on packed BCD numbers are add decimal with extend (ABCD), subtract decimal with extend (SBCD), and negate decimal with extend (NBCD).

4.5.7 Program Control Instructions

A set of subroutine call and return instructions and conditional and unconditional branch instructions perform program control operations.

4.5.8 System Control Instructions

Privileged instructions, trapping instructions, and instructions that use or modify the condition code register provide system-control operations. All of these instructions cause the processor to flush the instruction pipeline.

4.5.9 Power Control Instructions

For a complete understanding of how STOP and LPSTOP instructions are implemented, see Chapter 10, Power Control.

4.5.10 Modifications to CPU32 Instruction Compatibility

This section discusses the modifications to the instructions the fido1100 implements differently from the CPU32.

- Interpolate—This causes an Illegal Instruction exception in the fido1100
- Table—This causes an Illegal Instruction exception in the fido1100
- MOVEC instruction—These are new control codes to define fido1100-specific registers
- LPSTOP and BKPT instructions
  - Because the fido1100 exception handling is different from that of a CPU32, LPSTOP does not “export” interrupt priority via external-bus interface. See Chapter 10, Power Control, for more information.
  - BKPT does not require or generate an external bus cycle for breakpoint acknowledge. See Chapter 9, Debug Features, for more information.
- BGND—The fido1100 does not have a CPU32-style Background Debug Mode (BDM). The BDM instruction just executes a trap.
- Branch and Jump instructions have additional profiling support. (See Chapter 9, Debug Features, for details on call and branch trace capability.)
  - Bcc
  - BRA
  - BSR
  - Dbcc
  - JMP

- JSR
- RTE
- RTS

- Table Lookup and Interpolate instructions are not implemented—they will TRAP instead. The fido1100 tools will issue a warning if these instructions are used in assembly code.
  - TBLS and TBLSN
  - TBLU and TBLUN

4.5.11 New Instructions

This section discusses new instructions the fido1100 implements to support the multiple-hardware context architecture.

- SLEEP instruction:
  - Similar to STOP but does not alter the SR and only affects the current context. Sleep causes the current context to change from the Ready (RDY) state to the Not Ready (NOT_RDY) state allowing other pending contexts to run. See Section 4.9, Context Management, for complete details.

- TRAPX instruction
  - Sends a trap signal to Master Context_0. Can be used for inter-context communication and exception escalation.

4.6 Interrupts, Faults and Exceptions

4.6.1 Overview

This section presents the following:

- External Interrupts
- Interrupt Priorities and Control
- Interrupt, Fault, and Exception Processing
- Stack Frames
- Vector Table

The fido1100 has many internal interrupt sources, eight external interrupts, and many sources of faults and exceptions. Each of these has a unique vector in the vector table. With the fido1100, each context has its own vector table (256 entries each) (see Table 4-8, Exception Vector Table).

With the fido1100, each interrupt has its own priority and is "associated" with a single context. Each interrupt has an interrupt control register where the priority and context association is assigned. This is true for all interrupts, internal and external. Interrupts are potential wakeup events for sleeping contexts. When an interrupt occurs, its own priority and the interrupt priority mask in the context status register both come into play in determining subsequent processing. Details are provided in Section 4.6.3, Interrupt Priorities and Control, and in Section 4.8.11, Context Management Registers.

There are eight external interrupts for connection to external devices. The interrupt priority, context association, polarity, and edge/level triggering are all parameters in the interrupt control register. These are discussed in Section 4.6.2, External Interrupts.

Exceptions and faults may not produce the same stack frame format as a CPU32. The fido1100 is instruction-set compatible with the CPU32 but not binary compatible with it. The details of exception processing and stack frames are discussed in Section 4.6.4, Interrupt, Fault, and Exception Handling, and in Section 4.6.4.1., Short Format Stack Frame.

4.6.2 External Interrupts

The fido1100 communications controller can receive interrupt requests from eight external signals (see Table 4-1).

Table 4-1. External Hardware Signals

Signal Name	Description
INT0	Interrupt_0
INT1	Interrupt_1
INT2	Interrupt_2
INT3	Interrupt_3
INT4 DMA0 ACK	Muxed pin, Interrupt_4 or DMA Channel 0 Acknowledge
INT5 DMA1 ACK	Muxed pin, Interrupt_5 or DMA Channel 1 Acknowledge
INT6 DMA0 REQ	Muxed pin, Interrupt_6 or DMA Channel 0 Request
INT7 DMA1 REQ	Muxed pin, Interrupt_7 or DMA Channel 1 Request

These eight interrupts are mapped to vectors 24–31 in the vector table. Each interrupt channel must be assigned to a particular context by definition of its interrupt control register. These eight registers are "access-controlled" registers, and as such are writable by only the master context (see Chapter 11, Access-Controlled Registers). The detailed definition of one of the eight external interrupt control registers is presented in Table 4-2.

Table 4-2. Interrupt 'N' Control Registers (where N=0..7)

31–14	13	12	11	10	9–8	7–5	4–3	2–0
Reserved	Enable	Status	Polarity	Level	Reserved	Priority	Reserved	Context
–	RW	R	RW	RW	–	RW	–	RW

Note: Reset value is 0x00000000.

- Bits [31–14]—Reserved
- Bit [13]—Enable (0=interrupt channel disabled, 1=interrupt channel enabled)
- Bit [12]—Status, read this bit to determine interrupt channel status
  - 0—No interrupt pending
  - 1—Interrupt is pending

Note: Reading this bit acknowledges the interrupt and clears the bit.

- Bit [11]—Polarity of interrupt signal
  - 0—Low level for level sensitive or falling edge for edge sensitive
  - 1—High level for level sensitive or rising edge for edge sensitive
- Bit [10]—Type of interrupt signal
  - 0—Level sensitive
  - 1—Edge sensitive
- Bits [9–8]—Reserved
- Bits [7–5]—Priority of interrupt channel

Note: Sets the priority of the interrupt channel to be assigned, where zero is the lowest and seven is the highest, relative to the SR for the associated context.

- Bits [4–3]—Reserved
- Bits [2–0]—Associated context for this interrupt

Note: This field defines associated context for that interrupt channel. Only this context can be conditionally awakened by this interrupt (based on priority scheme). Context values are 0–4.

4.6.3 Interrupt Priorities and Control

As described in Chapter 3, Context Architecture, a context can change state from NOT_RDY to RDY when an interrupt associated with it fires. Whether it changes state and/or subsequently begins to run is all based on the two-tiered interrupt priority scheme of the fido1100. The following parameters are all used in resolving an interrupt:

- The priority of the associated interrupt
- The interrupt priority mask in the associated context Status Register
- The overall priority of the interrupted context

If the priority of the interrupt, the Status Register interrupt priority mask, and the priority of the context allow, the interrupt wakes up and begins to run. Figure 4-1 shows a diagram of this processing.

The Status Register contains condition codes, interrupt priority mask, supervisor/user state, and trace enable. Only the Condition codes (lower 8 bits) are manipulated by user-mode instructions. When in supervisor mode, the entire register can be manipulated. Each context has its own copy of the Status Register.

This register can be accessed by the owner context in supervisor mode via the MOVESR instruction. This register is also memory mapped, where it is an “access-controlled” register, and as such is writable by only the master context (see Table 4-3, Status Register).

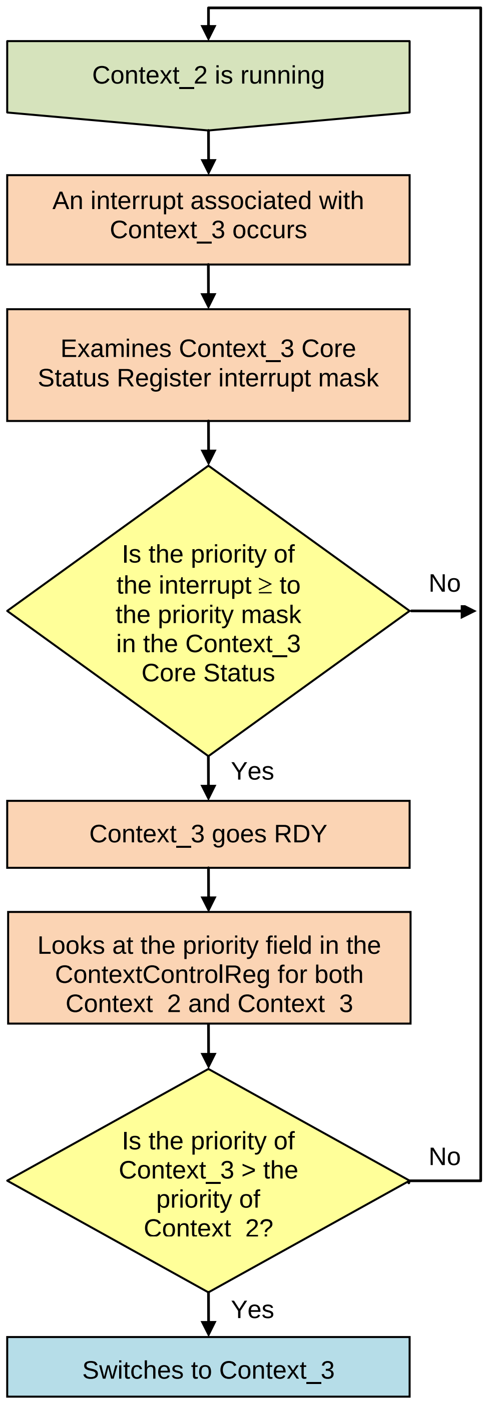
Table 4-3. Status Register

System Byte				User Byte (CCR)					
15–14	13	12–11	10–8	7–5	4	3	2	1	0
TE	Supervisor State	Reserved	Interrupt Priority Mask	Reserved	X	N	Z	V	C
RW	RW	–	RW	–	R	R	R	R	R

Note: Reset value is 0x2700 (trace disabled, supervisor mode, all interrupts masked, all condition codes cleared).

- Bits [15–14]—Trace Enable
  - 00—Trace disabled
  - 01—Branch trace(trace on change of flow)
  - 10—Instruction trace
  - 11—Reserved

Note: This is CPU32 trace control, for the fido1100 enhanced tracing features, see Chapter 9, Debug Features.


Context_2 is running
An interrupt associated with Context_3 occurs
Examines Context_3 Core Status Register interrupt mask
Is the priority of the interrupt ≥ to the priority mask in the Context_3 Core Status
No
Yes
Context_3 goes RDY
Looks at the priority field in the ContextControlReg for both Context 2 and Context 3
Is the priority of Context_3 > the priority of Context 2?
No
Yes
Switches to Context_3

1) Assume that Context_2 is running and an interrupt occurs. This interrupt is associated with Context_3.

2) Each interrupt has a priority and is associated with a context, both via that interrupt's control register. Every interrupt source has a control register.

3) Each context has a Core Status Register that contains an interrupt mask.

4a) When an interrupt occurs, the Core Status Register for the associated context (Context_3) is examined to see if the interrupt is masked or not. If the IntMask=2 and the interrupt priority = 2, the interrupt is masked. In this case, the state of Context_3 does not change and Context_2 continues executing.

4b) If the interrupt priority is ≥3, the state of Context_3 (which the interrupt is associated with) goes RDY.

5) Now, if the priority in the Context_3 control register is more than the priority of the Context_2 control register, Context_3 will run (the interrupt handler is executed).

In Step 4, if another (higher priority) event wakes up Context_3, and it sets its IMR to allow that previous interrupt, it will be serviced then.

The ContextControlReg can be accessed only by the master context (Context_0) in supervisor mode. This register is memory-mapped.

The CoreStatusReg can be accessed by any context if it is in supervisor mode (i.e., each context can set its own interrupt mask via the MOVESR instruction). This register is also memory-mapped where it is writable by only the master context.

Figure 4-1. Context and Interrupt Priority Processing

- Bit [13]—Supervisor State (0=user mode, 1=supervisor)
- Bits [12–11]—Reserved
- Bits [10–8]—Interrupt Priority Mask
  - 000—Minimum Priority Mask (no interrupts are masked)
  - 111—Maximum Priority Mask (all interrupts are masked)

Note: The priority of an interrupt must be greater than the Interrupt Priority Mask to be considered un-masked. However, if the priority mask is seven and the interrupt priority is seven, the interrupt is un-masked even though they are equal. This is retained CPU32 compatibility.

- Bits [7–5]—Reserved
- Bit [4]—X→Extend flag
- Bit [3]—N→Negative flag
- Bit [2]—Z→Zero flag
- Bit [1]—V→Overflow flag
- Bit [0]—C→Carry flag

Note: For details on how the condition codes are set/cleared, see The fido1100 Instruction Set Reference Guide.

4.6.4 Interrupt, Fault and Exception Handling

Generally, whenever an interrupt, exception, or fault occurs, the current instruction will always complete and the event is then serviced. There are two exceptions to this rule:

- The MOVEM instruction
- The DIVS and DIVU instructions

These are the only interruptible instructions in the fido1100.

The fido1100 generates two types of stack frames:

- Short Format Stack Frame (four 16-bit words)
- Instruction Error Stack Frame (six 16-bit words)

Each interrupt, fault, and exception has a unique entry in the vector table. The type of stack frame generated for these events and a definition of the stack frames follows.

4.6.4.1 Short Format Stack Frame

The Short Format Stack Frame contains four words and the format is:

- SP—Status Register Word
- SP +2—Next Instruction Program Counter High Word
- SP +4—Next Instruction Program Counter Low Word
- SP +6—Format/Vector Offset Word

The Offset Word Format/Vector is presented in Table 4-4.

Table 4-4. Short Format Stack Frame Offset Word Format/Vector

15–12	11–0
0	Vector Offset

The top four bits define the stack frame format (which will be zero for this frame type), the remaining bits are the exception vector offset (i.e., the eight-bit exception vector times four, or 0-1020 decimal) (see Table 4-8, Exception Vector Table).

This four-word stack frame will be created by:

- Interrupts (Standard Mode contexts only)
- Format Error Exceptions
- The TRAP #n and TRAPX #n Instructions
- Illegal Instruction Exceptions
- A-Line and F-Line un-implemented instructions
- Privilege Violation Exceptions

4.6.4.2 Instruction Error Stack Frame

This stack frame contains six words and the format is:

- SP—Status Register Word
- SP +2—Next Instruction Program Counter High Word
- SP +4—Next Instruction Program Counter Low Word
- SP +6—Format/Vector Offset Word
- SP +8—Faulted Instruction Program Counter High Word
- For MPU Fault: SP +8—Faulted Address High Word

- SP +10—Faulted Instruction Program Counter Low Word
- For MPU Fault: SP +8—Faulted Address Low Word

The Format/Vector Offset Word format is presented in Table 4-5.

Table 4-5. Instruction Error Stack Frame Offset Word Format/Vector

15–12	11–0
2	Vector Offset

The top four bits define the stack frame format (which will be a two for this frame type), the remaining bits are the exception vector offset (i.e., the eight-bit exception vector times 4, or 0-1020 decimal) (see Table 4-8, Exception Vector Table).

This six-word stack frame will be created by:

- Instruction related traps:
  - CHK instruction
  - CHK2 instruction
  - TRAPcc instruction
  - TRAPV instruction
  - DIVS/DIVU instructions (divide by zero)

- Trace Exceptions
- Address Errors
- Bus Errors
- MPU Errors
- Hardware breakpoints (the faulted instruction program counter is the address of the instruction executing when the breakpoint was sensed).

4.6.4.3 Interrupt Handling

As described in Chapter 3, Context Architecture, if an interrupt occurs that is associated with either a Fast Single-Thread or Fast-Vectored type context, no stack frame is generated. After the priority is resolved (as described above in this chapter), the processing is as follows:

- Fast Single-Thread—Execution begins at current PC value for that context
- Fast Vectored—Interrupt vector fetched from vector table, and PC is loaded with address of ISR

An interrupt associated with a Standard Context generates a Short Format Stack Frame.

4.6.4.4 Fault and Exception Handling

The general processing for fault and exception handling follows the flow described below:

- Any register altered by a faulted instruction effective address calculation is restored to its previous value.
- The S bit in the status register (SR) is set to establish supervisor privilege level.
- The T0 and T1 bits in the SR are cleared to disable tracing.
- The interrupt-priority mask bit in the context's status register is set to be equal to the priority of the requesting interrupt source's interrupt control register.

Note: This will cause the context's interrupt service routine code to assume the priority of the interrupt (masking it) and thus prevent the interrupt request from causing an infinite loop. It will be up to the software to reset the interrupt request by reading the corresponding interrupt control/status register.

- A stack frame is generated according to the nature of the fault or exception.
- The exception vector is multiplied by four to generate a vector offset.
- The Exception Vector table address is calculated by adding the vector offset to the Vector Base Register (VBR) for the current context.
- The result is used to fetch the vector from the exception vector table (see Table 4-8, Exception Vector Table).
- This value is placed into the PC and begins execution from the location.
- The return sequence is case-dependent. If the fault can be handled within the affected context, the return is based on the stack frame type. If the fault is fatal, control is transferred to the master context.

A benefit of having multiple contexts is that fault conditions that would be fatal to a typical processor (e.g., a double-bus fault) need only be fatal to the executing context. The fido1100 uses the Inter-Context fault (Vector #95) to handle fatal faults: The cases handled are:

- Reset—If the reset vector points to memory that causes a bus error when the execution unit fetches it. This can only occur in the master context and hence will halt the entire CPU.

- Double-Bus Fault
- Double-MPU Fault
- Address Error
- Any combination of Address/MPU/Bus Fault while stacking the exception frame (e.g., while stacking the exception frame for an MPU fault, a bus fault is generated)

If any of these events occur in Context_0, the processor will be halted. However, if any of these events occur in Context_1–Context_4, they are handled as follows:

- The faulting context is halted (the State field in Context Control Register set to HALT)
- The ID of the faulting context is placed in the Faulted Context Register
- Exception #95 (Inter-Context Fault) is forced, transferring control to the master context which can determine subsequent processing

4.6.5 Summary

All faults and exceptions follow the processing flow as described above. Some faults can be handled within the executing context and return from the fault based on the stack frame type generated. Some can fault initially in Context_1–Context_4, where a subsequent double-fault will fault to the master context. *This is important as each context has its own vector table.* Additional details are provided for the following faults:

- Reset
- Bus Error
- Address Error
- Illegal Instruction
- Instruction Execution Exception
- Instruction Execution Error
- Trace Exception
- Breakpoint
- Context Overtime Fault

- MPU Error
- Inter-Context Fault

4.6.5.1 Reset Additional Details (Vectors #0 and #1)

- If the reset vector points to memory that causes a bus error when the execution unit fetches it, it will occur only in the master context and therefore halt the entire CPU.

4.6.5.2 Bus Error Additional Details (Vector #2)

- Generates an Instruction Error Format Stack Frame in the context of execution
- This exception can occur if the CPU tries to access memory that does not exist or if the bus cycle is aborted due to an rdy_n timeout (see External Bus Timing Diagrams).

Note: An aborted pre-fetch does not cause an exception.

- All processing for this exception occurs while remaining in the same execution context unless a bus error occurs during the exception processing for the following:
  - An MPU error
  - A bus error
  - An address error
  - A reset
  - While the processor is loading, the stack information during an RTE instruction execution a double bus fault is generated.
- In any of these events and depending on the execution context, the processor:
  - Context_1–Context_4—Writes the context number to the Faulted Context Register, halts the current context, and forces exception #95(Inter-Context Fault) that switches to the Master Context.
  - Master Context—Is halted. Only a reset can restart the processor.

4.6.5.3 Address Error Additional Details (Vector #3)

- Generates an Instruction Error Format Stack Frame
- An Address Error exception occurs if the CPU tries to fetch an instruction from a non-word aligned boundary (an odd address.)

Note 1: The exception will occur only when CPU actually tries to access that memory. If the execution is stalled (by another exception, an interrupt, etc.) this exception will not occur. When the instruction is resumed (after the exception is handled) if the stalling condition does not re-occur the exception will occur.

Note 2: The exception will not occur on an instruction fetch if the instruction is not executed.

- All processing for this exception occurs while remaining in the same execution context unless an address error occurs during the exception processing for the following:
  - An MPU error
  - A bus error
  - An address error
  - A reset
  - While the processor is loading the stack information during a RTE instruction execution a double bus fault is generated

- In any of these events and depending on the execution context the processor:
  - Context_1–Context_4—Writes the context number to the Faulted Context Register, halts the current context, and forces exception #95 (Inter-Context Fault) that switches to the Master Context.
  - Master Context—Is halted. Only a reset can restart the processor.

4.6.5.4 Illegal Instruction Additional Details (Vector #4)

- Generates a Short Format Stack Frame

- An illegal instruction exception will occur under the following conditions:
  - If the illegal instruction is executed (opcode 0x4AFC)
  - When the first instruction word contains a bit pattern that does not correspond to a valid CPU32 instruction
  - If the instruction is a MOVEC instruction and the first extension word does not specify a valid control code.
  - If the instruction specifies an indexing addressing mode extension word with Bits [5–4] = 00 or Bits [3–0] = 0000

4.6.5.5 Instruction Execution Exception Additional Details (Vectors #5, #6, and #7)

- Generates an Instruction Error Format Stack Frame

- This exception will occur for the following conditions:
  - Divide by Zero—Exception Vector #5

- CHK, CHK2 instructions—Exception Vector #6
- TRAPcc, TRAPV Instructions—Exception Vector #7

4.6.5.6 Instruction Execution Error Additional Details (see Vectors below)

- Generates a Short Format Stack Frame

- This exception will occur under the following conditions:
  - Privilege Violation (executing a supervisor mode instruction while in user mode)—Exception Vector #8
  - A-line unimplemented instruction execution (instructions with Bits [15–12] = 1010b)—Exception Vector #10
  - F-line unimplemented instruction execution (instructions with Bits [15–12] = 1111b)—Exception Vector #11
  - TRAP #n instruction—Exception Vectors #32–47
  - TRAPX #n instruction—Exception Vectors #96–111
  - A stack frame format error is detected (CPU version number in BERR stack frame and stack frame format number validity during the RTE instruction)—Exception Vector #14

4.6.5.7 Trace Exception Additional Details (Vector #9)

- Generates an Instruction Error Format Stack Frame

- This exception will occur when an instruction executes with the Trace bits set in the SR. These bits will issue this exception according to the rules presented in Table 4-6.

Table 4-6. Trace Exception Instruction Error Generation

T1	T0	Tracing Exception
0	0	Tracing disabled
0	1	Trace on change of status/flow
1	0	Trace on instruction execution
1	1	Reserved

- This functions as follows:

  - With T[1–0] set to 01b, the trace exception will fire whenever the PC changes sequence during execution. All branches, jumps, subroutine calls, returns, and status register manipulations will cause this exception. No exception occurs if the branch is not taken.

  - With T[1–0] set to 10b, a trace exception will be generated after the instruction execution is complete. If the instruction is not executed (because of an interrupt, or because it is an illegal, unimplemented, or privileged instruction), a trace exception is not generated. If the instruction execution is aborted, the trace exception is deferred until the instruction is completed normally. This also includes bus errors or address errors. The exception handler for this exception is not traced.

- If tracing is enabled and an interrupt is pending when an instruction executes, the trace exception will be processed before the interrupt exception.
- If an instruction forces an exception, the forced exception is processed before the trace exception.
- If an instruction is executed and a breakpoint is pending upon completion of the instruction, the trace exception is processed before the breakpoint.
- Tracing also affects the operation of the LPSTOP and STOP instructions. If the T1 is set when one of these instructions executes, the trace exception will be processed and, upon return from trace, the next instruction will be executed. Hence the processor will not stop.
- To prevent a nested trace exception, tracing is disabled as described earlier in the general fault and exception processing flow.
- The CPU32 trace capability exists for compatibility. The advanced methods described in Chapter 9, Debug Features are recommended for better debug capability.

4.6.5.8 Breakpoint Additional Details (Vector #12)

- Generates an Instruction Error Format Stack Frame
- This exception will occur for the following reasons:
  - BKPT Instruction Executed with BKPT bit set in the Debug Control Register—Exception Vector #12
  - Hardware breakpoint or watchpoint trigger with break enabled.
- This exception will be processed in the same execution context, following the flow as described earlier in the general fault and exception processing flow.
- See Chapter 9, Debug Features, for details on hardware breakpoints and watchpoints.

4.6.5.9 Context Overtime Fault Additional Details (Vector #64)

- The fido1100 provides a set of registers that can be used to monitor and control context timing. The context timers can operate in two modes:

  - Time-slice Mode—A feature for running multiple threads within a single context. In this case, the fault occurs in the local context of operation that should deal with scheduling as required.
  - Standard Mode—A context-overtime monitor for critical safety applications. In this case, the fault occurs in the master context and the faulting context is set to the Halted state with the number of the faulting context placed into the Faulted Context Register.

- See Section 4.9, Context Management, for details on context timer functionality.

Note: This exception is classified as an "Interrupt" and therefore will generate a Short Format Stack Frame, but only if the targeted context is set to Standard Mode as defined by the Context Control Register.

4.6.5.10 MPU Error Additional Details (Vector #65)

- Generates an Instruction Error Stack Frame

- This exception can occur under the following conditions:
  - When the MPU disallows memory access. This includes the following conditions:
    - A context executes an instruction that results in an effective address pointing to a memory block that is not assigned to that context.
    - A context tries to write to a read-only restricted memory block.
    - A context tries to read or write to a disabled memory block.
    - An instruction fetch only generates the exception if the instruction is to be executed.

- The exception will occur only when CPU actually tries to access that memory. If the execution is stalled (by another exception, an interrupt, etc.) this exception will not occur. When the instruction is resumed (after the exception is handled) and if the stalling condition does not re-occur, the MPU exception will occur.

- The exception handler can look at the Faulted Context Register to determine who is the offending context and can act accordingly.

- All processing for this exception occurs while remaining in the same execution context unless an MPU error occurs during the exception processing for the following:
  - An MPU error
  - A bus error
  - An address error
  - A reset
  - While the processor is loading the stack information during an RTE instruction execution, a double-bus fault is generated

- In any of these events and depending on the execution context the processor:
  - Context_1–Context_4—Writes the context number to the Faulted Context Register, halts the current context, and forces exception #95 (Inter-Context Fault) that switches to the Master Context.
  - Master Context—Is halted. Only a reset can restart the processor.

- If an MPU block is set up to point to non-existent memory when an access to that memory is attempted, a fatal fault will occur rather than an MPU fault.

4.6.5.11 Inter-Context Fault Additional Details (Vector #95)

- This is the fido1100 mechanism to handle fatal faults and isolate a halting condition to a single context. The cases are as follows:

  - Reset (e.g., if the reset vector points to memory that causes a bus error when the execution unit fetches it.

Note: When this happens, it will always be in Context_0 and hence will always result in a total CPU halt.

  - Double Bus Fault

  - Double MPU Fault

  - Double Address Error

  - Any combination ADDRESS/MPU/BUS fault (e.g., while stacking exception frame for an MPU fault, a bus fault is generated).

- If any of these events occur in Context_1–Context_4 the following sequence will occur:
  - The faulting context is halted (the State field in the Context Control Register set to halt)
  - The ID of the faulting context is placed into the Faulted Context Register
  - The Inter-Context Fault (vector #95) is forced, resulting in control transfer to the master context.

- If any of these events occur in Context_0 execution, the processor will halt all execution. The Faulted Context Register is defined in Table 4-7.

Table 4-7. Faulted Context Register

31–8	7–0
Reserved	Context ID
–	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bits [7–0]—Faulted context ID (a value from 0 to 4)

4.7 Reset Processing

Activating the RESET_N pin will immediately abort all processing and cause a Reset exception. The Reset exception is the highest priority of all exceptions. Any processing in progress when this exception is recognized is lost and cannot be recovered. Reset will result in the following actions (as defined by the value of Bit [0] in the Power On Reset Register). See Chapter 10, Power Control, for additional information.

- Power On Reset Register Bit [0] = 0—Minor Reset:
  - Initialize the Vector Base Register (VBR) for Context_0 to 0x00000000.
  - Initialize the Status Register (SR) for the Context_0 to 0x2700.
  - The following is done using Context_0's register set:
    - Generates a vector number to reference the Reset exception vector.
    - Loads the first long word of the vector into Context_0's supervisor stack pointer. If a bus error exception occurs here, this is considered a double-bus fault and the processor halts.
    - Loads the second long word of the vector into Context_0's program counter. If a bus error exception occurs here, this is considered a double-bus fault and the processor halts
  - Sets the Context Control Register for Context_0 to 0x1F02. This will:
    - Place Context_0 in the Ready state.
    - Assign Context_0 the highest priority.
    - Cause the processor to fetch and initiate decode of the first instruction to be executed (in Context_0). If a bus error exception or an address error exception occurs here, this is considered a double-bus fault and the processor halts
  - Context Timer Clear Register to 0x00000000
  - Context Timer Register to 0x00000000
  - Faulted Context Register to 0x00000000
  - Context Idle Timer Register to 0x00000000

- Power On Reset Register Bit [0] = 1—(Major Reset)
  - Initializes the Vector Base Register (VBR) for *all contexts* to 0x00000000.
  - Initialize the Status Register (SR) for *all contexts* to 0x2700.
  - The following is done using Context_0's register set:
    - Generates a vector number to reference the Reset exception vector.
    - Loads the first long word of the vector into Context_0's supervisor stack pointer. If a bus-error exception occurs here, this is considered a double-bus fault and the processor halts.
    - Loads the second long word of the vector into Context_0's program counter. If a bus error exception occurs here this is considered a double bus fault and the processor halts
  - Sets the Context Control Register for Context_0 to 0x1F02. This will:
    - Place Context_0 into the Ready state.
    - Assign Context_0 the highest priority.
    - Cause the processor to fetch and initiate decode of the first instruction to be executed (in Context_0). If a bus error exception or an address error exception occurs here this is considered a double bus fault and the processor halts
  - Set all Registers to default values, as indicated in Table 5-20, Memory and Register Group Address Map Table

Table 4-8 presents the vector assignments implemented for the various exceptions. The Context Switch column refers to how the exception is handled.

Note: Each context has its own vector table.

Table 4-8. Exception Vector Table

Vector Number	Vector Offset (hex)	Vector Offset (dec)	Assignment	Context Switch?	Notes
0	000	0	RESET: Initial Stack Pointer	Yes, to Context_0	–
1	004	4	RESET: Initial Program Counter	Yes, to Context_0	–
2	008	8	Bus Error	First BERR = No, Second = Yes, to Context_0	–
3	00C	12	Address Error	First Address Error = No, Second = Yes, to Context_0	–
4	010	16	Illegal Instruction	No	–
5	014	20	Divide by Zero	No	–
6	018	24	CHK, CHK2 Instructions	No	–

7	01C	28	TRAPCC, TRAPV Instructions	No	–
8	020	32	Privilege Violation	No	–
9	024	36	Trace	No	–
10	028	40	A-Line unimplemented Instructions	No	–
11	02C	44	F-Line unimplemented Instructions	No	–

Table 4-8. Exception Vector Table *(Continued)*

Vector Number	Vector Offset (hex)	Vector Offset (dec)	Assignment	Context Switch?	Notes
12	030	48	Hardware Breakpoint	No	–
13	034	52	Reserved, (coprocessor)	–	3
14	038	56	Format Error	No	–
15	03C	60	Uninitialized Interrupt	No	1, 3
16	040	64	Software Interrupt	Maybe	2
17–23	044–05C	68–92	Unassigned, (reserved)	–	3
24	060	96	Interrupt_0	Maybe	2
25	064	100	Interrupt_1	Maybe	2
26	068	104	Interrupt_2	Maybe	2
27	06C	108	Interrupt_3	Maybe	2
28	070	112	Interrupt_4	Maybe	2
29	074	116	Interrupt_5	Maybe	2
30	078	120	Interrupt_6	Maybe	2
31	07C	124	Interrupt_7	Maybe	2
32–47	080–0BC	128–188	Trap #n instruction	No	–
48–58	0C0–0E8	192–232	Reserved, (coprocessor)	–	3
59–63	0EC-0FC	236–252	Unassigned, (reserved)	–	3
64	100	256	Context Overtime	Maybe	4
65	104	260	MPU Error	No	–
66	108	264	System Timer 0	Maybe	2
67	10C	268	System Timer 1	Maybe	2
68	110	272	System Timer 2	Maybe	2
69	114	276	System Timer 3	Maybe	2
70	118	280	System Timer 4	Maybe	2
71	11C	284	Watchdog Timer	Maybe	2
72	120	288	Timer Counter Unit 0	Maybe	2
73	124	292	Timer Counter Unit 1	Maybe	2
74	128	296	DMA Channel 0	Maybe	2
75	12C	300	DMA Channel 1	Maybe	2
76	130	304	ADC Conversion Complete	Maybe	2
77	134	308	PMU Ch 0 Interrupt	Maybe	2
78	138	312	PMU Ch 1 Interrupt	Maybe	2
79	13C	316	PMU Ch 2 Interrupt	Maybe	2
80	140	320	PMU Ch 3 Interrupt	Maybe	2
81–84	144–150	324–336	Unassigned, (reserved)	–	3
85–94	154–178	340–37C	Unassigned, (reserved)	–	3

Table 4-8. Exception Vector Table *(Continued)*

Vector Number	Vector Offset (hex)	Vector Offset (dec)	Assignment	Context Switch?	Notes
95	17C	380	Inter-Context Fault	Yes, (to Context_0)	–
96–111	180–1BC	384–444	Trapx instruction	Yes, (to Context_0)	–
112–255	1C0–3FC	344–1020	Unassigned, (reserved)	–	3

Notes:
1. An uninitialized interrupt (vector #15) can occur in a CPU32 part because the interrupt vector is obtained via an interrupt-acknowledge bus cycle. Because the fido1100 uses a fixed-interrupt vector scheme, this will not occur.
2. Interrupt exceptions are assigned to a context and may cause the hardware executive to switch to it if necessary.
3. The normal CPU32 system uses an interrupt-acknowledge bus cycle to fetch interrupt vectors. Because the fido1100 does not do this, the vectors not specifically assigned here are inaccessible to the user.
4. This operates in two modes. In one mode, there is a context switch to Context_0. In the other mode, the fault is handled by the faulting context.

4.8 Context Management

4.8.1 Overview

This section will cover the following context management features of fido:

- The Master Context
- Context priorities
- Context modes
- Context states
- Contexts and interrupts
- Contexts timers
- Context initialization
- Context claim registers
- Detail register definitions

Refer to Chapter 3, Context Architecture, for an overview on the fido architecture and context management. It describes context types, modes of operation, and contexts' transition states.

If a feature is not needed, do nothing. This applies to context management features or fido features such as memory protection, the ADC unit, etc. The default register state for context timers, priority inheritance—all of it—is disabled. Disabled features have no effect on system operation.

4.8.2 Master Context

Context_0 is the Master Context. By default, it is the highest priority context, regardless of priority settings in the Context Control Register. This manual uses the terms "Context_0" and "Master Context" interchangeably.

The Master Context can write to any writable field of any register in the register memory map (see Chapter 12, Register Map Reference). Many registers defined throughout this manual have access-controlled bit fields or may be entirely access-controlled registers. Write capability to access-controlled registers and bit fields within registers can be done only with code executing in the Master Context. See Chapter 11, Access-Controlled Registers, for more information.

From a reset condition, the fido chip powers up running in Context_0. At this point, it is a single-context machine; all other contexts (1–4) are disabled. All internal peripherals and features of fido power up in a disabled or unused state. This is why it makes sense to perform all initialization that the application requires in the Master Context and, when completed, sleep the Master Context, waking it up only when needed.

From the Master Context, all other contexts can be completely set up, armed, and ready to go as desired. Always consider the following attributes when setting up a context for operation:

- Context type (Standard, Fast-Vectored, or Fast Single-Thread)
- Context initial state (RDY, NOT_RDY, HALTED), initial program counter, and stacks
- Associated interrupts (both internal and external)
- Internal peripheral initialization (context timers, timer-counters, ADC, etc.)
- Memory protection and internal debug capability

See related chapters for internal peripherals and other fido features for details on initialization requirement details. If a feature is not needed, no action is necessary. The power up state of internal peripherals and features are disabled and will have no effect on system operation.

Although some applications call for multi-context use, some might run entirely out of the Master Context and not use other contexts. In this case, the Master Context could initialize everything it needs, then transfer control to an executive and continue from there.

4.8.3 Context Priorities

The priority each context operates at is specified in its Context Control Register (i.e., zero through seven, where zero is lowest and seven is highest). During initialization, the Master Context would set up this register for each context used by the application. Also priority-related is the interrupt priority mask found in the Core Status Register for each context that would also be initialized by the Master Context. Section 4.8.6, Contexts and Interrupts, provides an example of how these two fields relate and when a context gets to run.

4.8.4 Context Modes

The Context Control Register of the context specifies the mode a context operates in (Standard, Fast-Vectored, or Fast Single-Thread). During initialization, the Master Context would set up this register for each context used by the application. In many real-time embedded applications, there are usually interrupt-driven events of varying priorities that must be handled along with other lower priority work that can be done in the background (see Figure 4-2).

The fido solution to this example could be as shown in Table 4-9.

Table 4-9. Sample Context Control Register

Context	Mode	Priority	Initial State	Description
0	Fast-Vectored	7	RDY	Performs all initialization then becomes an exception handler.
1	Fast Single-Thread	6	NOT_RDY	This is the minimum response time for the highest priority in the system. It is coded such that the function terminates with a sleep followed by a branch to the entry point.
2	Fast-Vectored	5	NOT_RDY	All medium-priority interrupts are grouped here. Data processing could be done here.
3	Standard	1	RDY	Handle all background tasks and low-priority data processing needs here. Associate low-priority interrupts with this context and service them within it.
4	Reserved	0	HALTED	Held in reserve for future system needs.

4.8.5 Context States

The initial state of all contexts (RDY, NOT_RDY, and HALTED) are also set in the Context Control Register. During initialization, the Master Context should set all used contexts to the desired initial state, and when the master sleeps itself, the highest priority context that is in a RDY state will run. If a context has an interrupt wakeup event (i.e., Fast-Vectored or Fast Single-Thread), that context could be set to NOT_RDY so it will not run until the interrupt fires. In the sample system on the previous page, after the Master Context has finished initialization and sleeps itself, Context_3 would begin to run. Context_1 will run whenever the high-speed interrupt fires. Context_2 will run whenever an associated interrupt fires and Context_1 is not in a RDY state.

The sleep instruction is used to transition from an RDY state to the NOT_RDY state, facilitating execution control. See the following example code for a context that assumes Context_1 is an FST context, activated via the Context_1 software interrupt.

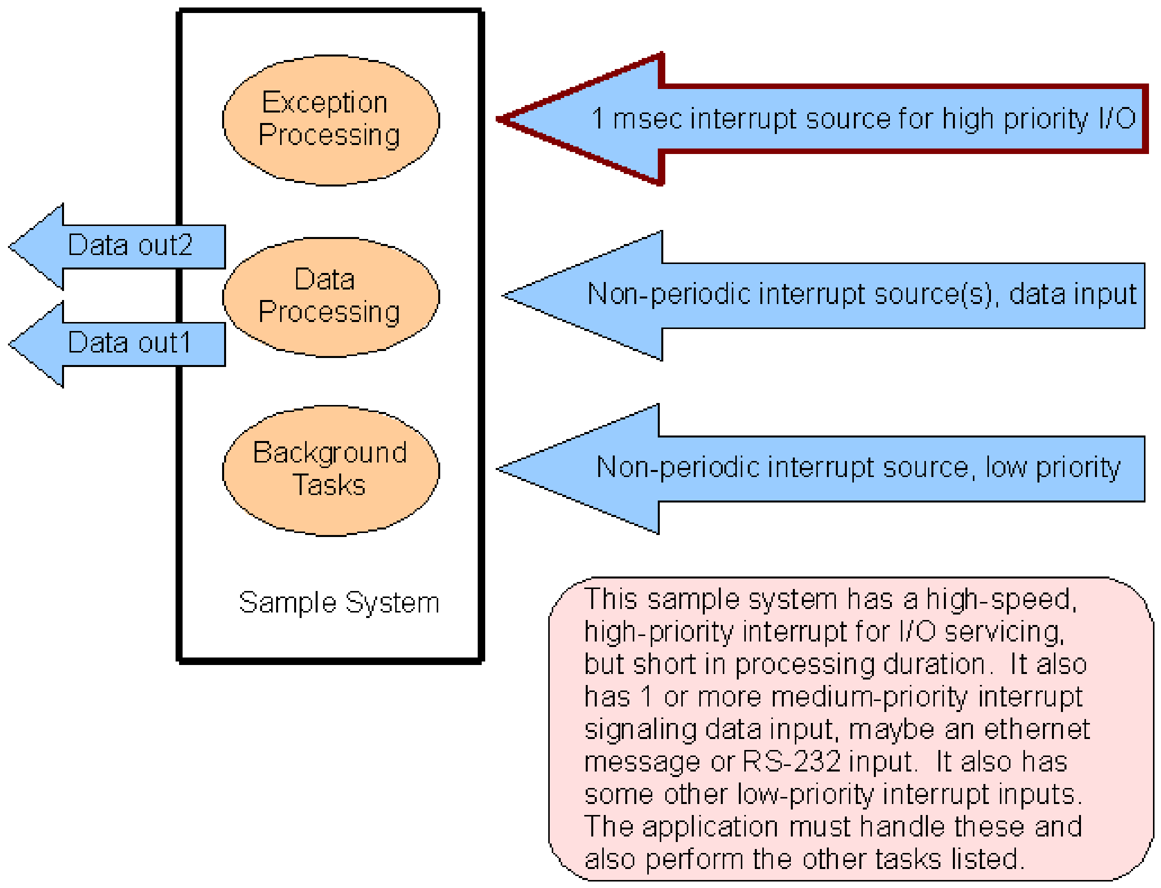

Exception Processing
Data Processing
Background Tasks
Sample System
Data out2
Data out1
1 msec interrupt source for high priority I/O
Non-periodic interrupt source(s), data input
Non-periodic interrupt source, low priority
This sample system has a high-speed, high-priority interrupt for I/O servicing, but short in processing duration. It also has 1 or more medium-priority interrupt signaling data input, maybe an ethernet message or RS-232 input. It also has some other low-priority interrupt inputs. The application must handle these and also perform the other tasks listed.

Figure 4-2. Sample System

```
CTX1_SW_INT_HDLR:                        /* code for FAST-SINGLE THREAD MODE */
  movel CHR_CTX1_INT_CTRL, ctx1swintstat/* acknowledge the S/W interrupt, else it pends */
  oril VALUE_1, IO_ADDR_1                /* perform some I/O, for example */
                                         /* optional 'C' handler call could be made here */
  andil VALUE_2, IO_ADDR_2               /* perform some more I/O */
  sleep                                  /* Context_1 is now in a NOT_RDY state */
  bra CTX1_SW_INT_HDLR                   /* with its program counter here...*/
```

Therefore, the next time the Context_1 software interrupt occurs, Context_1 wakes up and begins executing at its current program counter, which is now at the branch instruction. This is how the program counter needs to be controlled by software when using an FST-type context. *In addition, it is important to acknowledge the interrupt.* Any fido interrupt source has a status register that needs to be read via software to perform the interrupt acknowledge.

Note: This also serves as an example of how contexts can communicate with each other. In this case, Context_1 has been set up as an FST context. It was initialized to a NOT_RDY state by the Master Context so it will not go RDY until the Context_1 software interrupt fires. The Context_1 software interrupt fires when the Context_1 S/W Interrupt Actuation Register is written to, which is designed to be used by other contexts.

4.8.6 Contexts and Interrupts

To understand better how context priorities, interrupt priorities, and context states are interrelated, refer to the example shown in Figure 4-1, Context and Interrupt Priority Processing. Context_2 is currently running and Context_3 is in a NOT_RDY state. An interrupt associated with Context_3 fires. Note the logic the execution follows in resolving which context runs next.

In the final analysis, fido's context and interrupt priority mechanism results in a deterministic outcome. Whichever RDY context has the highest priority will run.

4.8.7 Context Timers

Another available option for use in context control is a set of timing registers that can track, specify, and limit execution time. Some applications require the ability to record timing statistics and other performance data. This register set, one for each context, facilitates the ability to do this (see Table 4-10).

Table 4-10. Description of Context Timers Registers

Register Name	Description
Context Timer Enable Register	Overall enable- and mode-control register used to manage context timers. Is a single register and an access-controlled register and writable by only the Master Context (Context_0).
Context Timer Register	Holds the value measuring context execution time. One for each context. Read/write access dependent upon timer mode.
Context Max Time Register	Used to limit context execution time in time-slice mode. One for each context. Is an access-controlled register and writable by only the Master Context (Context_0).
Context Timer Clear Register	Used to clear the Context Timer Register. A single register and an access-controlled register and writable by only the Master Context (Context_0).
Context Idle Timer Register	Timer used to track time during which no context is active. A single, read-only register.

The context timers can be used to track execution time and *when in use have no additional throughput overhead*. They can be used for any or all contexts via the Context Timer Enable Register, or not at all if not needed. Additionally, the context timer function has two different modes of operation, standard and time-sliced. This is selectable via the Context Timer Enable Register.

Note: In systems using an external bus master, the currently running context timer will continue to run when the fido1100 yields to an external master. This time will need to be accounted for when planning context maximum run times.

4.8.7.1 Standard Mode

In standard mode, the Context Max Time Register is treated as a critical time that, when exceeded, is a faulting event. If the Context Timer Register (for a context using this feature in standard mode) exceeds the Context Max Time Register, it will generate a context-overtime fault exception to the Master Context. The context ID will be scored in the Faulted Context Register. This implementation is taken to address time-critical applications requiring this level of safety.

4.8.7.2 Time-Slice Mode

In time-slice mode, the Context Max Time Register is not treated as a fault. It is used to limit the time a context runs before changing threads within a context or perhaps sleeping itself. If the Context Timer Register (for a context using this feature in time-slice mode) exceeds the Context Max Time Register, it generates a context-overtime fault exception to the local context. Therefore, even though an exception is generated, it is local to the operating context, which can service the exception and do something like schedule another thread or, if done temporarily, sleep the context. This feature was implemented to facilitate management of multiple threads within a single context.

4.8.8 Context Initialization

The ideal place for all context initialization is in the Master Context—and done once for all contexts during power-up. The following features associated with all used contexts would be set up:

- Priority, mode, and initial state
- Stacks, initial program counter, and vector base table
- Associated interrupts (external and internal)
- Context timer setup (if desired)
- Memory protection (if desired)

Because most of the above features require writing to access-controlled registers, perform this processing from the Master Context in supervisor mode.

4.8.9 Context Claim Registers

The context claim register set is used to share resources between or among contexts, while still retaining priority determinism and preventing priority inversion. Priority inversion is defined as any condition where a lower-priority task runs before a higher-priority task. If a lower-priority task is running initially and locks a shared resource, and then a higher-priority task begins to run needing the same resource, a method is needed to allow the lower-priority task to run just long enough to finish using the resource. This is a common problem for software operating systems. The fido addresses it with hardware using the register set shown in Table 4-11.

Table 4-11. Description of Context Claim Register

Register Name	Description
Context Claim Register Context Priority Inheritance Register Pending Context Register	A three-register set used to manage context execution when sharing resources among contexts. One register set is for each context except the Master Context. Only the Context Claim register is writable, the others are set by the execution engine based on events.

The Context Claim register is used to define a shared resource. It is the only software-writable register. The other registers are set by hardware based upon events. The value written to the Context Claim Register is an arbitrary assignment (except for 0 and 0xFFF). It would represent the object ID of a shared resource that is used in more than one context. See the example in Table 4-12.

Table 4-12. Example of Context Claim Register Implementation

Context	State	Priority
1	NOT_RDY	3
2	NOT_RDY	2
3	RDY	1

Initially, Context_3 is running. It writes 0x55 to its claim register, gets the claim, and continues to run. Context_1 receives a wakeup event, transitions to RDY and begins to run, and writes 0x55 to its claim register. Since object ID 0x55 is locked (write will not take), Context_1 is now pending on Context_3. The Ctx_1 pend bit is set (in Context_3's Pending Context Register), preventing it from running even though it is now in a RDY state. Simultaneously, because of the Context_1 claim, Context_3 now inherits Priority 3, and begins to run again. It runs, and writes a 0 to its claim register when finished with the resource.

Note 1: If Context_2 receives a wakeup event prior to Context_3 relinquishing its claim, Context_2 will not run because Context_3 is running at the inherited Priority 3, which prevents inversion.

When Context_3 writes a 0 to its claim register, the Ctx1_pend bit is cleared, Context_3 returns to Priority 1, and Context_1 runs.

Note 2: If Context_2 wakes up while Context_3 is running, it runs and posts a 0x55 claim. Context_3 will run at priority 2 while Context_2 pends in a RDY state. Then if Context_1 wakes up, runs, and posts a third 0x55 claim, Context_3 will resume at Priority 3. When Context_3 relinquishes its claim, Context_1 will run ahead of Context_2 because it is higher in priority than Context_2.

Software does not need to do anything to manage the context pending or priority inheritance, it only needs to set overall context priority and define the shared resource IDs. The hardware takes care of the rest and does so within a single clock cycle. See the Context Claim Register Set detailed definition for more information.

4.8.10 Software Interrupt Control and Actuation Registers

This register set can be used for inter-context communication and resource sharing. It allows a mailbox interrupt to pass messages or manage control. Each context has a pair of registers to implement this feature (see Table 4-13).

Table 4-13. Description of Software Interrupt Control and Actuation Registers

Register Name	Description
S/W Interrupt Control Register S/W Interrupt Actuation Register	Register pair used to communicate between contexts using mailbox-like interrupts—one register pair for each context. The interrupt control register is access-controlled and writable by only the Master Context and the owner context. The actuation registers are read/write to any context.

Any context can write to the S/W Interrupt Actuation Register of another context. A context can also write to its own if desired. The interrupt generated to the other context is a wakeup event subject to the rules of other interrupts. The Core Status Register priority mask, the priority of the S/W Interrupt for the interrupted context, and the context priority of the interrupted context control whether or when the interrupted context runs.

See Section 4.8.20, Software Interrupt Control Register, and Section 4.8.21, Software Interrupt Actuation Register, for more information.

4.8.11 Context Management Registers

This section contains the context-management register definitions. For addresses of all registers, see the register memory map in Chapter 12, Register Map Reference. These are all directly addressable memory-mapped registers, subject to the Memory Base Offset Register. These registers are defined in Table 4-14.

Table 4-14. Description of Context Management Registers

Register Name	Description
Context Control Register	Defines context type, priority, and execution state—one for each context. Is an access-controlled register and writable by only the Master Context (Context_0).
Context Timer Enable Register	Overall enable and mode control register used to manage context timers. A single register and an access-controlled register and writable by only the Master Context (Context_0).
Context Timer Register	Holds the value measuring context execution time—one for each context. Read/write by owner context.
Context Max Time Register	Used to limit context execution time in time-slice mode—one for each context. Is an access-controlled register and writable by only the Master Context (Context_0).
Context Timer Clear Register	Used to clear the Context Timer Register. A single register and an access-controlled register and writable by only the Master Context (Context_0).
Context Idle Timer Register	Timer used to track time during which no context is active. A single, read-only register.
Context Claim Register Context Priority Inheritance Register Pending Context Register	Three-register set used to manage context execution when sharing resources among contexts. One register set for each context *except* the Master Context. Only the Context Claim register is writable, the others are set by the execution engine based on events.
S/W Interrupt Control Register S/W Interrupt Actuation Register	Register pair used to communicate between contexts using mailbox-like interrupts. One register pair for each context. The interrupt control register is writable by the Master Context (Context_0) and the owner context. The actuation registers are read/write to any context.

4.8.12 Context Control Register

The context control register is the basic context definition register. This register is access-controlled and writable by only the Master Context. It specifies context control and status. There is one context control register for each context (see Table 4-15).

Table 4-15. Context Control Register

Context Control Register						
31–16	15	14–13	12–11	10–8	7–2	1–0
Reserved	Reserved	Interrupt Mode	Reserved	Priority	Reserved	State

–	R	RW	R	RW	–	RW

Note: Reset value for Context_0 is 0x1F02; for other contexts is 0x1800.

- Bits [31–16]—Reserved
- Bit [15]—Reserved, always reads 0
- Bits [14–13]—Defines the type of response the context makes to a pending interrupt
  - 00 → Standard mode. Operates like a standard microprocessor. If interrupt is of higher than or equal priority than the current interrupt mask in context status register, then the current state of the context is stacked and execution is begun at the interrupt vector.
  - 01 → Fast mode. All interrupts are masked while the context is in the Ready state, when a context is in the NOT READY state, an interrupt of priority higher than or equal to the mask in the status register for that context will cause a vector fetch and execution begins. Nothing is stacked.
  - 11 → Fast single-threaded mode. All interrupts are masked while the context is in the Ready state, when a context is in the NOT READY state, an interrupt of priority higher than or equal to the mask in the status register for that context will cause execution to begin at the current program counter location for that interrupt. Nothing is stacked—no vector fetch is performed.
- Bits [12–11] → Reserved, always read back 1 (future expansion for priority)
- Bits [10–8] → Priority of the context (0–7) where 7 is highest, 0 is lowest
- Bits [7–2] → Reserved, always reads 0
- Bits [1–0] → Execution state of the context
  - 00 → HALTED. The context will not execute when in the halted state. Only a write to this register can move a context out of this state.
  - 01 → NOT READY. The context can be made ready by any interrupt/exception targeted to it.
  - 10 → READY. The context is ready to run and will be placed into execution when it is the highest-priority ready context.

4.8.13 Context Timer Enable Register

The Context Timer Enable Register enables the context timing function on a context-by-context basis. Both the mode (standard versus time-slice) and the enable/disable of the context timer are controlled here. This register is access-controlled and writable by only the Master Context (see Table 4-16).

Table 4-16. Context Timer Enable Register

31	30–21	20	19	18	17	16	15–5	4	3	2	1	0
ENI	Reserved	Mode4	Mode3	Mode2	Mode1	Mode0	Reserved	EN4	EN3	EN2	EN1	EN0
RW	–	RW	RW	RW	RW	RW	–	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bit [31]—Write of “1” enables the Context Idle Timer, write of 0 disables, a read returns its status.
- Bits [30–21]—Reserved.
- Bit [20]—Context_4 timer mode control, write of 0 specifies standard context timer mode, write of 1 specifies time slice mode, a read returns current mode.
- Bit [19]—Context_3 timer mode control, write of 0 specifies standard context timer mode, write of 1 specifies time slice mode, a read returns current mode.
- Bit [18]—Context_2 timer mode control, write of 0 specifies standard context timer mode, write of 1 specifies time slice mode, a read returns current mode.
- Bit [17]—Context_1 timer mode control, write of 0 specifies standard context timer mode, write of 1 specifies time slice mode, a read returns current mode.
- Bit [16]—Context_0 timer mode control, write of 0 specifies standard context timer mode, write of 1 specifies time-slice mode, a read returns current mode.

Note: Concerning the mode bits [20–16]: When a timer is in standard mode, a context timer > the maximum time will generate a priority 6 overtime fault to the Master Context. When a timer is in time-slice mode, a context timer > the maximum time will fault to the local context which will remain in a RDY state. It is anticipated the fault handler for this context will be responsible for scheduling activity.

- Bits [15–5]—Reserved.
- Bit [4]—Context_4 timer enables, write of 1 enables, write of 0 disables, a read returns current status.
- Bit [3]—Context_3 timer enables, write of 1 enables, write of 0 disables, a read returns current status.

- Bit [2]—Context_2 timer enables, write of 1 enables, write of 0 disables, a read returns current status.
- Bit [1]—Context_1 timer enables, write of 1 enables, write of 0 disables, a read returns current status.
- Bit [0]—Context_0 timer enables, write of 1 enables, write of 0 disables, a read returns current status.

4.8.14 Context Timer Register

Associated with each context is a 24-bit timer used to measure/control execution time. These registers operate in two distinct modes as indicated in the Mode field of the Context Timer Enable Register.

4.8.14.1 Standard Mode Operation

When the Mode field of the Context Timer Enable Register is in Standard Mode, the following applies. The timer increments when the context is active and the associated EN flag of the Context Timer Enable Register is a 1. The timer remains constant when the context is inactive or the EN flag is 0. The register is cleared whenever the associated CLR flag of the Context Timer Clear Register is written with a 1. A write to the register has no effect. The register is cleared whenever the associated CLR flag of the Context Timer Clear Register is written with a 1. The register is read-only in standard-mode operation. See also Section 4.8.15, Context Maximum Time Register, for instruction on the use of this register as a means for guaranteeing time-partitioning.

4.8.14.2 Time-Slice Mode Operation

When the mode field of the Context Timer Enable Register is in Time-Slice Mode, the following applies. The timer increments when the context is active, the associated EN flag of the Context Timer Enable Register is a 1, *and* the supervisor flag of the status register is a 0 (in user mode). This means that only user-mode execution is timed. Interrupt service code is not counted.

A write to this register (in time-slice mode) is only valid when in supervisor mode (user-level writes have no effect), the value written is then used to carry on with counting. This feature is used to allow multiple software contexts to use the time-slice feature (at various priorities) and to allow a given task to give up the remainder of its slice (see Table 4-17).

Table 4-17. Context Timer Counter Register

31–24	23–0
Reserved	Count
–	RW

Note: Reset value is 0x00000000.

- Bits [31–24]—Reserved
- Bits [23–0]—Count → When active, the counter is incremented every 16 external clock cycles, for example:
  - 66-MHz base frequency
    - Counter LSB = 240 nsec
    - Counter Period (max) = 4.03 seconds
  - 50-MHz base frequency
    - Counter LSB = 320 nsec
    - Counter Period (max) = 5.36 seconds

4.8.15 Context Maximum Time Register

This register is duplicated for each context. These registers are access-controlled and writable by only the Master Context.

This register is used to generate a fault based on execution time for a context. The run time desired to be assigned to the context is written in this register with the LSB having the same scaling as that in the Context Timer Register. The lower 9 bits of this register are ignored (always return 0), while bits 23–9 are compared to the Context Timer Register. If the value in the Context Timer Register is greater than or equal to the value in the Context Maximum Time Register, what happens next depends upon the context timer mode:

- Standard Mode: The context is halted and a context overtime fault is generated to the Master Context (with the exception that if the overrunning context is the Master Context, then the context is not halted but the fault is generated). The context ID is placed in the Faulted Context Register.
- Time-Slice Mode: The context is not halted. A context overtime fault is generated to the local context of operation (for scheduling purposes).

If the value in the Context Maximum Time Register is zero then this feature is disabled (see Table 4-18).

Table 4-18. Context Maximum Time Register

31–24	23–9	8–0
Reserved	Max Value	Reserved
–	RW	–

Note: Reset value is 0x00000000.

- Bits [31–24]—Reserved
- Bits [23–9]—Max Value , for example:
  - System clock speed 50 MHz
    - Minimum timeout 163.8 µsec
    - Maximum timeout 5.36 seconds
  - System clock speed 66 MHz
    - Minimum timeout 124.1 µsec
    - Maximum timeout 4.03 seconds
- Bits [8–0] → Reserved, always read back 0

4.8.16 Context Timer Clear Register

The Context Timer Clear Register provides a single place to clear (reset) the context timers on an individual basis. This register is access-controlled and writable by only the Master Context (see Table 4-19).

Table 4-19. Context Timer Clear Register

31	30–5	4	3	2	1	0
CLRI	Reserved	Clr4	Clr3	Clr2	Clr1	Clr0
RW	–	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bit [31]—Write of 1 clears the Context Idle Timer, always reads 0
- Bits [30–5]—Reserved
- Bit [4]—Write of 1 clears Context_4 Timer Register, always reads 0
- Bit [3]—Write of 1 clears Context_3 Timer Register, always reads 0
- Bit [2]—Write of 1 clears Context_2 Timer Register, always reads 0
- Bit [1]—Write of 1 clears Context_1 Timer Register, always reads 0
- Bit [0]—Write of 1 clears Context_0 Timer Register, always reads 0

4.8.17 Context Idle Timer Register

This is a 24-bit timer that measures the time during which no context is active (effectively equivalent for idle). This register is READ ONLY. The timer increments when all the contexts are inactive and the ENI flag of the Context Timer Enable Register is a 1. If any context is active or the ENI flag is a 0, the timer remains constant. The register is cleared whenever the CLRI flag of the Context Timer Clear Register is written with a “1” (see Table 4-20).

Table 4-20. Context Idle Timer Register

31–24	23–0
Reserved	Count
–	R

Note: Reset value is 0x00000000.

- Bits [31–24]—Reserved

- Bits [23–0]—Count → When active, the counter is incremented every 16 external clock cycles, for example:
  - 66-Mhz base frequency
    - Counter LSB = 240 nsec
    - Counter Period (max) = 4.03 seconds

  - 50-Mhz base frequency
    - Counter LSB = 320 nsec
    - Counter Period (max) = 5.36 seconds

4.8.18 Context Claim Register Set

A problem facing any system is how to manage limited resources among numerous threads of execution. This is no less true with a hardware kernel. While this can be managed in software by the Master Context (which can modify priorities, sleep contexts, etc.), the overhead is not desirable, also it effectively locks out any intermediate priority contexts (intermediate between the Master Context and the context[s] whose priority is being manipulated) while the manipulations are taking place.

To manage this problem, the following registers are defined for each context except the Master Context Table 4-21).

Table 4-21. Context Claim Priority Inheritance Register

31–3	2–0
Reserved	Priority
–	R

The value in this register overrides the value in the Context Control Register if it is greater (higher priority). This register is not writable by software, it has a reset value of 0x000 (lowest priority). See Table 4-22 for how other values are loaded into it.

Table 4-22. Pending Contexts Register

31–5	4	3	2	1	0
Reserved	Ctx4 pend	Ctx3 pend	Ctx2 pend	Ctx1 pend	Reserved
–	R	R	R	R	–

If one of the flags in this register is set then the corresponding context is held from execution until the flag is cleared. This register is not writable by software. See Table 4-23 for how it is modified.

Table 4-23. Claim Register

31–12	11–0
Reserved	Object ID
–	RW

Note: Reset value is 0x00000000.

- Bits [31–12]—Reserved
- Bits [11–0]—Object ID

This register can be written according to the following:

- A write of a non-zero value by the context associated with the register.
  - If no other Claim register (no claim register associated with another context) contains the same value then the value is written into the Object ID field.
  - If another Claim register has the same value already in its Object ID field then:
    - No write takes place to the Claim register
    - In the registers for the context that has already "claimed" the value, the flag corresponding to the current context is set in the pending contexts register.
    - If the priority of the current context is higher than that in the Priority Inheritance Register of the context that holds the claim then the current priority overwrites it (raising the priority of the context that holds the claim).

Note: This mechanism is used by fido to handle the problem of priority inversion, and allows a higher priority context to temporarily yield to a lower priority context until a shared resource is released.

Take the following example assuming these initial conditions (see Table 4-24).

Table 4-24. Claim Register Example

Context	State	Priority
1	NOT_RDY	3
2	NOT_RDY	2
3	RDY	1

Initially, Context_3 is running. It writes 0x55 to its claim register, gets the claim and continues to run. Context_1 receives a wakeup event, transitions to RDY and begins to run, and writes 0x55 to its claim register. Because object ID 0x55 is locked, the write does not take and Context_1 is now pending on Context_3. The Ctx_1 pend bit is set (in Context_3's Pending Context Register); preventing it from running even though it is now in a RDY state. Simultaneously, Context_3 now inherits Priority 3, and begins to run again. It runs, and writes a 0 to its claim register when finished with the resource.

Note 1: If Context_2 receives a wakeup event prior to Context_3 relinquishing its claim, Context_2 will not run because Context_3 is running at the inherited Priority 3, which prevents inversion.

When Context_3 writes a 0 to its claim register, the Ctx1 pend bit is cleared, Context_3 priority returns to 1, and Context_1 runs.

Note 2: If Context_2 wakes up while Context_3 is running, it runs and posts a 0x55 claim, Context_3 will run at inherited priority 3 while Context_2 pends in a RDY state. Then if Context_1 wakes up, runs, and posts a third 0x55 claim, Context_3 will resume at priority 3 while both Context_1 and Context_2 pend in a RDY state. When Context_3 relinquishes its claim, Context_1 will run ahead of Context_2, based on its higher priority.

- A write of zero by the context associated with the register.
  - The Priority Inheritance Register cleared, returning precedence to the Context Control Register
  - The Claim Register is cleared, relinquishing the context's claim on the object
  - The Pending Contexts Register is cleared, freeing for execution any pending contexts

- A write of all ones by the context associated with the register.
  - The Priority Inheritance Register is cleared, returning precedence to the Context Control Register
  - The Claim Register is cleared, relinquishing the context's claim on the object
  - The Pending Contexts Register is cleared, freeing for execution any pending contexts
  - The context associated will be put to the "NOT READY" state

- A write by a context not associated with the register
  - No effect

4.8.19 Software Interrupt Register Set

Associated with each context is a pair of registers that allows an interrupt to be generated by software action. Any context (including the context with which the registers are associated) that writes to the appropriate one will cause an interrupt (assuming it is enabled and of sufficient priority). These registers can be used to facilitate context communication and provide for another method to manage shared resources. The two registers consist of a control register and an actuation register.

4.8.20 Software Interrupt Control Register

These registers (one for each context) provide a means to control the software interrupt for the associated hardware context. In addition, the priority (relative to the SR for that context) is controlled here. This register is access-controlled and writable by only the Master Context and the owner context (see Table 4-25).

Table 4-25. Software Interrupt Control Register

31–14	13	12	11–8	7–5	4–0
Reserved	Enable	Status	Reserved	Priority	Reserved
–	RW	R	–	RW	–

Note: Reset value is 0x00000000.

- Bits [31–14]—Reserved

- Bit [13]—Interrupt enable bit for this context (1=enabled, 0=disabled)

- Bit [12]—Interrupt status bit for this context (read this bit to determine status), 1=interrupt pending, a read will clear the bit, acknowledging the interrupt

- Bits [11–8]—Reserved

- Bits [7–5]—This field allows the priority of the interrupt channel to be assigned, 0 is lowest and 7 is highest

- Bits [4–0]—Reserved

4.8.21 Software Interrupt Actuation Register

These registers (one for each context) provide a means to generate an interrupt to any context via software. A write of any value will generate the interrupt action. The register can be used by software as a mailbox for identifying the source of the interrupt (Table 4-26).

Table 4-26. Software Interrupt Actuation Register

31–0
Signature Data
RW

5. Memory Management and Protection

5.1 Overview

The fido1100 has built-in memory and memory management/protection features. This section will cover the following aspects of the fido1100:

- The fido1100 internal SRAM
- The fido1100 internal Relocatable Rapid Execution Memory (RREM)
- Endian Mode Control
- MOVEC Access Based Registers
- Memory and Register Group Address Map
- Memory Protection Unit
- Programmable Chip Select Registers
- Complete Register Address Map

5.2 The fido1100 Internal Memory and Registers

The fido1100 provides a general-purpose use 24-Kbyte Internal SRAM, a 32-Kbyte RREM that improves execution speed of heavily used code segments, and an MPU to control access to blocks of memory on a context-by-context basis. It also has an Endian Mode Control feature to perform byte swapping automatically that is controllable using the most significant bit (MSB) of the address. The fido1100 interfaces to external memory via chip-select control registers and an on-board SDRAM controller (see Figure 5-1).

5.3 Internal SRAM

The fido1100 provides a general-purpose use 24-Kbyte Internal SRAM. This directly addressable RAM is intended for frequently accessed data. Because the memory is internal, accesses are much quicker than using the external bus. There are no restrictions on how to use this RAM. Some ideas follow and are application-dependent.

- Stacks
- Data tables
- Key data structures
- Any other frequently accessed data where time constraints are critical

Note: The base address of Internal SRAM is set by the value in the Memory Base Offset (MBO) register.

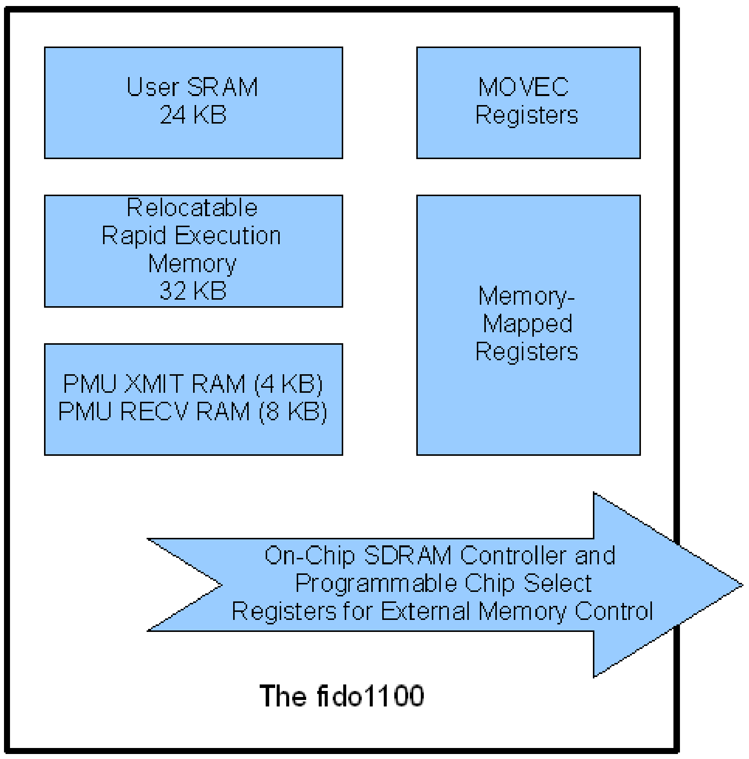


Figure 5-1. The fido1100 Internal Memory and Registers

5.4 Internal Relocatable Rapid Execution Memory

The fido1100 provides a 32-Kbyte relocatable rapid execution memory (RREM) targeted for code execution. Sections of code that are high frequency in use can be copied from external memory (flash, for example) to this internal memory, where they will execute more quickly. The Relocatable RAM Control Register is used to re-map the external address block to a RREM block after the copy has been done, thus overlaying the two address blocks. When the CPU issues an address to fetch an instruction that would have resulted in an external bus cycle, a faster internal bus cycle occurs instead. The advantage of this scheme is that the software does not need to know whether it is executing a piece of code from RREM or external memory. A re-mapping example is provided in Table 5-1, Example of Re-Mapping Relocatable RAM Memory.

The RREM is 32 Kbyte of internal memory partitioned into 16, 2-Kbyte segments. Each segment is independently mappable via the RREM Control Registers to overlay an external memory area on 2-Kbyte boundaries (this is the only restriction). There are 16 Relocatable RAM Control Registers, one for each block of Relocatable RAM. They are access-controlled and writable by only the Master Context. Typically, the Master Context would re-map the desired code sections to RREM during the initialization sequence (see Table 5-1).

Table 5-1. Relocatable RAM Control Register

31–11	10–1	0
Base Address	Reserved	EN
RW	–	RW

Note: Reset value is 0x00000000.

- Bits [31–11]—Base address of memory block (on a 2-Kbyte boundary) that will be overlaid to RREM.

Note: This base address value must not overlap any fido1100 internal memory (User SRAM, register space, etc.).

- Bits [10–1]—Reserved, set to 0
- Bit [0]—Enable bit for RREM block (0=disabled, 1=enabled)

Note: The Relocatable RAM memory can be loaded like any other until any of the enable flags are set. Once any block is enabled, the only path to access all blocks is through fetching instructions (the memory can only be used for code, not for operand space). Thus all blocks to be used must be set up prior to enabling any block.

5.5 Re-Mapping Example

Assuming an 8112-byte block of code to accelerate. It begins at address 0x22_1248 in flash memory and is contiguous. The steps to map this to RREM are presented in Table 5-2.

Table 5-2. Example of Re-Mapping Relocatable RAM Memory

Step No.	Description
1	Set sourceAddrPtr=0x221000 (must adhere to 2-Kbyte address boundary).
2	Set destAddrPtr=0x180000 (base address of RREM assuming MBO register = 0x100000).
3	Set numWordsToCopy=0x2000 (8192 bytes) and perform the copy from source memory to RREM. At this point, the code is address aligned in RREM.
4	Set Relocatable RAM Control Register_0 to 0x221001.
5	Set Relocatable RAM Control Register_1 to 0x221801.
6	Set Relocatable RAM Control Register_2 to 0x222001.
7	Set Relocatable RAM Control Register_3 to 0x22_801. At this point, all of the 8112 bytes of code are now re-mapped and enabled to operate out of RREM. There are also some bytes of code both before and after the 8112-byte block that will execute out of RREM.

Note that in order to comply with the 2-Kbyte boundary, there will be 0x248 bytes of code prior to the desired 8112-byte block, and 0x50 bytes of code after the desired block that has an address range also mapped to RREM. Once a RREM block is enabled, all instruction fetches within the 2-Kbyte address range it is configured for are made from RREM. An additional concern exists at addresses 0x221000 and 0x222FFF. Given the multi-word nature of many CPU instructions, it is possible for part of an instruction to be in flash and the other part in RREM. This will cause an instruction fault and should be avoided.

The above example shows how to re-map code to RREM, and points out some potential issues with using RREM. The best way to use RREM safely and effectively is to use the compiler/linker to force critical blocks of code to 2-Kbyte, aligned-address boundaries. This will result in reduced memory waste and avoids an instruction fault.

5.6 Endian Mode Control

The fido1100 has a 32-bit physical memory mapped address space, with Bit [31] of the address acting as a little-endian/big-endian control mode bit. This address space covers all internal and external resources available to the fido1100. Internal resource examples are register banks, internal peripherals, internal User SRAM, RREM, and PMU Transmit and Receive Buffers. External resources are any address not mapped as an internal resource. Examples of external resources would be flash memory, SDRAM memory, and external peripherals.

5.7 Definitions

- Big Endian—The most significant data byte of multi-byte data operand is stored at the least significant address. The subsequent address stores next to the most significant byte and so on.
- Little Endian—The least significant data byte of multi-byte data operand is stored at the least significant address. The subsequent address stores next to the least significant byte and so on.
- Byte → 8 bits (moveb instruction)
- Word → 16 bits (movew instruction)
- Long word → 32 bits (movel instruction)

This feature provides the user with a method to read/write data efficiently in a little-endian byte order (even though the fido1100 is a big-endian machine). The implementation is that all memory accesses to addresses greater than 0x7fffffff (i.e., with address-Bit [31] set) will reverse the byte order of the read/written data as follows:

- Byte Access—No change

- Word Access–Swaps bytes 0 and 1
- Long-Word Access—Swaps bytes 0 and 3, swaps bytes 1 and 2 (same as swapping 0 and 1, swapping 2 and 3, then swapping words)

Examples of a word write (16-bit) and a long-word write (32-bit) are provided in Tables 5-3 and 5-4.

Table 5-3. Word Write Operation Example (Original Data in Register 1 = 0x1234)

Column Index	1	2
Effective Address	0x00000000 (Big Endian)	0x80000000 (Little Endian)
Memory (byte location) at address 0 gets	12	34
Memory (byte location) at address 1 gets	34	12

Table 5-4. Long-Word Write Operation Example (Original Data in Register 1 = 0x12345678)

Column Index	1	2
Effective Address	0x00000000 (Big Endian)	0x80000000 (Little Endian)
Memory (byte location) at address 0 gets	12	78
Memory (byte location) at address 1 gets	34	56
Memory (byte location) at address 2 gets	56	34
Memory (byte location) at address 3 gets	78	12

The side effects of the endian-mode control feature are:

- Only 31 address pins are available externally
- The address space has 2 rather than 4 Gbytes

5.8 MOVEC Access-Based Registers

A set of special registers in the fido1100 are accessible only using the MOVEC instruction. Table 5-5 summarizes these registers and their address for the MOVEC instruction.

Table 5-5. Summary of MOVEC Access-Based Registers

Name	MOVEC Control Code	Major-Reset Affect	Minor-Reset Affect	Reset-Instruction Affect	Read Register Mapping	Notes
Source Function Code (SFC)	0x000	Set to 0x0000	Unaffected	None	N/A	Refers to SFC for current context.
Destination Function Code (DFC)	0x001	Set to 0x0000	Unaffected	None	N/A	Refers to DFC for current context.
User Stack Pointer (USP)	0x800	Context-Dependent	Unaffected	None	N/A	Refers to User Stack Pointer for current context.
Vector Base Register (VBR)	0x801	Set to 0x0000	Unaffected	None	N/A	Refers to Vector Base Register for current context.
Configuration Access Control (CAC)	0xFFE	Set to 0x00000000	Unaffected	None	0xFFFE	Only accessible from Master Context (Context_0).
Memory Base Offset (MBO)	0xFFF	Set to 00100000	Unaffected	None	0xFFFF	Only accessible from Master Context (Context_0).

Detailed descriptions of all MOVEC mapped registers follow.

5.9 Source Function Code Register (0x000)

This register has been retained in support of the CPU32 architecture in conjunction with the MOVES instruction to provide eight separate address spaces. *This feature is not supported on the fido1100.* However, the registers are available for reading and writing to provide compatibility with legacy CPU32 code (but they have no functional effect) (see Table 5-6).

Table 5-6. Source Function Code Register

31–3	2–0
Reserved	FC0
–	RW

Note: Reset value is 0x00000000.

- Bits [31–3]—Reserved
- Bits [2–0]—FC0 value (unused, has no effect)

Note: Via the MOVEC instruction, any context can set its own SFC register as desired. This register also shows up in the memory-mapped register address space, where it is directly accessible, but only by the Master Context.

5.10 Destination Function Code Register (0x001)

This register has been retained in support of the CPU32 architecture in conjunction with the MOVES instruction to provide eight separate address spaces. *This feature is not supported on the fido1100.* However, the registers are available for reading and writing to provide compatibility with legacy CPU32 code (but they have no functional effect) (see Table 5-7).

Table 5-7. Destination Function Code Register

31–3	2–0
Reserved	FC1
–	RW

Note: Reset value is 0x00000000.

- Bits [31–3]—Reserved
- Bits [2–0]—FC1 value (unused, has no effect)

Note: Via the MOVEC instruction, any context can set its own DFC register as desired. This register also shows up in the memory-mapped register address space, where it is directly accessible, but only by the Master Context.

5.11 User Stack Pointer Register (0x800)

This register is the user mode stack pointer for the context. Recall that each context has its own stack, (for both user and privileged mode) (see Table 5-8).

Table 5-8. User Stack Pointer Register

31–0
User SP Address
RW

Note: Reset value is 0x00000000.

- Bits [31–0]—Current user mode SP address

Note: Via the MOVEC instruction, any context can set its own User SP register as desired. This register also shows up in the memory-mapped register address space, where it is directly accessible, but only by the Master Context.

5.12 Vector Base Register (0x801)

This register is the base address for the context vector table. Recall that each context has its own interrupt/exception table. The vector base address must be on a 1024-byte boundary (256 possible vectors at 4 bytes each) (see Table 5-9).

Table 5-9. Vector Base Register

31–10	9–0
Vector Base Address	Reserved, set to 0
RW	–

Note: Reset value is 0x00000000.

- Bits [31–10]—Interrupt/exception vector table address
- Bits [9–0]—Reserved, set to 0 for boundary restriction

Note: Via the MOVEC instruction, any context can set its own vector base register as desired. This register also shows up in the memory-mapped register address space, where it is directly accessible, but only by the Master Context.

5.13 Configuration Access Control Register (0xFFE)

This register has a single field that controls access to the processor registers that are access-controlled. The Master Context is the only context that can write to access-controlled registers, and this register can prevent (or allow) even Master Context write ability. Only the Master Context can write to the Configuration Access Control (CAC) register (see Chapter 11, Access-Controlled Registers).

At power up, this register defaults to 0 such that the Master Context can write to all access-controlled registers. If the application has some safety requirement to guarantee certain registers are never changed after initialization, the Master Context could perform the initialization then lock itself out via the Lock bit (see Table 5-10).

Table 5-10. Configuration Access Control Register

31–1	0
Reserved	Lock
–	RW

Note: Reset value is 0x00000000.

- Bits [31–1]—Reserved

- Bit [0]—Lock bit
  - 0 → Write access is allowed to configuration register space (for Master Context only)
  - 1 → Write access disabled to configuration register space

5.14 Memory Base Offset Register (0xFFF)

All internal fido1100 resources are located relative to the value of the Memory Base Offset (MBO) Register. The MBO allows the internal resources of the fido1100 to be conveniently located anywhere in the 31-bit physical address of the fido1100 on a 1-Mbyte boundary.

Internal addresses are OR-ed with the value in the MBO Register to adjust bits 30–20. For example, the MBO Register has a value of 0x00100000 at POR, thus the internal User SRAM which has an offset of 0x00000000 would be accessed at address 0x00100000. If the MBO Register was changed to address 0x70F00000, then the internal User SRAM which has an offset of 0x00000000 would be accessed at address 0x70F00000.

The Master Context would typically set this register early in the initialization sequence, as it would likely access many of the internal registers.

Table 5-11. Memory Base Offset Register

31–20	19–0
Offset	Reserved
RW	–

Note: Reset value is 0x00100000.

- Bits [31–20]—Forms the upper 12 bits of address decode

Note: The value of this field should never be set to 0 as it will cause a conflict with User SRAM to overlay the flash address area.

- Bits [19–0] – Reserved and are set to 0

5.15 Memory and Register Group Address Map

Table 5-12, Memory and Register Group Address Map Table, lists the default address range for memory and registers. It is important to note that the address shown (other than CS0 and SDRAM) refer to internal locations and will be offset by the default value of the Memory Base Offset register. The default POR value of the Memory Base Offset register is 0x00100000. For example, if the Memory Base Offset was 0x00100000, the DMA Control Register would be at address 0x001A0000. The offset programmed into the Memory Base Offset register applies to every internal register shown in the Complete Register Address Map Table. All addresses refer to 32-bit registers. Addresses with the same start and finish values indicate a single 32-bit register for the specified function.

Note: The user must not write or attempt reads from any address in the reserved rows. Accessing these locations will result in indeterminate read or write behavior.

Table 5-12. Memory and Register Group Address Map Table

Description	Allocation	Address Range	Notes
CS0 (Boot Area)	256 Kbytes	0x00000000–0x0003FFFF	1, 2, 7, 8, 10
User SRAM	24 Kbytes	0x00100000–0x00105FFF	2, 4
RREM	32 Kbytes	0x00180000–0x00187FFF	2, 4
DMA control registers	8 x 32	0x001A0000–0x001A001C	3, 4
Available	128 x 32	0x001A0080–0x001A027C	4, 12
TCU control registers	32 x 32	0x001A0280–0x001A02FC	3, 4
System Timer registers	7 x 32	0x001A0300–0x001A0318	3, 4
Watchdog Timer registers	2 x 32	0x001A0340–0x001A0344	3, 4
POR Reset Register	1 x 32	0x001A0360	3, 4
Available	7 x 32	0x001A0364–0x001A037C	4, 12
Clock Mask Register	1 x 32	0x001A0380	3, 4
Available	31 x 32	0x001A0384–0x001A03FC	4, 12
Device ID Register	1 x 32	0x001A0400	3, 4, 11
Debug Control Registers	35 x 32	0x001A0404–0x001A048C	3, 4
ADC control registers	10 x 32	0x001A0600–0x001A0624	3, 4
BIU control registers	16 x 32	0x001A0680–0x001A06BC	3, 4
BIU Priority register	1 x 32	0x001A0700	3, 4
BIU Default Timing register	1 x 32	0x001A0704	3, 4
Available	62 x 32	0x001A0708–0x001A07FC	4, 12
SDRAM Controller Registers	12 x 32	0x001A0800–0x001A082C	3, 4, 9
Interrupt Control Registers	8 x 32	0x001A0900–0x001A091C	3, 4
Software Interrupts	5 x 32	0x001A0980–0x001A0990	3, 4
Available	27 x 32	0x001A0994–0x001A09FC	4, 12
PMU control registers	142 x 32	0x001A0A00–0x001A0C34	3, 4
PMU Reserved	304 x 32	0x001A0C40–0x001A10FC	4, 12, 13
Context Claim Registers	20 x 32	0x001A1100–0x001A114C	3, 4
Available	7148 x 32	0x001A1150–0x001A80FC	4, 12
Context_0 Core Registers	22 x 32	0x001A8100–0x001A8154	3, 4
Context_1 Core Registers	22 x 32	0x001A8180–0x001A81D4	3, 4
Context_2 Core Registers	22 x 32	0x001A8200–0x001A8254	3, 4
Context_3 Core Registers	22 x 32	0x001A8280–0x001A82D4	3, 4
Context_4 Core Registers	22 x 32	0x001A8300–0x001A8354	3, 4
Reserved	27 x 22 x 32	0x001A8380–0x001A90FC	4, 12
Available	960 x 32	0x001A9100–0x001A9FFC	4, 12
MPU Block Control Registers	16 x 32	0x001AA000–0x001AA03C	3, 4
MPU Attribute Registers	16 x 32	0x001AA080–0x001AA0BC	3, 4

Table 5-12. Memory and Register Group Address Map Table *(Continued)*

Description	Allocation	Address Range	Notes
MPU Allocation Registers	5 x 32	0x001AA100–0x001AA110	3, 4
RREM Control Registers	16 x 32	0x001AA180–0x001AA1BC	3, 4, 6
Available	128 x 32	0x001AA200–0x001AA3FC	4, 12
Context Control Registers	5 x 32	0x001AA400–0x001AA410	3, 4
Context Max Time Registers	5 x 32	0x001AA480–0x001AA490	3, 4
Context Timer Registers	5 x 32	0x001AA500–0x001AA510	3, 4
Context Timer Clear Reg	1 x 32	0x001AA580	3, 4
Context Idle Timer	1 x 32	0x001AA584	3, 4
Context Timer Enable Reg	1 x 32	0x001AA588	3, 4
Available	29 x 32	0x001AA58C–0x001AA5FC	4, 12
Faulted Context ID	1 x 32	0x001AA600	3, 4, 11
Current Context ID	1 x 32	0x001AA604	3, 4, 11
Available	5758 x 32	0x001AA608–0x001AFFFF	4, 12
PMU Transmit Memory	1 Kbyte x 32	0x001B0000–0x001B0FFF	3, 4
PMU Receive Memory	2 Kbyte x 32	0x001B4000–0x001B5FFF	3, 4
UIC0 control registers	82 x 32	0x001B8000–0x001B8144	3, 4
UIC1 control registers	82 x 32	0x001B8200–0x001B8344	3, 4
UIC2 control registers	82 x 32	0x001B8400–0x001B8544	3, 4
UIC3 control registers	82 x 32	0x001B8600–0x001B8744	3, 4
UIC Reserved	82 x 32	0x001B8800–0x001B8F44	3, 4, 13
SDRAM	128 Mbytes	0x00000000–0x07FFFFFF	2, 7, 8, 9

Notes:
1. Must contain reset vector and initial exception vector table, max allocation assumes no relocation of remaining register map.
2. May be addressed as bytes, words, or long words.
3. All registers are accessed on long-word aligned boundaries, even if they are not 32-bit registers. Only the registers defined in the Complete Register Address Map Table will return data. All other addresses will return 0.
4. The Address Range shown is power on default and the base can be adjusted via the Memory Base Offset Register (initialized to 0x00100000).
5. The SDRAM overlaps all other memory-mapped peripherals. SDRAM will not be selected within the address range of any overlapped peripheral.
6. RREM will be initialized at this address. The MPU registers allow it to be relocated on a block-by-block basis.
7. External Memory, shown based on POR values of registers.
8. Cannot be relocated but can be expanded in size.
9. SDRAM Control Registers control address of SDRAM.
10. For both POR and Reset the Chip Select 0 will select 256 Kbytes starting at 0x00000000 and ending at 0x0003ffff.
11. Read only.
12. Accessing these locations will cause an external bus cycle.
13. These registers are reserved for future expansion, reading them will always return 0.

5.16 Memory Protection Unit

The Memory Protection Unit (MPU) provides 16 Memory Control Blocks. Each block has an associated register that defines the starting point, size, and attributes of the memory range defined by that block. In addition to these registers, each hardware context has an associated MPU allocation register that identifies which blocks are active for the corresponding context. These access-controlled registers are writable by only the Master Context (Context_0). Typically, the Master Context would set up the MPU as required for the application in the initialization sequence. The basic rules for the MPU are as follows:

- If a memory range is not in any block then it is accessible by all contexts.
- If a memory range is defined but is not enabled for the current context, then an access to that range will generate an MPU fault (exception #65, see also Chapter 4,Core CPU, for details on exception processing).

Note: DMA violation of protected space do not raise exception #65, there is an MPU fault bit in the DMA Control Register that is set in this event (see Section 7.3, Direct Memory Access Controllers).

- If a memory range is read only and defined for the current context, a write to that range will generate an MPU fault.
- Multiple controls can be placed on a single-address range (e.g., one block could be set up allowing read/write access to some range for Context_2 while another block allows read-only access to the same range for Context_3).

Both instruction space and data space can be protected. Details and an MPU example of these registers are provided in this section (see Table 5-13).

Table 5-13. Memory Protection Units

Register Name	Description
MPU_Block_Control_Base_Register MPU_Block_Control_Attributes_Register	A two-register set used to define a protected address range in memory and also its attributes (size, RO, R/W etc.) There are 16 register pairs (MPU Block Control Base Register and MPU Block Control Attributes Register) allowing 16 separate areas to be protected. These are access-controlled registers and writable by only the Master Context (Context_0).
CTX0_MPU_Allocation CTX1_MPU_Allocation CTX2_MPU_Allocation CTX3_MPU_Allocation CTX4_MPU_Allocation	Five independent registers, one for each context. The above 16 defined blocks can be assigned to individual contexts via these registers. These are access-controlled registers and writable by only the Master Context (Context_0).

5.17 MPU Block Control Base Register

This register defines the address block in memory to be protected. The smallest block size allowed is 64 bytes, so the address is restricted to a 64-byte boundary. There are 16 of these registers, which are used in conjunction with the MPU Block Control Attributes Registers (see Table 5-14).

Table 5-14. MPU Block Control Base Register

31–6	5–0
Base Address	Reserved
RW	–

Note: Reset value is 0x00000000.

- Bits [31–6]—Base Address (addresses bits 31–6 of the protected block)

  Note: Must be on a 64-byte boundary

- Bits [5–0]—Reserved

5.18 MPU Block Control Attributes Register

This register defines the attributes of the block in memory to be protected. There are 16 of these registers, which are used in conjunction with the MPU Block Control Base Registers (see Table 5-15).

Table 5-15. MPU Block Control Attributes Register

31–7	6–2	1	0
Reserved	Size	RO	EN
–	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–7]—Reserved
- Bits [6–2]—Block size (see Table 5-16)
- Bit [1]—Read-Only Control bit
  - 0 → block can be written and read
  - 1 → block is read only
- Bit [0]—Block Enable bit
  - 0 → Block is disabled, has no effect on address space

– 1 → Block enabled, address rules apply to contexts as defined in CTXx_MPU_Allocation registers

Table 5-16. Value and Block Sizes

Size Value	Block Size	Size Value	Block Size
0	64 bytes	16	4 Mbytes
1	128 bytes	17	8 Mbytes
2	256 bytes	18	16 Mbytes
3	512 bytes	19	32 Mbytes
4	1 Kbytes	20	64 Mbytes
5	2 Kbytes	21	128 Mbytes
6	4 Kbytes	22	256 Mbytes
7	8 Kbytes	23	512 Mbytes
8	16 Kbytes	24	1 Gbyte
9	32 Kbytes	25	2 Gbytes
10	64 Kbytes	26	Undefined
11	128 Kbytes	27	Undefined
12	256 Kbytes	28	undefined
13	512 Kbytes	29	undefined
14	1 Mbyte	30	undefined
15	2 Mbytes	31	undefined

5.19 CTX MPU Allocation Registers

The block definitions in the MPU Block Control Base Registers/MPU Block Control Attributes Registers for the MPU are allocated to some set of contexts. The CTX MPU Allocation Registers are used to make that allocation. An Allocation register is associated with each context; each bit in the Allocation Register is associated with an MPU Block Control Register. Thus, to allocate a block to a particular context, set the corresponding bit in the Context Allocation Register. These are access-controlled registers and writable by only the Master Context, Context_0 (see Table 5-17).

There are five CTX MPU Allocation Registers, one for each context.

When the MPU disallows a memory access, it will raise exception #65 (MPU Error Exception) (see Chapter 4, Core CPU, on exception handling for details). It should also be noted that a DMA transaction that violates memory protection does not raise exception #65—the DMA is halted with an error indicating the violation (see Chapter 7, Peripheral Management Unit, for details).

Table 5-17. CTX MPU Allocation Register

31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
–	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved
- Bit [15]—Block 15 application bit
  - 0 → block does not apply to this context
  - 1 → block applies to this context
- Bit [14]—Block 14 application bit
  - 0 → block does not apply to this context
  - 1 → block applies to this context
- Bits [13–0] Block 13 to Block 0 individual application bits
  - 0 → block does not apply to this context
  - 1 → block applies to this context

Following is an example problem on how to set up the MPU.

5.20 MPU Example

Table 5-18 presents an example of data space protection.

Table 5-18. Example of MPU Data Space

MPU Block	Address Range	CTX0	CTX1	CTX2	CTX3	CTX4	Remarks
0	0x300_0000–0x300_1FFF	RW	None	None	None	RW	Only contexts 0 and 4 have access. Any other context access should MPU fault.
1	0x300_8000–0x300_87FF	RW	None	RW	None	None	Only contexts 0 and 2 have access as indicated. Contexts 3 and 4 will fault if they access this area.
2	0x300_8000–0x300_87FF	None	RO	None	None	None	Context_1 has RO access.

The MPU registers could be set up as shown in Table 5-19 to establish this protection.

Table 5-19. Example of MPU Protection

Register	Value
MPU Block Control Base Register_0	0x0300_0000
MPU Block Control Attributes Register_0	0x0000_001D (size=8 Kbyte, RW)
MPU Block Control Base Register_1	0x0300_8000
MPU Block Control Attributes Register_1	0x0000_0015 (size=2 Kbyte, RW)
MPU Block Control Base Register_2	0x0300_8000
MPU Block Control Attributes Register_2	0x0000_0017 (size=2 Kbyte, RO)
CTX0 MPU Allocation Register	0x0000_0003 (MPU blocks 0 and 1, RW on both)
CTX1 MPU Allocation Register	0x0000_0004 (MPU block 2, RO)
CTX2 MPU Allocation Register	0x0000_0002 (MPU block 1, RW)
CTX3 MPU Allocation Register	0x0000_0000 (no access to protected blocks)
CTX4 MPU Allocation Register	0x0000_0001 (MPU block 0, RW)

In this example, the first three register pairs in the MPU were used. Any of the register pairs in any order can be used. Using memory protection has no impact on system timing or performance.

Note: Although the above register assignments are based on how the fido1100 is designed to work, there is a known error. If more than one MPU block is set up for the same address range, and if one block is RW, and the other is RO, the context that is assigned RO access will not generate an MPU fault if it writes to this range. It erroneously picks up write ability from the write-enabled MPU block. A correction is planned for the next release of the part.

In this example, therefore, Context_1 will not fault if it writes to Block 2.

5.21 Programmable Chip Select Registers

The fido1100 interfaces to external memory and peripherals through a set of programmable chip select registers. A built-in SDRAM controller interfaces to SDRAM separately. The highlights of these features are as follows:

- Programmable Chip Selects and External Bus Timing
- External Memory

External memory attached to the fido1100 must not use addresses that overlap those functions or the reserved spaces listed in Table 5-12, Memory and Register Group Address Map Table.

- Eight chip selects provide eight configurable banks
  - Supports 8- or 16-bit external devices
  - External device can insert wait states into bus cycle

- Programmable memory timing per chip-select
  - Chip-select delay
  - Output-enable delay
  - Write-enable timing
  - Variable number of wait states

- SDRAM configuration, timing, and bank select

Refer to Chapter 6, External Bus Interface, for details on managing the external bus and for definitions of registers associated with this feature.

5.22 Complete Register Address Map

Table 5-20 provides the complete memory-mapped register address map for the fido1100.

Table 5-20. Complete Register Address Map Table

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
DMA Channel 0 Control Register	DMACh0_Control	0x000A0000	0x00000000	0x00000000	2
DMA Channel 0 Count Register	DMACh0_Count	0x000A0004	0x00000000	0x00000000	–
DMA Channel 0 Destination Address Register	DMACh0_Dest	0x000A0008	0x00000000	0x00000000	–
DMA Channel 0 Source Address Register	DMACh0_Source	0x000A000C	0x00000000	0x00000000	–
DMA Channel 1 Control Register	DMACh1_Control	0x000A0010	0x00000000	0x00000000	2
DMA Channel 1 Count Register	DMACh1_Count	0x000A0014	0x00000000	0x00000000	–
DMA Channel 0 Destination Address Register	DMACh1_Dest	0x000A0018	0x00000000	0x00000000	–
DMA Channel 0 Source Address Register	DMACh1_Source	0x000A001C	0x00000000	0x00000000	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
Reserved	Reserved	0x000A0020–0x000A027C	–	–	–
TCU 0 Status Register	TCU00_Status	0x000A0280	0x00000000	Unaffected	–
TCU 0 Mode Register	TCU00_Mode	0x000A0284	0x00000000	Unaffected	2
TCU 0 Counter Register	TCU00_Counter	0x000A0288	0x00000000	Unaffected	–
TCU0 Channel 0 IO Mode Register	TCU00_Ch0_IOMode	0x000A0290	0x00000000	Unaffected	–
TCU0 Channel 1 IO Mode Register	TCU00_Ch1_IOMode	0x000A0294	0x00000000	Unaffected	–
TCU0 Channel 2 IO Mode Register	TCU00_Ch2_IOMode	0x000A0298	0x00000000	Unaffected	–
TCU0 Channel 3 IO Mode Register	TCU00_Ch3_IOMode	0x000A029C	0x00000000	Unaffected	–
TCU0 Channel 0 Input Capture Register	TCU00_Ch0_InputCapture	0x000A02A0	0x00000000	Unaffected	–
TCU0 Channel 1 Input Capture Register	TCU00_Ch1_InputCapture	0x000A02A4	0x00000000	Unaffected	–
TCU0 Channel 2 Input Capture Register	TCU00_Ch2_InputCapture	0x000A02A8	0x00000000	Unaffected	–
TCU0 Channel 3 Input Capture Register	TCU00_Ch3_InputCapture	0x000A02AC	0x00000000	Unaffected	–
TCU0 Channel 0 Output Capture Register	TCU00_Ch0_OutputCompare	0x000A02B0	0x00000000	Unaffected	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
TCU0 Channel 1 Output Capture Register	TCU00_Ch1_OutputCompare	0x000A02B4	0x00000000	Unaffected	–
TCU0 Channel 2 Output Capture Register	TCU00_Ch2_OutputCompare	0x000A02B8	0x00000000	Unaffected	–
TCU0 Channel 3 Output Capture Register	TCU00_Ch3_OutputCompare	0x000A02BC	0x00000000	Unaffected	–
TCU 1 Status Register	TCU01_Status	0x000A02C0	0x00000000	Unaffected	–
TCU 1 Mode Register	TCU01_Mode	0x000A02C4	0x00000000	Unaffected	2
TCU 1 Counter Register	TCU01_Counter	0x000A02C8	0x00000000	Unaffected	–
TCU1 Channel 0 IO Mode Register	TCU01_Ch0_IOMode	0x000A02D0	0x00000000	Unaffected	–
TCU1 Channel 1 IO Mode Register	TCU01_Ch1_IOMode	0x000A02D4	0x00000000	Unaffected	–
TCU1 Channel 2 IO Mode Register	TCU01_Ch2_IOMode	0x000A02D8	0x00000000	Unaffected	–
TCU1 Channel 3 IO Mode Register	TCU01_Ch3_IOMode	0x000A02DC	0x00000000	Unaffected	–
TCU1 Channel 0 Input Capture Register	TCU01_Ch0_InputCapture	0x000A02E0	0x00000000	Unaffected	–
TCU1 Channel 1 Input Capture Register	TCU01_Ch1_InputCapture	0x000A02E4	0x00000000	Unaffected	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
TCU1 Channel 2 Input Capture Register	TCU01_Ch2_InputCapture	0x000A02E8	0x00000000	Unaffected	–
TCU1 Channel 3 Input Capture Register	TCU01_Ch3_InputCapture	0x000A02EC	0x00000000	Unaffected	–
TCU1 Channel 0 Output Capture Register	TCU01_Ch0_OutputCompare	0x000A02F0	0x00000000	Unaffected	–
TCU1 Channel 1 Output Capture Register	TCU01_Ch1_OutputCompare	0x000A02F4	0x00000000	Unaffected	–
TCU1 Channel 2 Output Capture Register	TCU01_Ch2_OutputCompare	0x000A02F8	0x00000000	Unaffected	–
TCU1 Channel 3 Output Capture Register	TCU01_Ch3_OutputCompare	0x000A02FC	0x00000000	Unaffected	–
System Timer Interrupt_0 Control Register	SYSTimer_Int00Control	0x000A0300	0x00000000	Unaffected	2
System Timer Interrupt_1 Control Register	SYSTimer_Int01Control	0x000A0304	0x00000000	Unaffected	2
System Timer Interrupt_2 Control Register	SYSTimer_Int02Control	0x000A0308	0x00000000	Unaffected	2
System Timer Interrupt_3 Control Register	SYSTimer_Int03Control	0x000A030C	0x00000000	Unaffected	2

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
System Timer Interrupt_4 Control Register	SYSTimer_Int04Control	0x000A0310	0x00000000	Unaffected	2
System Timer Control Register	SYSTimer_Control	0x000A0314	0x00000000	Unaffected	1
System Timer Prescale Register	SYSTimer_Prescale	0x000A0318	0x00000000	Unaffected	1
Watchdog Timer Control Register	WDTTimer_Control	0x000A0340	0x00000000	Unaffected	2
Watchdog Timer Reload Value Register	WDTTimer_Reload	0x000A0344	0x0000000F	Unaffected	1
Power On Reset Register	POR_reg	0x000A0360	0x00000001	Unaffected	1
Clock Mask Register	Clock Mask Register	0x000A0380	0x00003F0F	Unaffected	1
Device ID Register	Device_ID_Register	0x000A0400	Constant–0x11100531	Constant–0x11100531	3
Debug Control Register	DBG_Ctrl	0x000A0404	0x00000000	Unaffected	1
Debug Trace Buffer Control Register	Dbg_TrcBuf_Ctrl	0x000A0408	0x00000000	Unaffected	1
Debug Trace Buffer Base Address Register	Dbg_TrcBuf_Base	0x000A040C	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 0 Base Address Register	Dbg_Brk00_Base	0x000A0410	0x00000000	Unaffected	1

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
Debug Breakpoint/ Watchpoint 0 Data Register	Dbg_Brk00_Data	0x000A0414	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 0 Data Mask Register	Dbg_Brk00_DataMask	0x000A0418	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 0 Mode Control Register	Dbg_Brk00_Ctrl	0x000A041C	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 1 Base Address Register	Dbg_Brk01_Base	0x000A0420	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 1 Data Register	Dbg_Brk01_Data	0x000A0424	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 1 Data Mask Register	Dbg_Brk01_DataMask	0x000A0428	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 1 Mode Control Register	Dbg_Brk01_Ctrl	0x000A042C	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 2 Base Address Register	Dbg_Brk02_Base	0x000A0430	0x00000000	Unaffected	1

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
Debug Breakpoint/ Watchpoint 2 Data Register	Dbg_Brk02_Data	0x000A0434	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 2 Data Mask Register	Dbg_Brk02_DataMask	0x000A0438	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 2 Mode Control Register	Dbg_Brk02_Ctrl	0x000A043C	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 3 Base Address Register	Dbg_Brk03_Base	0x000A0440	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 3 Data Register	Dbg_Brk03_Data	0x000A0444	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 0 Data Mask Register	Dbg_Brk03_DataMask	0x000A0448	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 3 Mode Control Register	Dbg_Brk03_Ctrl	0x000A044C	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 4 Base Address Register	Dbg_Brk04_Base	0x000A0450	0x00000000	Unaffected	1

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
Debug Breakpoint/ Watchpoint 4 Data Register	Dbg_Brk04_Data	0x000A0454	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 0 Data Mask Register	Dbg_Brk04_DataMask	0x000A0458	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 4 Mode Control Register	Dbg_Brk04_Ctrl	0x000A045C	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 5 Base Address Register	Dbg_Brk05_Base	0x000A0460	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 5 Data Register	Dbg_Brk05_Data	0x000A0464	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 5 Data Mask Register	Dbg_Brk05_DataMask	0x000A0468	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 5 Mode Control Register	Dbg_Brk05_Ctrl	0x000A046C	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 6 Base Address Register	Dbg_Brk06_Base	0x000A0470	0x00000000	Unaffected	1

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
Debug Breakpoint/ Watchpoint 6 Data Register	Dbg_Brk06_Data	0x000A0474	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 6 Data Mask Register	Dbg_Brk06_DataMask	0x000A0478	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 6 Mode Control Register	Dbg_Brk06_Ctrl	0x000A047C	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 7 Base Address Register	Dbg_Brk07_Base	0x000A0480	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 7 Data Register	Dbg_Brk07_Data	0x000A0484	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 7 Data Mask Register	Dbg_Brk07_DataMask	0x000A0488	0x00000000	Unaffected	1
Debug Breakpoint/ Watchpoint 7 Mode Control Register	Dbg_Brk07_Ctrl	0x000A048C	0x00000000	Unaffected	1
ADC Control Register	A2D0_ControlRegister	0x000A0600	0x00000000	Unaffected	2
ADC Start Register	A2D0_StartRegister	0x000A0604	0x00000000	Unaffected	–
ADC Data Available Flag Register	A2D0_DataAvailableRegister	0x000A0608	0x00000000	Unaffected	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
ADC Channel 0 Data Register	A2D0_Ch0DataRegister	0x000A060C	0x00000000	Unaffected	–
ADC Channel 1 Data Register	A2D0_Ch1DataRegister	0x000A0610	0x00000000	Unaffected	–
ADC Channel 2 Data Register	A2D0_Ch2DataRegister	0x000A0614	0x00000000	Unaffected	–
ADC Channel 3 Data Register	A2D0_Ch3DataRegister	0x000A0618	0x00000000	Unaffected	–
ADC Channel 4 Data Register	A2D0_Ch4DataRegister	0x000A061C	0x00000000	Unaffected	–
ADC Channel 5 Data Register	A2D0_Ch5DataRegister	0x000A0620	0x00000000	Unaffected	–
ADC Channel 6 Data Register	A2D0_Ch6DataRegister	0x000A0624	0x00000000	Unaffected	–
ADC Channel 7 Data Register	A2D0_Ch7DataRegister	0x000A0628	0x00000000	Unaffected	–
Chip Select 0 Control Register	BIU_CS0_Control	0x000A0680	0x00000245 or 0x00000205, based on size pin	Unaffected	1
Chip Select 0 Timing Register	BIU_CS0_Timing	0x000A0684	0x31811031	Unaffected	1
Chip Select 1 Control Register	BIU_CS1_Control	0x000A0688	0x00000000	Unaffected	1
Chip Select 1 Timing Register	BIU_CS1_Timing	0x000A068C	0x31811031	Unaffected	1
Chip Select 2 Control Register	BIU_CS2_Control	0x000A0690	0x00000000	Unaffected	1

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
Chip Select 2 Timing Register	BIU_CS2_Timing	0x000A0694	0x31811031	Unaffected	1
Chip Select 3 Control Register	BIU_CS3_Control	0x000A0698	0x00000000	Unaffected	1
Chip Select 3 Timing Register	BIU_CS3_Timing	0x000A069C	0x31811031	Unaffected	1
Chip Select 4 Control Register	BIU_CS4_Control	0x000A06A0	0x00000000	Unaffected	1
Chip Select 4 Timing Register	BIU_CS4_Timing	0x000A06A4	0x31811031	Unaffected	1
Chip Select 5 Control Register	BIU_CS5_Control	0x000A06A8	0x00000000	Unaffected	1
Chip Select 5 Timing Register	BIU_CS5_Timing	0x000A06AC	0x31811031	Unaffected	1
Chip Select 6 Control Register	BIU_CS6_Control	0x000A06B0	0x00000000	Unaffected	1
Chip Select 6 Timing Register	BIU_CS6_Timing	0x000A06B4	0x31811031	Unaffected	1
Chip Select 7 Control Register	BIU_CS7_Control	0x000A06B8	0x00000000	Unaffected	1
Chip Select 7 Timing Register	BIU_CS7_Timing	0x000A06BC	0x31811031	Unaffected	1
External Bus Priority Register	BIU_Priority	0x000A0700	0x00000000	Unaffected	1
External Bus Default Timing Register	BIU_Def_Timing	0x000A0704	0x18E11011	Unaffected	1

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
SDRAM Timing Parameter 0 Register	SDRAM_Timing_0	0x000A0800	0x00a22602	Unaffected	1
SDRAM Timing Parameter 1 Register	SDRAM_Timing_1	0x000A0804	0x00480820	Unaffected	1
SDRAM Configuration 0 Register	SDRAM_Config_0	0x000A0808	0x00001226	Unaffected	1
SDRAM Configuration 1 Register	SDRAM_Config_1	0x000A080C	0x00000000	Unaffected	1
SDRAM External Bank 0 Configuration Register	SDRAM_Ext_Bank_0	0x000A0810	0x00001800	Unaffected	1
SDRAM External Bank 1 Configuration Register	SDRAM_Ext_Bank_1	0x000A0814	0x00000820	Unaffected	1
SDRAM External Bank 2 Configuration Register	SDRAM_Ext_Bank_2	0x000A0818	0x00000840	Unaffected	1
SDRAM External Bank 3 Configuration Register	SDRAM_Ext_Bank_3	0x000A081C	0x00000860	Unaffected	1
SDRAM External Bank 4 Configuration Register	SDRAM_Ext_Bank_4	0x000A0820	0x00000880	Unaffected	1
SDRAM External Bank 5 Configuration Register	SDRAM_Ext_Bank_5	0x000A0824	0x000008A0	Unaffected	1

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
SDRAM External Bank 6 Configuration Register	SDRAM_Ext_Bank_6	0x000A0828	0x000008C0	Unaffected	1
SDRAM External Bank 7 Configuration Register	SDRAM_Ext_Bank_7	0x000A082C	0x000008E0	Unaffected	1
External Interrupt_0 Control Register	IntControlCh0	0x000A0900	0x00000000	Unaffected	2, 5
External Interrupt_1 Control Register	IntControlCh1	0x000A0904	0x00000000	Unaffected	2, 5
External Interrupt_2 Control Register	IntControlCh2	0x000A0908	0x00000000	Unaffected	2, 5
External Interrupt_3 Control Register	IntControlCh3	0x000A090C	0x00000000	Unaffected	2, 5
External Interrupt_4 Control Register	IntControlCh4	0x000A0910	0x00000000	Unaffected	2, 5
External Interrupt_5 Control Register	IntControlCh5	0x000A0914	0x00000000	Unaffected	2, 5
External Interrupt_6 Control Register	IntControlCh6	0x000A0918	0x00000000	Unaffected	2, 5
External Interrupt_7 Control Register	IntControlCh7	0x000A091C	0x00000000	Unaffected	2, 5

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
Context_0 Software Interrupt Control Register	CTX0_INT_CTRL	0x000A0980	0x00000000	Unaffected	–
Context_1 Software Interrupt Control Register	CTX1_INT_CTRL	0x000A0984	0x00000000	Unaffected	–
Context_2 Software Interrupt Control Register	CTX2_INT_CTRL	0x000A0988	0x00000000	Unaffected	–
Context_3 Software Interrupt Control Register	CTX3_INT_CTRL	0x000A098C	0x00000000	Unaffected	–
Context_4 Software Interrupt Control Register	CTX4_INT_CTRL	0x000A0990	0x00000000	Unaffected	–
MAC Filter Mode and Status Register	PDMA_MAC_filter_mode	0x000A0A00	0x00000000	Unaffected	–
MAC Filter Data Write Register	PDMA_filter_data_write_head	0x000A0A04	0x00000000	Unaffected	–
MAC Filter Data Read Register	PDMA_filter_data_read_head	0x000A0A08	0x00000000	Unaffected	–
PMU Channel 0A Control Register	PDMACh0A_Control	0x000A0A40	0x00000000	Unaffected	2
PMU Channel 0A Status Register	PDMACh0A_Status	0x000A0A44	0x00000400	Unaffected	5

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
PMU Channel 0A Transmit Packet Size Register	PDMACh0A_PckXmitSize	0x000A0A48	0x00000000	Unaffected	–
PMU Channel 0A Receive Packet Size Register	PDMACh0A_PckRcvSize	0x000A0A4C	0x00000000	Unaffected	–
PMU Channel 0A Transmit Frame Buffer Start Address Register	PDMACh0A_XmitFBufStart	0x000A0A50	0x00000000	Unaffected	–
PMU Channel 0A Transmit Frame Buffer End Address Register	PDMACh0A_XmitFBufEnd	0x000A0A54	0x00000003	Unaffected	–
PMU channel 0A Transmit Frame Buffer Read Pointer	PDMACh0A_XmitFBufRdPtr	0x000A0A58	0x00000000	Unaffected	–
PMU channel 0A Transmit Frame Buffer Write Pointer	PDMACh0A_XmitFBufWrPtr	0x000A0A5C	0x00000000	Unaffected	–
PMU Channel 0A Receive Frame Buffer Start Address Register	PDMACh0A_RcvFBufStart	0x000A0A60	0x00000000	Unaffected	–
PMU Channel 0A Receive Frame Buffer End Address Register	PDMACh0A_RcvFBufEnd	0x000A0A64	0x00000003	Unaffected	–
PMU channel 0A Receive Frame Buffer Read Pointer	PDMACh0A_RcvFBufRdPtr	0x000A0A68	0x00000000	Unaffected	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
PMU channel 0A Receive Frame Buffer Write Pointer	PDMACh0A_RcvFBufWrPtr	0x000A0A6C	0x00000000	Unaffected	–
PMU Channel 0A Transmit Data FIFO Register	PDMACh0A_Xmit_Data	0x000A0A70	N/A	Unaffected	–
PMU Channel 0A Receive Data FIFO Register	PDMACh0A_Rcv_Data	0x000A0A74	N/A	Unaffected	–
PMU Channel 0B Control Register	PDMACh0B_Control	0x000A0A80	0x00000000	Unaffected	2
PMU Channel 0B Status Register	PDMACh0B_Status	0x000A0A84	0x00000400	Unaffected	5
PMU Channel 0B Transmit Packet Size Register	PDMACh0B_PckXmitSize	0x000A0A88	0x00000000	Unaffected	–
PMU Channel 0B Receive Packet Size Register	PDMACh0B_PckRcvSize	0x000A0A8C	0x00000000	Unaffected	–
PMU Channel 0B Transmit Frame Buffer Start Address Register	PDMACh0B_XmitFBufStart	0x000A0A90	0x00000000	Unaffected	–
PMU Channel 0B Transmit Frame Buffer End Address Register	PDMACh0B_XmitFBufEnd	0x000A0A94	0x00000003	Unaffected	–
PMU channel 0B Transmit Frame Buffer Read Pointer	PDMACh0B_XmitFBufRdPtr	0x000A0A98	0x00000000	Unaffected	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
PMU channel 0B Transmit Frame Buffer Write Pointer	PDMACh0B_XmitFBufWrPtr	0x000A0A9C	0x00000000	Unaffected	–
PMU Channel 0B Receive Frame Buffer Start Address Register	PDMACh0B_RcvFBufStart	0x000A0AA0	0x00000000	Unaffected	–
PMU Channel 0B Receive Frame Buffer End Address Register	PDMACh0B_RcvFBufEnd	0x000A0AA4	0x00000003	Unaffected	–
PMU channel 0B Receive Frame Buffer Read Pointer	PDMACh0B_RcvFBufRdPtr	0x000A0AA8	0x00000000	Unaffected	–
PMU channel 0B Receive Frame Buffer Write Pointer	PDMACh0B_RcvFBufWrPtr	0x000A0AAC	0x00000000	Unaffected	–
PMU Channel 0B Transmit Data FIFO Register	PDMACh0B_Xmit_Data	0x000A0AB0	N/A	Unaffected	–
PMU Channel 0B Receive Data FIFO Register	PDMACh0B_Rcv_Data	0x000A0AB4	N/A	Unaffected	–
PMU Channel 1A Control Register	PDMACh1A_Control	0x000A0AC0	0x00000000	Unaffected	2
PMU Channel 1A Status Register	PDMACh1A_Status	0x000A0AC4	0x00000400	Unaffected	5
PMU Channel 1A Transmit Packet Size Register	PDMACh1A_PckXmitSize	0x000A0AC8	0x00000000	Unaffected	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
PMU Channel 1A Receive Packet Size Register	PDMACh1A_PckRcvSize	0x000A0ACC	0x00000000	Unaffected	–
PMU Channel 1A Transmit Frame Buffer Start Address Register	PDMACh1A_XmitFBufStart	0x000A0AD0	0x00000000	Unaffected	–
PMU Channel 1A Transmit Frame Buffer End Address Register	PDMACh1A_XmitFBufEnd	0x000A0AD4	0x00000003	Unaffected	–
PMU channel 1A Transmit Frame Buffer Read Pointer	PDMACh1A_XmitFBufRdPtr	0x000A0AD8	0x00000000	Unaffected	–
PMU channel 1A Transmit Frame Buffer Write Pointer	PDMACh1A_XmitFBufWrPtr	0x000A0ADC	0x00000000	Unaffected	–
PMU Channel 1A Receive Frame Buffer Start Address Register	PDMACh1A_RcvFBufStart	0x000A0AE0	0x00000000	Unaffected	–
PMU Channel 1A Receive Frame Buffer End Address Register	PDMACh1A_RcvFBufEnd	0x000A0AE4	0x00000003	Unaffected	–
PMU channel 1A Receive Frame Buffer Read Pointer	PDMACh1A_RcvFBufRdPtr	0x000A0AE8	0x00000000	Unaffected	–
PMU channel 1A Receive Frame Buffer Write Pointer	PDMACh1A_RcvFBufWrPtr	0x000A0AEC	0x00000000	Unaffected	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
PMU Channel 1A Transmit Data FIFO Register	PDMACh1A_Xmit_Data	0x000A0AF0	N/A	Unaffected	–
PMU Channel 1A Receive Data FIFO Register	PDMACh1A_Rcv_Data	0x000A0AF4	N/A	Unaffected	–
PMU Channel 1B Control Register	PDMACh1B_Control	0x000A0B00	0x00000000	Unaffected	2
PMU Channel 1B Status Register	PDMACh1B_Status	0x000A0B04	0x00000400	Unaffected	5
PMU Channel 1B Transmit Packet Size Register	PDMACh1B_PckXmitSize	0x000A0B08	0x00000000	Unaffected	–
PMU Channel 1B Receive Packet Size Register	PDMACh1B_PckRcvSize	0x000A0B0C	0x00000000	Unaffected	–
PMU Channel 1B Transmit Frame Buffer Start Address Register	PDMACh1B_XmitFBufStart	0x000A0B10	0x00000000	Unaffected	–
PMU Channel 1B Transmit Frame Buffer End Address Register	PDMACh1B_XmitFBufEnd	0x000A0B14	0x00000003	Unaffected	–
PMU channel 1B Transmit Frame Buffer Read Pointer	PDMACh1B_XmitFBufRdPtr	0x000A0B18	0x00000000	Unaffected	–
PMU channel 1B Transmit Frame Buffer Write Pointer	PDMACh1B_XmitFBufWrPtr	0x000A0B1C	0x00000000	Unaffected	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
PMU Channel 1B Receive Frame Buffer Start Address Register	PDMACh1B_RcvFBufStart	0x000A0B20	0x00000000	Unaffected	–
PMU Channel 1B Receive Frame Buffer End Address Register	PDMACh1B_RcvFBufEnd	0x000A0B24	0x00000003	Unaffected	–
PMU channel 1B Receive Frame Buffer Read Pointer	PDMACh1B_RcvFBufRdPtr	0x000A0B28	0x00000000	Unaffected	–
PMU channel 1B Receive Frame Buffer Write Pointer	PDMACh1B_RcvFBufWrPtr	0x000A0B2C	0x00000000	Unaffected	–
PMU Channel 1B Transmit Data FIFO Register	PDMACh1B_Xmit_Data	0x000A0B30	N/A	Unaffected	–
PMU Channel 1B Receive Data FIFO Register	PDMACh1B_Rcv_Data	0x000A0B34	N/A	Unaffected	–
PMU Channel 2A Control Register	PDMACh2A_Control	0x000A0B40	0x00000000	Unaffected	2
PMU Channel 2A Status Register	PDMACh2A_Status	0x000A0B44	0x00000400	Unaffected	5
PMU Channel 2A Transmit Packet Size Register	PDMACh2A_PckXmitSize	0x000A0B48	0x00000000	Unaffected	–
PMU Channel 2A Receive Packet Size Register	PDMACh2A_PckRcvSize	0x000A0B4C	0x00000000	Unaffected	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
PMU Channel 2A Transmit Frame Buffer Start Address Register	PDMACh2A_XmitFBufStart	0x000A0B50	0x00000000	Unaffected	–
PMU Channel 2A Transmit Frame Buffer End Address Register	PDMACh2A_XmitFBufEnd	0x000A0B54	0x00000003	Unaffected	–
PMU channel 2A Transmit Frame Buffer Read Pointer	PDMACh2A_XmitFBufRdPtr	0x000A0B58	0x00000000	Unaffected	–
PMU channel 2A Transmit Frame Buffer Write Pointer	PDMACh2A_XmitFBufWrPtr	0x000A0B5C	0x00000000	Unaffected	–
PMU Channel 2A Receive Frame Buffer Start Address Register	PDMACh2A_RcvFBufStart	0x000A0B60	0x00000000	Unaffected	–
PMU Channel 2A Receive Frame Buffer End Address Register	PDMACh2A_RcvFBufEnd	0x000A0B64	0x00000003	Unaffected	–
PMU Channel 2A Receive Frame Buffer Read Pointer	PDMACh2A_RcvFBufRdPtr	0x000A0B68	0x00000000	Unaffected	–
PMU Channel 2A Receive Frame Buffer Write Pointer	PDMACh2A_RcvFBufWrPtr	0x000A0B6C	0x00000000	Unaffected	–
PMU Channel 2A Transmit Data FIFO Register	PDMACh2A_Xmit_Data	0x000A0B70	N/A	Unaffected	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
PMU Channel 2A Receive Data FIFO Register	PDMACh2A_Rcv_Data	0x000A0B74	N/A	Unaffected	–
PMU Channel 2B Control Register	PDMACh2B_Control	0x000A0B80	0x00000000	Unaffected	2
PMU Channel 2B Status Register	PDMACh2B_Status	0x000A0B84	0x00000400	Unaffected	5
PMU Channel 2B Transmit Packet Size Register	PDMACh2B_PckXmitSize	0x000A0B88	0x00000000	Unaffected	–
PMU Channel 2B Receive Packet Size Register	PDMACh2B_PckRcvSize	0x000A0B8C	0x00000000	Unaffected	–
PMU Channel 2B Transmit Frame Buffer Start Address Register	PDMACh2B_XmitFBufStart	0x000A0B90	0x00000000	Unaffected	–
PMU Channel 2B Transmit Frame Buffer End Address Register	PDMACh2B_XmitFBufEnd	0x000A0B94	0x00000003	Unaffected	–
PMU Channel 2B Transmit Frame Buffer Read Pointer	PDMACh2B_XmitFBufRdPtr	0x000A0B98	0x00000000	Unaffected	–
PMU Channel 2B Transmit Frame Buffer Write Pointer	PDMACh2B_XmitFBufWrPtr	0x000A0B9C	0x00000000	Unaffected	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
PMU Channel 2B Receive Frame Buffer Start Address Register	PDMACh2B_RcvFBufStart	0x000A0BA0	0x00000000	Unaffected	–
PMU Channel 2B Receive Frame Buffer End Address Register	PDMACh2B_RcvFBufEnd	0x000A0BA4	0x00000003	Unaffected	–
PMU Channel 2B Receive Frame Buffer Read Pointer	PDMACh2B_RcvFBufRdPtr	0x000A0BA8	0x00000000	Unaffected	–
PMU Channel 2B Receive Frame Buffer Write Pointer	PDMACh2B_RcvFBufWrPtr	0x000A0BAC	0x00000000	Unaffected	–
PMU Channel 2B Transmit Data FIFO Register	PDMACh2B_Xmit_Data	0x000A0BB0	N/A	Unaffected	–
PMU Channel 2B Receive Data FIFO Register	PDMACh2B_Rcv_Data	0x000A0BB4	N/A	Unaffected	–
PMU Channel 3A Control Register	PDMACh3A_Control	0x000A0BC0	0x00000000	Unaffected	2
PMU Channel 3A Status Register	PDMACh3A_Status	0x000A0BC4	0x00000400	Unaffected	5
PMU Channel 3A Transmit Packet Size Register	PDMACh3A_PckXmitSize	0x000A0BC8	0x00000000	Unaffected	–
PMU Channel 3A Receive Packet Size Register	PDMACh3A_PckRcvSize	0x000A0BCC	0x00000000	Unaffected	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
PMU Channel 3A Transmit Frame Buffer Start Address Register	PDMACh3A_XmitFBufStart	0x000A0BD0	0x00000000	Unaffected	–
PMU Channel 3A Transmit Frame Buffer End Address Register	PDMACh3A_XmitFBufEnd	0x000A0BD4	0x00000003	Unaffected	–
PMU Channel 3A Transmit Frame Buffer Read Pointer	PDMACh3A_XmitFBufRdPtr	0x000A0BD8	0x00000000	Unaffected	–
PMU Channel 3A Transmit Frame Buffer Write Pointer	PDMACh3A_XmitFBufWrPtr	0x000A0BDC	0x00000000	Unaffected	–
PMU Channel 3A Receive Frame Buffer Start Address Register	PDMACh3A_RcvFBufStart	0x000A0BE0	0x00000000	Unaffected	–
PMU Channel 3A Receive Frame Buffer End Address Register	PDMACh3A_RcvFBufEnd	0x000A0BE4	0x00000003	Unaffected	–
PMU Channel 3A Receive Frame Buffer Read Pointer	PDMACh3A_RcvFBufRdPtr	0x000A0BE8	0x00000000	Unaffected	–
PMU Channel 3A Receive Frame Buffer Write Pointer	PDMACh3A_RcvFBufWrPtr	0x000A0BEC	0x00000000	Unaffected	–
PMU Channel 3A Transmit Data FIFO Register	PDMACh3A_Xmit_Data	0x000A0BF0	N/A	Unaffected	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
PMU Channel 3A Receive Data FIFO Register	PDMACh3A_Rcv_Data	0x000A0BF4	N/A	Unaffected	–
PMU Channel 3B Control Register	PDMACh3B_Control	0x000A0C00	0x00000000	Unaffected	2
PMU Channel 3B Status Register	PDMACh3B_Status	0x000A0C04	0x00000400	Unaffected	5
PMU Channel 3B Transmit Packet Size Register	PDMACh3B_PckXmitSize	0x000A0C08	0x00000000	Unaffected	–
PMU Channel 3B Receive Packet Size Register	PDMACh3B_PckRcvSize	0x000A0C0C	0x00000000	Unaffected	–
PMU Channel 3B Transmit Frame Buffer Start Address Register	PDMACh3B_XmitFBufStart	0x000A0C10	0x00000000	Unaffected	–
PMU Channel 3B Transmit Frame Buffer End Address Register	PDMACh3B_XmitFBufEnd	0x000A0C14	0x00000003	Unaffected	–
PMU Channel 3B Transmit Frame Buffer Read Pointer	PDMACh3B_XmitFBufRdPtr	0x000A0C18	0x00000000	Unaffected	–
PMU Channel 3B Transmit Frame Buffer Write Pointer	PDMACh3B_XmitFBufWrPtr	0x000A0C1C	0x00000000	Unaffected	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
PMU Channel 3B Receive Frame Buffer Start Address Register	PDMACh3B_RcvFBufStart	0x000A0C20	0x00000000	Unaffected	–
PMU Channel 3B Receive Frame Buffer End Address Register	PDMACh3B_RcvFBufEnd	0x000A0C24	0x00000003	Unaffected	–
PMU Channel 3B Receive Frame Buffer Read Pointer	PDMACh3B_RcvFBufRdPtr	0x000A0C28	0x00000000	Unaffected	–
PMU Channel 3B Receive Frame Buffer Write Pointer	PDMACh3B_RcvFBufWrPtr	0x000A0C2C	0x00000000	Unaffected	–
PMU Channel 3B Transmit Data FIFO Register	PDMACh3B_Xmit_Data	0x000A0C30	N/A	Unaffected	–
PMU Channel 3B Receive Data FIFO Register	PDMACh3B_Rcv_Data	0x000A0C34	N/A	Unaffected	–
Reserved	Reserved	0x000A0C40–0x001A10FC	–	–	–
Reserved	Reserved	0x000A1100	–	–	–
Reserved	Reserved	0x000A1104	–	–	–
Reserved	Reserved	0x000A1108	–	–	–
Context_0 Software Interrupt Actuation Register	SWINTACT0	0x000A110C	0x00000000	Unaffected	–
Context_1 Claim Register	CTX1_CLAIM	0x000A1110	0x00000000	0x00000000	4

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
Context_1 Pending Contexts Register	CTX1_PENDING	0x000A1114	0x00000000	0x00000000	3
Context_1 Priority Inheritance Register	CTX1_PRI_INHER	0x000A1118	0x00000000	0x00000000	3
Context_1 Software Interrupt Actuation Register	SWINTACT1	0x000A111C	0x00000000	Unaffected	–
Context_2 Claim Register	CTX2_CLAIM	0x000A1120	0x00000000	0x00000000	4
Context_2 Pending Contexts Register	CTX2_PENDING	0x000A1124	0x00000000	0x00000000	3
Context_2 Priority Inheritance Register	CTX2_PRI_INHER	0x000A1128	0x00000000	0x00000000	3
Context_2 Software Interrupt Actuation Register	SWINTACT2	0x000A112C	0x00000000	Unaffected	–
Context_3 Claim Register	CTX3_CLAIM	0x000A1130	0x00000000	0x00000000	4
Context_3 Pending Contexts Register	CTX3_PENDING	0x000A1134	0x00000000	0x00000000	3
Context_3 Priority Inheritance Register	CTX3_PRI_INHER	0x000A1138	0x00000000	0x00000000	3
Context_3 Software Interrupt Actuation Register	SWINTACT3	0x000A113C	0x00000000	Unaffected	–

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
Context_4 Claim Register	CTX4_CLAIM	0x000A1140	0x00000000	0x00000000	4
Context_4 Pending Contexts Register	CTX4_PENDING	0x000A1144	0x00000000	0x00000000	3
Context_4 Priority Inheritance Register	CTX4_PRI_INHER	0x000A1148	0x00000000	0x00000000	3
Context_4 Software Interrupt Actuation Register	SWINTACT4	0x000A114C	0x00000000	Unaffected	–
Reserved	Reserved	0x000A1150–0x000A80FC	–	–	–
Context_0 Data Registers	CTX0_D0–CTX0_D7	0x000A8100–0x000A811C	Unaffected	Unaffected	1
Context_0 Address Registers	CTX0_A0–CTX0_A6	0x000A8120–0x000A8138	Unaffected	Unaffected	1
Context_0 A7/User Stack Pointer (USP)	CTX0_A7 (USP)	0x000A813C	Unaffected	Unaffected	1
Context_0 A7/Supervisor Stack Pointer (SSP)	CTX0_A7’ (SSP)	0x000A8140	Loaded from Vector 0	Loaded from Vector 0	1
Context_0 Program Counter (PC)	CTX0_PC	0x000A8144	Loaded from Vector 0	Loaded from Vector 0	1
Context_0 Status Register (SR)	CTX0_SR	0x000A8148	0x2700	0x2700	1
Context_0 Vector Base Register (VBR)	CTX0_VBR	0x000A814C	0x0000	0x0000	1

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
Context_0 Alternate Function Code (SFC)	CTX0_SFC	0x000A8150	Unaffected	Unaffected	1
Context_0 Alternate Function Code (DFC)	CTX0_DFC	0x000A8154	Unaffected	Unaffected	1
Context_1 Data Registers	CTX1_D0–CTX1_D7	0x000A8180–0x000A819C	Unaffected	Unaffected	1
Context_1 Address Registers	CTX1_A0–CTX1_A6	0x000A81A0–0x000A81B8	Unaffected	Unaffected	1
Context_1 A7/User Stack Pointer (USP)	CTX1_A7 (USP)	0x000A81BC	Unaffected	Unaffected	1
Context_1 A7/Supervisor Stack Pointer (SSP)	CTX1_A7' (SSP)	0x000A81C0	Unaffected	Unaffected	1
Context_1 Program Counter (PC)	CTX1_PC	0x000A81C4	Unaffected	Unaffected	1
Context_1 Status Register (SR)	CTX1_SR	0x000A81C8	0x2700	Unaffected	1
Context_1 Vector Base Register (VBR)	CTX1_VBR	0x000A81CC	0x0000	Unaffected	1
Context_1 Alternate Function Code (SFC)	CTX1_SFC	0x000A81D0	Unaffected	Unaffected	1
Context_1 Alternate Function Code (DFC)	CTX1_DFC	0x000A81D4	Unaffected	Unaffected	1
Context_2 Data Registers	CTX2_D0–CTX2_D7	0x000A8200–0x000A821C	Unaffected	Unaffected	1

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
Context_2 Address Registers	CTX2_A0–CTX2_A6	0x000A8220–0x000A8238	Unaffected	Unaffected	1
Context_2 A7/User Stack Pointer (USP)	CTX2_A7 (USP)	0x000A823C	Unaffected	Unaffected	1
Context_2 A7/Supervisor Stack Pointer (SSP)	CTX2_A7' (SSP)	0x000A8240	Unaffected	Unaffected	1
Context_2 Program Counter (PC)	CTX2_PC	0x000A8244	Unaffected	Unaffected	1
Context_2 Status Register (SR)	CTX2_SR	0x000A8248	0x2700	Unaffected	1
Context_2 Vector Base Register (VBR)	CTX2_VBR	0x000A824C	0x0000	Unaffected	1
Context_2 Alternate Function Code (SFC)	CTX2_SFC	0x000A8250	Unaffected	Unaffected	1
Context_2 Alternate Function Code (DFC)	CTX2_DFC	0x000A8254	Unaffected	Unaffected	1
Context_3 Data Registers	CTX3_D0–CTX3_D7	0x000A8280–0x000A829C	Unaffected	Unaffected	1
Context_3 Address Registers	CTX3_A0–CTX3_A6	0x000A82A0–0x000A82B8	Unaffected	Unaffected	1
Context_3 A7/User Stack Pointer (USP)	CTX3_A7 (USP)	0x000A82BC	Unaffected	Unaffected	1
Context_3 A7/Supervisor Stack Pointer (SSP)	CTX3_A7' (SSP)	0x000A82C0	Unaffected	Unaffected	1

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
Context_3 Program Counter (PC)	CTX3_PC	0x000A82C4	Unaffected	Unaffected	1
Context_3 Status Register (SR)	CTX3_SR	0x000A82C8	0x2700	Unaffected	1
Context_3 Vector Base Register (VBR)	CTX3_VBR	0x000A82CC	0x0000	Unaffected	1
Context_3 Alternate Function Code (SFC)	CTX3_SFC	0x000A82D0	Unaffected	Unaffected	1
Context_3 Alternate Function Code (DFC)	CTX3_DFC	0x000A82D4	Unaffected	Unaffected	1
Context_4 Data Registers	CTX4_D0–CTX4_D7	0x000A8300–0x000A831C	Unaffected	Unaffected	1
Context_4 Address Registers	CTX4_A0–CTX4_A6	0x000A8320–0x000A8338	Unaffected	Unaffected	1
Context_4 A7/User Stack Pointer (USP)	CTX4_A7	0x000A833C	Unaffected	Unaffected	1
Context_4 A7/Supervisor Stack Pointer (SSP)	CTX4_A7’	0x000A8340	Unaffected	Unaffected	1
Context_4 Program Counter (PC)	CTX4_PC	0x000A8344	Unaffected	Unaffected	1
Context_4 Status Register (SR)	CTX4_SR	0x000A8348	0x2700	Unaffected	1
Context_4 Vector Base Register (VBR)	CTX4_VBR	0x000A834C	0x0000	Unaffected	1

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
Context_4 Alternate Function Code (SFC)	CTX4_SFC	0x000A8350	Unaffected	Unaffected	1
Context_4 Alternate Function Code (DFC)	CTX4_DFC	0x000A8354	Unaffected	Unaffected	1
MPU Block 0 to 15 Control Registers	MPU_Blk00Ctrl–MPU_Blk15Ctrl	0x000AA000–0x000AA03C	0x0000	Unaffected	1
MPU Block 0 to 15 Attribute Registers	MPU_Blk00Attrib–MPU_Blk15Attrib	0x000AA080–0x000AA0BC	0x0000	Unaffected	1
Context_0 MPU Allocation Register	CTX0_MPUAllocation	0x000AA100	0x0000	Unaffected	1
Context_1 MPU Allocation Register	CTX1_MPUAllocation	0x000AA104	0x0000	Unaffected	1
Context_2 MPU Allocation Register	CTX2_MPUAllocation	0x000AA108	0x0000	Unaffected	1
Context_3 MPU Allocation Register	CTX3_MPUAllocation	0x000AA10C	0x0000	Unaffected	1
Context_4 MPU Allocation Register	CTX4_MPUAllocation	0x000AA110	0x0000	Unaffected	1
Relocatable Rapid Execution Memory (RREM) Block 0 to 15 Control Registers	DCACHE_RelocateBlk00–DCACHE_RelocateBlk15	0x000AA180–0x000AA1BC	0x0000	Unaffected	1
Context_0 Control Register	CTX0_Control	0x000AA400	0x1F02	0x1F02	1
Context_1 Control Register	CTX1_Control	0x000AA404	0x1800	Unaffected	1

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
Context_2 Control Register	CTX2_Control	0x000AA408	0x1800	Unaffected	1
Context_3 Control Register	CTX3_Control	0x000AA40C	0x1800	Unaffected	1
Context_4 Control Register	CTX4_Control	0x000AA410	0x1800	Unaffected	1
Context_0 Maximum Time Register	CTX0_MaxTime	0x000AA480	0x00fffe00	Unaffected	1
Context_1 Maximum Time Register	CTX1_MaxTime	0x000AA484	0x00fffe00	Unaffected	1
Context_2 Maximum Time Register	CTX2_MaxTime	0x000AA488	0x00fffe00	Unaffected	1
Context_3 Maximum Time Register	CTX3_MaxTime	0x000AA48C	0x00fffe00	Unaffected	1
Context_4 Maximum Time Register	CTX4_MaxTime	0x000AA490	0x00fffe00	Unaffected	1
Context_0 Timer Register	CTX0_Time	0x000AA500	0x0000	0x0000	1
Context_1 Timer Register	CTX1_Time	0x000AA504	0x0000	0x0000	1
Context_2 Timer Register	CTX2_Time	0x000AA508	0x0000	0x0000	1
Context_3 Timer Register	CTX3_Time	0x000AA50C	0x0000	0x0000	1
Context_4 Timer Register	CTX4_Time	0x000AA510	0x0000	0x0000	1
Context Timer Clear Register	CTXN_TimeClr	0x000AA580	0x0000	0x0000	1
Context Idle Timer Register	CTX_IdleTimer	0x000AA584	0x0000	0x0000	–
Context Timer Enable Register	CTXN_TimeEn	0x000AA588	0x0000	Unaffected	1

Table 5-20. Complete Register Address Map Table *(Continued)*

Block	Name	Base Address*	Major Reset value	Minor Reset value	Notes
Faulted Context Register	CTX_FaultID	0x000AA600	0x0000	0x0000	3
Current Context Register	Current Context Register	0x000AA604	0x0000	0x0000	3

Notes:
* Base address shown in the table is the base offset. This address is OR-ed with the value in the Memory Base Offset Register to adjust bits 30–20. For example, the Memory Base Offset Register has a value of 0x00100000 at POR, thus the Faulted Context Register (CTX_FaultID) would be accessed at address 0x001AA600. If the Memory Base Offset Register is changed to address 0x70F00000 then the Faulted Context Register (CTX_FaultID) is accessed at address 0x70FAA600.
1. Readable by all contexts, writable by only Context_0 (see Chapter 11, Access-Controlled Registers).
2. Certain fields in the register have restricted writes (see Chapter 11, Access-Controlled Registers).
3. Read only.
4. Context-specific. See Section 4.8.9, Context Claim Registers, for details.
5. Certain fields (typically status bits) of this register are evaluated on a continuous basis and therefore the value of this register may be modified after reset to reflect the condition evaluated.

6. External Bus Interface

The fido1100 external bus interface to external memory and peripherals is configured by a set of chip-select and bus-timing registers. The fido1100 also has a built-in SDRAM controller to control the interface to SDRAM separately. The external address bus of the fido1100 is a 31-bit bus, and the external data bus is configurable to support either an 8- or a 16-bit bus. This chapter covers the following:

- Address and Data Bus
- External Bus Chip Select Control and Timing Registers (8 reg. pairs)
- External Bus Default Timing Register
- External Bus Priority Register
- SDRAM Controller Registers
- Startup and Operation of SDRAM Controller
- SDRAM Module Types Address Mapping (16-Bit Bus Width)
- SDRAM Module Types Address Mapping (8-Bit Bus Width)
- SDRAM External I/O Signal List
- External Bus Arbitration

The fido1100 is a 32-bit data bus internally and supports either a 16- or 8-bit data bus externally. The external data bus size is selected at Reset depending on the value of the A_26_SIZE signal (0 = 8-bit data bus, 1 = 16-bit data bus). See *The fido Data Sheet* Chapter 7, Reset, for a description of reset processing. The primary features of the External Bus Interface include:

- Eight chip selects that provide eight configurable banks
  - Support of 8- or 16-bit external devices
  - An external device that can insert wait states into bus cycle
- Support of external bus master (with internal priority based on priority of currently executing context/DMA)
- Management of non-aligned data accesses over the external bus
- Programmable memory timing per chip-select
  - Chip select delay

- Output enable delay
- Write-enable timing
- Variable number of wait states

If no on-board chip-select logic responds to a memory cycle, then it is assumed that external logic is decoding the access and controlling the access timing via the RDY_N signal. If the RDY_N signal is not asserted within 255 clocks from the beginning of the memory cycle, a bus fault is generated. Table 6-1 lists the external bus interface signals.

Table 6-1. External Bus Interface Signal List

Signal Name	Type	Description
MEMCLK	Output	Memory clock used by external memory
D15–D0	Bi-directional	External data bus bits 15..0
A24–A0	Output, tri-stateable	External address bus bits 24..0 (Note: MSB of address bus is Endian Control bit, and A30–A25 are muxed)
A_25_RESET_DELAY	Muxed, Internal Pull-up	Muxed pin, External Bus Interface address Bit [25] or POR counter bypass
A_26_SIZE	Muxed, Internal Pull-up	Muxed pin, External Bus Interface address Bit [26] or data bus size select (0 = 8-Bit, 1 = 16-Bit)
A27_CS7_N	Muxed	Muxed pin, External Bus Interface address Bit [27] or Chip select 7 (chip select active low)
A28_CS6_N	Muxed	Muxed pin, External Bus Interface address Bit [28] or Chip select 6 (chip select active low)
A29_CS5_N	Muxed	Muxed pin, External Bus Interface address Bit [29] or Chip select 5 (chip select active low)
A30_CS4_N	Muxed	Muxed pin, External Bus Interface address Bit [30] or Chip select 4 (chip select active low)
CS0_N	Output	Chip select 0 (chip select active low)
CS1_N	Output	Chip select 1 (chip select active low)
CS2_N	Output	Chip select 2 (chip select active low)
CS3_N	Output	Chip select 3 (chip select active low)
BE1_N BE0_N	Output, tri-stateable	Byte Enables, active low
OE_N	Output, tri-stateable	Output Enable
RW_N	Output, tri-stateable	Read/Write Control (write is active low)
RDY_N	Input	External Bus Ready (active low)
HOLDREQ_N	Input	External Bus Master Request (active low)
HOLDGNT_N	Output	External Bus Master Grant (active low)

6.1 Address and Data Bus

The fido1100 has a 32-bit address bus, supporting a 2-Gbyte memory space that can be accessed as a little-endian or big-endian address space. Address Bits [30–0] are used to address external devices and address Bit [31] serves as a big-endian/little-endian mode bit. Accesses to addresses between 0x00000000 and 0x7FFFFFFF will be big-endian accesses. Accesses to addresses between 0x80000000 and 0xFFFFFFFF will be little-endian accesses of memory space 0x00000000 to 0x7FFFFFFF (see Section 5.6, Endian Mode Control, for details).

6.2 External Bus Chip Select Control and Timing Registers

The fido1100 uses the registers listed below to control the operation and characteristics of the External Bus. They are detailed in sections that follow:

- External Bus Chip Select Control Register
- External Bus Chip Select Timing Register
- External Bus Default Timing Register
- External Bus Priority Register

6.2.1 External Bus Chip Select Control Register_N (where N=0..7)

The External Bus Chip Select Control Register defines the location and size of the programmable Chip Select. There are eight of these registers, one for each chip select. They are access-controlled and writable by only the Master Context, Context_0 (see Chapter 11, Access-Controlled Registers) (see Table 6-2).

Table 6-2. External Bus Chip Select Control Register

31–12	11	10	9	8	7–6	5	4	3–0
Base Address	Reserved	Mode Select	Output Enable	SDRAM Enable	Width	Byte Enable	Reserved	Size
RW	–	RW	RW	RW	RW	RW	–	RW

Note: Reset value is CS0 =0x00000205 or 0x00000245 (256 Kbytes, 8/16-bit-wide devices dependent on Size input at reset). CS1–CS7 = 0x00000000.

- Bits [31–12]—Base Address → Specifies the beginning address for the CS block. Must be a multiple of bank size (bank size is specified by bits 3..0)

Note: If the SDRAM Enable bit is 1, then only the upper 12 bits [31–20] are valid, and must match the value set in Bits [11–0] of the appropriate SDRAM External Bank register. The remaining bits of this field [19–12] must be set to “0.”

- Bit [11]—Reserved

- Bit [10]—Mode Select (0 → use chip select output, 1 → use address output)

Notes:

- This field only applies to muxed chip-select pins:
CS7 muxed with A27 (hardware signal A27_CS7_N)
CS6 muxed with A28 (hardware signal A28_CS6_N)
CS5 muxed with A29 (hardware signal A29_CS5_N)
CS4 muxed with A30 (hardware signal A30_CS4_N)

-Bit 10 only controls muxed cs/addr signals CS7/A27 - CS4/A30. On non-muxed signals, it has no affect.

- Bit [9]—Chip Select Enable
  - 0—Disable chip select signal
    - Muxed CS/Addr signals will tristate. Non-muxed signals will drive high.
  - 1—Enable chip select signal
    - The muxed cs/addr signals will drive appropriately as set by bit 10. Non-muxed signals will drive low as appropriate or operation.

- Bit [8]—SDRAM Enable
  - 0—Associated memory is not SDRAM.
  - 1—Associated memory is SDRAM.

Note: When this flag is set all other fields (except Base Address, Mode Select, Output Enable, and Size) in this register, as well as the External Bus Chip Select Timing Register, are invalid. This enables the SDRAM Control Registers.

- Bits [7–6]—Width → Specifies the bus width of the attached peripheral. Determines the byte lane of the bus used to access the external device (via the BE_N byte enable signals):
  - 00—8-bit device, all data transferred as bytes on D[7–0]
  - 01—16-bit device, all data transferred as words on D[15–0]. Unaligned word accesses require two bus cycles
  - 10—Reserved
  - 11—Reserved

Note: Size input (sampled at reset) sets the Size field for CS0. This field cannot be written to on CS0. It is latched at reset and cannot be overwritten

- Bit [5]—Byte Enable → Controls byte enable signal behavior when this chip select is active
  - 0—Byte enable signals active for reads and write cycles

  - 1—Byte enable signals become byte write enables (OR-ed with RW_N)

- Bit [4]—Reserved

- Bits [3–0]—Size → Specifies the total range covered by this CS:
  - 0000—8 Kbytes
  - 0001—16 Kbytes
  - 0010—32 Kbytes
  - 0011—64 Kbytes
  - 0100—128 Kbytes
  - 0101—256 Kbytes
  - 0110—512 Kbytes
  - 0111—1 Mbyte
  - 1000—2 Mbytes
  - 1001—2 Mbytes
  - 1010—8 Mbytes
  - 1011—16 Mbytes
  - 1100—32 Mbytes
  - 1101—64 Mbytes
  - 1110—128 Mbytes
  - 1111—256 Mbytes

To allow the SDRAM controller to operate properly, some fields of this register must be set up as follows to enable the chip select to be in SDRAM mode:

- SDRAM_enable must be set to "1."

- output_enable must be set to "1." If configuring CS4-CS7, the mode_select must be set to "0."

- The base_address Bits [31–20] must be set to the base address of the SDRAM memory attached to this chip select.
  - base_address Bits [19–12] must be set to "0."
  - The 12-bit value set in Bits [31–20] must match exactly with Bits [11–0] of the appropriate SDRAM external bank.

- The size must be properly set to the size of the SDRAM memory attached to this chip select.

Note: The binary pattern for the size field in this register is different from the BNKSIZE field in the SDRAM Configuration 0 Register.

There are eight Chip Selects for External Memory, and eight External Banks for the SDRAM controller (further information is provided in the SDRAM Controller Registers section). When a chip select is in SDRAM mode (SDRAM_enable = 1) it is controlled by the appropriate SDRAM controller's external bank. CS0 is controlled by SDRAM External Bank 0. CS1 is connected to External Bank 1, etc.

6.2.2 External Bus Chip Select Timing Register_N (where N=0..7)

The External Bus Chip Select Timing Register defines the timing of the programmable Chip Selects. There are eight of these registers, one for each chip select. Some important notes follow:

- There are eight chip selects but one OE and one WE, these signals will adjust on an individual basis to the CS in use for that cycle.

- For the internally generated chip selects, the external ready input signal (RDY_N) can be enabled/disabled and the wait timing defined. However, if no on-board chip-select logic responds to a memory cycle, then it is assumed that external logic is decoding the access. For this type of memory access, the external RDY_N input signal will always be enabled and will determine whether the cycle ends in a bus fault.

Note: If the RDY_N signal is not asserted (low) within 256 clocks from the end of the TxWAIT period (see the External Bus Default Timing Register), a bus fault is generated.

When the sdram_enable flag of the External Bus Chip Select Control Register is enabled, this register is disabled for the given chip select (see Table 6-3). They are access-controlled and writable by only the Master Context, Context_0 (see Chapter 11, Access-Controlled Registers).

Table 6-3. Chip Select Timing Register

31–27	26–22	21	20–19	18–16	15–14	13–12	11–10	9–8	7–6	5–4	3–2	1–0
TwWAIT	TrWait	RDY_ENABLE	Reserved	THLD	Reserved	TCS	Reserved	TOE	Reserved	TWEF	Reserved	TWER
RW	RW	RW	–	RW	R	RW	–	RW	–	RW	–	RW

Note: The default settings for the Chip Select Timing Register at POR or external reset are as follows:
CS0 loaded to 0x31811031 (suitable for most FLASH ROM; assumes CS0 will be used for RESET vector at address 0x00000000):
TwWAIT—6 clocks
TrWAIT—6 clocks
THLD—1 clock
TCS—1 clock
TOE—0, coincident with CSn_N
TWEF—3 clocks
TWER—1 clock
RDY_ENABLE—0, disable external ready.
CS1–CS7 unaffected by external reset (value retained).
CS1–CS7 set to 0x31811031 as above by POR reset.

- Bits [31–27]—TwWAIT → Depends on RDY_ENABLE (Bit [21])
  - If RDY_ENABLE=0, TwWAIT specifies the width of the chip select active period for the write cycle, allows for 0–31 resulting in a wait time of 1–32 clocks.
  - If RDY_ENABLE=1, TwWAIT specifies the wait time before the RDY_N line is first sampled for the write cycle. This provides a max wait time of 484nS at 66 MHz, anything greater than this will require the external RDY_N line and external logic.

- Bits [26–22]—TrWAIT → Depends on RDY_ENABLE (Bit [21])
  - If RDY_ENABLE=0, specifies the width of the chip select active period for the read cycle, allows for 0–31 resulting in a wait time of 1–32 clocks.
  - If RDY_ENABLE=1, specifies the wait time before the RDY_N line is first sampled for the read cycle. This provides a max wait time of 484nS at 66 MHz. Anything greater than this will require the external RDY_N line and external logic.

- Bit [21]—Ready Enable → Use is described above

- Bits [20–19]—Reserved

- Bits [18–16]—THLD → Specifies the time between when the CSn_N and BEn_N signals go inactive (hi) and the address is removed. Value is 0–7 clocks.

- Bits [15–14]—Reserved

- Bits [13–12]—TCS → Specifies the time between when the address bus is driven and the CSn_N and BEn_N signals go active (low). Value is 0–3 clocks.

- Bits [11–10]—Reserved

- Bits [9–8]—TOE → Specifies the time between when the CSn_N and BEn_N signals go active (low) and the OE signal goes active (low). Value is 0–3 clocks.

- Bits [7–6]—Reserved

- Bits [5–4]—TWEF → Specifies the time between when the CSn_N and BEn_N signals go active (low) and the WE_N signal goes active (low). Value is 0–3 clocks.

- Bits [3–2]—Reserved

- Bits [1–0]—TWER → Specifies the time between when the WE_N signal goes inactive (hi) and the CSn_N and BEn_N signals go inactive (hi). Value is 0–3 clocks.

6.3 External Bus Default Timing Register

This register defines the default timing for all external bus memory accesses not specifically mapped to either of the following: one of the internal peripherals or one of the 8 external chip selects. There is but one of these registers. Once defined, all the unmapped external memory cycles will be defined alike. This register is access-controlled and writable by only the Master Context, Context_0 (see Chapter 11, Access-Controlled Registers).

For this type of memory access, the external RDY_N input signal will always be enabled and will determine whether the cycle ends in a bus fault. If the RDY_N signal is not asserted (low) within 256 clocks from the end of the TwWAIT/TrWAIT period, a bus fault is generated (see Table 6-4).

Table 6-4. External Bus Default Timing Register

31–27	26–22	21	20–19	18–16	15–14	13–12	11–10	9–8	7–6	5–4	3–2	1–0
TwWAIT	TrWait	RDY_ENABLE = 1	Reserved	THLD	Reserved	TCS	Reserved	TOE	Reserved	TWEF	Reserved	TWER
RW	RW	R	–	RW	–	RW	–	RW	–	RW	–	RW

Note: Reset value is 0x18E11011:
TwWAIT—3 clocks
TrWAIT—3 clocks
THLD—1 clock
TCS—1 clock
TOE—0, coincident with CSn_N
TWEF—1 clock
TWER—1clock
RDY_ENABLE—1, enable external ready

- Bits [31–27]—TwWait → Specifies the wait time before the RDY_N line is first sampled for the write cycle. This provides a max wait time of 484nS at 66 MHz.
- Bits [26–22]—TrWait → Specifies the wait time before the RDY_N line is first sampled for the read cycle. This provides a max wait time of 484nS at 66 MHz.
- Bit [21]—RDY_ENABLE → The RDY_N input is always used for these external memory cycles and as such, the RDY_ENABLE bit of this register is always forced to 1.
- Bits [20–19]—Reserved
- Bits [18–16]—THLD → Specifies the time between when the CSn_N and BEn_N signals go inactive (hi) and the address is removed. Value is 0–7 clocks.
- Bits [15–14]—Reserved
- Bits [13–12]—TCS → Specifies the time between when the address bus is driven and the CSn_N and BEn_N signals go active (low). Value is 0–3 clocks.

- Bits [11–10]—Reserved
- Bits [9–8]—TOE → Specifies the time between when the CSn_N and BEn_N signals go active (low) and the OE signal goes active (low). Value is 0–3 clocks.
- Bits [7–6]—Reserved
- Bits [5–4]—TWEF → Specifies the time between when the CSn_N and BEn_N signals go active (low) and the WE_N signal goes active (low). Value is 0–3 clocks.
- Bits [3–2]—Reserved
- Bits [1–0]—TWER → Specifies the time between when the WE_N signal goes inactive (hi) and the CSn_N and BEn_N signals go inactive (hi). Value is 0–3 clocks.

6.4 External Bus Priority Register

The External Bus Priority Register is used to assign the priority that an external bus master (via request/grant) has relative to the internal fido1100 resources (i.e., the five contexts, the CPU DMA controller, etc.) (see Table 6-5). This register is access-controlled and writable by only the Master Context, Context_0 (see Chapter 11, Access-Controlled Registers).

Table 6-5. External Bus Priority Register

31–3	2–0
Reserved	Priority
Reserved	Priority

Note: Reset value is 0x00000000.

- Bits [31–3]—Reserved
- Bits [2–0]—Priority → The priority the external bus is assigned, 0 is the lowest, 7 the highest.

6.5 SDRAM Controller Registers

The fido1100 has a built-in SDRAM controller that can handle up to eight banks of external SDRAM. These registers are access-controlled and writable by only the Master Context, Context_0 (see Chapter 11, Access-Controlled Registers).

The registers used to program the SDRAM controller are presented in Table 6-6.

Table 6-6. Registers Used to Program the SDRAM Controller

Register Name	Description
SDRAM_Timing_Parameter_0_Register	SDRAM timing control register 0. A single, access-controlled register, writable by only the Master Context (Context_0).
SDRAM_Timing_Parameter_1_Register	SDRAM timing control register 1. A single, access-controlled register, writable by only the Master Context (Context_0).
SDRAM_Configuration_0_Register	SDRAM configuration control register 0. A single, access-controlled register, writable by only the Master Context (Context_0).
SDRAM_Configuration_1_Register	SDRAM configuration control register 1. A single, access-controlled register, writable by only the Master Context (Context_0).
SDRAM_Ext_Bank_0_Register SDRAM_Ext_Bank_1_Register SDRAM_Ext_Bank_2_Register SDRAM_Ext_Bank_3_Register SDRAM_Ext_Bank_4_Register SDRAM_Ext_Bank_5_Register SDRAM_Ext_Bank_6_Register SDRAM_Ext_Bank_7_Register	SDRAM external bank control registers (8 of these), one per chip select region. Used to enable the bank and specify base address. Access-Controlled registers, writable by only the Master Context (Context_0).

It should be noted that the SDRAM controller is inactive unless Bit [10] in the Clock Mask Register is set to logic 0 (see Chapter 10, Power Control, for details).

Listed below are the SDRAM registers. They are detailed in sections that follow:

- SDRAM Timing Parameter 0 Register
- SDRAM Timing Parameter 1 Register
- SDRAM Configuration 0 Register
- SDRAM Configuration 1 Register
- SDRAM External Bank Configuration Register_N (where N=0..7)

6.5.1 SDRAM Timing Parameter 0 Register

This is a single register used to set SDRAM timing parameters (see Table 6-7). It is access-controlled and writable by only the Master Context, Context_0 (see Chapter 11, Access-Controlled Registers).

Table 6-7. SDRAM Timing Parameter 0 Register

31–20	19–16	15	14–12	11–8	7–6	5–4	3–2	1–0
Reserved	TRP	Reserved	TRCD	TRF	Reserved	TWR	Reserved	TCL
–	RW	–	RW	RW	–	RW	–	RW

Note: Reset value is 0x00A22602.

- Bits [31–20]—Reserved
- Bits [19–16]—TRP → Pre-charge cycle time. This parameter specifies the cycles needed by the pre-charge command. That is, the next valid SDRAM command can be issued after the time specified in this parameter.
- Bit [15]—Reserved
- Bits [14–12]—TRCD → RAS-to-CAS delay. This parameter specifies the minimum period between active command and the following read/write command. The maximum allowed. Value is 3.
- Bits [11–8]—TRF → Auto-refresh cycle time. This parameter specifies the time needed by SDRAM to execute auto-refresh command. That is, the next valid SDRAM command can be issued after the time specified in this parameter.

Note: The minimum value for this field is 3. If 1 or 2 is put here, it will cause "double refreshes." Although not harmful, they can be distracting.

- Bits [7–6]—Reserved
- Bits [5–4]—TWR → Write-recovery time. This parameter specifies the period between pre-charge and the last valid write data and the period between the last read data out and write command.
- Bits [3–2]—Reserved
- Bits [1–0]—TCL → CAS-latency. This parameter specifies the time between read command and the first data out. Due to limitation of the read pipeline, the allowed CAS latency is 2 or 3.
  - 00—Illegal
  - 01—Illegal
  - 10—CAS=2
  - 11—CAS=3

6.5.2 SDRAM Timing Parameter 1 Register

This is a single register used to set SDRAM timing parameters (see Table 6-8). It is access-controlled and writable by only the Master Context, Context_0 (see Chapter 11, Access-Controlled Registers).

Table 6-8. SDRAM Timing Parameter 1 Register

31–24	23–20	19–16	15–0
Reserved	INI_PREC	INI_REFT	REF_INTV
–	RW	RW	RW

Note: Reset value is 0x00480820.

- Bits [31–24]—Reserved
- Bits [23–20]—INI_PREC → Initial pre-charge times. The default value of this field is 4.
- Bits [19–16]—INI_REFT → Initial refresh times. The default value of this field is 8.
- Bits [15–0]—REF_INTV → Refresh interval. One refresh command should be issued if the refresh counter equals the refresh interval.

6.5.3 SDRAM Configuration 0 Register

This is a single register used to set SDRAM configuration parameters (see Table 6-9). It is access-controlled and writable by only the Master Context, Context_0 (see Chapter 11, Access-Controlled Registers).

Table 6-9. SDRAM Configuration 0 Register

31–17	16	15–14	13–12	11	10–8	7–6	5–4	3–0
Reserved	MA2T	Reserved	DDW	Reserved	DSZ	Reserved	MBW	BNKSIZE
–	RW	–	RW	–	RW	–	RW	RW

Note: Reset value is 0x00001226.

- Bits [31–17]—Reserved
- Bit [16]—MA2T → Double Memory Address Cycle Enable. This register is used to control if the address should be validated before the SDRAM command is issued. *This bit should always be set to "0" for the fido1100.*
- Bits [15–14]—Reserved

- Bits [13–12]—DDW → SDRAM Data Width. This register indicates the data width of each individual SDRAM Module. This indicates the width of a single SDRAM device attached to fido1100.
  - 00—×4 device
  - 01—×8 device
  - 10—×16 device
  - 11—Reserved

- Bit [11]—Reserved

- Bits [10–8]—DSZ → SDRAM Size. This register indicates the size of each individual SDRAM Module (in bits). This indicates the size of a single SDRAM device attached to the fido1100:
  - 000—16 Mbit
  - 001—64 Mbit
  - 010—128 Mbit
  - 011—256 Mbit
  - 100—512 Mbit
  - 101—Reserved
  - 110—Reserved
  - 111—Reserved

- Bits [7–6]—Reserved

- Bits [5–4]—MBW → Memory Bus Width. This register indicates the bus size of external memory bus. This is width of the fido1100 external data bus connected to the SDRAM devices:
  - 00—Memory data width is 8
  - 01—Memory data width is 16
  - 10—Reserved
  - 11—Reserved

- Bits [3–0]—BNKSIZE → Bank Size. The following encoding shows the size of bank (in bytes). Bank sizes other than the following values may cause an unexpected error. This identifies the total size of the memories attached to a single Chip Select:
  - 0000—Reserved
  - 0001—2 Mbyte
  - 0010—4 Mbyte
  - 0011—8 Mbyte
  - 0100—16 Mbyte
  - 0101—32 Mbyte
  - 0110—64 Mbyte
  - 0111—128 Mbyte

– 1000—256 Mbyte
– 1001—Reserved
– 1010—Reserved
– 1011—Reserved
– 1100—Reserved
– 1101—Reserved
– 1110—Reserved
– 1111—Reserved

6.5.4 SDRAM Configuration 1 Register

This is a single register used to set SDRAM configuration parameters (see Table 6-10). It is access-controlled and writable by only the Master Context, Context_0 (see Chapter 11, Access-Controlled Registers).

Table 6-10. SDRAM Configuration 1 Register

31-5	4	3	2	1	0
Reserved	IPREC	IREF	ISMR	PWDN	SREF
–	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–5]—Reserved
- Bit [4]—IPREC → Initial pre-charge start flag. If IPREC is set to “1,” the SDRAM controller will start pre-charging if SDRAM stays in the IDLE state. This flag will be cleared to zero if the executed pre-charge command is equal to the specified initial pre-charge count. Writing 0 to this bit has no affect.
- Bit [3]—IREF → Initial refresh start flag. If IREF is set to “1,” refresh controller will start sending refresh command to control engine until initial refresh time. This flag will be cleared if the executed refresh command is equal to the specified initial refresh time. Writing 0 to this bit has no affect.
- Bit [2]—ISMR → Start set-mode-register. If ISMR is set to “1,” refresh controller will send a set-mode-register command to control engine. This bit will be cleared if set-mode-register command is done. Writing 0 to this bit has no affect.
- Bit [1]—PWDN → Power-down operation mode. If this parameter is set to “1,” SDRAM controller will pull CKE low to suspend the clock while SDRAM controller is in IDLE state. That is, all queued write buffers in SDRAM controller are cleared and all SDRAM banks are pre-charged.
  – The power-down command will cause the controller to clear the CKE line during a no op command. CKE will remain low while in power-down mode.

- After entering power-down mode, the PWDN bit will remain set until a write to the register clears it.
- While in power-down mode, the controller will continue to perform refreshes and read/write operations will also be allowed. CKE will go high to perform the operation, and then will go low to put memory back into power-down mode.

- Bit [0]—SREF → Self-refresh mode. If this parameter is set to “1,” SDRAM controller will send self-refresh-entry command to SDRAM while controller is in IDLE state (all write buffers queued are cleared and all SDRAM banks are pre-charged). After entering into self-refresh state, this bit will be cleared to 0. Setting this bit to 0 has no effect.

  - The self-refresh command will cause the controller to clear the CKE line during a refresh command. CKE will remain low while in self-refresh mode.
  - While in self-refresh mode, the controller will not make any bus requests nor perform any refresh commands. However data in the SDRAM memory will be retained (the SDRAM memory will be performing internal self-refreshes).
  - Any read or write operation will cause the controller to leave self-refresh mode. CKE will go back high, and the controller will resume refresh operations.

6.5.5 SDRAM External Bank Configuration Register_N (where N=0..7)

There are eight registers used to set SDRAM bank configuration parameters, one per chip select region (see Table 6-11). They are access-controlled and writable by only the Master Context, Context_0 (see Chapter 11, Access-Controlled Registers).

Table 6-11. SDRAM External Bank Configuration Register

31–13	12	11–0
Reserved	BNK_EN	BNK_BASE
–	RW	RW

Note: Reset value is:
SDRAM_Ext_Bank_0 → 0x1800
SDRAM_Ext_Bank_1 → 0x0820
SDRAM_Ext_Bank_2 → 0x0840
SDRAM_Ext_Bank_3 → 0x0860
SDRAM_Ext_Bank_4 → 0x0880
SDRAM_Ext_Bank_5 → 0x08A0
SDRAM_Ext_Bank_6 → 0x08C0
SDRAM_Ext_Bank_7 → 0x08E0
The default values of these registers puts the Bank base address in the “little-endian” memory space.

- Bits [31–13]—Reserved

- Bit [12]—BNK_EN → Bank enable flag.
  - 0—Bank is disabled

  – 1—Bank is enabled

Note 1: External Bank 0 is enabled by default. All others are disabled

Note 2: The SDRAM Enable flag in the associated External Bus Chip Select Register must be set.

- Bits [11–0]—BNK_BASE → 12-bit base address of external bank. This field is equivalent to Bits [31–20] of the fido1100 address bus.

6.6 Startup and Operation of SDRAM Controller

The External Data bus can be configured at either 8 or 16 bits wide; however, the SDRAM devices must fill the entire width. If using a 16-bit-wide bus, at least one ×16 device, two ×8 devices, or four ×4 devices must be used.

There is only one SDRAM Controller in the fido1100. However, the controller can be set up to control eight separate Chip Selects (via the SDRAM External Bank Configuration Registers).

There is a one-to-one interaction between the SDRAM External Banks and the External Bus Chip Selects. External Bank 0 will operate via Chip Select 0; External Bank 1 will operate via Chip Select 1, etc.

The settings for Timing 0, Timing 1, and Config 0 Registers are global; hence, the SDRAM devices attached to each chip select must have the exact same timing and configuration. CS1 cannot be configured as a 16-bit-wide bus with a single ×16 8-Mbyte device and have CS2 configured as an 8-bit-wide bus with two ×4 devices (see Setups for SDRAM Controller Examples provided in this section).

6.6.1 Initial Setups

1. Wait for MEMCLK to stabilize. MEMCLK is clocked at the same rate as the system clock.

2. Wait 200 µsecs for the SDRAM device to stabilize (this delay may be less depending upon the SDRAM device requirements).

3. Ensure that Bit [10] of the Clock Mask Register is set to "0." This will turn on the master clock to the SDRAM Controller.

4. Set the External Bus Chip Select Register for the chip select connected to the SDRAM Device.
   – SDRAM Enable must be set to "1."

- Output Enable must be set to "1." If configuring CS4-CS7, the Mode Select must be set to "0."
- The Base Address Bits [31–20] must be set to the base address of the SDRAM memory attached to this chip select.
  - Base Address Bits [19–12] must be set to "0."
  - The 12-bit-value set in Bits [31–20] must match exactly with Bits [11–0] of the appropriate SDRAM External Bank Configuration Register BNK_BASE field.
- The Size field must be properly set to the size of the SDRAM memory attached to this chip select.

5. Set SDRAM Timing Parameter 0 Register.
   - The TRP, TRCD, TRF, and TWR fields represent the number of clocks that must occur to meet the timing requirements of the SDRAM Device. The data sheet for the SDRAM device must be evaluated against the clock speed that the fido1100 is running at (see Example 1 and Example 2 below).
   - The TCL field (number of clocks before data is ready from SDRAM Device) can be set to "2" for most SDRAMS. Because fido1100 max clock is 66 MHz, and most SDRAM are set for ≥100 MHz, CAS Latency can be set to "2."

6. Set SDRAM Timing Parameter 1 Register.
   - The INI_PREC field identifies the number of initial pre-charges that will occur before setting the SDRAM Device Mode Register. The SDRAM Device should be consulted for the appropriate number.
   - The INI_REFT field identifies the number of initial refreshes (after the initial pre-charges) that will occur before setting the SDRAM Device Mode Register. The SDRAM Device should be consulted for the appropriate number.
   - The REF_INTV field should be computed by the following formula (REF_INTV = [Tref / num_rows] * Fclk).
     - Tref—The time interval that all rows should be refreshed. Consult the SDRAM Device data sheet.
     - num_rows—The number of rows that the SDRAM Device contains. Consult the SDRAM Device data sheet.
     - Fclk—The frequency of the system clock.

Note: Drop any fractions and convert the value to hex for the register.

7. Set SDRAM Configuration 0 Register.
   - The MA2T field must be set to "0."
   - The MBW field identifies the width of the External Data bus connected to the SDRAM Devices. The fido1100 supports only 8- and 16-bit-wide buses.
   - The DDW field identifies the width of a single SDRAM Device. The external data bus must be fully populated (i.e., a 16-bit-wide bus requires either one ×16 device, two ×8 devices, or four ×4 devices).

- The DSZ field identifies the size (in bits) of a single SDRAM device connected to the Data Bus.
  - This represents the size of a single ×4, ×8, or ×16 device.
  - The options are 16, 64, 128, 256, and 512 Mbits
- The BNKSIZE field identifies the size (in bytes) of all SDRAM devices tied to a single External Bank.
  - If MBW—16 bits, and there are four ×4 64 Mbit, the bank size would be 32 Mbytes.
  - If MBW—16 bits, and there are two ×8 16 Mbit, the bank size would be 4 Mbytes.
  - If MBW—8 bits, and there are 1 x8 64Mbit, the bank size would be 8 Mbytes.

Note: This register affects all external banks. The exact same External Bus Width and types of SDRAM devices must be the same on each external bank. Only the base address can be changed for an external bank.

8. Set SDRAM Configuration 1 Register (Command Register)
   - To initialize the SDRAM devices:
     - Set the IPREC Bit to cause the number of initial Pre-charges (identified by INI_PREC in Timing Parameter 1) to occur.
     - Set the IPREC Bit to cause the number of initial Refreshes (identified by INI_REFT in Timing Parameter 1) to occur. These will occur after the initial Pre-charges.
     - Set the ISMR Bit to cause the SDRAM Controller to set the SDRAM device Mode register. The Mode register will be set to Sequential burst length of 4 and the CAS Latency will be set the same as the TCL field of Timing Parameter 0.
   - All 3 bits (IPREC, IREF, and ISMR) can be set at the same time.
   - After setting the bits, read the register until it equals zero.

Note: A short delay (e.g., 4 NOP commands) may be needed before and after writing to this register to allow the setting of the previous registers to propagate inside of the SDRAM Controller.

Warning: After performing the ISMR function, do not perform another as indeterminate results may occur.

9. Set SDRAM External Bank Configuration Register(s)
   - Each External Bank corresponds to an external chip select.
   - The BNK_BASE field identifies the upper 12 bits of the base address of the SDRAM Device(s) on the Chip Select. The address space of the SDRAM Device(s) is defined in the BNKSIZE field of Configuration Register 0.
   - The BNK_EN will enable/disable accesses to the SDRAM memory.

10. SDRAM is now ready for normal operations
    - Refresh Commands to the SDRAM devices should occur at the correct interval

  - Transfers between the SDRAM Device and the SDRAM Controller will always occur via a 4 -column burst.
  - Bursts via the DMA system will perform multiple 4-column bursts

6.6.2 Setups for SDRAM Controller Examples

Micron SDRAM Models MT48LC16M4A2-75 (64-Mbit ×4 device), MT48LC8M8A2-75 (64-Mbit ×8 device), MT48LC4M16A2-75 (64-Mbit ×16 device)

- 4096 rows, must be refreshed every 64 milliseconds.
- tRP—20 ns min
- tRCD—20 ns min
- tRFC—66 ns min (Auto Refresh Period)
- tWR—1 Clk + 7.5 ns min
- Startup requires one initial pre-charge and two initial refreshes

6.6.2.1 SDRAM Controller Example 1

A 66-MHz; 16-bit Data Bus, single ×16 device:

- Timing 0—0x00022522 // TRP=2, TRCD=2, TRF=5, TWR=2, TCL=2.
  - Clock is 15ns period
    - Need two clocks to meet TRP, TRCD, and TWR.
    - Need five clocks to meet TRF (tRFC) period
    - Because it is running slow compared to SDRAM, the device can use CAS Latency (TCL) = 2.

- Timing 1—0x00120407//INI_PREC=1, INI_REFT=2, REF_INTV=0x0407.
  - REF_INTV = 64 msec/4096 × 66 MHz = 1031 = ≥0x0407

- Config 0—0x00002113//MA2T = 0, DDW = ×16 device, dsz = 64 Mbit, mbw = 16 bit, bnksz = 8 Mbyte.
  - Single, 64-Mbit device = 8 Mbytes

6.6.2.2 SDRAM Controller Example 2

A 22-MHz, 8-bit Data Bus, two ×4 devices:

- Timing 0—0x00011112//TRP=1, TRCD=1, TRF=1, TWR=1, TCL=2
  - Clock is 45ns period
    - Need 1 clock to meet TRP, TRCD, and TWR.
    - Need 1 clock to meet TRF (tRFC) period
    - Because it is running slow compared to SDRAM, the device can use CAS Latency (TCL) = 2
- Timing 1—0x00120157//INI_PREC=1, INI_REFT=2, REF_INTV=0x0157
  - REF_INTV = 64 msec/4096 × 22 MHz = 343.8 = ≥0x0157

- Config 0—0x00000104//MA2T=0, DDW= ×4 device, dsz = 64 Mbit, mbw = 8 bit, bnksz = 16 Mbyte
  - Two, 64-Mbit devices = 16 Mbytes

6.7 SDRAM Module Types Address Mapping (16-Bit Bus Width)

Table 6-12 provides address-mapping information for the listed SDRAM module types when using a 16-bit bus width. The "AP" is Auto Pre-Charge.

Table 6-12. SDRAM Module Types Address Mapping for 16-Bit Bus Width

Module Type	Total Bank Size	Number of SDRAM Devices	Bank Select	Row Address	Column Address													Rows	Cols
					12	11	10	9	8	7	6	5	4	3	2	1	0		
16M (×16)	2 Mbytes	1	9	20–10	–	–	AP	–	–	8	7	6	5	4	3	2	1	2048	256
16M (×8)	4 Mbytes	2	10	21–11	–	–	AP	–	9	8	7	6	5	4	3	2	1	2048	512
16M (×4)	8 Mbytes	4	11	22–12	–	–	AP	10	9	8	7	6	5	4	3	2	1	2048	1024
64M (×16)	8 Mbytes	1	10, 9	22–11	–	–	AP	–	–	8	7	6	5	4	3	2	1	4096	256
64M (×8)	16 Mbytes	2	11, 10	23–12	–	–	AP	–	9	8	7	6	5	4	3	2	1	4096	512
64M (×4)	32 Mbytes	4	12, 11	24–13	–	–	AP	10	9	8	7	6	5	4	3	2	1	4096	1024
128M (×16)	16 Mbytes	1	11, 10	23–12	–	–	AP	–	9	8	7	6	5	4	3	2	1	4096	512
128M (×8)	32 Mbytes	2	12, 11	24–13	–	–	AP	10	9	8	7	6	5	4	3	2	1	4096	1024
128M (×4)	64 Mbytes	4	13, 12	25–14	–	11	AP	10	9	8	7	6	5	4	3	2	1	4096	2048
256M (×16)	32 Mbytes	1	11, 10	24–12	–	–	AP	–	9	8	7	6	5	4	3	2	1	8196	512
256M (×8)	64 Mbytes	2	12, 11	25–13	–	–	AP	10	9	8	7	6	5	4	3	2	1	8196	1024
256M (×4)	128 Mbytes	4	13, 12	26–14	–	11	AP	10	9	8	7	6	5	4	3	2	1	8196	2048
512M (×16)	64 Mbytes	1	12, 11	25–13	–	–	AP	10	9	8	7	6	5	4	3	2	1	8196	2048
512M (×8)	128 Mbytes	2	13, 12	26–14	–	11	AP	10	9	8	7	6	5	4	3	2	1	8196	4096
512M (×4)	256 Mbytes	4	14, 13	27–15	–	11	AP	10	9	8	7	6	5	4	3	2	1	8196	8192

6.8 SDRAM Module Types Address Mapping (8-Bit Bus Width)

Table 6-13 provides address-mapping information for the listed SDRAM module types when using an 8-bit bus width. The "AP" is Auto Pre-Charge.

Table 6-13. SDRAM Module Types Address Mapping for 8-bit Bus Width

Module Type	Total Bank Size	Number of SDRAM Devices	Bank Select	Row Address	Column Address													Rows	Cols
					12	11	10	9	8	7	6	5	4	3	2	1	0		
16M (×8)	2 Mbytes	1	9	20–10	–	–	AP	–	8	7	6	5	4	3	2	1	0	2048	512
16M (×4)	4 Mbytes	2	10	21–11	–	–	AP	9	8	7	6	5	4	3	2	1	0	2048	1024
64M (×8)	8 Mbytes	1	10, 9	22–11	–	–	AP	–	8	7	6	5	4	3	2	1	0	4096	512
64M (×4)	16 Mbytes	2	11, 10	23–12	–	–	AP	9	8	7	6	5	4	3	2	1	0	4096	1024
128M (×8)	16 Mbytes	1	11, 10	23–12	–	–	AP	9	8	7	6	5	4	3	2	1	0	4096	1024
128M (×4)	32 Mbytes	2	12, 11	24–13	–	10	AP	9	8	7	6	5	4	3	2	1	0	4096	2048
256M (×8)	32 Mbytes	1	11, 10	24–12	–	–	AP	9	8	7	6	5	4	3	2	1	0	8196	1024
256M (×4)	64 Mbytes	2	12, 11	25–13	–	10	AP	9	8	7	6	5	4	3	2	1	0	8196	2048
512M (×8)	64 Mbytes	1	12, 11	25–13	–	10	AP	9	8	7	6	5	4	3	2	1	0	8196	4096
512M (×4)	128 Mbytes	2	13, 12	26–14	11	10	AP	9	8	7	6	5	4	3	2	1	0	8196	8192

6.9 SDRAM External I/O Signal List

In addition to the External Bus Interface Signal List presented in Table 6-1, the hardware signals used for SDRAM external I/O are presented in Table 6-14.

Table 6-14. SDRAM External I/O Signal List

Signal Name	Type	Description
RAS_N	Output	Row Address Strobe (active low)
CAS_N	Output	Column activate signal (active low)
BA_0 BA_1	Outputs	Bank Enables (active low)
CKE	Output	Clock Enable used in conjunction with MEMCLK (active high)

6.10 External Bus Arbitration

An external bus arbitration takes place as follows:

1. External master asserts HOLDREQ_N

2. The fido1100 evaluates current internal bus priority (the higher of current context priority and active DMA priority) versus value in External Bus Priority Register. If external bus

priority is higher than internal priority, HOLDGNT_N is asserted (after the current bus cycle completes) for the duration of the external access and all bus outputs from the fido1100 are tri-stated.

3. External bus can then begin controlling address and data bus. The external bus master must continue to assert HOLDREQ_N as long as it requires control of the bus.

4. The fido1100 continuously monitors the relative priorities of internal and external bus, if internal bus priority rises higher than or equal to external bus (e.g., a higher-priority context is enabled) then the fido1100 will deassert the HOLDGNT_N signal.

5. The fido1100 waits for the external bus master to deassert HOLDREQ_N before driving the bus (regardless whether the external master terminated on its own time or in response to deassertion of HOLDGNT_N).

Note 1: When the HOLDGNT_N is active, the fido1100 will tri-state the address and data bus so that it can be externally driven.

Note 2: Even while HOLDGNT_N is active, the fido1100 execution unit will continue to run, executing code, unless an external bus access is required by the application, at which point the fido1100 execution will temporarily stall, waiting for the external bus to be released.

7. Peripheral Management Unit

7.1 PMU

7.1.1 Overview

The Peripheral Management Unit (PMU) provides data transfers between the CPU and the UIC using Dual-Port RAM. Dual-Port RAM (as used in this document) is a RAM that can be simultaneously read from and written to (i.e., it has a read port and a write port). On the PMU, the Transmit Dual-Port has a write port associated with the CPU bus and a read port associated with the UIC bus. The Receive Dual-Port has a write port associated with the UIC bus and a read port associated with the CPU bus. Although the CPU can read from the Transmit Dual-Port and write to the Receive Dual-Port, these are slower than actions made in the usual direction.

The PMU can allocate a fixed set of resources (primarily transmit and receive Dual-Port RAM) among a variable set of resources (UICs with various protocols). The PMU Dual-Port RAM is a subset block of memory within the total fido1100 memory map. There are two channels of PMU available for each UIC, a primary and secondary channel. If the UIC is operating a single protocol (i.e., Ethernet), it is managed using the primary PMU channel—the secondary channel is not used. If the UIC is operating a dual protocol (e.g., dual UARTS, UART, and GPIO), then each protocol uses a channel.

The PMU Dual-Port RAM is divided into two sections; one for transmit-buffer allocations and one for receive-buffer allocations (see Figures 7-1 and 7-2). Transmit memory can be allocated a maximum buffer size of 1K $\times$ 32 bits. Receive memory can be allocated a maximum buffer size of 2K $\times$ 32 bits. Both receive and transmit memory sections are addressed using 32-bit (long-word) accesses and can be used for single-channel operation (one UIC only) or allocated among the four channels for multiple UIC operations.

7.1.1.1 PMU Register Overview

The PMU contains eight sets of registers that are used to configure transmit and receive buffers for the four UIC primary and secondary channels. PMU registers will be referred to by a generic naming system. For each set of PMU registers, the register names are preceded by “Chxy,” where the “x” corresponds to which UIC the PMU is associated with and the “y” corresponds to either the primary or secondary PMU channel (“A” for primary and “B” for secondary). For example, “Ch0A” would signify UIC 0 using a PMU primary channel.

- Control and status registers
  - PMUChxy_Control—Used to configure PMU channel operational mode and enable/disable PMU channel interrupts.
  - PMUChxy_Status—Read-only register that contains bits for interrupt status and state of PMU channel.

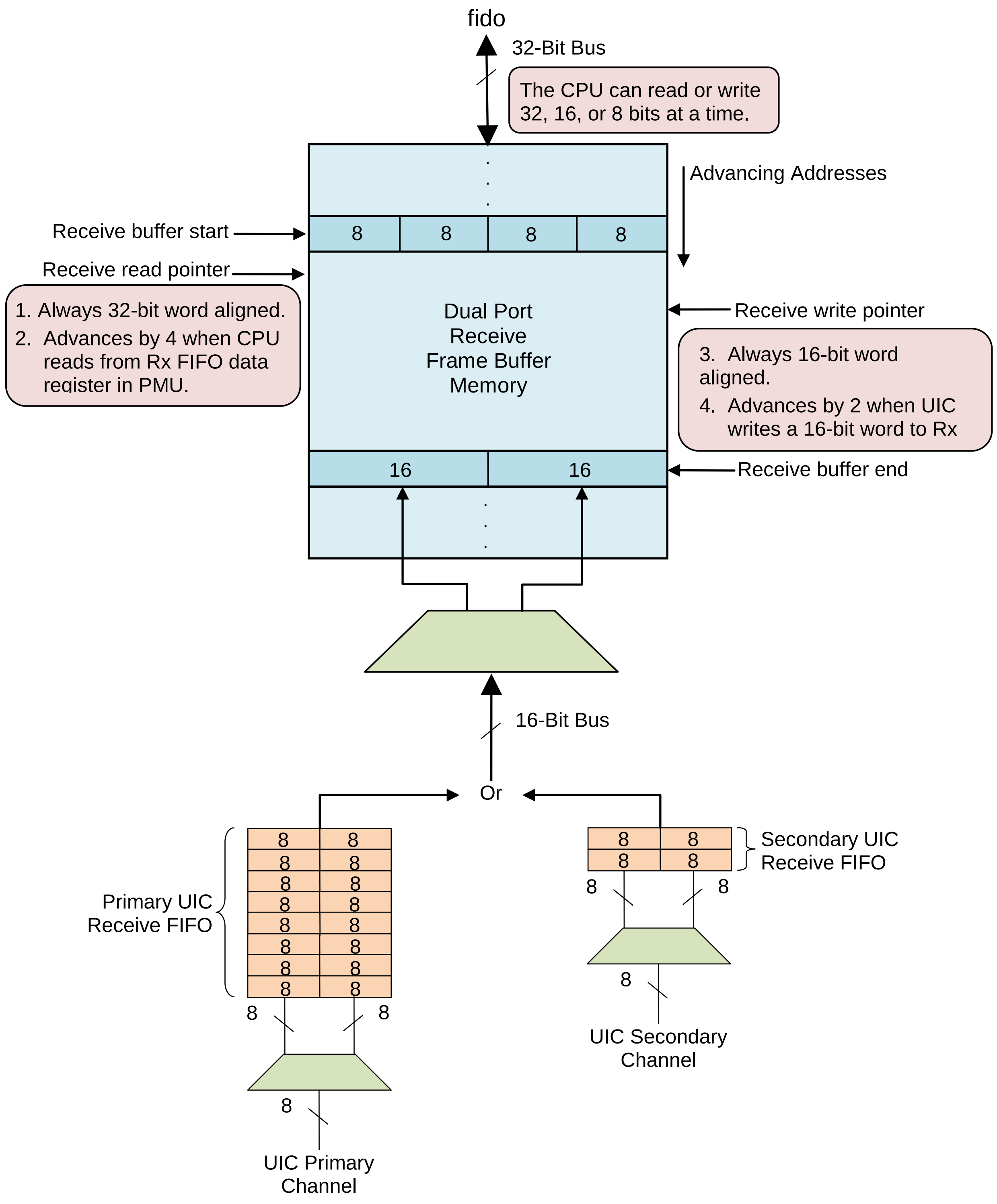

fido
32-Bit Bus
The CPU can read or write 32, 16, or 8 bits at a time.
Advancing Addresses
Receive buffer start
8
8
8
8
Receive read pointer
1. Always 32-bit word aligned.
2. Advances by 4 when CPU reads from Rx FIFO data register in PMU.
Dual Port Receive Frame Buffer Memory
Receive write pointer
3. Always 16-bit word aligned.
4. Advances by 2 when UIC writes a 16-bit word to Rx
16
16
Receive buffer end
16-Bit Bus
Or
Primary UIC Receive FIFO
Secondary UIC Receive FIFO
8
8
UIC Secondary Channel
8
8
8
UIC Primary Channel

Figure 7-1. Diagram of the Dual-Port Receive RAM

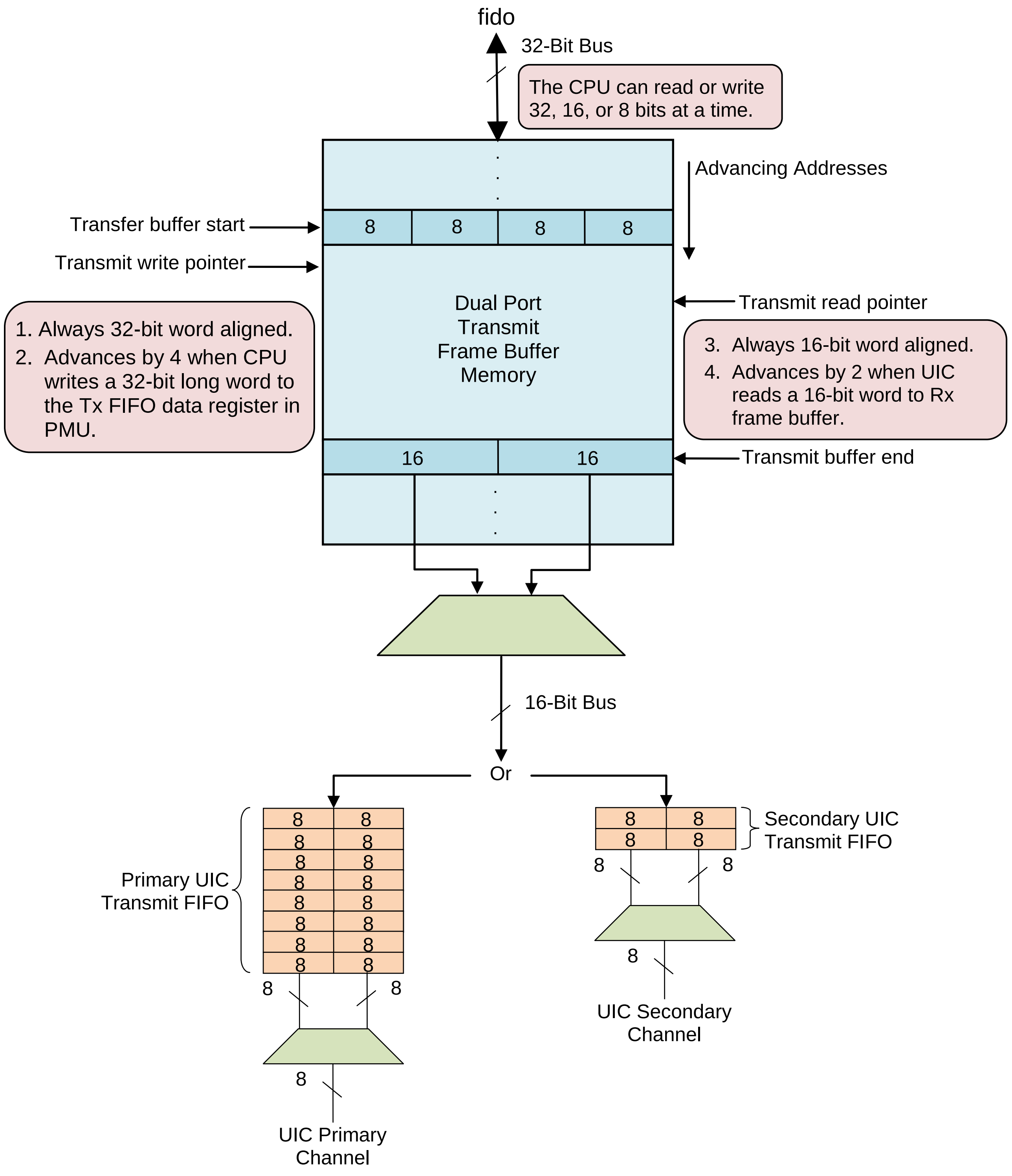

fido
32-Bit Bus
The CPU can read or write 32, 16, or 8 bits at a time.
Advancing Addresses
Transfer buffer start
8
8
8
8
Transmit write pointer
Dual Port Transmit Frame Buffer Memory
Transmit read pointer
1. Always 32-bit word aligned.
2. Advances by 4 when CPU writes a 32-bit long word to the Tx FIFO data register in PMU.
3. Always 16-bit word aligned.
4. Advances by 2 when UIC reads a 16-bit word to Rx frame buffer.
16
16
Transmit buffer end
16-Bit Bus
Or
Primary UIC Transmit FIFO
Secondary UIC Transmit FIFO
8
UIC Primary Channel
UIC Secondary Channel

Figure 7-2. Diagram of the Dual-Port Transmit RAM

- Transmit Registers
  - PMUChxy_XmitFBufStart—Start address of channel's transmit frame buffer in PMU Transmit RAM.
  - PMUChxy_XmitFBufEnd—Ending address of channel's transmit frame buffer in PMU Transmit RAM.
  - PMUChxy_XmitFBufWrPtr—Address in transmit frame buffer where next long-word written to PMUChxy_XmitData register will be stored.
  - PMUChxy_XmitFBufRdPtr—Address in transmit frame buffer where next data to be transferred to UIC will be read from.
  - PMUChxy_XmitData—Transmits FIFO write head when PMU channel is in FIFO mode. Long word written here will be stored at address pointed to by PMUChxy_XmitFBufWrPtr.
  - PMUChxy_PckXmitSize—Number of bytes to transmit.
- Receive Registers
  - PMUChxy_RcvFBufStart—Start address of channel's receive frame buffer in PMU Receive RAM.
  - PMUChxy_RcvFBufEnd—Ending address of channel's receive frame buffer in PMU Receive RAM.
  - PMUChxy_RcvFBufWrPtr—Address in receive frame buffer where next data received from UIC will be stored.
  - PMUChxy_RcvFBufRdPtr—Address in receive frame buffer where first data received from UIC was stored and where a read of PMUChxy_RcvData will take data from.
  - PMUChxy_RcvData—Receives FIFO read head when PMU channel is in FIFO mode. Long word read here will be taken from address pointed to by PMUChxy_RcvFBufrdPtr.
  - PMUChxy_PckRcvSize—Number of bytes received by PMU in last packet (rounded up to next even value).

Two modes of operation for data transfers, the FIFO or Random Access, are available between the CPU and the PMU. The FIFO mode treats each allocated frame buffer as circular. Hardware manages read and write pointers, providing an easy method for using circular buffers with minimal software. Data transfer into the transmit buffer and out of the receive buffer is accomplished by writing and reading 32 bits (long words) to the transmit (PMUChxy_Xmit_Data) and receive (PMUChxy_Rcv_Data) FIFO registers, respectively. The FIFO mode is especially effective when coupled with DMA transfers between memory and transmit or receive frame buffers. Using DMA and PMU, FIFO mode can significantly reduce the CPU processing required to transmit or receive data packets.

In FIFO mode, each long-word CPU or DMA write to the PMU FIFO transmit head (PMUChxy_XmitData) results in the data being stored at the location pointed to by the Transmit Frame Buffer Write Pointer (PMUChxy_XmitFBufWrPtr), incrementing it by four bytes. Once a transmit is started, the Transmit Frame Buffer Read Pointer (PMUChxy_XmitFBufRdPtr) increments by two bytes for every word read by the UIC. The PMU maintains both transmit pointers within the boundaries of the Transmit Frame Buffer set by its start-address register (PMUChxy_XmitFBufStart) and its end-address register (PMUChxy_XmitFBufEnd) including wrapping addresses as needed to create a circular buffer.

In FIFO mode, each long-word CPU or DMA read from the PMU FIFO receive head (PMUChxy_RcvData) results in the data being read from the location pointed to by the Receive Frame Buffer Read Pointer (PMUChxy_RcvFBufRdPtr), incrementing it by four bytes. As data is received from the UIC, the Receive Frame Buffer Write Pointer (PMUChxy_RcvFBufWrPtr) is updated. The PMU maintains both receive pointers within the boundaries of the Receive Frame Buffer set by its start-address register (PMUChxy_RcvFBufStart) and its end-address register (PMUChxy_RcvFBufEnd) to create a circular buffer.

In Random Access mode, the CPU reads or writes PMU transmit or receive buffers in the same way as any other memory mapped internal memory. This differs from FIFO mode where the CPU writes full packets of data to the transmit buffer and reads them from the receive buffer. This approach works well when there are many different packets to transmit or a need to process all bytes of a received packet.

If there is a need to transmit periodically the same message with just a few bytes changed or to receive some standard packet and only one or two bytes need to be read, Random Access mode provides a more efficient method. The CPU would load the standard packet into the allocated transmit buffer, manually controlling the Transmit Frame Buffer Write and Read Pointers. The CPU must manually set the read address in the PMU Transmit Frame Buffer Read Pointer (PMUChxy_XmitFBufRdPtr) to the starting location where the CPU placed the data. The PMU Transmit Frame Buffer Write Pointer (PMUChxy_XmitFBufWrPtr) is set to the next long word after the data. When a transmit start occurs, the data packet is then transferred to the UIC starting at the Transmit Frame Buffer Read Pointer (Chxy_XmitFBufRdPtr). After the data packet has been transmitted, the CPU could alter a small number of bytes in the packet, reset the PMU Transmit Frame Buffer Read Pointer (PMUChxy_XmitFBufRdPtr) to where the data packet was loaded, and start a new transmit without reloading the entire data packet.

The CPU can read any address within the PMU Receive Frame Buffer memory for PMU Random Access receive-buffer memory reads. Data is initially written by the UIC to the PMU within the Receive Frame Buffer boundaries allocated by its start-address register (PMUChxy_RcvFBufStart) and its end-address register (PMUChxy_RcvFBufEnd). Before a message is received, the CPU must set the Receive Frame Buffer Read Pointer (PMUChxy_RcvFBufRdPtr) to be less than the Receive Frame Buffer Write Pointer (PMUChxy_RcvFBufWrPtr). This will prevent the PMU from falsely triggering a receive-overflow error. The PMU tracks the data written from the UIC using the Receive Frame Buffer Write Pointer (PMUChxy_RcvFBufWrPtr). The CPU can read the Receive Frame Buffer Write Pointer (PMUChxy_RcvFBufWrPtr) to find the end of the data packet. The CPU must manage the Receive Frame Buffer Write Pointer (PMUChxy_RcvFBufWrPtr) in Random Access mode.

Note: In Random Access mode, the Transmit Data (PMUChxy_Xmit_Data) and Receive Data (PMUChxy_Rcv_Data) registers are not used because data placement in PMU memory is directly done by the CPU.

For both FIFO and Random Access modes, the Frame-Buffer-Start-address registers (PMUChxy_RcvFBufStart and PMUChxy_XmitFBufStart) and Frame-Buffer-End-address registers (PMUChxy_RcvFBufEnd and PMUChxy_XmitFBufEnd) are usually initialized with values that allocate enough space for data packets received and transmitted by that UIC.

When setting values for the frame buffer start and end pointers, the PMU memory is referenced using relative addressing. The addresses can range in value from 0x000 to 0xFFF and 0x0000 to 0x1FFF for Transmit and Receive Frame Buffers, respectively.

Note: Relative addressing does not apply for CPU accesses because the CPU uses 32-bit (long-word) addresses for accesses to the PMU.

The PMU interfaces with the UIC using a managed, FIFO-type buffering system. For both PMU Primary and Secondary channels, data is transferred to and from the UIC as 16-bit words. Both the primary transmit- and receive-channel FIFOs will buffer eight words (16 bytes) and the secondary transmit-and-receive-channel FIFOs will buffer two words (4 bytes). The UIC reads eight-bit (byte) data transfers from the primary- and secondary-channel FIFOs and writes 16-bit data transfers to both the primary- and secondary-channel FIFOs.

7.1.2 Interrupts

For each UIC, the PMU provides a single interrupt to the CPU (see Figure 7-3). This interrupt is an OR-ed combination of the PMU Primary-Channel and Secondary-Channel interrupts. Each of these sources can be enabled or disabled through the primary- and secondary-channel configuration registers.

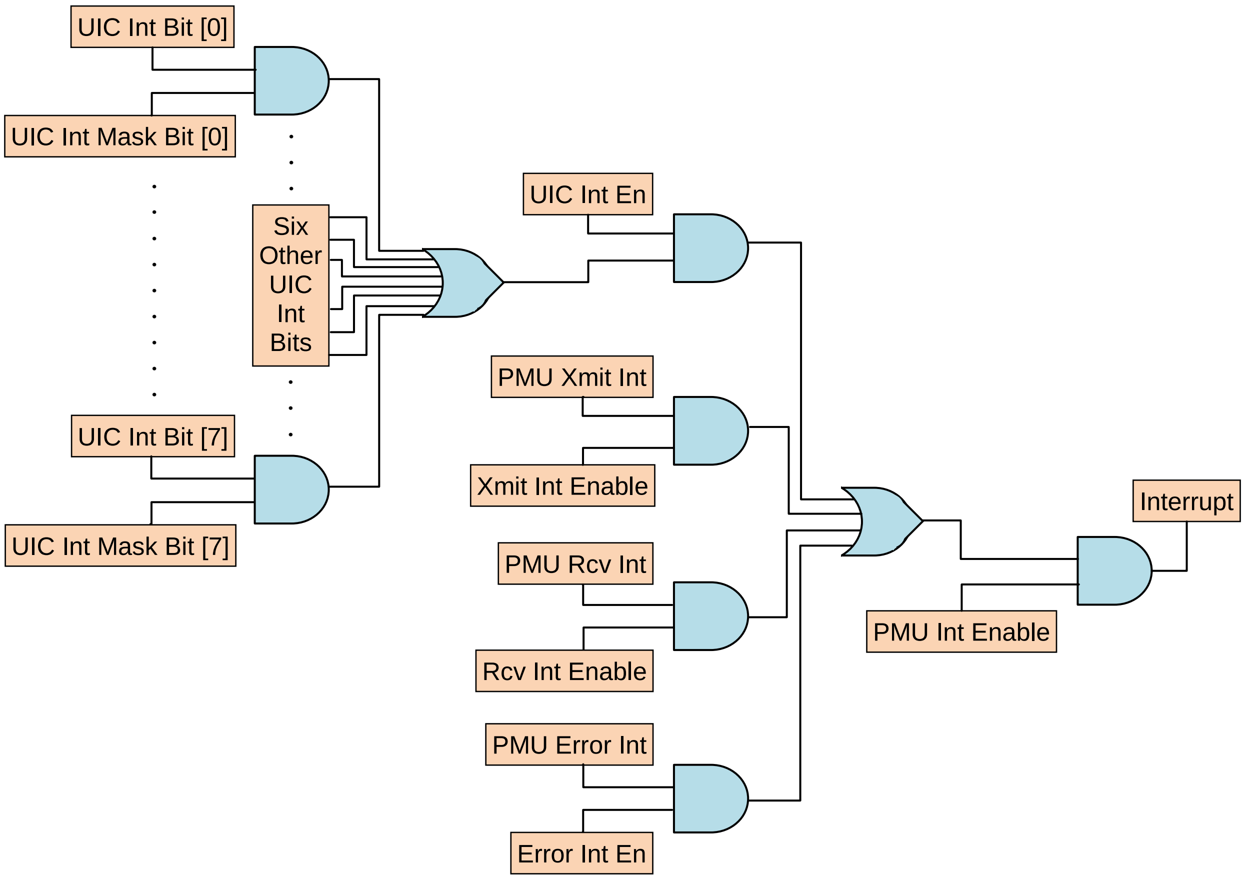


Figure 7-3. Diagram of PMU Interrupts

For each PMU primary and secondary channel, there are three data transfer interrupts between the PMU and UIC—transmit, receive, and error. These interrupts are OR-ed together to form the PMU channel interrupt. Each of these interrupts can be enabled or disabled through the channel configuration register (PMUChxy_Control).

Only in the PMU primary channel is an additional interrupt available from the UIC. This interrupt is controlled by UIC firmware and is OR-ed with the output of the three data-transfer interrupts as part of the PMU primary-channel interrupt. The UIC interrupt can be enabled or disabled through the PMU primary-channel control register. The UIC interrupt enable in the secondary-channel control register has no effect.

For interrupt context and priority assignment of the PMU for a particular UIC, the primary-channel control register is used. Settings are available for contexts zero through four and for interrupt priorities of one through seven. These settings have no effect in the secondary-channel configuration register.

Because there is only one PMU interrupt for both the primary and secondary channels, the interrupt handler for the PMU must evaluate more than one status register to determine the cause of an interrupt accurately. For example, if running single-protocol firmware on a UIC that uses the primary channel, the interrupt handler should first read the protocol-specific UIC Register status, and then the PMU primary channel's status register to evaluate an interrupt condition.

If running dual-protocol firmware on a UIC, the interrupt handler should read the UIC status register as well as both the primary and secondary channel's PMU status registers.

The following pseudo-code example demonstrates how an interrupt handler should operate for a UIC that is using both a primary and secondary PMU channel.

```
PMU_interrupt_handler()
{
        if(UIC_interrupt_flag)
        {
                handle_UIC_interrupt();
                clear_UIC_interrupt_flag();
        }

        if(PMU_primary_channel_interrupt_flag)
        {
                 if(transmit_interrupt)
                {
                        handle_transmit_interrupt();
                }
                else if(receive_interrupt_flag)
                {
                        handle_receive_interrupt();
                }
                else if(error interrupt_flag)
                {
                        handle_error_interrupt();
                }
        }
        else if(PMU_secondary_channel_interrupt_flag)
        {
                if(transmit_interrupt_flag)
                {
                        handle_transmit_interrupt();
                }
                else if(receive_interrupt_flag)
                {
                        handle_receive_interrupt();
                }
                else if(error_interrupt_flag)
                {
                        handle_error_interrupt();
                }
        }
}
```

7.1.3 PMU Registers

Each PMU channel has a set of registers where in “Chxy” the “x” represents which UIC the PMU is associated with and the “y” represents either primary or secondary PMU channel (“A” for primary and “B” for secondary). For example, “Ch0A” would signify UIC 0 using a PMU primary channel.

Note: These registers are dedicated, visible on the CPU bus, and not located in Dual-Port RAM. The following subsections explain each PMU register.

Listed below are the PMU registers. They are detailed in sections that follow:

- PMU Configuration/Control Register
- PMU Status Register
- PMU Packet Transmit Size Register
- PMU Packet Received Size Register
- PMU Receive Buffer Start Register
- PMU Receive Frame Buffer End Register
- PMU Transmit Frame Buffer Start Register
- PMU Transmit Frame Buffer End Register
- PMU Receive Frame Buffer Write Pointer Register
- PMU Receive Frame Buffer Read Pointer Register
- PMU Transmit Frame Buffer Read Pointer Register
- PMU Transmit Frame Buffer Write Pointer Register
- PMU Transmit Data Register
- PMU Receive Data Register

7.1.3.1 PMU Configuration/Control Register

The PMU configuration register sets up each PMU channel for operation with the UICs. Configurations include data packet sizes, interrupt allocation, and mode determination (see Table 7-1).

Table 7-1. PMUChxy_Control

31–19	18	17	16	15–13	12–11	10–8	7	6	5	4	3	2	1–0
Reserved	rcv packet size en	uic int en	pmu int en	int priority	Reserved	int context	channel en	xmit start	Reserved	xmt int enable	rcv int enable	error int enable	mode
–	RW	RW	RW	RW	RW	RW	RW	RW	–	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–19]—Reserved
- Bit [18]—Receive Packet Size Enable (rcv packet size en) → Determines whether the size of the received packet is written to the beginning of the packet buffer by hardware.
  - 0—Size not written to buffer, full buffer is used for received content.
  - 1—First long word in buffer is the size of the received packet.
- Bit [17]—UIC Interrupt Enable (uic int en) → Mask to allow UIC to interrupt directly the CPU via the PMU interrupt. The PMU interrupt (in proxy for the UIC) will be fired per the settings of the context and priority interrupt settings.

Note: This field applies only to the primary-channel control registers. It has no affect on secondary-channel registers.

- Bit [16]—PMU Interrupt Enable (pmu int en) → Interrupt enable for PMU Channel. The PMU interrupt will be fired using the settings of the Primary Channel Context and Priority interrupt settings.

Note: The UIC Interrupt Enable is independent of the PMU Interrupt Enable but requires the PMU Interrupt is enabled for operation.

- Bits [15–13]—Interrupt Priority (int priority) → Interrupt priority for the assigned context. Priorities range from zero through seven (000–111).

Note: This field applies only to the primary-channel control registers. It has no affect on secondary-channel registers.

- Bits [12–11] → Reserved

- Bits [10–8]—Interrupt Context (int context) → Assigns the PMU channel and its associated UIC to a CPU context
  - 000—Context_0
  - 001—Context_1
  - 010—Context_2
  - 011—Context_3
  - 100—Context_4
  - 101–111—Reserved

Note: This field applies only to the primary-channel control register. It has no affect on secondary-channel registers.

- Bit [7]—Channel Enable (channel en) → Enables this PMU channel

- Bit [6]—Transmit Start (xmit start) → Starts transmit of data from PMU to UIC
  - 0—No affect
  - 1—Start transmit (if enabled)

Note: Asserting this bit will clear the Transmit Complete (xmit complete) flag in the PMU status register.

- Bit [5]—Reserved

- Bit [4]—Transmit Interrupt Enable (xmit int enable) → Enables PMU Interrupt to fire when PMU has completed transferring data to UIC

- Bit [3]—Receive Interrupt Enable (rcv int enable) → Enables PMU Interrupt to fire when PMU has received a full block of data from the UIC

- Bit [2]—Error Interrupt Enable (error int enable) → Enables PMU Interrupt to fire when UIC indicates an error

- Bits [1–0]—PMU Mode (mode)
  - 00—FIFO Mode → Dual-Port Memory operates as a FIFO (CPU should read/write from Data Registers of PMU)
  - 01—Random Access Mode → Dual-Port Memory is random access (CPU can directly access Dual-Port Memory in PMU)
  - 10–11—Not Used

7.1.3.2 PMU Status Register

The PMU status register provides all indications for a PMU channel, based on interrupt and data transactions. All register values are read-only with most status bits cleared upon reading (see Table 7-2).

Table 7-2. PMUChxy_Status

31–11	10	9	8	7	6–4	3	2	1	0
Reserved	rcv FIFO empty	xmit FIFO full	Reserved	interrupt flag	Reserved	Rx overflow	uic error	rcv complete	xmit complete
–	R	R	–	R	–	R	R	R	R

Note: Reset value is 0x00000400.

- Bits [31–11]—Reserved
- Bit [10]—Receive FIFO Empty (rcv FIFO empty) → Indicates that the receive FIFO buffer is empty (PMUChxy_RcvFBufRdPtr equals PMUChxy_RcvFBufWrPtr)
- Bit [9]—Transmit FIFO Full (xmit FIFO full) → Indicates that the transmit FIFO buffer has no space available (PMUChxy_XmitFBufRdPtr is one location greater than PMUChxy_XmitFBufWrPtr)
- Bit [8]—Reserved
- Bit [7]—PMU Interrupt Flag (interrupt flag) → Indicates the PMU has fired an interrupt (cleared on read)

Note: Serves as interrupt acknowledge for PMU-generated interrupt and must read UIC status registers first to clear interrupt conditions originating there.

- Bits [6–4]—Reserved
- Bit [3]—Receive Overflow (rx overflow) → Write Frame Buffer pointer has passed the Read Frame Buffer pointer for the receive FIFO
- Bit [2]—UIC Error (uic error) → Indicates an error has occurred in the UIC (cleared on read)
- Bit [1]—Receive Complete (rcv complete) → The PMU has received a packet of data from the UIC (cleared on read)

- Bit [0]—Transmit Complete (xmit complete) → The PMU has completed transmitting data to the UIC (cleared on read and on Transmit Start [xmit start])

7.1.3.3 PMU Packet Transmit Size Register

The PMU Packet Transmit Size Register contains the number of bytes to be transmitted to the UIC from the PMU. This register is valid in both FIFO and Random Access modes.

Table 7-3. PMUChxy_PckXmitSize

31–12	11–0
Reserved	tx_size
–	RW

Note: Reset value is 0x00000000.

- Bits [31–12]—Reserved
- Bits [11–0]—Transmit Size (tx_size) → Size in bytes

7.1.3.4 PMU Packet Received Size Register

The PMU Packet Received Size Register contains the number of bytes the PMU received from the UIC. This value is reset for each new data packet. This register is used in both FIFO and Random Access modes (see Table 7-4).

Table 7-4. PDMAChxy_PckRcvSize

31–13	12–0	
Reserved	rx_size	0
–	RW	R

Note: Reset value is 0x00000000.

- Bits [31–13]—Reserved
- Bits [12–0]—Receive Size (rx_size) → Size in bytes (rounded up to next even value)

7.1.3.5 PMU Receive Buffer Start Register

The PMU Receive Frame Buffer Start Register identifies the starting address of the Receive Frame Buffer memory location for received data from the UIC. This register is used in both FIFO and Random Access modes. Address locations are relative to the PMU using long-word boundaries (lower bits default to 00) and are zero-based (memory starts at zero reference) (see Table 7-5).

Table 7-5. PMUChxy_RcvFBufStart

31–13	12–0		
Reserved	rcv_fb_start	0	0
–	RW	R	R

Note: Reset value is 0x00000000.

- Bits [31–13]—Reserved
- Bits [12–0]—Receive Frame Buffer Start (rcv_fb_start) → Start Address of the Receive Frame Buffer for this channel based on a long-word (32-bit) boundary.

7.1.3.6 PMU Receive Frame Buffer End Register

The PMU Receive Buffer End Register identifies the ending address of the Receive Frame Buffer memory location for received data from the UIC. This register is used in both FIFO and Random Access modes. Address locations are relative to the PMU using long-word boundaries (lower 2 bits default to 11) and are zero-based (memory starts at zero reference) (see Table 7-6).

Table 7-6. PMUChxy_RcvFBufEnd

31–13	12–0		
Reserved	rcv_fb_end	1	1
–	RW	R	R

Note: Reset value is 0x00000011.

- Bits [31–13]—Reserved
- Bits [12–0]—Receive Frame Buffer End (rcv_fb_end) → End Address of the Receive Frame Buffer for this channel based on a long-word (32-bit) boundary.

7.1.3.7 PMU Transmit Frame Buffer Start Register

The PMU Transmit Frame Buffer Start Register identifies the starting address of the Transmit Frame Buffer memory location transmit data to the UIC. This register is used in both FIFO and Random Access modes. Address locations are relative to the PMU using long-word boundaries (lower 2 bits default to 00) and are zero-based (memory starts at zero reference) (see Table 7-7).

Table 7-7. PMUChxy_XmitFBufStart

31–12	11–0		
Reserved	xmt_fb_start	0	0
–	RW	R	R

Note: Reset value is 0x00000000.

- Bits [31–12]—Reserved
- Bits [11–0]—Transmit Frame Buffer Start (xmt_fb_start) → Start Address of the Transmit Frame Buffer for this channel based on a long-word (32-bit) boundary.

7.1.3.8 PMU Transmit Frame Buffer End Register

The PMU Transmit Frame Buffer End Register identifies the ending address of the Transmit Frame Buffer memory location for transmit data to the UIC. This register is used in both FIFO and Random Access modes. Address locations are relative to the PMU using long-word boundaries (lower 2 bits default to 11) and are zero-based (memory starts at zero reference) (see Table 7-8).

Table 7-8. PMUChxy_XmitFBufEnd

31–12	11–0		
Reserved	xmt_fb_end	1	1
–	RW	R	R

Note: Reset value is 0x00000011.

- Bits [31–12]—Reserved
- Bits [11–0]—Transmit Frame Buffer End (xmt_fb_end) → End address of the Transmit Frame Buffer for this channel based on a long word (32 bit) boundary.

7.1.3.9 PMU Receive Frame Buffer Write Pointer Register

The PMU Receive Frame Buffer Write Pointer Register points to the current address of the PMU memory location being written for data transfers from the UIC to PMU. The address of this pointer ranges in value bounded by the Receive Frame Buffer Start (PMUChxy_RcvFBufStart) and Receive Frame Buffer End (PMUChxy_RcvFBufEnd) registers. This pointer is controlled by the PMU in both FIFO and Random Access modes. Data from the UIC will be written at this address in the PMU memory 16 bits (word) at a time. The write-pointer address space is relative to the PMU using word boundaries (lowest bit defaults to 0) and is zero-based (pointer starts at zero reference) (see Table 7-9).

Table 7-9. PMUChxy_RcvFBufWrPtr

31–13	12–0	
Reserved	rcv_fb_wr_ptr	0
–	RW	R

Note: Reset value is 0x00000000.

- Bits [31–13]—Reserved

- Bits [12–0]—Receive Frame Buffer Write Pointer (rcv_fb_wr_ptr) → Current write address of the Receive Frame Buffer for this channel based on a word-aligned (16-bit) boundary.

7.1.3.10 PMU Receive Frame Buffer Read Pointer Register

The PMU Receive Frame Buffer Read Pointer register points to the current address of the PMU memory location being read for data transfers from the PMU to CPU. The address of this pointer is controlled by the PMU and ranges in value bounded by the Receive Frame Buffer Start (PMUChxy_RcvFBufStart) and Receive Frame Buffer End (PMUChxy_RcvFBufEnd) registers.

In FIFO mode, data read from the PMU by the CPU is retrieved from this address and transferred 32 bits (long words) at a time to the CPU. The write pointer address space is relative to the PMU using long-word boundaries (lower bits default to 00) and is zero-based (pointer starts at zero reference). In FIFO mode, this register is managed automatically by the PMU. It must be managed manually by the CPU in Random Access mode (see Table 7-10).

Table 7-10. PMUChxy_RcvFBufRdPtr

31–13	12–0		
Reserved	rcv_fb_rd_ptr	0	0
–	RW	R	R

Note: Reset value is 0x00000000

- Bits [31–13]—Reserved
- Bits [12–0]—rcv_fb_rd_ptr → Address in memory of rcv frame buffer (FIFO) for reading data from buffer (FIFO) (CPU reading received data inserted by UIC), value in bytes, long-word aligned

7.1.3.11 PMU Transmit Frame Buffer Read Pointer Register

The PMU Transmit Frame Buffer Pointer Register points to the current address of the PMU memory location being written to for data transfers from the PMU to UIC. The address of this pointer ranges in value bounded by the Transmit Frame Buffer Start (PMUChxy_XmitFBufStart) and Transmit Frame Buffer End (PMUChxy_XmitFBufEnd) registers. This pointer is controlled by the PMU in both FIFO and Random Access modes. Data will be written from this address in the PMU memory 16 bits (one word) at a time to the UIC. The read-pointer address space is relative to the PMU using word boundaries (lower bit defaults to 0) and is zero-based (pointer starts at zero reference) (see Table 7-11).

Table 7-11. PMUChxy_XmitFBufRdPtr

31–12	11–0	
Reserved	xmt_fb_rd_ptr	0
–	RW	R

Note: Reset value is 0x00000000.

- Bits [31–12]—Reserved
- Bits [11–0]—Transmit Frame Buffer Read Pointer (xmt_fb_rd_ptr) → Current read address of the Transmit Frame Buffer for this channel based on a word-aligned (16-bit) boundary.

7.1.3.12 PMU Transmit Frame Buffer Write Pointer Register

The PMU Transmit Frame Buffer Write Pointer Register points to the current address of the PMU memory location being written for data transfers from the CPU to the PMU in FIFO mode. The address of this pointer ranges in value bounded by the Transmit Frame Buffer Start (PMUChxy_RcvFBufStart) and Transmit Frame Buffer End (PMUChxy_RcvFBufEnd) registers and is controlled by the PMU in FIFO mode only. Data from the CPU to the PMU will be written to this address using long-word (32 bits) boundaries at a time where the lower address bits defaults to 00. The read-pointer address space is relative to the PMU and is zero-based (pointer starts at zero reference). This register is managed by the PMU in FIFO mode and must be managed by the CPU in Random Access mode (see Table 7-12).

Table 7-12. PMUChxy_XmitFBufWrPtr

31–12	11–0		
Reserved	xmt_fb_wr_ptr	0	0
–	RW	R	R

Note: Reset value is 0x00000000.

- Bits [31–13]—Reserved
- Bits [11–0]—Transmit Frame Buffer Write Pointer (xmt_fb_wr_ptr) → Current write address of the Transmit Frame Buffer for this channel based on a long-word-aligned (32-bit) boundary.

7.1.3.13 PMU Transmit Data Register

The PMU Transmit Data Register is accessed by the CPU or DMA to write data to the transmit FIFO buffer in FIFO mode. Data written to this register will be placed in the address pointed to by the Transmit Frame Buffer Write Pointer (PMUChxy_XmitFBufWrPtr) and the Transmit Frame Buffer Write Pointer will be incremented by four for each data write (one increment for each word of 32-bit-long data). This register is not used in Random Access mode (see Table 7-13).

Table 7-13. PMUChxy_Xmit_Data

31–0
xmit data FIFO head
W

Note: Reset value is 0x00000000.

- Bits [31–0]—32-bit long word from PMU Tx FIFO.

> *Note: Each read by the UIC from the PMU will increment the Transmit Frame Buffer Read Pointer (PMU_Chxy_XmitFBufRdPtr) by two bytes, indicating 16 bits have been read.*

7.1.3.14 PMU Receive Data Register

The PMU Receive Data Register is accessed by the CPU to quickly read data from the receive FIFO buffer. Data read from this register will return the data at the address pointed to by the Receive Frame Buffer Read Pointer (PMUChxy_RcvBufRdPtr) and the Receive Frame Buffer Read Pointer is incremented by four per each data read (one increment for each word of 32-bit-long data). This register is not used in Random Access mode (see Table 7-14).

Table 7-14. PMUChxy_Rcv_Data

31–0
rcv data FIFO head
R

Note: Reset value is 0x00000000.

- Bits [31–0]—32-bit long word from PMU Rx FIFO.

> *Note: Each write by the UIC to the PMU will increment the Receive Frame Buffer Write Pointer (PMUChxy_RcvFBufWrPtr) by two bytes, indicating 16 bits have been written.*

7.1.4 PMU Usage

PMU setup is typically performed through the master context since certain register fields have restricted access to other contexts.

> *Note: Unused UICs or UICs used strictly for GPIO do not require memory allocation for PMU channels.*

To set up the PMU for operation with the UIC, the following steps are used:

1. Determine which protocol(s) to program and load into the UIC (see Section 7.2.5, UIC Programming, for more information on UIC firmware and firmware loading).

2. Divide the transmit Dual-Port appropriately among the protocol(s) using the Transmit Frame Buffer Start Register (PMUChxy_XmitFBufStart) and Transmit Frame Buffer End Register (PMUChxy_XmitBufEnd) for each channel.

3. Set the Transmit Frame Buffer Read Pointer (PMUChxy_XmitFBufRdPtr) and Transmit Frame Buffer Write Pointer (PMUChxy_XmitFBufWrPtr) to the value in the corresponding Transmit Frame Buffer Start Register (PMUChxy_XmitFBufStart) for each channel.

4. Divide the receive Dual-Port appropriately among the protocol(s) using the Receive Frame Buffer Start Register (PMUChxy_RcvFBufStart) and Receive Frame Buffer End Register (PMUChxy_RcvFBufEnd) for each channel.

5. Set the Receive Frame Buffer Read Pointer (PMUChxy_RcvFBufRdPtr) and Receive Frame Buffer Write Pointer (PMUChxy_RcvFBufWrPtr) to the value in the corresponding Receive Frame Buffer Start Register (PMUChxy_RcvFBufStart) for each channel.

6. Assign interrupts and priorities using the PMU primary-channel configuration registers to the desired context.

7. Enable PMU channels to be used. (As a rule, use the primary channel for a UIC running a single-channel protocol such as Ethernet, and the primary and secondary channel for a UIC running a two-channel protocol such as dual-UART.)

7.1.5 Buffer Access Modes

After PMU channels have been set up as described above, the PMU Channel control register (PMUChxy_Control) can be set to either the FIFO or Random Access mode. Details on how to use these modes are presented below.

For the transmit mode, described in Section 7.1.5.1, The FIFO Transmit Mode and Section 7.1.5.3, Random Access Transmit Mode; the following assumptions apply:

- The Transmit buffer allocated is larger than the message to be transmitted.
- The CPU is writing the data to be transmitted (i.e., the DMA is not being used).
- The PMU interrupt is not being used to recognize a Transmit Complete condition.

For the receive mode, described in Section 7.1.5.2, The FIFO Receive Mode, and Section 7.1.5.4, Random Access Receive Mode, the following assumptions apply:

- The Receive buffer allocated is larger than the message to be received.
- The CPU is reading the received data from the Receive FIFO (i.e., the DMA is not being used).
- The PMU interrupt is not being used to recognize a Receive Complete condition.

7.1.5.1 The FIFO Transmit Mode

To operate the PMU in FIFO transmit mode for a particular channel, use the following procedures:

1. Write the total number of data bytes for the entire message to be transmitted in the Transmit Size Register (PMUChxy_PckXmitSize).

2. Write the first 32 bits (long word) of message data into the Transmit Data Register (PMUChxy_Xmit_Data). This contains the first four bytes of the message.

3. Continue writing the remaining message data bytes to the FIFO Transmit Data Register (PMUChxy_Xmit_Data) until the message is loaded into the Transmit buffer.

4. Set the Transmit Start bit in the PMU Control Register (PMUChxy_Control).

5. After a message has been fully transferred to the UIC and transmission is complete, the Transmit complete bit (xmit complete) will be set in the PMU Channel Status register (PMUChxy_Status).

6. Repeat the above procedures to transmit the next message data.

7.1.5.2 The FIFO Receive Mode

To operate the PMU in FIFO receive mode for a particular channel, use the following procedures:

1. Ensure that the Receive Complete flag (rcv complete) in the PMU Channel Status register (PMUChxy_Status) has been set.

2. Read the message data from the Receive Data register (PMUChxy_Rcv_Data) in long-word increments (data in FIFO order).

3. The size of the message in number of bytes can be retrieved using the Receive Size Register (PMUChxy_PckRcvSize) or packet-size insertion.

4. Repeat the steps above for the next message data.

7.1.5.3 Random Access Transmit Mode

To operate the PMU in Random Access transmit mode for a particular channel, use the following procedures:

1. Write the entire message data into the transmit buffer at the desired address in PMU memory using long-word boundaries.

2. Write the beginning address of the data packet, relative to the start of the transmit buffer, to the Transmit Frame Buffer Read Pointer register (PMUChxy_XmitFBufRdPtr).

3. Write the ending address of the data packet (rounded up to the next long-word boundary), relative to the start of the transmit buffer, to the Transmit Frame Buffer Write Pointer register (PMUChxy_XmitFBufWrPtr).

4. Write the total number of data bytes for the entire message to be transmitted into the Transmit Size register (PMUChxy_PckXmitSize).

5. Set the transmit start flag bit in the PMU Channel Control register (PMUChxy_Control) to start the transmission of data to the UIC.

6. Wait for the data packet to have been transmitted as indicated by the Transmit Complete bit being set in the PMU Status registers (PMUChxy_Status).

7. Repeat the above procedures to transmit the next message data.

7.1.5.4 Random Access Receive Mode

To operate the PMU in Random Access receive mode for a particular channel, use the following procedures:

1. Set the Receive Buffer Read pointer (PMUChxy_RcvFBufRdPtr) and Receive Buffer Write pointer (PMUChxy_RcvFBufWrPtr) to the Receive Buffer Start value (PMUChxy_RcvFBufStart).

2. Wait for the Receive Complete flag (rcv complete) in the PMU Channel Status register (PMUChxy_Status) to be set.

3. Using memory reads, retrieve the message data starting at the Receive Buffer Read pointer (PMUChxy_RcvFBufRdPtr) of the PMU memory.

4. The size of the message in number of bytes can be retrieved using the Receive Size Register (PMUChxy_PckRcvSize).

5. Repeat the steps above for the next message data.

7.1.6 Packet Size Insertion

The PMU has the capability of inserting the size of a received packet at the beginning of the message in the Receive Frame Buffer via hardware. This feature is enabled by setting the Receive-Packet-Size-Enable bit (rcv packet size en) in the PMU Channel Control Register (PMUChxy_Control).

This allows a software program to scan a Receive Frame Buffer having multiple packets by reading the first word of a buffer. This is useful in quickly determining the offset to the next buffer and proceeding on to the next until a zero-size field is detected, indicating that all available data has been processed.

When Packet Size Insertion is enabled, the packet data is written beginning at the second long word of the Receive Frame Buffer. When packet reception is complete, the word immediately following the packet is written with zeros (0x00000000) and the number of bytes received is written to the first long word of the buffer.

Note: When setting up a Receive Frame Buffer to operate with Packet Size Insertion enabled, zero the first long word of the buffer before enabling the PMU and/or UIC. The receive-write pointer must be set to the second long word of the buffer.

When Packet Size Insertion is disabled, setting the rcv-packet-size-en bit in the PMU Channel Control Register (PMUChxy_Control) to zero, the first word of received data is written to the location indicated by the receive-write pointer.

7.2 Universal I/O Controller (UIC)

7.2.1 Overview

The fido1100 is equipped with four on-board UICs. Each UIC is a powerful communications controller designed with ample horsepower and firmware flexible enough to handle a wide range of communication protocols from 10/100 Ethernet to UART, along with General-Purpose I/O (GPIO) interfacing. The Peripheral Management Unit (PMU) and the UIC work together to minimize the communication, task-processing efforts that burden a standard CPU’s resources.

For communication protocols, the UIC executes firmware to provide the signal-handling specific to a protocol. Each UIC can transfer data to/from the PMU through two channels of FIFO, the primary and the secondary channel. The primary and secondary channels support 16- and 4-byte FIFOs, respectively. The UIC firmware is runtime executable and also loaded into the UIC via the primary-channel FIFO.

Each UIC can be used as direct GPIO to the 18 external pins of a UIC without the use of firmware. Each UIC controls 3 GPIO ports and 18 pins. Ports A and B contain eight pins each, and GPIO Port C contains two pins. Using all four UICs, up to 72 pins of GPIO are possible with the fido1100 chip.

Note: UIC GPIO works without the need for UIC firmware, however it is necessary to take the UIC out of PGM mode for GPIO to work. This is done by clearing Bit [0] of the UIC configRegA (see Table 7-24. ConfigRegA—Program Control).

This manual covers only basic interfacing between the fido1100 core processor and the UIC. A set of register definitions is provided here to set up the UIC for configuring GPIO and controlling the download of firmware to a UIC.

Note: All UIC registers must be read or written using 32-bit long-word operations.

7.2.2 UIC GPIO Registers

Each UIC GPIO port is controlled by three registers. Descriptions of these registers are presented in Section 7.2.2.1, GPIO Data Direction Registers; Section 7.2.2.2, GPIO Data Inversion Registers; and Section7.2.2.3, GPIO Data Registers.

7.2.2.1 GPIO Data Direction Registers

The GPIO Data Direction Register is an 8-bit register for Ports A and B and a 2-bit register for Port C that determines the direction of data flow on a pin-by-pin basis. All GPIO pins default to inputs at reset (see Table 7-15).

Table 7-15. GPIO_DIR_A

31–8	7	6	5	4	3	2	1	0
Reserved	GPIO7 Dir	GPIO6 Dir	GPIO5 Dir	GPIO4 Dir	GPIO3 Dir	GPIO2 Dir	GPIO1 Dir	GPIO0 Dir
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x000000FF.

- Bits [31–8]—Reserved
- Bit [7]—GPIO7 Dir
  - 0—Pin set as output
  - 1—Pin set as input
- Bit [6]—GPIO6 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [5]—GPIO5 Dir
  - 0—Pin set as output
  - 1—Pin set as input
- Bit [4]—GPIO4 Dir
  - 0—Pin set as output
  - 1—Pin set as input
- Bit [3]—GPIO3 Dir
  - 0—Pin set as output
  - 1—Pin set as input
- Bit [2]—GPIO2 Dir
  - 0—Pin set as output
  - 1—Pin set as input
- Bit [1]—GPIO1 Dir
  - 0—Pin set as output
  - 1—Pin set as input
- Bit [0]—GPIO0 Dir
  - 0—Pin set as output
  - 1—Pin set as input

Table 7-16. GPIO_DIR_B

31–8	7	6	5	4	3	2	1	0
Reserved	GPIO15 Dir	GPIO14 Dir	GPIO13 Dir	GPIO12 Dir	GPIO11 Dir	GPIO10 Dir	GPIO9 Dir	GPIO8 Dir
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x000000FF.

- Bits [31–8]—Reserved
- Bit [7]—GPIO15 Dir
  - 0—Pin set as output
  - 1—Pin set as input
- Bit [6]—GPIO14 Dir
  - 0—Pin set as output
  - 1—Pin set as input
- Bit [5]—GPIO13 Dir
  - 0—Pin set as output

  - 1—Pin set as input

- Bit [4]—GPIO12 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [3]—GPIO11 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [2]—GPIO10 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [1]—GPIO09 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [0]—GPIO08 Dir
  - 0—Pin set as output
  - 1—Pin set as input

The lower two bits of this register operate as GPIO Direction bits, with the remaining bits having other purposes (Port C has only two pins) (see Table 7-17).

Table 7-17. PIO_DIR_C

31–5	4–2			1	0
Reserved	UICOP			GPI017 Dir	GPI016 Dir
–	RW	R	R	RW	RW

Note: Reset value is 0x03.

- Bits [31–5]—Reserved

- Bits [4–2]—UICOP (UIC other purposes)

- Bit [1]—GPIO17 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [0]—GPIO16 Dir
  - 0—Pin set as output
  - 1—Pin set as input

Note: To avoid indeterminate behavior, a read/modify/write operation is recommended for Port C registers (i.e., read the register, modify the value using OR/AND, then write the modified value to the register).

7.2.2.2 GPIO Data Inversion Registers

The GPIO Data Inversion Register is an 8-bit register for Ports A and B and a 2-bit register for Port C that determines the inversion attribute of data on the GPIO pins. Each bit of each port can be set to inverting or non-inverting. Inversion applies to both input and output signals. All GPIO pins default to non-inverting at reset (see Table 7-18).

Table 7-18. GPIO_INV_A

31–8	7	6	5	4	3	2	1	0
Reserved	GPIO7 Invert	GPIO6 Invert	GPIO5 Invert	GPIO4 Invert	GPIO3 Invert	GPIO2 Invert	GPIO1 Invert	GPIO0 Invert
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bit [7]—GPIO7 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted
- Bit [6]—GPIO6 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted
- Bit [5]—GPIO5 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted
- Bit [4]—GPIO4 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted
- Bit [3]—GPIO3 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted
- Bit [2]—GPIO2 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [1]—GPIO1 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [0]—GPIO0 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

Table 7-19. GPIO_INV_B

31—8	7	6	5	4	3	2	1	0
Reserved	GPIO15 Invert	GPIO14 Invert	GPIO13 Invert	GPIO12 Invert	GPIO11 Invert	GPIO10 Invert	GPIO9 Invert	GPIO8 Invert
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved

- Bit [7]—GPIO15 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [6]—GPIO14 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [5]—GPIO13 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [4]—GPIO12 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [3]—GPIO11 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [2]—GPIO10 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [1]—GPIO9 Invert
  - 0—Pin set as not inverted

  - 1—Pin set as inverted

- Bit [0]—GPIO8 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

Note: The lower two bits of this register operate as GPIO Inversion bits. The remaining bits have other purposes (Port C has only two GPIO pins).

Table 7-20. GPIO_INV_C

31–8	7–2	1	0
Reserved	UICOP	GPIO17 Invert	GPIO16 Invert
–	R	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved

- Bits [7–2]—UICOP (UIC other purposes)

- Bit [1]—GPIO17 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [0]—GPIO16 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

7.2.2.3 GPIO Data Registers

The GPIO Data Register is an 8-bit register for Ports A and B and a 2-bit register for Port C that reflects the current state of the I/O. When the corresponding bit is set to input in the Data Direction Register, the data register reflects the state of the external I/O; a write has no affect. When the corresponding bit is set to output, the data register can be written and its value will be driven on the corresponding output pin (subject to the setting in the inversion register). All GPIO Data Registers default to zero at reset (see Tables 7-21, 7-22, and 7-23).

Table 7-21. GPIO_Data_A

31–8	7	6	5	4	3	2	1	0
Reserved	GPIO7 Data	GPIO6 Data	GPIO5 Data	GPIO4 Data	GPIO3 Data	GPIO2 Data	GPIO1 Data	GPIO0 Data
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bit [7]—GPIO7 Pin Data
- Bit [6]—GPIO6 Pin Data
- Bit [5]—GPIO5 Pin Data
- Bit [4]—GPIO4 Pin Data
- Bit [3]—GPIO3 Pin Data
- Bit [2]—GPIO2 Pin Data
- Bit [1]—GPIO1 Pin Data
- Bit [0]—GPIO0 Pin Data

Table 7-22. GPIO_Data_B

31–8	7	6	5	4	3	2	1	0
Reserved	GPIO15 Data	GPIO14 Data	GPIO13 Data	GPIO12 Data	GPIO11 Data	GPIO10 Data	GPIO9 Data	GPIO8 Data
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bit [31–8]—Reserved
- Bit [7]—GPIO15 Pin Data
- Bit [6]—GPIO14 Pin Data
- Bit [5]—GPIO13 Pin Data
- Bit [4]—GPIO12 Pin Data
- Bit [3]—GPIO11 Pin Data
- Bit [2]—GPIO10 Pin Data
- Bit [1]—GPIO9 Pin Data

- Bit [0]—GPIO8 Pin Data

Note: The lower two bits of this register operate as GPIO Data bits. The remaining bits have other purposes (Port C has only two GPIO pins).

Table 7-23. GPIO_Data_C

31–8	7–2	1	0
Reserved	UICOP	GPIO17 Data	GPIO16 Data
–	R	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bits [7–2]—UICOP (UIC other purposes)
- Bit [1]—GPIO17 Pin Data
- Bit [0]—GPIO16 Pin Data

Note: If using the GPIO pins with pull-up resistors (i.e., open collector or open drain interfaces), set the following configuration using the registers associated with the desired GPIO pin. Set the inversion to OFF (clear bit in GPIO_INV_x), the data to zero (clear bit in GPIO_DATA_x), and the direction to output (set bit in GPIO_DIR_x).

7.2.3 UIC Configuration Registers

The UIC Configuration Registers (UIC Configuration Register A through UIC Configuration Register L) set options that configure and control how the UIC operates. Nine configuration registers are loaded along with the protocol firmware at programming time and generally left unchanged during firmware execution. Configuration registers are not accessible by the UIC processor, and are typically not modified while UIC firmware is executing.

To change any of the configuration register values, the first value written to the target register must be 0xAA to unlock the register. The next value written will be the new data value for the register. After the second write to the register or if a write to a different address occurs after the unlock write, the UIC register will be automatically locked. This feature prevents the UIC configuration registers from accidentally being written to during runtime in case unplanned memory accesses occur.

This manual shows only those configuration registers necessary for programming and setting up the UIC. These are Configuration Registers A, D, K, and L.

- Configuration Register A controls the programming of the UIC and sets the runtime mode.
- Configuration Register D controls the sleep mode of the UIC.
- Configuration Registers K and L are read-only and contain the firmware ID and checksum bytes, respectively.

7.2.3.1 UIC Configuration Register A

This register is used to control the programming of the UIC and enable runtime operation. Only bits zero through two are used for this operation; the remaining bits are set by UIC firmware for other UIC functions (see Table 7-24).

Table 7-24. ConfigRegA—Program Control

31–8	7	6	5	4–3	2	1	0
Reserved	TXE	TXEN	TXNB	THC	SSR	PGC	PGM
–	RW	RW	RW	RW	R	R	RW

Note: Reset value is 0x00000001.

- Bits [31–8]—Reserved
- Bit [7]—Transmit/Receive Register (TxReg/RxReg) Endian formats (TXE)
  - 0—TxReg/RxReg is set as little-Endian and does not reverse bit order on a byte (reset value)
  - 1—TxReg/RxReg are big-Endian and reverses bit order on a byte
- Bit [6]—Transmit/Receive Register (TxReg/RxReg) Enables (TXEN)
  - 0—UIC controls the TxReg/RxReg enable internally (reset value)
  - 1—TxReg enable is controlled by an external pin (set in UIC ConfigRegC)
- Bit [5]—Transmit/Receive Register (TxReg/RxReg) Nibble Mode (TXNB)
  - 0—TxReg/RxReg transmits and receives one bit at a time (reset value)
  - 1—TxReg/RxReg transmits and receives one nibble at a time
- Bits [4–3]—UIC Thread Count (THC)
  - 11—Four threads, program memory is segmented into four banks
  - 10—Reserved
  - 01—Two threads with program memory segmented into two banks using threads one and four only
  - 00—One thread with program memory non segmented using thread one only
- Bit [2]—UIC is executing (SSR) → Set by UIC Configuration Manager hardware
  - 0—UIC is idle (reset value)

- 1—UIC is executing programs after being programmed and CPU has cleared Program Mode (bit zero)

- Bit [1]—Programming Complete (PGC) → Set by UIC Configuration Manager hardware
  - 0—UIC program has not been loaded (reset value)
  - 1—UIC program and configuration have been loaded

- Bit [0]—Program Mode (PGM)
  - 0—Normal Operation
  - 1—Programming Mode (Reset value)

7.2.3.2 UIC Configuration Register D

This register is used to control the sleep state of the UIC. Only bits zero and one are used for this operation. The remaining bits are used for other UIC functions (see Table 7-25).

Table 7-25. ConfigRegD—Master Control Register

31–8	7	6	5	4–3	2	1	0
Reserved	HDLC	ALT	Edge	CRC Size	NRZI	RMAS	PYJO
–	RW	RW	RW	RW	RW	R	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved

- Bit [7]—High Level Data Link Mode (HDLC) (when using UIC HDLC firmware)
  - 0—Bit Stuffing for Transmit (TxReg) and Receive (RxReg) registers not used
  - 1—Transmit (TxReg) and Receive (RxReg) registers use bit stuffing, recognizing 0x01111110 as a frame delimiter, seven or more "1"s as an abort sequence.

- Bit [6]—Alternate Transmit/Receive Register (TxReg/RxReg) pin mapping (ALT). Bit is exclusive with UIC Configuration Register C Bit [0]
  - 0—UIC Pins 0 and 1 will be tied to GPIO_DATA_A register Bit [0] and Bit [1], respectively
  - 1—UIC Pins 0 and 1 will be tied to Transmit (TxReg) and Receive (RxReg) registers, respectively (primarily used for HDLC mapping)

- Bit [5]—Clock Edge Setting (Edge)
  - 0—Transmit (TxReg) and Receive (RxReg) registers use rising edge of clock
  - 1—Transmit (TxReg) and Receive (RxReg) registers use falling edge of clock

- Bits [4–3]—CRC Size Setting (CRC Size)
  - 00—CRC uses 8 bits
  - 01—CRC uses 16 bits

  - 10—CRC uses 32 bits
  - 11—Reserved

- Bit [2]—Non Return To Zero Inverted mode (NRZI)
  - 0—Transmit (TxReg) and Receive (RxReg) registers use straight encoding
  - 1—Transmit (TxReg) and Receive (RxReg) registers use a zero representation as a change in wire state, and a data 1 as no change

- Bit [1]—Read Me A Story (RMAS)
  - 0—UIC still preparing to enter sleep mode after PYJO = 1
  - 1—UIC has entered sleep mode

- Bit [0]—Put Your Jammies On (PYJO) (Initiate UIC sleep mode)
  - 0—UIC is in RUN mode
  - 1—Command UIC to enter sleep mode

7.2.3.3 UIC Configuration Register K

This register displays a unique firmware ID allowing the tracking of protocol firmware loaded into the UIC (see Table 7-26). .

Table 7-26. ConfigRegK—UIC Firmware ID

31–8	7–4	3–0
Reserved	ID	Version
–	R	R

- Bits [31–8]—Reserved

- Bits [7–4]—ID (see Table 7-27)

- Bits [3–0]—Versions can range from 0x0 to 0xF

Table 7-27. Firmware ID Codes

Protocol	ID[7–4]
Ethernet10	0xF
Ethernet100	0xE
UART	0xD
HDLC	0xC
CAN	0xB
SMBUS	0xA
GPIO	0x9
Unused	0x1–0x8
Reserved	0x0

7.2.3.4 UIC Configuration Register L

This register holds the checksum computed by the UIC during programming of the UIC. The value of this register can be used for confirmation of program download when compared to the checksum value provided during UIC firmware assembly time (see Table 7-28).

Table 7-28. ConfigRegL—UIC Programming Checksum

31–8	7–0
Reserved	Checksum
–	R

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bits [7–0]—Checksum value

7.2.4 UIC Interrupt Registers

The UIC uses a pair of registers for generating interrupts to the fido1100 CPU, the Interrupt Register and the Interrupt Mask Register. Any time that a bit is set in the Interrupt Register, it is ANDed with the corresponding bit in the Interrupt Mask Register, and the result is forwarded through the PMU logic and delivered to the CPU as an interrupt. This allows various firmware protocols to use software defined interrupts for PMU to CPU communications. A read of the Interrupt Register by the CPU clears the latched interrupt status.

As an alternative to UIC-firmware-generated interrupts, the Port A GPIO pins can be mapped directly to the Interrupt Register to provide eight additional external hardware interrupt sources. Any flag set in either mode that is also set in the mask register will generate the UIC interrupt.

7.2.4.1 Interrupt Status Register

This eight-bit register can be written by the UIC firmware or configured to be mapped directly to Port A GPIO pins. Interrupts are latched until read by the CPU (see Table 7-29).

Table 7-29. INTERRUPT_STATUS_REG

31–8	7	6	5	4	3	2	1	0
Reserved	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bits [7–0]—UIC definable interrupt flags.

7.2.4.2 Interrupt Mask Register

This 8-bit register is ANDed with the Interrupt Status Register to determine which bits will generate an interrupt. To mask an interrupt, the software (either UIC or fido CPU) should write a zero to the corresponding bit in this register (see Table 7-30).

Table 7-30. INTERRUPT_MASK_REG

31–8	7–0							
Reserved	INT MASK7	INT MASK6	INT MASK5	INT MASK4	INT MASK3	INT MASK2	INT MASK1	INT MASK0
–	RW							

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bits [7–0]—Masks bits for each UIC definable interrupt flag.

7.2.5 UIC Programming

UIC programming is accomplished using the PMU Transmit FIFO Frame Buffer when the UIC Program Mode bit is set in the UIC Configuration A (ConfigRegA) register. The entire UIC must be programmed in order and cannot be partially programmed. The UIC will exit programming mode only after the Program Mode (PGM) bit is cleared in the UIC Configuration A (ConfigRegA) register. The UIC Configuration Manager will flush all PMU to UIC FIFOs, CRC register, and the Transmit/Receive (TxReg/RxReg) upon completion of programming.

The following procedure is used to program a UIC.

1 Set up the PMU for transmit with a PMU Transmit Frame Buffer in either FIFO or Random Access mode. See Section 7.1.4, PMU Usage, for setup procedures.

2 Load the PMU Transmit Frame Buffer for the desired UIC with the firmware to download and set the transmit size register to the firmware size.

3 Set the Transmit Start (xmit start) bit in the appropriate PMU Control (PMUChxy_Control) register to begin transmission.

4 Wait for the Transmit Complete (xmit complete) interrupt flag in the PMU Status (PMUChxy_Status) register.

5 Read both the UIC Interrupt Status (INTERRUPT_STATUS_REG) and the PMU Status (PMUChxy_Status) registers to clear the interrupts.

6 Clear the Program Mode (PGM) bit in the UIC Configuration A register to allow the start of program execution.

7 Read the PMU Status (PMUChxy_Status) register a second time to ensure all interrupts have been properly cleared.

8 Repeat for the remaining UICs, if necessary.

7.2.6 UIC Sleep Mode

The fido1100 CPU can request that the UIC be placed in sleep mode (only with firmware loaded). In order for sleep mode to occur, the UIC firmware must have an instruction execution point within the firmware that allows a pause in execution. To put the UIC in sleep mode, the following procedures are used:

1 Set the Put Your Jammies On (PYJO) bit in the Configuration D (ConfigRegD) register in the desired UIC to request sleep mode.

2 Wait for the UIC to set the Read Me A Story (RMAS) bit in the Configuration D (ConfigRegD) register indicating that execution of firmware has paused and that the UIC is ready for sleep mode.

3 Set the clock mask bit for the desired UIC in the Clock Master Register to turn off the clock to the UIC, placing it in low-power mode.

To wake up the UIC out of low-power mode, the following steps are used:

1 Clear the clock mask bit for the desired UIC in the Clock Master Register to enable the clock to the UIC.

2 Clear the Put Your Jammies On (PYJO) bit in the Configuration D (ConfigRegD) register to resume program execution.

7.3 DMA (Direct Memory Access Controllers)

7.3.1 Overview

The Direct Memory Access (DMA) module provides direct data transfers between memory and the various peripherals. Data transfers are made without interaction from the CPU. The DMA can transfer data to and from any of the memory interfaces of the fido1100, which includes User RAM, RREM, the external bus, and the PMU buffers.

The DMA supports two channels with a set of control, count, source, and destination registers dedicated to each channel. Each DMA channel can transfer data in either byte (8-bit), word (16-bit), or long-word (32-bit) formats. Each DMA channel can be automatically triggered by the general-purpose peripheral timers (Timer Counter Units), the completion of a data transfer to/from PMU buffers, an external DMA request, or on command by the CPU. Each DMA channel has a transfer priority that is compared to the current executing context's priority. If the priority setting for the DMA channel is greater than the context priority, the DMA will alternate burst transfers of data with CPU fetches of instructions or operands. If the DMA channel priority is lower than the currently executing context's priority, the DMA will idle during context execution times. This prevents a large DMA transfer from starving the CPU of instructions.

For DMA use with a Timer Counter Unit (TCU), the DMA is triggered upon the activation of the TCU's Termination Event Interrupt. This allows for DMA data transfers at designated timing intervals. Refer to the TCU section of this document for further information.

For external DMA requests, each DMA channel has a dedicated request and acknowledge signal provided through shared (muxed) pins. When enabled, an external peripheral can assert the DMA request pin indicating a DMA transfer is needed. The DMA acknowledges the request by asserting the DMA ACK pin to the requesting external peripheral. The DMA ACK pin remains asserted until the completion of the DMA transfer cycle. If the DMA request pin is still asserted by the external peripheral after completion of a DMA cycle, another DMA cycle is issued and the ACK pin remains asserted. Upon completion of the DMA transfer cycle, the ACK pin is again de-asserted.

DMA channels can indicate DMA transfer complete using interrupts or through polling by the CPU of the DMA Channel Control register. For DMA Channel use with interrupts, priorities are assignable from zero to seven and to any context. For polling mode, a DMA transfer complete is indicated when the End Of Process (EOP) flag is asserted in the DMA Channel Control (DMAChx_Control) register.

A primary use of a DMA channel is to transfer data between the PMU (Dual-Port RAM) and the CPU memory. This allows continuous data transfers to and from the UIC via the PMU for various protocol applications. A special DMA bypass mode can be used for very fast transfers between PMU buffer FIFO heads and Internal User RAM. This can be very helpful for communication protocols requiring large amounts of data to be transferred.

In DMA bypass mode, a data transfer between PMU FIFO heads and Internal User RAM can happen every clock cycle. DMA data transfers will use multiple clock cycles if not using the special bypass mode.

7.3.2 DMA Registers

Each DMA channel has a set of the following registers where in DMA_Chx the "x" represents the channel number. For example, DMA_Ch0 would represent DMA channel 0. Listed below are the PMU registers. They are detailed in sections that follow:

- DMA Control Register
- DMA Channel 0/1 Source Address Registers
- DMA Channel 0/1 Destination Address Registers
- DMA Channel 0/1 Count Registers

7.3.2.1 DMA Control Register

This DMA Channel Control Register provides the control for the DMA channel. Note that several fields within the DMA Channel Control register are accessible only through the Master Context, Context_0 (see Table 7-31).

Table 7-31. DMAChx_Control

31	30	29	28	27	26–25	24	23–22	21	20–17	16	15–13	12–11	10–8	7	6–4	3–2	1–0
EN	EOP	MPU Fault	Bus Fault	SDRAM Dest	Dest Addr Control	SDRAM Src	Source Addr Control	PMU Dir	Transfer Control	Ack Enable	Xfer Priority	Reserved	Context	Int En	Int Priority	Burst Size	Xfer Size
RW	R	R	RW	RW	RW	RW	RW	RW	RW	RW	RW	–	RW	RW	R/W	RW	RW

Note: Reset value is 0.00000000.

- Bit [31]—DMA Channel Enable (EN)
  - 0—DMA channel is disabled
  - 1—DMA channel is enabled

- Bit [30]—End of process (EOP) → Set when DMA has completed transferring data and indicates an interrupt has been generated by the DMA (cleared on read)

- Bit [29]—Memory Protection Fault (MPU) Fault → Set if an MPU Fault occurs during data transfer (cleared on read)

- Bit [28]—Bus Fault → Set if a Bus Fault occurs during transfer (cleared on read)

- Bit [27]—SDRAM Destination (SDRAM Dest)
  - 0—DMA destination is not in SDRAM
  - 1—DMA destination is in SDRAM

- Bits [26–25]—DMA Destination Address Control (Dest Addr Control)
  - 00—Increment destination address on transfer
  - 01—Decrement destination address on transfer
  - 10—Hold destination address on transfer
  - 11—PMU Transmit FIFO write → Enables bypass mode transfers to the PMU Transmit FIFO Head and holds destination address on transfer

Note 1: For PMU Transmit FIFO writes in bypass mode, the DMA Destination Address Register (DMAChx_Destination) holds a value from zero to seven specifying which PMU Transmit Buffer FIFO will be written. Only the lower three bits of the Destination Address register will be evaluated when PMU Transmit FIFO write mode is selected.

Note 2: The source address for PMU Transmit FIFO writes in bypass mode must be in one of the following fido1100 internal memory spaces:
- *Internal URAM*
- *PMU Receive FIFO Head*
- *PMU Receive Buffer in Random Access mode*

- Bit [24]—SDRAM Source (SDRAM Src)
  - 0—DMA destination is not in SDRAM
  - 1—DMA destination is in SDRAM

- Bits [23–22]—Source Address Control (Source Addr Control)
  - 00—Increment source address on transfer
  - 01—Decrement source address on transfer
  - 10—Hold source address on transfer
  - 11—PMU Receive FIFO Read → Enables bypass mode transfers to the PMU Transmit FIFO Head and holds source address on transfer

Note 1: For PMU Receive FIFO reads in bypass mode, the DMA Source Address Register (DMAChx_Source) holds a value from zero to seven specifying which PMU Receive buffer FIFO will be read. Only the lower 3 bits of the Source Address register will be evaluated when PMU Receive FIFO Read mode is selected.

Note 2: The destination address for PMU Receive FIFO reads in bypass mode must be in one of the following fido1100 internal memory spaces:
- *Internal URAM*
- *PMU Transmit FIFO Head*
- *PMU Transmit Buffer in Random Access mode*

- Bit [21]—PMU Direction (PMU Dir) → Determines which PMU status bit of the selected PMU channel (set with the Transfer Control bits) to monitor for PMU FIFO data transfers to and from the DMA. This only applies if the Transfer Control bits have been set to binary 1000 through 1111.
  - 0—Read from PMU, transfer occurs when the PMU Receive FIFO Empty (Recv FIFO Empty) flag is zero in the PMU Channel Status (PMUChxy_Status) register
  - 1—Write to PMU, transfer occurs when the PMU Transmit FIFO Full (Xmit FIFO Full) flag is zero in the PMU Channel Status (PMUChxy_Status) register

- Bits [20–17]—DMA Transfer Control (Transfer Control)
  - 0000—Transfer when ready (Trigger → data transfer runs to completion)
  - 0001—Timer Counter 0 (Trigger → data transfer runs to completion)
  - 0010—Timer Counter 1 (Trigger → data transfer runs to completion)
  - 0011—Reserved
  - 0100—Reserved
  - 0101—Reserved
  - 0110—Reserved
  - 0111—External DMA Request (Throttle → data transfer only occurs when active)
  - 1000—PMU Channel 0A ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)
  - 1001—PMU Channel 0B ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)
  - 1010—PMU Channel 1A ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)
  - 1011—PMU Channel 1B ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)
  - 1100—PMU Channel 2A ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)
  - 1101—PMU Channel 2B ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)

- 1110—PMU Channel 3A ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)
- 1111—PMU Channel 3B ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)

Note: The use of the term "Throttle" refers to the speed at which data is transferred by the DMA. Depending upon the amount of empty space present in memory (e.g., PMU FIFO buffer), the DMA will transfer at a rate until the memory is full at which point data transfers will stop (throttle down). As soon as memory becomes available, the DMA will again start to transfer data (throttle up).

- Bit [16]—Acknowledge Enable (ACK Enable) → When set, enables DMA Acknowledge/Interrupt (ACK/INT) muxed pin as DMA ACK output.

- Bits [15–13]—DMA Transfer Priority (Xfer Priority) → Sets the priority of DMA transfers with respect to CPU context priorities. Priority levels range between zero and seven (000–111). Only writable through the master context (Context_0)

- Bits [12–11]—Reserved

- Bits [10–8]—DMA Context Assignment (Context) → Sets the context the DMA interrupt is associated with. Only writable through the master context (Context_0)
  - 000..100—Context zero through four
  - 101..111—Reserved

- Bit [7]—Interrupt Enable (Int En) → Enables the DMA interrupt when the DMA Count (DMAChx_Count) register reaches zero
  - 0—DMA interrupt is not enabled
  - 1—DMA interrupt is enabled

- Bits [6–4]—Interrupt Priority (Int Priority) → Interrupt priority level for the assigned context. Priority levels range between zero and seven (000–111)

- Bits [3–2]—DMA transfer data burst size (Burst Size) → Number of transfers DMA should do before allowing a CPU access to occur
  - 00—Single transfer
  - 01—4 transfer burst
  - 10—32 transfer burst
  - 11—128 transfer burst

- Bits [1–0]—DMA Transfer Size (Xfer Size) → Set the data size of each data transfer
  - 00—Byte

- 01—Word (16-bits)
- 10—Long word (32-bits)
- 11—Reserved

7.3.2.2 DMA Channel 0/1 Source Address Registers

The DMA Source Address registers reflects the current value of the source address. If a Bus Fault occurs during the DMA transfer, this register will retain the value of the address where the bus fault occurred (see Table 7-32).

Table 7-32. DMAChx_Source

31–0
Source address
RW

Note: Reset value is 0x00000000.

- Bits [31–0]—Source address for DMA data or, in bypass mode, a number 0–7 indicating which PMU FIFO to read from.

7.3.2.3 DMA Channel 0/1 Destination Address Registers

The DMA Destination Address register reflects the current value of the destination address of the next data transfer for that channel. If a Bus Fault occurs during the DMA transfer, this register will retain the value of the address where the bus fault occurred (see Table 7-33).

Table 7-33. DMAChx_Destination

31–0
Destination address
RW

Note: Reset value is 0x00000000.

- Bits [31–0]—Destination address for DMA data or, in bypass mode, a number 0–7 indicating which PMU FIFO to read from.

7.3.2.4 DMA Channel 0/1 Count Registers

The DMA count register controls the number transfers for a single DMA channel. This register reflects the "live" value of the count and indicates how many data transfers remain. If a Bus Fault occurs during the DMA Transfer, this register will hold the value of the count when the bus fault occurred (see Table 7-34).

Table 7-34. DMAChx_Count

31–0
Count
RW

Note: Reset value is 0x00000000.

- Bits [31–0]—DMA transfer count.

Note: This is not a byte count. This register holds the number of transfers per Bits [0–1] of the DMAChx_Control register.

7.3.3 DMA Setup

To set up the DMA for memory data transfer operations, the following procedures are used for each channel. Changes in DMA setup configurations are allowed by the various contexts during program execution for necessary memory data transfer changes with the exception of context assignment and DMA priority.

1. From the Master Context (Context_0), assign a context and a priority to the DMA channel using the bits in the DMA Control (DMAChx_Control) register.

2. Set up the Transfer Control bits in the DMA Control (DMAChx_Control) register for the desired peripheral.

3. Set up the Destination and Source Address modes in the DMA Control (DMAChx_Control) register for address pointer control.

4. Set the data transfer increment size using the Transfer Size bits in the DMA Control (DMAChx_Control) register.

5. If using external DMA requests, enable the External DMA request pin by setting the DMA transfer control bits and the Acknowledge Enable bit in the DMA Control (DMAChx_Control) register.

6. If using DMA interrupts, assign the interrupt to a context and set the priority using the bits in the DMA Control (DMAChx_Control) register.

7. Write the DMA data transfer count into the DMA Channel Count (DMAChx_Count) register.

8. Write the destination-memory start address and source-memory start address into the DMA Channel Destination (DMAChx_Destination) and DMA Channel Source (DMAChx_Destination) registers, respectively.

9. Enable the DMA Channel by setting the Enable bit in the DMA Control (DMAChx_Control) register.

10. Repeat if necessary for the other DMA Channel.

7.4 MAC Filter

7.4.1 MAC Filter Overview

The UIC and PMU operate together to perform Media Access Control (MAC) filtering of received Ethernet packets. For each incoming Ethernet packet, a comparison is made of the packet's address with an address in the MAC filter. If the CRC of the Ethernet packet is valid and the incoming MAC address matches an entry in the MAC filter, the data packet becomes available for processing. If there is a mismatch in either the CRC or MAC address, then the incoming packet is rejected and the resources are quickly reset and cleared to receive the next incoming packet.

The MAC filter is an integrated, user-programmable filter capable of executing multiple address comparisons without CPU intervention. MAC filtering occurs prior to the transfer of a data packet from the UIC to the PMU Receive Frame Buffer. The MAC filter is capable of operating using the following table modes:

- As a single lookup table (256 entries)
- As dual lookup tables (2 × 128 entries)
- As a single Hash table (256 entries using the first 128 locations in partition 0)
- As both a 256-entry Hash Table and a lookup table (128 entries)

The MAC filter allows a specific way of eliminating the servicing of unwanted Ethernet packet data in multicast modes. Rather than using a Hash table filter with limited ability, packet MAC destination addresses are compared against a list of either 256 or 128 allowed addresses depending upon the configuration of the MAC filter. If the current address of a packet is not within the table, the MAC Filter will indicate to the PMU to abort the transfer of data from the appropriate UIC. The MAC Filter address tables are shared among all four UICs.

For MAC filtering, the internal hardware either uses a pass or fail status from the filtering result to determine when to transfer data from the UIC to the PMU. If the filtering result returns a pass indication, then the data packet is transferred. If the filtering result returns a fail indication, then the data packet is abandoned with no indication provided to the CPU of the packet's existence. MAC filtering is available with the following filtering modes for use with various Ethernet protocols.

7.4.1.1 Hash Mode

In this mode, the PMU computes the 8-bit CRC of the destination MAC address and uses this as an index to lookup a byte within a 256-entry hash table. If the corresponding byte has a value of all “1”s (0xFF), the filter issues a pass, else it returns fail.

7.4.1.2 Destination Address Mode

In this mode, a 48-bit value (e.g., the first 48 bits designated as the destination address in an Ethernet message) is compared against the table. If a match is found, then the filter issues a pass—if not, it issues a fail. Broadcast messages are not accepted unless their destination address is an exact match to the table.

7.4.1.3 Destination Address with Broadcast and Multicast Mode

In this mode, a 48-bit value is compared against the table (e.g., the first 48 bits designated as the destination address in an Ethernet message). If a match is found, the filter issues a pass. If the message is a broadcast or multicast message (the LSB of the destination address is “1”), the filter will also issue a pass. Otherwise, the filtering result is a fail.

7.4.1.4 Destination Address with Broadcast Override Mode

In this mode, filtering operates the same as Destination Address mode unless the least significant bit of the 48-bit value is set. If set, the next 48-bit value is used for the filter operation (i.e., the second set of 48 bits designated as the source address in an Ethernet message).

7.4.1.5 Source Address Mode

In this mode, a 48-bit value is compared against the table (i.e., the second set of 48 bits designated as the source address in an Ethernet message). If a match is found, then the filter issues a pass—if not, it issues a fail.

7.4.1.6 MAC Filter Partitioning

The MAC filter table uses a RAM-based FIFO-style memory buffer. The table can either be used as a single partition with 256 entries or be divided into two partitions, each having 128 entries for MAC filtering. Partition configuration is controlled through the MAC Filter Mode Configuration Register. If in single MAC filter table partition mode (Partition 0), then the configuration bits for partition one have no effect.

Note: To achieve real-time filtering of MAC addresses and to deal with 48-bit MAC addresses, the following restrictions to the data in the MAC Filter RAM apply:

- MAC addresses in the table must be sorted in ascending order. This allows the MAC filtering hardware to use a binary search approach requiring a maximum of eight comparisons. It is the responsibility of the CPU to sort the MAC Address data properly before loading it into the MAC filter.
- MAC Filter partitions are written and read using the MAC Filter Data Write and MAC Filter Data Read registers. Internal hardware pointers automatically keep track of the filter data for reading and writing. Data words are read and written in ascending order within the filter 16 bits at a time. The internal pointers can be reset to zero by setting the Reset pointer bits in the MAC Filter Mode Configuration register.

7.4.2 MAC Filter Registers

7.4.2.1 MAC Filter Mode Configuration Register

The MAC Filter Mode Configuration register is used to set up and configure the MAC table and filtering functions. Filtering modes can be set for each UIC and for each partition of the MAC Filter RAM table. This register also allows the resetting of the internal read and write pointers for the MAC Filter RAM table (see Table 7-35).

Table 7-35. MAC Filter Mode Configuration Register

31–25	24	23	22–15	14–13	12–11	10–9	8–7	6–4	3–1	0
Reserved	Reset read pointer in MAC filter RAM	Reset write pointer in MAC filter RAM	Reserved	UIC_3_ mode	UIC_2_ mode	UIC_1_ mode	UIC_0_ mode	P1 Mode	P0 Mode	Partition
–	RW	RW	–	RW	RW	RW	RW	RW	RW	RW

Note: Reset address is 0x00000000.

- Bits [31–25]—Reserved
- Bit [24]—Reset read pointer in MAC filter RAM → When set, resets the read pointer to the beginning of the table (cleared automatically).
- Bit [23]—Reset write pointer in MAC filter RAM → When set, resets the write pointer to the beginning of the table (cleared automatically).
- Bits [22–15]—Reserved
- Bits [14–13]—UIC_3_mode → Sets the MAC filtering mode for UIC 3
  - 00—No Filtering
  - 01—Reserved
  - 10—MAC Filtering uses HASH Table Partition 0
  - 11—MAC Filtering uses HASH Table Partition 1

- Bits [12–11]—UIC_2_mode → Sets the MAC filtering mode for UIC 2
  - 00—No Filtering
  - 01—Reserved
  - 10—MAC Filtering uses HASH Table Partition 0
  - 11—MAC Filtering uses HASH Table Partition 1

- Bits [10–9]—UIC_1_mode → Sets the MAC filtering mode for UIC 1
  - 00—No Filtering
  - 01—Reserved
  - 10—MAC Filtering uses HASH Table Partition 0
  - 11—MAC Filtering uses HASH Table Partition 1

- Bits [8–7]—UIC_0_mode → Sets the MAC filtering mode for UIC 0
  - 00—No Filtering
  - 01—Reserved
  - 10—MAC Filtering uses Table Partition 0
  - 11—MAC Filtering uses Table Partition 1

- Bits [6–4]—Partition 1 mode → Sets the mode for how RAM Partition Table 1 is used
  - 000—Destination address filtering mode
  - 001—Destination address filtering with broadcast override (broadcast messages are not filtered) mode
  - 010—Accept all broadcast messages mode
  - 011—Source address filtering mode
  - 100—Hash filtering mode
  - 101—Reserved
  - 110—Reserved
  - 111—Reserved

- Bits [3–1]—Partition 0 mode → Sets the mode for how RAM Partition Table 0 is used
  - 000—Destination address filtering mode
  - 001—Destination address with broadcast override (broadcast messages are not filtered) mode
  - 010—Accept all broadcast messages mode
  - 011—Source address filtering mode
  - 100—Hash filtering mode
  - 101—Reserved
  - 110—Reserved
  - 111—Reserved

- Bit [0]—Partition table setup
  - 0—Single partition → A single 256 entry partition shared by all UICs
  - 1—Dual partition → Two independent 128 entry partition tables

7.4.2.2 MAC Filter Data Write Register

Writing to this register will write the data word to the next address in the MAC Filter RAM and will automatically increment an internal address pointer to the next MAC Filter RAM table data word. Data words must be written in ascending order by the CPU. The internal address pointer is reset to zero upon the setting of the reset-write-pointer bit in the MAC Filter Mode Configuration register (see Table 7-36).

Table 7-36. MAC_Filter_Data_Write

31–16	15–0
Reserved	Write data
–	W

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved
- Bits [15–0]—MAC filter data. This data is written 16 bits at a time. Three writes are required to set a single 48-bit filter location.

7.4.2.3 MAC Filter Data Read Register

This register displays the current data word read from the MAC Filter RAM table. Reading this register automatically increments an internal address pointer to the next MAC Filter RAM table data word. Data words are read in ascending order. The internal read pointer is reset to zero upon the setting of the reset read pointer bit in the MAC Filter Mode Configuration register (see Table 7-37).

Table 7-37. MAC_Filter_Data_Read

31–16	15–0
Reserved	Read data
–	R

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved
- Bits [15–0]—MAC filter data. This data reads 16 bits at a time. Three reads are required to set a single 48-bit filter location.

7.4.3 MAC Filter RAM organization for HASH Table Mode

When the MAC Filter RAM table is set to HASH mode, data words are organized as two words (16 bits) of HASH data and four bytes (32 bits) of don’t cares. Only the first two bytes of the HASH are used for MAC address filtering decreasing the time necessary for a table lookup.

The HASH table is written by the CPU using a repeated sequence configuration of 16 bits of hash data, followed by four bytes (32 bits) of don’t cares. The example HASH tables below show the arrangement of each partition with MAC HASH bytes from zero to 255. In the examples, a capital “H” represents a byte of HASH data and a capital “X” represents a don’t care (see Tables 7-38 and 7-39).

Table 7-38. MAC Filter HASH Table Partition 0

ADDR	D[47–40]	D[39–32]	D[31–24]	D[23–16]	D[15–8]	D[7–0]
0	H1	H0	X	X	X	X
1	H3	H2	X	X	X	X
...	...	...	...	...	...	...
19	H39	H38	X	X	X	X
20	H41	H40	X	X	X	X
21	H43	H42	X	X	X	X
...	...	...	...	...	...	...
126	H253	H252	X	X	X	X
127	H255	H254	X	X	X	X

Table 7-39. MAC Filter HASH Table Partition 1

ADDR	D[47–40]	D[39–32]	D[31–24]	D[23–16]	D[15–8]	D[7–0]
128	H1	H0	X	X	X	X
129	H3	H2	X	X	X	X
...	...	...	...	...	...	...
147	H39	H38	X	X	X	X
148	H41	H40	X	X	X	X
149	H43	H42	X	X	X	X
...	...	...	...	...	...	...
254	H253	H252	X	X	X	X
255	H255	H254	X	X	X	X

7.4.4 MAC Filter Setup

The following steps are used in setting up the MAC filter for operation:

1. Set up the Partition bit in the MAC Filter Mode Configuration Register for either single- or dual-mode table.

2. Load the table partition(s) with the necessary MAC addresses.

    a. To load a table partition with address data requires three writes to fill one entry.

    b. Data should be written in the following order for an entry: most significant word to least significant word.

    c. For example, using an Ethernet MAC address (lsb - msb) 00:01:02:03:04:05, write the bytes using the following order:

        i. D[47–32] → 0x0504

        ii. D[31–16] → 0x0302

        iii. D[15–0] → 0x0100

3. Set the filtering mode of the partition(s) using the Partition Mode bits in the MAC Filter Mode Configuration Register

4. Assign UICs to MAC filter table partitions using the UIC mode bits in the MAC Filter Mode Configuration Register. UIC mode bits cannot be set for use with Table Partition one (0x11) if the Table Partition bit in the MAC Filter Mode Configuration Register is set for only Partition Table 0.

8. Internal Peripherals

8.1 System Timer

8.1.1 Overview

The System Timer provides multiple periodic System Timer interrupts in hardware. These interrupts can be optionally assigned to the fast-context switching hardware providing a zero overhead system executive. Alternatively, the System Timer interrupts can simply produce a traditional vectored interrupt request to provide a system with basic timing needs. The System Timer removes the necessity of using the peripheral Timer Counter Units (TCUs) as resources for system timing requirements (see Figure 8-1).

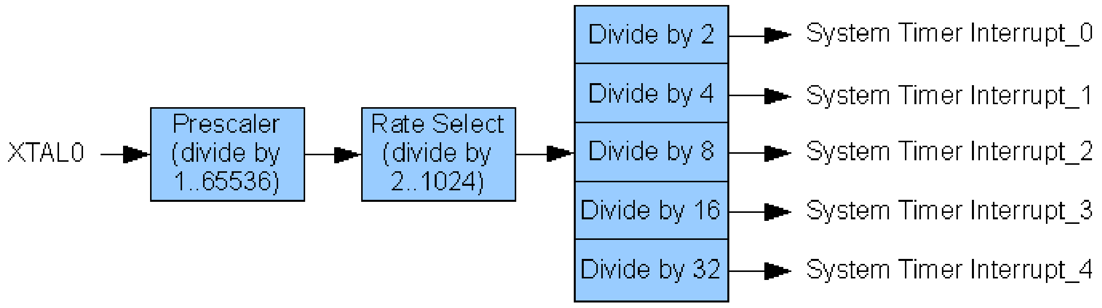


Figure 8-1. System Timer

The System Timer is composed of three stages combined to provide the desired set of interrupts for timing. These stages consist of a Prescaler counter, Rate Selector counter, and a System Interrupt counter. Each of these counters can be configured to provide several timing options as necessary. System timer operation is controlled by the following registers:

- System Timer Control Register
  - Handles overall control of System Timer
- System Timer Prescale Register
- System Timer Interrupt_0 Control Register
  - Controls priority, context association, and enabling of System Timer 0 interrupt
- System Timer Interrupt_1 Control Register
  - Controls priority, context association, and enabling of System Timer 1 interrupt
- System Timer Interrupt_2 Control Register
  - Controls priority, context association, and enabling of System Timer 2 interrupt

- System Timer Interrupt_3 Control Register
  - Controls priority, context association, and enabling of System Timer 3 interrupt

- System Timer Interrupt_4 Control Register
  - Controls priority, context association, and enabling of System Timer 4 interrupt

The Prescaler is a 16-bit, down-counter that converts the CPU fixed clock to a scaled clock signal required for the operation of the Rate Selector counter. The Prescaler operates by counting down from a preload value between 0 and 65535. When the counter reaches zero, the next CPU clock will send a clock pulse to the Rate Selector counter and reload the Prescaler with the count value on the next clock cycle. The Prescaler can be bypassed to allow the Rate Selector counter to operate using the CPU clock by using a preload value of zero.

The Rate Selector counter is a ten-bit, divide-by counter with a selectable output clock rate. The clock rate produced by the rate selector counter is sent to the system-interrupt counter. Divide rates are selectable from divide-by-two to divide-by-1024 in 2^N increments.

The System Timer interrupt counter is a five-bit counter that accepts the output of the rate select counter to generate five different System Timer interrupts, one per counter output bit. Interrupts are assignable with various levels of priorities to separate contexts and can be set up to clear the context timers if necessary.

Each System Timer interrupt occurs at twice the period or half the frequency of the predecessor system's timer interrupt. System Timer Interrupt_0 is half the input clock frequency of the Rate Selector counter, system Interrupt_1 is half the rate of system Interrupt_0 (one-fourth the rate of the select counter), system Interrupt_2 is half the rate of system Interrupt_1 (one-eighth the rate of the select counter), etc. The first System Timer Interrupt_0 will occur after the configured period has elapsed from when the System Timer is enabled. If multiple system timer interrupts are enabled, multiple interrupts may occur at the same time. Tables 8-1 and 8-2 show examples of System Timer interrupt sequencing when multiple System Timer interrupts are enabled. The Interrupt Counter Count columns show the output of the Rate Selector counter.

Table 8-1. Sequence Showing System Timer Interrupts_0 and _1 Enabled

Interrupt Counter Count	Sequence
×	System Timer Interrupt_0
×+1	System Timer Interrupt_0, System Timer Interrupt_1

Table 8-2. Sequence Showing System Timer Interrupts_0, _1, and _2 Enabled

Interrupt Counter Count	Sequence
×	System Timer Interrupt_0
×+1	System Timer Interrupt_0, System Timer Interrupt_1
×+2	System Timer Interrupt_0
×+3	System Timer Interrupt_0, System Timer Interrupt_1, System Timer Interrupt_2

8.1.2 System Timer Interrupt Calculation

To calculate the timing of a particular System Timer interrupt, the formula in Equation 8-1 is used:

$$IRQx_t = 2^{(IRQx+1)} \times \left(\frac{(Prescaler + 1)(RateSelect)}{CPU_{clk}} \right) \qquad \textbf{Equation 8-1}$$

where:

$IRQx_t$	= Time in which a particular interrupt will periodically occur
$IRQx$	= Number of the System Timer interrupt
$Prescaler$	= Prescaler counter value
$RateSelect$	= Rate Select counter value
CPU_{clk}	= CPU clock frequency.

For example, Equation 8-2 calculates the maximum time for System Timer Interrupt_2 using a CPU clock of 66 MHz, the maximum prescaler value of 65535, and the maximum rate select of 1024.

$$IRQ2_t = 2^{(2+1)} \times \left(\frac{(65535 + 1)(1024)}{66{,}000{,}000} \right)$$
$$IRQ2_t = 8 \times 1.0168 = 8.1344 \text{ seconds} \qquad \textbf{Equation 8-2}$$

where:

$IRQ2_t$ = Time in which System Timer Interrupt_2 will periodically occur.

The System Timer Interrupt_2 will occur every 8.1344 seconds.

8.1.3 System Timer Registers

8.1.3.1 System Timer Control Register

The System Timer Control Register sets up the Bit Rate Select value for the Rate Selector counter and enables the overall System Timer circuitry (see Table 8-3). It is access-controlled and writable by only the Master Context, Context_0 (see Chapter 11, Access-Controlled Registers).

Table 8-3. System Timer Control Register

31–5	4–1	0
Reserved	Bit Rate Select	En
–	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–5]—Reserved

- Bits [4–1]—Bit Rate Select → Selects which bit of the System Timer feeds the five-bit System Interrupt Counter
  - 0000—Bit [0] (Divides by 2)
  - 0001—Bit [1] (Divides by 4)
  - 0010—Bit [2] (Divides by 8)
  - 0011—Bit [3] (Divides by 16)
  - 0100—Bit [4] (Divides by 32)
  - 0101—Bit [5] (Divides by 64)
  - 0110—Bit [6] (Divides by 128)
  - 0111—Bit [7] (Divides by 256)
  - 1000—Bit [8] (Divides by 512)
  - 1001—Bit [9] (Divides by 1024)
  - 1010–1111—Reserved

- Bit [0]—System Timer Enable (En) → Enables the System Timer circuitry
  - 0—System Timer disabled
  - 1—System Timer enabled

8.1.3.2 System Timer Prescale Register

The System Timer Prescale Register contains the value that the Prescale counter uses to count down from (see Table 8-4). This value is reloaded automatically into the Prescale counter on the next clock cycle after the counter reaches zero. It is access-controlled and writable by only the Master Context, Context_0 (see Chapter 11, Access-Controlled Registers).

Table 8-4. System Timer Prescale Register

31–16	15–0
Reserved	Prescale Value
–	RW

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved
- Bits [15–0]—Prescale Value → Reloads value when Prescale counter reaches zero
  - Equation 8-3 calculates the prescale value for a desired System Timer Interrupt_0 period.

$$Prescale = \frac{(Desired\ Period\ in\ Seconds)\,(System\ Clock)}{2\,(Bitrate)} \qquad \textbf{Equation 8-3}$$

  - Equation 8-4 presents an example of a System Timer Interrupt_0 with a one second period using a system clock of 12 MHz and a bit rate select value of 1024.

$$Prescale = \frac{1 \times 12{,}000{,}000}{2 \times 1024} = 5859 = 0x16E3 \qquad \textbf{Equation 8-4}$$

8.1.3.3 System Timer Interrupt Control Registers

There is an Interrupt Control Register for each of the five System Timer interrupts generated by the System Timer. These registers define the interrupt priority, context association, mask, enable, and status for each of the System Timer interrupts. In the register definition shown in Table 8-5, the “x” indicates which System Timer interrupts the register controls.

Table 8-5. System Timer Interrupt Control Register

31–12	11	10	9	8	7–6	5–3	2–0
Reserved	Clear CT	Mode	Int Flag	Enable	Reserved	Context	Priority
–	RW	RW	RW	RW	–	RW	RW

Note: Reset value is 0x00000000. x = 0 to 4.

- Bits [31–12]—Reserved
- Bit [11]—Clear Context Timers (Clear CT) → Determines if the interrupt clears the context timer registers.
- Bit [10]—System Timer Interrupt Mode (Mode) → Determines how the associated System Timer interrupt is handled
  - 0—System timer interrupt acts as a standard interrupt request to the assigned context

- 1—Sets the state of the specified context from not ready to ready. If the context is halted or already ready, then this interrupt has no effect.

- Bit [9]—Interrupt Flag (Int Flag) → If this bit is set, it indicates the System Timer interrupt has occurred (cleared on read).

- Bit [8]—Enable → Enables the interrupt
  - 0—Disables the interrupt
  - 1—Enables the interrupt

- Bits [7–6]—Reserved

- Bits [5–3]—Sets which context the System Timer interrupt is associated with
  - 000—Context_0
  - 001—Context_1
  - 010—Context_2
  - 011—Context_3
  - 100—Context_4
  - 101–111—Reserved

- Bits [2–0]—Interrupt Priority (Priority) → Interrupt priority for the assigned interrupt. Interrupt priority range is zero through seven (000–111), where seven is the highest priority.

8.1.4 System Timer Setup

The following steps are used in setting up the System Timer interrupts for operation:

1. Determine and calculate the desired period for the System Timer interrupts needed using the available Prescale and Rate Select values.

2. Using the Master Context (Context_0), load the System Timer Prescale Register and the System Timer Control Register with the values calculated in step one.

3. Using the Master Context (Context_0), assign and enable the System Timer interrupts to the desired contexts using the System Timer Interrupt Control Registers for each interrupt.

4. Set the mode and priority for each System Timer interrupt using the System Timer Interrupt Control Registers for each interrupt.

5. Set the Clear Context Timers (Clear CT) bit in the System Timer Interrupt Control Register associated with the interrupt intended to clear the context timers if required.

Note: Only one System Timer interrupt should have the Clear CT bit set.

6. Using the Master Context (Context_0), set the enable bit in the System Timer Control register to start the System Timer.

Note: The instructions above assume that an Interrupt Service Routine (ISR) exists in each context's exception vector table for System Timer Interrupt_0 to _4.

8.2 Watchdog Timer

8.2.1 Overview

The Watchdog Timer (WDT) is part of the fido1100 fail-safe features. The WDT is an internal programmable timer that is used to prevent system lockup or runaway code. The WDT uses a reloadable countdown value that generates a warning interrupt upon reaching a set value. If this warning interrupt is ignored, the WDT continues counting down and upon reaching zero, generates a system reset (see Figure 8-2). It is the responsibility of the application software periodically to reset the WDT to prevent a timeout from occurring. When the WDT has been enabled, only a power on reset (POR) can disable it.

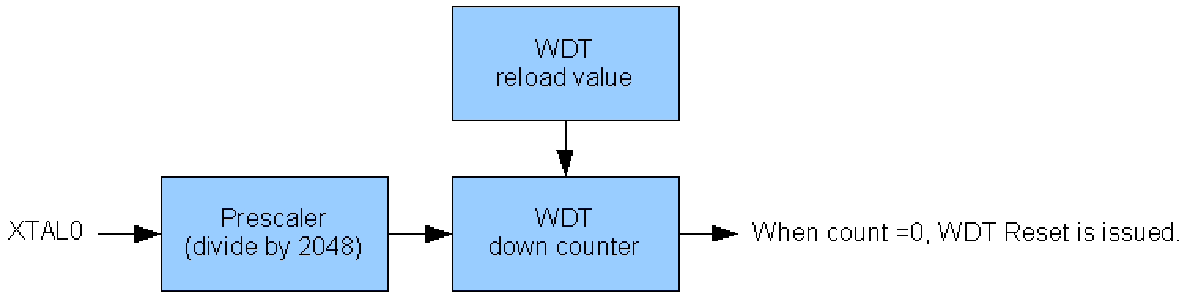


Figure 8-2. Watchdog Timer

The WDT is configured and controlled by two registers:

- The Watchdog Timer Control Register
- The Watchdog Timer Reload Register

The WDT is a 16-bit countdown timer that has a clock input based on the CPU clock divided by a fixed 2048 prescaler. The prescaler takes the CPU clock as input, divides it by 2048 to provide a slower clock for the WDT of approximately 32.3 kHz (assuming the CPU clock is 66 MHz). The WDT uses a value loaded from the 16-bit Watchdog Timer Reload Register. The 12 most significant bits of the reload register can be set by the user, while the four least significant bits are fixed at all ones.

Upon enabling the WDT or on WDT reset, the counter is loaded with the Timer Reload register value and counts down until the value of 0x000F is reached. A WDT warning interrupt is generated indicating a reset will occur within 500 µsec (assuming CPU clock is 66 MHz) if the WDT is not reset. The WDT warning interrupt causes an automatic switch to the Master Context (Context_0) upon occurrence and has a non-maskable priority level (interrupt priority level 7). The WDT is reset or reloaded by a write from any context to the Watchdog Timer Control Register. For any context other than Context_0, the value written to the Watchdog Timer Control Register is ignored. The action of writing to the register accomplishes reloading the WDT.

The maximum time the WDT can be set for is approximately two seconds (assuming the CPU clock is 66 MHz) before the WDT warning interrupt occurs, or a total of 2.5 seconds before reset. During debug operations, the WDT can be frozen by setting the Freeze Watchdog Timer bit in the JTAG Debug Register. Freezing or halting the WDT during a JTAG debug session prevents the WDT from causing a chip reset while the CPU is halted during the debug session.

Note: The JTAG Debug Register is accessible only through the JTAG port.

8.2.2 Watchdog Timer Registers

8.2.2.1 Watchdog Timer Control Register

The Watchdog Timer Control Register enables the WDT and the WDT warning interrupt. This WDT warning interrupt is a non-maskable interrupt (priority-level seven) and will automatically cause a switch to Context_0 upon occurrence (see Table 8-6).

Table 8-6. Watchdog Timer Control Register

31–3	2	1	0
Reserved	Int En	En	Reserved
–	RW	RW	–

Note: Reset value is 0x00000000.

- Bits [31–3]—Reserved
- Bit [2]—Interrupt Enable (Int En)
  - 0—Disables the WDT warning interrupt
  - 1—Enables the WDT warning interrupt
- Bit [1]—Enable (En)
  - 0—WDT is disabled
  - 1—Enables the WDT. Once enabled, the WDT can only be disabled by a power on reset (POR).
- Bit [0]—Reserved

Note: This register is only writeable by the Master Context (Context_0) when the lock bit in the Configuration Access Control Register is deasserted. It is readable by any context.

8.2.2.2 Watchdog Timer Reload Register

The Watchdog Timer Reload Register provides the value from which the WDT counter begins decrementing. This value is loaded into the WDT counter upon reset and when first enabled. The WDT counter is reloaded with this value any time the Watchdog Timer Control Register is written to. Only the upper 12 bits of the Reload Count are writable. The lowest nibble is reserved for the Watchdog Timer warning interrupt (see Table 8-7).

Table 8-7. Watchdog Timer Reload Register

31–16	15–0				
Reserved	Reload Count	1	1	1	1
–	RW	R	R	R	R

Note: Reset value is 0x0000000F.

- Bits [31–16]—Reserved
- Bits [15–0]—Reload Count → Countdown value for the Watchdog Timer counter. The least significant 4 bits of the Reload Count are forced to be ones.

Notes: This register is writeable by only the Master Context (Context_0) when the lock bit in the Configuration Access Control Register is deasserted. It is readable by any context.

The Watchdog Timer Reload Register can be written only if the Enable bit (En) of the Watchdog Timer Control Register is zero.

A Minor Reset reloads the Watchdog Timer but does not disable it.

8.2.3 Watchdog Timer Setup

The following steps are used in setting up the Watchdog Timer for operation.

1. Using the Master Context (Context_0), load the Watchdog Timer Reload register with the desired value.
2. Using the Master Context (Context_0), set the Enable (En) bit in the Watchdog Timer Control register to start the Watchdog Timer.

3. Ensure one of the contexts periodically writes the Watchdog Timer Control register to reset the timer.

8.3 Timer Counter Units

8.3.1 Overview

The fido1100 is equipped with two Timer Counter Units for general-purpose timing requirements. Each Timer Counter Unit (TCU) includes four channels with each channel connected to a dedicated external pin. Channels are bidirectional and configurable for either input data capture or for output data comparison. Registers used by Timer Counter Units include:

- Timer Counter 0 Mode Register
  - Sets Prescaler value for TCU 0
  - Configures TCU 0 operational mode

- Timer Counter 0 Status Register
  - Input capture and output compare status flags for the four channels of TCU 0

- Timer Counter 0 Value Register
  - Write the preload value for TCU 0 to this register
  - Read the current TCU 0 counter value from this register

- Timer Counter 0 Channel N Mode Register (N = 0..3)
  - Four registers to configure the operation of the four channels of TCU 0

- Timer Counter 0 Channel N Input Capture Register (N = 0..3)

- Timer Counter 0 Channel N Output Compare Register (N = 0..3)

- Timer Counter 1 Mode Register
  - Sets Prescaler value for TCU 1
  - Configures TCU 1 operational mode

- Timer Counter 1 Status Register
  - Input capture and output compare status flags for the four channels of TCU 1

- Timer Counter 1 Value Register
  - Write the preload value for TCU 1 to this register
  - Read the current TCU 1 counter value from this register

- Timer Counter 1 Channel N Mode Register (N = 0..3)
  - Four registers to configure the operation of the four channels of TCU 0

- Timer Counter 1 Channel N Input Capture Register (N = 0..3)
- Timer Counter 1 Channel N Output Compare Register (N = 0..3)

External pins used by Timer Counter Units include:

- T0IC0_T0OC0—TCU 0, Input Capture channel 0 or Output Compare channel 0
- T0IC1_T0OC1—TCU 0, Input Capture channel 1 or Output Compare channel 1
- T0IC2_T0OC2—TCU 0, Input Capture channel 2 or Output Compare channel 2
- T0IC3_T0OC3—TCU 0, Input Capture channel 3 or Output Compare channel 3
- T0IC0_T0OC0—TCU 1, Input Capture channel 0 or Output Compare channel 0
- T0IC1_T0OC1—TCU 1, Input Capture channel 1 or Output Compare channel 1
- T0IC2_T0OC2—TCU 1, Input Capture channel 2 or Output Compare channel 2
- T0IC3_T0OC3—TCU 1, Input Capture channel 3 or Output Compare channel 3

Each TCU is composed of a Prescaler and Timer Counter register to provide various timing options as necessary. The Prescaler is a 16-bit, divide-by counter that converts either the fixed CPU clock or an external clock (set by the Clock Mode bits in the TCU Mode register) to a scaled input signal required for the operation of the Timer Counter register. The Prescaler is configurable in divide-by steps of 2^N from one to 32768 using the Prescaler bits in the Timer Counter Mode Register (TCUx_Mode). The Timer Counter register is a 16-bit configurable up/down counter that provides a current count value.

The Timer Counter register can operate using several modes of count configurable through the Autoreload bits in the Timer Counter Mode Register (TCUx_Mode). The Timer Counter register can also be initialized with a preload value between 0 and 65535. With Autoreload disabled, the Timer Counter register will count (depending upon the Up/Down bit in the TCU Mode register) up or down from 0x0000 or the preload value, if used. In single count, the Timer Counter register continues counting until the termination value of 0xFFFF for up count or 0x0000 for down count is reached, stopping the register count.

In free run, the Timer Counter register starts at either up or down from 0x0000 or the preloaded value, if used, and continues counting without stopping at a termination point. Upon reaching 0xFFFF for up count or 0x0000 for down count, the counter rolls over and continues counting without the preload, if used. With Autoreload enabled, upon rollover of the Timer Counter register at 0xFFFF for up count or 0x0000 for down count, the preload value is reloaded and used as the starting count.

At any time, the Timer Counter register can be reset to zero in all modes or the preload value, if used, by setting the reload bit in the TCU Mode register or through an input-capture condition. At any time, the Timer Counter register also can be reset to zero when the Clear-on-Capture bit is set in the TCU Channel Mode register (TCUxx_Chx_Mode) for a particular TCU channel.

The TCU can be configured to provide either a termination count interrupt or an interrupt based upon channel configurations. In single-count mode, a termination count interrupt is generated prior to the rollover of the Timer Counter register from 0x0001 to 0x0000 on down-counts and 0xFFFE to 0xFFFF on up-counts. In free-run mode, the interrupt is generated upon rollover from 0x0000 to 0xFFFF on down-counts and 0xFFFF to 0x0000 on-up counts. The Termination Event Interrupt must be enabled in the TCU Mode register for this interrupt to occur. Interrupts based upon channel configurations can occur either using input-capture or output-compare events.

Within the TCU, each of the four channels is configurable with either input-capture or output-compare functionality. For input-capture functionality, the current Timer Counter register value can be captured to a channel's Input Channel Capture register upon the occurrence of a trigger through a channel's dedicated external pin. Triggers can consist of a rising edge, falling edge, or both edges of an external signal. The trigger also can generate a TCU interrupt or terminate the count and reset the Timer Counter register to zero or to the preload value, when enabled.

For output-compare functionality, a value in the Channel Output Compare register can be used as a termination point. The current Timer Counter register value can be compared against the value of a channel's Output Channel Compare register to trigger an output level to a channel's dedicated external pin. The trigger level will be generated when the current Timer Counter register value equals that of the Output Channel Compare register.

Trigger levels can consist of an output high, an output low, or a toggling of the current output level. When equal, a comparison also can generate a TCU interrupt, if enabled, or terminate the count and reset the Timer Counter register to zero or to the preload value, when enabled.

Each TCU channel has a pulse mode that allows for pulse generation or pulse width measurement. For TCU pulse generation, a channel's dedicated external pin will toggle when the Timer Counter register reaching its termination level. This functionality can be combined with the Channel Output Compare to create various pulse lengths.

For example, with a TCU channel setup using an initialized low level on the channel's dedicated external pin (Pulse mode enabled), with the Output Compare mode set for toggle, and with the Timer Counter register set for counting up (at TCU enable), the Timer Counter register will count up from zero until the count equals the value in the channel's Output Compare register (i.e., the Timer Counter register termination value).

When the two values are equal, the output level will toggle from a low to high level. The Timer Counter register rolls over to zero and counts up until the Timer Counter register count equals the value in the channel's Output Compare register. The output level is toggled again from high to low, completing the pulse. Pulses are created repeatedly in this manner with the pulse width set by the count value of the Timer Counter register.

To measure TCU pulse duration, the count in the Timer Counter register is triggered using a level from a channel's dedicated external pin, configurable using the Pulse Width Measurement Level bit in the TCU Channel Mode register. Pulse duration is measured by the period the Timer Counter register counts while the given level is present. When a level is reached, the Timer Counter register counts until the level changes, causing it to stop. The result corresponds to the duration of the pulse.

8.3.2 TCU Timing Calculation

The following formula is used to calculate the frequency (in counts per second) for the Timer Counter Unit. The TCU timing is calculated using the Prescale setting, Clock source, Preload value, and count direction. See Equation 8-5 for calculating TCU frequencies using a down counter and an up counter.

For calculating a TCU frequency using a down counter :

$$TCU_f = \frac{CLK_f}{(Prescaler)\,(Preload)}$$

Equation 8-5

For calculating a TCU frequency using an up counter :

$$TCU_f = \frac{CLK_f}{(Prescaler)\,(65535 - Preload)}$$

where :

TCU_f	= Timer Counter frequency (in Hz)
CLKf	= CPU clock frequency
Prescaler	= Prescaler value (1 to 32768 in 2^N steps)
Preload	= Counter Preload value.

For example, to calculate a TCU frequency for a down-counter using a CPU clock of 66 MHz, a prescale value of 16, and preload value of 8250 (0x203A), see Equation 8-6.

$$TCU_f = \frac{66{,}000{,}000}{(16 \times 8250)}$$

$$TCU_f = 500\ \text{Hz}$$

Equation 8-6

where :

TCU_f = Timer Counter frequency (in Hz)

Table 8-8 shows examples of frequencies generated with the Timer Counter Unit using a 66-MHz System (CPU) Clock.

Table 8-8. Examples of Frequencies Generated with the Timer Counter Unit

Frequency	Prescale	Preload
0.05 Hz	32768	40,283 (0x9D5B)
1 Hz	1024	64,453 (0xFBC5)
60 Hz	32	34,375 (0x8647)
100 Hz	16	41,250 (0xA122)
1000 Hz	2	33,000 (0x80E8)
50 kHz	1	1,320 (0x528)
100 kHz	1	660 (0x294)

8.3.3 Timer Counter Unit Registers

The Timer Counter registers consist of a TCU control, counter, and status register as well as dedicated channel registers. Each TCU channel has a mode, input capture, and output compare register. For “TCUx” and “CHy,” “x” corresponds to which TCU and “y” corresponds to which channel. For example, TCU0_CH2 would represent TCU0, channel two.

8.3.3.1 Timer Counter Status Registers

The Timer Counter Status register for each TCU provides the status of input captures and output compares for each channel of the TCU and indicates when the Timer Counter register reaches a termination count. Status bits remain set until the register is read, which results in clearing all status bits to zero (see Table 8-9.

Table 8-9. TCUx_Status

31–16	15	14	13	12	11	10	9	8	7	6–0
Reserved	CH0_IC	CH1_IC	CH2_IC	CH3_IC	CH0_OC	CH1_OC	CH2_OC	CH3_OC	TE	Reserved
–	RW	RW	RW	RW	RW	RW	RW	RW	RW	–

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved
- Bit [15]—Channel 0 Input Capture Flag (CH0_IC) → When set, indicates an event occurred and the value from the Timer Counter register has been captured to the TCU Channel 0 Input Capture register.
- Bit [14]—Channel 1 Input Capture Flag (CH1_IC) → When set, indicates an event occurred and the value from the Timer Counter register has been captured to the TCU Channel 1 Input Capture register.

- Bit [13]—Channel 2 Input Capture Flag (CH2_IC) → When set, indicates an event occurred and the value from the Timer Counter register has been captured to the TCU Channel 2 Input Capture register.
- Bit [12]—Channel 3 Input Capture Flag (CH3_IC) → Is set when the value in the TCU Channel 3 Output Compare register is equal to the value of the Timer Counter register.
- Bit [11]—Channel 0 Output Compare Flag (CH0_OC) → Is set when the value in the TCU Channel 0 Output Compare register is equal to the value of the Timer Counter register.
- Bit [10]—Channel 1 Output Compare Flag (CH1_OC) → Is set when the value in the TCU Channel 1 Output Compare register is equal to the value of the Timer Counter register.
- Bit [9]—Channel 2 Output Compare Flag (CH2_OC) → Is set when the value in the TCU Channel 2 Output Compare register is equal to the value of the Timer Counter register.
- Bit [8]—Channel 3 Output Compare Flag (CH0_IC) → Is set when the value in the TCU Channel 3 Output Compare register is equal to the value of the Timer Counter register.
- Bit [7]—Termination Event Flag (TE) → When set, indicates the value in the Timer Counter register has reached a termination point.
- Bits [6–0]—Reserved

8.3.3.2 Timer Counter Mode Registers

The Timer Counter Mode register is used to configure the TCU functions. Only the master context (Context_0) may use the context assignment (Bits [22–20]) to assign the TCU interrupt to a particular context (see Table 8-10).

Table 8-10. TCUx_Mode

31	30–23	22–20	19–17	16	15	14	13–12	11	10–7	6–5	4–0
TC IRQ En	Reserved	Context	Priority	TE IRQ En	Up/ Dn	En	Auto Reload	Reload	Prescaler	CLK Mode	Reserved
RW	–	RW	RW	RW	RW	RW	RW	RW	RW	RW	–

- Bit [31]—Timer Counter Master Interrupt Enable (TC IRQ En) → Enables all interrupts for the Timer Counter Unit.

- Bits [30–23]—Reserved

- Bits [22–20]—Context → Sets which context the Timer Counter Unit interrupt is associated with
  - 000—Context_0
  - 001—Context_1
  - 010—Context_2
  - 011—Context_3
  - 100—Context_4
  - 101–111—Reserved

- Bits [19–17]—Interrupt Priority (IRQ Priority) → Interrupt priority for the assigned interrupt. Priorities range from zero to seven (000–111).

- Bit [16]—Termination Event Interrupt Enable (TE IRQ En) → Will fire an interrupt if TC_IRQEN is set and a Termination Event occurs.

- Bit [15]—Up/Down (Up/Dn) → Determines the direction of count.
  - 0 − Counts down
  - 1 − Counts up

- Bit [14]—TCU Enable (En) → When set, enables the counter to begin counting on the next valid clock edge. Counter is not reloaded upon enabling.

- Bits [13–12]—Auto Reload → Determines the configuration of the Timer Counter auto reload functionality
  - 00—Disables auto reload. Timer will count up/down to full count and stop on Termination Value clearing the En bit.
  - 01—Disables auto reload. Timer will free run by counting up/down to full count without stopping or reloading.
  - 10—Enables auto reload. Timer counts up/down to the Termination Value. On reaching it (rollover), the Timer reload value will be automatically reloaded into the Timer Counter. ICM preload functionality also is enabled.
  - 11—Reserved

- Bit [11]—Reload → When enabled, forces the Timer reload value to be loaded into the Timer Counter (Self Clearing).

- Bits [10–7]—Prescaler → Sets what clock divider value to use. The prescaler counter is reset each time the Timer Counter is reloaded (see Table 8-11).

Table 8-11. Default Prescaler Settings

Prescaler Value	Divide By
0000	1
0001	2
0010	4
0011	8
0100	16
0101	32
0110	64
0111	128
1000	256
1001	512
1010	1024
1011	2048
1100	4096
1101	8192
1110	16384
1111	32768

- Bits [6–5]—Clock Mode (Clk Mode) → Determines whether the internal or external clock is used. For external clock use, allows the selection of edge (see Table 8-12).

Table 8-12. Clock Mode Values

Clock Mode Value	Type of Clock
00	Internal system clock
01	External Clock with input rising edge
10	External Clock with input falling edge
11	External Clock with input rising and falling edge

Note: Reset value is 0x00000000.

- Bits [4–0]—Reserved

8.3.3.3 Timer Counter Register

The Timer Counter Register provides the current up or down count based on a frequency determined by the TCU timing calculations. A preload value may be written to this register for starting a count other than zero (see Table 8-13).

Table 8-13. TCUx_Counter

31–16	15–0
Reserved	Count Value
–	RW

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved
- Bits [15–0]—Count Value → Contains the current count value

8.3.3.4 Timer Counter Channel Mode Registers

The Timer Counter Channel Mode Register configures a channel for operation in either input capture or output compare mode. This register also sets up the channel's dedicated external pin as either an input or output and controls pulse mode operations (see Table 8-14).

Table 8-14. TCUx_Chy_Mode

31–16	15–14	13	12	11–10	9	8	7	6	5–4	3	2	1	0
Reserved	ICM	ICM IRQ En	ICM Pload	OCM	OCM IRQ En	PM	PWM	PWM Level	Reserved	Pin Ctrl	ICM SRC	COC	Reserved
–	RW	RW	RW	RW	RW	RW	RW	RW	–	RW	RW	RW	–

- Bits [31–16]—Reserved
- Bits [15–14]—Input Capture Mode (ICM) → Determines which clock edge the data is captured on (see Table 8-15)

Table 8-15. Input Capture Mode Values

ICM Value	Mode
00	Disabled
01	Use rising edge of clock
10	Use falling edge of clock
11	Use both rising and falling edge of clock

- Bit [13]—Input Capture Mode Interrupt Enable (ICM IRQ En) → When set, enables the Input Capture Mode Interrupt when the input capture condition occurs.
- Bit [12]—Input Capture Mode Preload Enable (ICM Pload) → When set, enables the counter to preload the next count value upon the occurrence of an input capture condition. The preload will only occur if the Autoreload has been enabled in the register.

- Bits [11–10]—Output Compare Mode (OCM) → Sets the level of output upon a compare (see Table 8-16).

Table 8-16. Output Compare Mode Values

OCM Value	Mode
00	Disabled
01	Output a high
10	Output a low
11	Toggle Output

Note: Reset value is 0x00000000.

- Bit [9] — Output Compare Mode Interrupt Enable (OCM IRQ En) → When set, enables the Output Compare Mode Interrupt when the output compare condition occurs.
- Bit [8] — Pulse Mode (PM) → When set, enables the output compare to toggle each time the counter rolls over.
- Bit [7] — Pulse Width Measurement (PWM) → When set, enables the counter to count only when the corresponding TCU input capture input matches the selected level.
- Bit [6] — Pulse Width Measurement Level (PWM Level) → When cleared, counter will count when the corresponding TCU input capture input is low. When set, the counter will count when the corresponding TCU input capture input is high. Pulse Width Measurement must be enabled.
- Bits [5–4] — Reserved
- Bit [3] — Pin Control (Pin Ctrl) → When cleared, sets the associated hardware pin for input capture. When set, enables the associated hardware pin for output compare.

- Bit [2] — Input Capture Source (ICM Src) → When cleared, selects the corresponding pin (subject to the setting of the Pin Crtl bit) as the trigger for input capture mode. When set, selects the internal UIC signal as the trigger for input capture mode.

NOTES:

1. *The UIC can be used to trigger the TCU input capture function. The purpose of this is to support time-based protocols on the UIC. It is specifically intended to support a portion of the IEEE 1588 protocol that involves synchronizing clocks around a network by timing the arrival of special sync messages. This is accomplished with the proper firmware and this bit of the TCU. The UIC's are internally assigned to the TCU channels as follows:*

    - UIC0 Timer 0, Input Capture 0
    - UIC1 Timer 0, Input Capture 1
    - UIC2 Timer 1, Input Capture 0
    - UIC3 Timer 1, Input Capture 1

2. *If using one of the UICs as a source as an Input Capture Source, the Input Capture Mode (ICM) bits must be set for the rising edge of the clock.*

- Bit [1]—Clear on Capture (COC) → When set, enables the counter to reset when an output compare occurs.

- Bit [0]—Reserved

8.3.3.5 Timer Counter Channel Input Capture Registers

The Timer Counter Channel Input Capture register holds the value captured from the Timer Counter register upon the occurrence of the event set by the channel’s mode register. Data remains in the register until the next capture event or the TCU is disabled (see Table 8-17).

Table 8-17. TCUx_Chy_Input_Capture

TCUx_Chy_Input_Capture	
31–16	15–0
Reserved	Capture Value
–	R

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved
- Bits [15–0]—Capture Value → Holds the value captured by the Timer Counter Register upon an event occurrence.

8.3.3.6 Timer Counter Channel Output Compare Registers

The Timer Counter Channel Output Compare register holds the value that is compared to the Timer Counter register. When the two values are equal, an event is generated as configured in the channel's mode register. Data remains in the register until overwritten by software or the TCU is disabled (see Table 8-18).

Table 8-18. TCUx_Chy_Output_Compare

TCUx_Chy_Output_Compare	
31–16	15–0
Reserved	Compare Value
–	R

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved
- Bits [15–0]—Compare Value → the value to be compared with the value in the Timer Counter Register to create an event.

8.3.4 TCU Setup

The following steps are used in setting up the TCU for operation.

1. Determine and calculate the desired frequency for the TCU count using the available Prescale settings, preload value, clock source frequency, and count direction.
2. Using the Master Context (Context_0), assign the TCU interrupt to the desired context using the TCU Mode Register (TCUx_Mode).
3. Set up and configure the TCU for the clock mode, reload type, interrupt mode and priority, counter direction, and operation mode using the TCU Mode Register (TCUx_Mode).
4. If necessary, write a preload value into the Timer Counter Register (TCUx_Counter).
5. Configure the desired channels for either input capture, output compare or pulse width mode operation using each channel's Timer Counter Channel Mode register (TCUx_Chy_Mode).

6. If using the channel for a Pulse mode, set the appropriate pulse width level and mode bits using the channel's Timer Counter Channel Mode register (TCUx_Chy_Mode).

7. If necessary, configure the event trigger levels and external dedicated hardware pins for each channel as appropriate for each channel's mode using the Timer Counter Channel Mode register (TCUx_Chy_Mode).

8. If necessary, load compare values into each channel's TCU Channel Output Compare register (TCUx_Chy_Output_Compare) for those channels set for output mode comparisons.

9. If necessary, configure and enable either the input capture or output compare interrupts for each channel using the channel's Timer Counter Channel Mode register (TCUx_Chy_Mode).

10. Set the enable bit in the Timer Counter Channel Mode register (TCUx_Chy_Mode) to enable the TCU operations by using the Master Context (Context_0).

8.4 Analog-to-Digital Converter

8.4.1 Overview

The Analog-to-Digital Converter (ADC) is an eight-channel, ten-bit linear data acquisition system used to convert analog data inputs to digital values for use by the fido1100 CPU core. The ADC will accept analog input voltages that are between the ADC Voltage Reference High (VRH) and the ADC Voltage Reference Low (VRL) pins. The analog-to-digital conversions are calculated based upon the differential value between VRH and VRL at the time of input. Inputs are digitized using a ten-bit resolution at a maximum throughput rate of 200K samples per second. The upper and lower reference voltage levels are bounded by the analog voltage levels set on external pins VDDA (analog supply voltage) and GNDA (analog ground). The maximum analog input voltage is 3.3 volts DC. ADC registers are as follows:

- ADC Control Register—Configuration of ADC
- ADC Start Register—Used to start the ADC sequencer
- ADC Data Available Register—Status flags for converted data available for each input analog channel
- ADC Data 0 Register—Last conversion value for analog channel 0
- ADC Data 1 Register—Last conversion value for analog channel 1

- ADC Data 2 Register—Last conversion value for analog channel 2
- ADC Data 3 Register—Last conversion value for analog channel 3
- ADC Data 4 Register—Last conversion value for analog channel 4
- ADC Data 5 Register—Last conversion value for analog channel 5
- ADC Data 6 Register—Last conversion value for analog channel 6
- ADC Data 7 Register—Last conversion value for analog channel 7

The ADC can operate in either single-channel or multiple-channel mode. In single-channel operation, one of the eight available channels is selected and can be converted continuously or a single conversion can be done. The Conversion Done (CD) bit in the ADC Control Register will be set after the selected channel's data is stored in its Data Result Register. In multiple-channel mode, all eight channels are used and sequentially converted. Conversions will be done in sequential order from channel zero to channel seven. As each is converted, the channel's data is stored in its Data Result Register. The Conversion Done (CD) bit in the ADC Control Register will be set after channel seven's data has been stored in its Data Result Register.

The ADC implements two modes of scanning: single-channel or multiple-channel. In single-channel scan mode, the ADC continuously converts a designated channel and stores the result in its Data Result register (ADC_ChxDataRegister). In multiple-channel scan mode, all channels are converted continuously and sequentially from zero to seven. New updated data is written to the Data Result register for each channel. If scan mode is disabled, then channels are converted once.

In both non-scan and scan modes, the ADC Start Register (ADC_StartRegister) controls the start of ADC conversions for either single or multiple channels. In non-scan mode, ADC conversions begin when the Start Bit in the ADC Start Register (ADC_StartRegister) is set. The Start Bit remains set until conversion has completed either for a single channel in single channel configuration or for all seven channels in a multi-channel configuration. The Start Bit clears automatically upon completion of the conversion process. Additional conversions are accomplished by setting the Start Bit to 1.

In scan mode, setting the Start Bit in the ADC Start Register starts the continuous conversion process. The Scan Mode Bit must be set in the ADC Control Register (ADC_ControlRegister) for scan mode conversions to start. The Start Bit remains set and conversions are continuous until either the Scan Mode Bit is cleared or the ADC circuitry is disabled by clearing the ADC enable bit in the ADC Control Register.

Either the ADC can be polled by the CPU or it can use the ADC Interrupt to determine when conversions have completed. For a polling setup, the CPU can monitor the Conversion Done Bit (CD) in the ADC Control Register.

The ADC Interrupt can be assigned to any context and has a selectable priority. The ADC Interrupt will cause an Exception #76 to be generated to the assigned context. If enabled, the ADC Interrupt is generated when the Conversion Done Bit in the ADC Control Register is asserted. For single-channel setup in either non-scan or scan mode, the interrupt will be generated upon the completion of a conversion for that channel. For multi-channel setup in either non-scan or scan mode, interrupt will activate upon completion of a sequential conversion pass for all channels from zero to seven. The ADC Interrupt is acknowledged by reading the ADC Control Register.

The ADC Data Available Register indicates what channels have been converted and for which the resulting data has not been read. A channel's data available bit in the ADC Data Available Register is cleared upon reading its data result register.

8.4.2 ADC Registers

8.4.2.1 ADC Control Register

The ADC Control Register is used to setup and configure the ADC functions (see Table 8-19). Context assignment (Bits [13–11]) of the ADC Interrupt to a particular context through the ADC Control register is writable by only the master context, Context_0.

Table 8-19. ADC_ControlRegister

31–14	13–11	10–8	7	6	5	4	3	2–0
Reserved	Context	IRQ Priority	En	Scan	Mult	CD	IRQ Enable	Channel Select
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–14]—Reserved
- Bits [13–11]—Context → Sets which context the ADC interrupt is associated with
  - 000—Context_0
  - 001—Context_1
  - 010—Context_2
  - 011—Context_3
  - 100—Context_4
  - 101–111—Reserved

- Bits [10–8]—Interrupt Priority (IRQ Priority) → Interrupt priority for the ADC interrupt. Priorities range from zero to seven (000–111).
- Bit [7]—ADC Enable (En) → Enables and powers the ADC circuitry
- Bit [6]—Scan Mode (Scan) → Determines the scan mode of the ADC channel inputs
  - 0—Scan is disabled allowing single-channel conversion if in single mode. If in multiple mode, each channel is converted once
  - 1—Scan mode is enabled allowing single continuous channel conversion if in single mode. If in multiple mode, each channel is converted continuously in order zero to seven (upon rollover from channel seven to zero, data in result registers will be overwritten if it has not already been read).
- Bit [5]—Multiple Channel Conversion (Mult) → Enables multiple channel functionality for ADC channels zero through seven
  - 0—Single channel conversion operation
  - 1—Multiple conversion operation (channels zero to seven)
- Bit [4]—Conversion Done (CD) → Indicates the ADC conversion has completed (cleared on read)
- Bit [3]—Interrupt Enable (IRQ En) → Enables the ADC interrupt upon completion of an ADC conversion
- Bits [2–0]—Channel Select → Selects the channel in single-conversion-mode operation. These bits are ignored in multiple-conversion operation.
  - 000—Channel 0
  - 001—Channel 1
  - 010—Channel 2
  - 011—Channel 3
  - 100—Channel 4
  - 101—Channel 5
  - 110—Channel 6
  - 111—Channel 7

8.4.2.2 ADC Conversion Start Register

The ADC Conversion Start Register starts the ADC conversion process. Once started, the Start Bit will indicate “1.” Any subsequent writes of “0” or “1” will have no effect. If not in scan mode, the Start Bit will clear upon completion of the conversion process. In scan mode, the Start Bit remains set until either the Scan mode (Bit [6]) is cleared or the ADC En (Bit [7]) is cleared in the ADC Control Register (see Table 8-10).

Table 8-20. ADC_StartRegister

31–1	0
Reserved	Start
–	RW

Note: Reset value is 0x00000000.

- Bits [31–1]—Reserved
- Bit [0]—Start ADC Conversion (Start) → Starts the ADC conversion process

8.4.2.3 ADC Data Available Register

The ADC Data Available Register provides status indications for each ADC channel. Data available flags are set upon the completion of a conversion for the associated channel. Data available flags are cleared by reading the associated channel's data result register (see Table 8-21).

Table 8-21. ADC Data Available Register

31–8	7	6	5	4	3	2	1	0
Reserved	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bit [7]—Channel 7 Data Available Flag (CH7) → Indicates an ADC conversion has completed and data is available in the channel 7 result register.
- Bit [6]—Channel 6 Data Available Flag (CH6) → Indicates an ADC conversion has completed and data is available in the channel 6 result register.
- Bit [5]—Channel 5 Data Available Flag (CH5) → Indicates an ADC conversion has completed and data is available in the channel 5 result register.
- Bit [4]—Channel 4 Data Available Flag (CH4) → Indicates an ADC conversion has completed and data is available in the channel 4 result register.
- Bit [3]—Channel 3 Data Available Flag (CH3) → Indicates an ADC conversion has completed and data is available in the channel 3 result register.

- Bit [2]—Channel 2 Data Available Flag (CH2) → Indicates an ADC conversion has completed and data is available in the channel 2 result register.
- Bit [1]—Channel 1 Data Available Flag (CH1) → Indicates an ADC conversion has completed and data is available in the channel 1 result register.
- Bit [0]—Channel 0 Data Available Flag (CH0) → Indicates an ADC conversion has completed and data is available in the channel 0 result register.

8.4.2.4 ADC Data Registers

The ADC Data Registers provide results of the converted data for each of the associated ADC channels. Each channel has a dedicated data result register where the "x" represents the channel number. Reading the associated channel data clears the channel's Data Available flag in the ADC Data Available Register (see Table 8-22).

Table 8-22. ADC Channel × Data Register

31–10	9–0
Reserved	Result
–	RW

Note: Reset value is 0x00000000.

- Bits [31–10]—Reserved
- Bits [9–0]—ADC Data Result (Result) → The result of the ADC converted data for a particular channel

8.4.3 ADC Setup

The following steps are used in setting up the ADC for operation.

1. Enable the power to the ADC circuitry by setting the Enable bit in the ADC Control Register.
2. Set up a Multi-bit in the ADC Control Register for either single channel or multiple channel operation.
3. If set up for single channel operation, assign the channel for ADC conversion operation using the Channel Select bits in the ADC Control Register.
4. If using the ADC Interrupt:
   – Assign the ADC Interrupt to the desired context using the Context bits in the ADC Control Register.

Note: The Context field is writable by only the Master Context, Context_0.

- Set the Interrupt Priority bits to the desired priority level and the Interrupt Enable bit in the ADC Control Register.

5. Set the Scan Mode Bit in the ADC Control Register, if scanning mode is desired.

6. Set the Start Bit in the ADC Start Register to start the conversion process.

9. Debug Features

9.1 Overview

Another one of fido's useful features for software developers is an internal debug capability. This section will cover the following internal debug features of fido:

- Brief JTAG discussion
- Debug Feature Associated Registers
- The Software Breakpoint Instruction
- Hardware Breakpoints (chainable instruction events) and Hardware Watchpoints (chainable memory access events)
- Trace Control and Buffering
- Call and branch instruction tracing
- Memory data read/write tracing
- Single address, simple buffer, or circular buffer modes

9.1.1 Context Awareness

The fido1100 debug features are context-specific or what is referred to as "context aware." This means if a breakpoint or watchpoint set up to trigger storage of some parameter writes, this trace can be configured to trigger only if the trigger condition is met while a specific context is executing. For example, consider a system where an error buffer is memory-shared between Context_2 and Context_3. Furthermore, say that both Context_2 and Context_3 could possibly write the same exact data value to the same memory location, but for some strange reason the system only experiences a fault when Context_3 does the write. The debug feature can be set up to only trigger if the write occurs while Context_3 is executing. This is what is meant by the term context-aware debug.

9.1.2 Breakpoints, Watchpoints, and Tracing

In addition to the S/W BKPT instruction, fido has eight hardware breakpoint/watchpoints that can be used to create single or chained events. Breakpoints are used to trigger on instruction-based events and watchpoints are used to trigger on read/writes to memory. Breakpoints and watchpoint events can be used to trigger tracing or storing events to a buffer for later examination. The trace modes are as follows:

- Call Tracing—Records subroutine calls.
- Branch Tracing—Records all jump/branch instruction executions
- Memory Tracing—Records all data written to or read from a certain memory block

The buffer types used to store trace data are also selectable:

- Single Address—All writes to one 32-bit address.
- Simple Buffer—Fills a buffer (32 bytes to 1 Mbyte in selectable size) and stops
- Circular Buffer—Writes to a buffer (32 bytes to 1 Mbyte in selectable size) and wraps continually until explicitly stopped

Chaining a series of breakpoints and/or watchpoints together to create a complex trigger is also supported in fido, subject to a few rules discussed later in this chapter.

9.1.3 Examples

Debug examples are provided in Section 9.6, Debug Feature Examples, which discusses the handling of exception #12 that occurs when the trigger is satisfied. Recall that fido allows each context to have its own unique vector table, so there are really up to five exception #12 handlers. So, given that fido debug is context aware, be sure to have the exception #12 handler covered in any/all contexts in which embedded debug is to run. See Chapter 4, Core CPU, for details on vector table discussion.

9.1.4 Summary

The software BKPT instruction is only the beginning. The fido debug capability goes beyond the typical debug environment where applications are loaded into RAM, and then a stop is performed with the software BKPT instruction. With fido's hardware-based debug features, even a flash-based application can be debugged. Hardware breakpoints or watchpoints can be used to trigger storing instruction or data trace to a memory buffer based on a single or chained sequence of events. These events can be defined to be either instruction-based or data-access-based. Using the debug features has no impact to a system throughout (except for trace buffer writes if a trace buffer is being used).

The fido debug capability is useful during development and for fielded applications that have those once-every-week-the-system-crashes type of debug needs. *It is ROM-able*. A breakpoint or a combination of breakpoints and watchpoints could be set up to trigger call tracing, and left to run. When the trigger occurs, the call sequence (for example) would be written out, and the system or offending context left to run or halted if desired. The trace buffer holds the data for analysis.

9.2 JTAG

The fido1100 supports a JTAG interface through which an external debugger can be connected. Via the external debugger, registers and memory can be written or read, breakpoints can be set, single stepping can be done, and the system can halt/run, etc. JTAG is the debugger interface, and it communicates to fido using the various JTAG scan chains to accomplish these things.

For the purpose of this chapter, however, it is assumed that the debugger is not connected to the system, and the embedded debug capability of fido is being used by the application. All debug features discussed here make this assumption.
One of the JTAG registers has an effect on breakpoint and internal debug behavior of fido. The Debug Control Register has a bit in it (Bit [4], which is the H/W Breakpoint Mode Control bit) that will be 0, after reset, when the debugger is not connected. When the debugger is not connected, leave this bit at 0 or indeterminate behavior can result.

A discussion about the register set that is used to control internal debugging follows. Hardware and software breakpoints are related to the registers that control them. Table 9-1 describes these registers.

9.3 Debug Feature Associated Registers

This section introduces the debug feature registers. For addresses of all registers, see the register memory map in Chapter 12, Register Map Reference. These are all directly addressable memory-mapped registers, subject to the Memory Base Offset Register (see Table 9-1). These registers are access-controlled and writable by only the Master Context, Context_0 (see Chapter 11, Access-Controlled Registers).

Table 9-1. Debug-Feature Register Descriptions

Register Name	Description
Debug Control Register	Defines the mode hardware breakpoints and how the S/W BKPT instruction is treated. A single register and an access-controlled register and writable by only the Master Context (Context_0).
Watchpoint Block Size and Break/Watch Mode Control Register	A four-register set used to manage hardware breakpoints and watchpoints. There are eight sets of these 4 registers such that up to 8 different breakpoints or watchpoints can be set up. These are access-controlled registers and writable by only the Master Context (Context_0).
Breakpoint Address/Watchpoint Base Address Register	
Watchpoint Data Register	
Watchpoint Data Mask Register	
Debug Trace Buffer Control Register	A register pair used to specify the trace-buffer address, buffer size, and buffer mode. These are access-controlled registers and writable by only the Master Context (Context_0). Used in conjunction with the mode field in the Watchpoint Block Size and Break/Watch Mode Control Register, when the mode field is set to trace enable, the trace and buffering controls kick in. Tracing can also be explicitly enabled by a write of the enable bit by the Master Context.
Base Memory Write Address Register	

These registers can be viewed in three classes:

- Overall debug control
- Hardware breakpoint and watchpoint control
- Tracing

The Debug Control Register controls how the S/W BKPT instruction is used.

Breakpoints and watchpoints can be used without invoking the trace feature if desired. Tracing is invoked only when the mode field in the Watchpoint Block Size and Break/Watch Mode Control Register is set to enable trace. That is how tracing and breakpoints are optionally linked.

The registers for tracing (Debug Trace Buffer Control Register and the Base Memory Write Address Register) allow for the type of tracing (instruction versus data) and for the type/size of trace buffer used.

9.3.1 Software Breakpoint Instruction

The Debug Control Register is used to control how the S/W BKPT instruction is treated. This is a memory-mapped register. It is access-controlled and writable by only the Master Context, Context_0 (see Chapter 11, Access-Controlled Registers). From the register table, the register discussed here is provided in Table 9-2.

Table 9-2. Debug Control Register

Register Name	Description
Debug Control Register	Defines the mode hardware breakpoints and how the S/W BKPT instruction is treated. A single register and an access-controlled register and writable by only the Master Context (Context_0).

The detailed definition of this register is provided in Table 9-3.

Table 9-3. Debug Control Register Definition

31–5	4	3–1	0
Reserved	H/W Breakpoint Mode	Reserved	S/W BKPT instruction ctrl
RW	–	RW	–

Note: Reset value is 0x00000000.

- Bits [31–5]—Reserved
- Bit [4]—H/W Breakpoint mode bit, only applies if the debugger is connected

Note 1: When the debugger is not connected, this bit should be left at its default setting of logic 0 to prevent indeterminate behavior.

Note 2: When the debugger is not connected, exception #12 is raised to the running context when an S/W BKPT or H/W breakpoint/watchpoint is encountered.

- Bits [3–1]—Reserved
- Bit [0]—S/W Breakpoint instruction control bit
  - 0 → BKPT instruction executes as a NOP; instruction execution uninterrupted
  - 1 → BKPT instruction will generate exception #12 to the local context (no HALT occurs—what happens next is up to the exception handler, which is application-dependent)

Note 3: If the debugger is connected, the BKPT instruction will not generate exception #12 but rather will HALT either just the running context or the entire CPU, depending upon the breakpoint mode bit in the JTAG Debug Control Register (which is not software-accessible).

9.3.2 S/W BKPT

The S/W BKPT instruction can be useful in debugging ROM-able applications. If the S/W BKPT instruction ctrl bit (Bit [0]) is set to 0, the BKPT instruction is treated as an NOP. Therefore, debugging with this instruction can be easily turned on and off. When set to 1, exception #12 (the BKPT exception in the vector table) is generated to the running context. No HALT occurs here—what happens next it is up to the running context BKPT exception handler. This handler could be coded to store information, halt the context if desired, or trapx to the Master Context, etc.

Therefore, to support embedded debugging, S/W BKPTs could be strategically sprinkled in code where desired and the exception handler could be used to determine the origin of some problem. They can be turned on and off with a single line of code change or compiler directive. (For example, the code in the Master Context could simply write a 1 to this register to turn it on, and this code could be subject to a #ifdef compiler directive.)

Although software breakpoints are handy, they are limited in ability. Hardware breakpoints, discussed below, have more options and greater debug capability.

9.4 Hardware Breakpoints and Watchpoints

The Debug Breakpoint and Watchpoint Registers are used to control how breakpoints and/or watchpoints are defined. From the register table, the registers discussed here are shown in Table 9-4.

Table 9-4. Debug Breakpoint and Watchpoint Registers

Register Name	Description
Watchpoint Block Size and Break/Watch Mode Control Register	This 4-register set is used to manage H/W breakpoints and watchpoints. There are eight sets of these 4 registers such that up to 8 different breakpoints or watchpoints can be set up. This is an access-controlled register and writable by only the Master Context (Context_0).
Breakpoint Address/Watchpoint Base Address Register	
Watchpoint Data Register	
Watchpoint Data Mask Register	

There are eight break/watch points available to the user (8 sets of 4 registers). Breakpoints and watchpoints can be chained based on numerical order, but the last one not marked as chained must be marked break or trace. The differences are:

- Break points monitor code execution and conditionally stop the processor to allow debugging. *Breakpoints occur upon instruction execution, not pre-fetch.*

- Watch Points monitor data read/written to memory and conditionally stop the processor to allow debugging.

Both can trigger the trace buffer to record information for later examination.

- Call tracing—Records subroutine calls
- Branch Tracing—Records all jump/branch instruction executions

- Memory Tracing—Record all data written to or read from a certain memory block

9.4.1 Watchpoint Block Size and Break/Watch Mode Control Register

This register is the key to defining breakpoints and watchpoints. It determines a breakpoint versus a watchpoint, triggering on a read versus write, context awareness, or an enable bit. For watchpoints, it allows a block size to monitor. Lastly, the Mode controls what happens when the event occurs (raise an exception, go to next breakpoint/watchpoint in a chain, or turn on tracing) (see Table 9-5).

Table 9-5. Watchpoint Block Size and Break/Watch Mode Control Register

31–16	15	14–13	12	11–10	9–7	6	5–4	3–0
Reserved	Enable	R/W Control	Context Aware	Reserved	Context	Type	Mode	Block Size
–	RW	RW	RW	–	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved

- Bit [15]—Enable (0 → disable, 1 → enable)

- Bits [14–13]—Read/Write trigger control
  - 00 → Trigger on read or write
  - 01 → Trigger on write
  - 10 → Trigger on read
  - 11 → Reserved

- Bit [12]—Context Aware control
  - 0 → Trigger in any context
  - 1 → Trigger in specified context only (as defined in context field in bits 9–7)

- Bits [11–10]—Reserved

- Bits [9–7]—Context specific trigger control → this field specifies which context (0–4) in which the event must occur to cause a trigger

Note: If Bit [12] is logic 0, this field is ignored.

- Bit [6]—Type (0 → Breakpoint, 1 → Watchpoint)

- Bits [5–4]—Mode
  - 00 → Reserved
  - 01 → Enable trace (turn on tracing when event triggers)
  - 10 → Breakpoint (raise exception on event trigger)

Note: It will HALT if the debugger is connected.

  - 11 → Chained (enable next breakpoint/watchpoint in chain)

Note: The last event in a chain must be breakpoint or enable trace.

- Bits [3–0]—Block size (for watchpoints only), will watch memory in this block size
  - 0000 → 1 byte
  - 0001 → 2 bytes
  - 0010 → 4 bytes
  - 0011 → 8 bytes
  - 0100 → 16 bytes
  - 0101 → 32 bytes
  - 0110 → 64 bytes
  - 0111 → 128 bytes
  - 1000 → 256 bytes
  - 1001 → 512 bytes

- 1010 → 1024 bytes
- 1011 → 2048 bytes
- 1100 → 4096 bytes
- 1101 → 8192 bytes
- 1110 → 16384 bytes
- 1111 → 32768 bytes

9.4.2 Breakpoint and Watchpoint Chaining

There are eight sets of four registers defining breakpoints/watchpoints. They are numbered in order, zero to seven. For example, assuming the user starts out with breakpoint_0 to trigger on an instruction address and the mode for this breakpoint is chained. Breakpoint_1 must be used for the next event the user sets up (which could be another instruction address or it could write to memory of some data value). If breakpoint_1 is chained, breakpoint_2 must be used for the next event, and so on. The last breakpoint in the chain must have its mode set to either breakpoint or enable trace. The user could set up multiple chained sequences, as long as they both abide by the numerical ordering. The user can have as few or as many as needed, but must abide by the numerical ordering and limit of eight.

9.4.3 Breakpoint Address/Watchpoint Base Address Register

This register defines the breakpoint/watchpoint address. For a breakpoint, it is the instruction address the user wants to trigger on. For a watchpoint, it can be either the single address of the data location or the base address of a block of memory defined by the block size (see Table 9-6).

Table 9-6. Breakpoint Address/Watchpoint Base Address Register

31–0
Base Address
RW

Note: Reset value is 0x00000000.

- Bits [31–0]—Instruction address or base data address for the breakpoint/watchpoint

> *Note: For a watchpoint, the base address is subject to a boundary based on the block-size field in the Watchpoint Block Size and Break/Watch Mode Control Register.*

For example, if a trace on a single write to memory occurs within a block, the address of that single location must fall within the range of the block size, and the base address here must accommodate that. Typically, the user just needs to mask off the trigger address by the block size and set the base address with that value (see Table 9-7).

Table 9-7. Watchpoint Block Size and Break/Watch Mode Control Register

Sample Trigger Address	Block Size	Breakpoint Address/Watchpoint Base Address Register
0x10A_4814	128 bytes	0x10A_4800
0x10A_714C	1024 bytes	0x10A_7000

9.4.4 Watchpoint Data Register

This register allows the user to qualify the data at the address contained in the Breakpoint Address/Watchpoint Base Address Register. *For a breakpoint, this register is ignored because the address is an instruction address.* For a watchpoint, this would contain the data value the user is looking for (anywhere in the block size). The user can specify whether looking for a read or a write (or don't care) in the Watchpoint Block Size and Break/Watch Mode Control Register (see Table 9-8).

Table 9-8. Watchpoint Data Register

31–0
Data
RW

Note: Reset value is 0x00000000.

- Bits [31–0]—Data value for watchpoint

9.4.5 Watchpoint Data Mask Register

This register allows the user to qualify further the data at the address contained in the Breakpoint Address/Watchpoint Base Address Register. *For a breakpoint, this register is ignored because the address is an instruction address.* For a watchpoint, this mask allows the user to trigger on the data in the Watchpoint Data Register through its mask value.

Here is how the mask works. For any bit set to logic 1 in the mask, a hardware comparator is enabled for that bit. For any bit set to logic 0, the comparator is disabled for that bit. For example, if the data register holds 0x12345678 and the data mask is 0xFFFFFFFF, the data value read from or written to must be 0x12345678 in order for it to trigger. If the data mask is 0x00000000, any data value read or written to at the address in the Breakpoint Address/Watchpoint Base Address Register would cause a trigger (see Table 9-9).

Table 9-9. Watchpoint Data Mask Register

31–0
Data Mask
RW

Note: Reset value is 0x00000000.

- Bits [31–0]—Data mask for watchpoint

9.5 Trace Capability

Two registers are used to control the tracing mode and the trace buffering. From the register table, the registers presented here are shown in Table 9-10.

Table 9-10. Trace Control Register

Register Name	Description
Debug Trace Buffer Control Register	A register pair used to specify trace buffer address, buffer size, and buffer mode. These are both access-controlled registers and writable by only the Master Context (Context_0). Used in conjunction with the mode field in the Watchpoint Block Size and Break/Watch Mode Control Register, when the mode field is set to trace enable, the trace and buffering controls kick in. Tracing can also be explicitly enabled by a write of the enable bit by the Master Context.
Base Memory Write Address Register	

Trace capability is invoked when the mode field in the Watchpoint Block Size and Break/Watch Mode Control Register is set to enable trace. This opens up further debug opportunities because information can be stored upon breakpoint/watchpoint events.

It is possible to invoke tracing without using a breakpoint/watchpoint to trigger it. If desired, the Master Context can turn on tracing at any time by setting the enable bit (Bit [15]). This option is very useful in capturing data prior to an event.

Table 9-11. Debug Trace Buffer Control Register

31–16	15	14	13–12	11–10	9–8	7–6	5–4	3–0
Reserved	Enable	Full Flag	Reserved	Trace Mode	Reserved	Buffer Mode	Reserved	Buffer Size
–	RW	RW	–	RW	–	RW	–	RW

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved

- Bit [15]—Trace enable (enabled explicitly by Context_0 or break/watch point; disabled explicitly by Context_0 or when buffer full)
  - 0 → Disable
  - 1 → Enable

- Bit [14]—Full flag
  - In simple buffer trace mode, the full flag bit is set to 1 when the trace buffer fills. The full flag bit is cleared when Context_0 writes a 1 to this bit.
  - If the full flag bit is set when a break/watch point is evaluated as "true," a new trace will not be started.

- Bits [13–12]—Reserved
- Bits [11–10]—Trace mode
  - 00 → Reserved
  - 01 → Call tracing mode
  - 10 → Branch tracing mode
  - 11 → Data tracing mode
- Bits [9–8]—Reserved
- Bits [7–6]—Specifies trace buffer mode
  - 00 → Single address mode
  - 01 → Circular buffer mode
  - 10 → Simple buffer mode
- Bits [5–4]—Reserved
- Bits [3–0]—Buffer size (used in the case of the buffer modes)
  - 0000 → 32 bytes
  - 0001 → 64 bytes
  - 0010 → 128 bytes
  - 0011 → 256 bytes
  - 0100 → 512 bytes
  - 0101 → 1 Kbyte
  - 0110 → 2 Kbytes
  - 0111 → 4 Kbytes
  - 1000 → 8 Kbytes
  - 1001 → 16 Kbytes
  - 1010 → 32 Kbytes
  - 1011 → 64 Kbytes
  - 1100 → 128 Kbytes
  - 1101 → 256 Kbytes
  - 1110 → 512 Kbytes
  - 1111 → 1 Mbyte

The second register of this pair specifies where the trace buffer is in memory (see Table 9-12).

Table 9-12. Trace Buffer Base Address Register

31–0
Base Address of Trace Buffer
RW

Note: Reset value is 0x00000000.

- Bits [31–0]—Address in memory of trace buffer

Note: This base address must be on a boundary defined by the buffer size field of the Debug Trace Buffer Control Register.

9.5.1 Instruction Tracing Mode

The trace modes are as follows:

- Call Tracing Mode—Records the following address control transfers
  - Subroutine calls
  - Subroutine returns
  - Interrupts and exceptions
  - Context switches

- Branch Tracing Mode—Records the following address control transfers
  - Unconditional branches and jumps
  - Conditional branches and jumps (only if taken)
  - Subroutine calls
  - Subroutine returns
  - Interrupts and exceptions
  - Context switches

- Data Tracing Mode—Records the following data for a specified address range
  - Data written or data read (only the data, not the address)

Note: DMA reads/writes are not supported. SDRAM reads/writes are supported.

  - Context switches

These modes are mutually exclusive; only one type of tracing can active at any given time. The call and branch instructions that generate address strobes during instruction tracing is further clarified in Table 9-13.

Table 9-13. Call and Branch Instructions that Generate Address Strobes

CPU Instruction	Call Tracing	Branch Tracing
BRA (branch always)	–	Yes
BSR (branch subroutine)	Yes	Yes
Bcc (branch conditionally)	–	Yes
Dbcc (decrement and branch conditionally)	–	Yes
JMP (jump)	–	Yes
JSR (jump subroutine)	Yes	Yes
RTD (return and deallocate)	Yes	Yes
RTE (return from exception)	Yes	Yes
RTR (return and restore condition codes)	Yes	Yes
RTS (return from subroutine)	Yes	Yes
Interrupts	Yes	Yes
Exceptions	Yes	Yes
Context switches (recorded as 0–4)	Yes	Yes

Therefore, for call and branch tracing, what is recorded in the trace buffer is the call/branch address history, along with any context switches, if any, that occurred. This can be useful in trying to reconstruct the path before or after something goes wrong. For data tracing, any reads or writes defined by the watchpoint block size will be written to the trace buffer, along with context switch signatures if any occurred. The user would have to analyze this data using a current link map. Some examples are given later in the section.

9.5.2 Trace Buffer Mode

The buffer mode for tracing can be set to one of three methods:

- Single-Address Mode—All trace data written to the same address. This would likely be captured by a logic analyzer and post processed as needed.

- Circular Buffer Mode—The data is written to a memory buffer and, when full, loops around to the beginning. *This is perhaps the desired mode for capturing events occurring prior to a trigger condition*. Tracing would have to be explicitly enabled by the Master Context. A breakpoint/watchpoint defining the trigger is set up using breakpoint mode. When the event occurs, the BKPT exception fires to the local context that would trapx to the Master Context and, in turn, disable tracing explicitly. The sequence leading up to the trigger could be deciphered from the buffer by finding the context switch to Context_0 (knowing that context switches are recorded in the trace buffer). (See Table 9-16, Example 3 of Debug-Feature Register Implementation.)

- Simple-Buffer Mode—Filled to capacity and then disables tracing, Clears Enable Bit, and break/watch points cannot re-enable until Clear Full bit is toggled. This would likely be the mode to use when the wishes to capture events after a trigger condition has been satisfied. The BKPT exception #12 will occur after the buffer has been filled.

Note: Remember that one restriction for the simple/circular mode buffering is that the base address contained in Memory Write Address Register must be on a boundary defined by the trace-buffer-size field in the Debug Trace Buffer Control Register.

9.6 Debug Feature Examples

In this section, three debug examples will be presented, along with a suggested implementation to use in setting up the debug registers. In all three examples, the setting up of the debug registers would take place at the end of initialization in the Master Context. Therefore, all contexts used are presumed to have been set up and are either RDY to run or will run when an interrupt places them in a RDY state.

- Example 1—Capture call sequence after a function named routine_1 is entered in Context_3, store this call sequence to a 1024-byte trace buffer. Stop trace when buffer full.

- Implementation—Set the debug registers as shown in Table 9-14.

Table 9-14. Example 1 of Debug-Feature Register Implementation

Debug Feature Register Name	Hex Value	Notes
Debug Control Register	0x00000001	Makes S/W BKPT instruction an NOP.
Watchpoint Block Size and Break/Watch Mode Control Register	0x0000D190	Enabled, breakpoint type, read, Context_3 aware, mode is enable trace, block size is N/A for a breakpoint.
Breakpoint Address/Watchpoint Base Address Register	&routine_1	This is the address of function routine_1.
Watchpoint Data Register	0x00000000	This is N/A for a breakpoint.
Watchpoint Data Mask Register	0x00000000	This is N/A for a breakpoint.
Debug Trace Buffer Control Register	0x00004485	Tracing initially disabled, will be enabled by breakpoint, clear full flag, call mode, simple buffer; buffer size is 1024 bytes to capture at least 256 calls.
Base Memory Write Address Register	address of trace buffer, for example 0xXXXXX400	Puts the address in memory where the user wants the trace buffer to reside. In this case, it must be on 1024-byte boundary.

For this example, the setup is complete. The Context_3 BKPT exception handler (vector #12) would need to be coded to handle the exception as required. It could simply return and allow the system to continue running, it could trapx to the Master Context, or it could perform some I/O as an external signal to the outside world. It could pipe the contents of the trace buffer out to GPIO. Those options are up to the user.

Note 1: In this example, only one of the eight breakpoint/watchpoint register sets is being used. Any of the eight sets (of four registers) could be used to define this breakpoint.

Note 2: In this example, the exception #12 occurs when the simple buffer is full.

- Example 2—Sets a chained breakpoint. Makes first event the instruction address of a function named routine_3 in Context_2, followed by a Context_2 write to RAM (variable named test_1, data value must be 0x12345678), and followed by an any context read from RAM (variable name test_2, data is don't care). Then triggers, enables branch tracing to a 2048-byte trace buffer, and stops trace when buffer is full.

- Implementation—Sets the debug registers as shown in Table 9-15.

Table 9-15. Example 2 of Debug-Feature Register Implementation

Debug Feature Register Name	Hex Value	Notes
Debug Control Register	0x00000001	Make S/W BKPT instruction an NOP.
Watchpoint Block Size and Break/Watch Mode Control Register_0	0x0000D130	Enabled, breakpoint type, read, Context_2 aware, mode is chained, block size is N/A for a breakpoint.
Breakpoint Address/Watchpoint Base Address Register_0	&routine_3	This is the address of function routine_3.
Watchpoint Data Register_0	0x00000000	This is N/A for a breakpoint.
Watchpoint Data Mask Register_0	0x00000000	This is N/A for a breakpoint.
Watchpoint Block Size and Break/Watch Mode Control Register_1	0x0000B172	Enabled, watchpoint type, Context_2 aware, mode is chained, memory write to a block size of 4 for specific address.
Breakpoint Address/Watchpoint Base Address Register_1	&test_1	This is the address in memory of variable test_1.
Watchpoint Data Register_1	0x12345678	Data value of 0x12345678.
Watchpoint Data Mask Register_1	0xFFFFFFFF	Enables all mask bits, data value must now be 0x12345678.
Watchpoint Block Size and Break/Watch Mode Control Register_2	0x0000C052	Enables, watchpoint type, any context, mode is enable trace, read from a block size of 4 for a specific address.
Breakpoint Address/Watchpoint Base Address Register_2	&test_2	This is the address in memory of variable test_2.
Watchpoint Data Register_2	0x12345678	Data value of 0x12345678.

Table 9-15. Example 2 of Debug-Feature Register Implementation *(Continued)*

Debug Feature Register Name	Hex Value	Notes
Watchpoint Data Mask Register_2	0x00000000	Mask disabled, will trigger on read of any value.
Debug Trace Buffer Control Register	0x00004886	Tracing initially disabled, will be enabled by watchpoint, clear full flag, branch mode, simple buffer, buffer size is 2048 bytes to capture at least 512 calls/branches.
Base Memory Write Address Register	Address of trace buffer, for example 0xXXXXX800	Puts the address in memory where the user wants the trace buffer to reside. In this case, it must be on 2048-byte boundary.

For this example, the setup is complete. Since the final event in the chain is not context specific, the BKPT exception handlers for all used contexts would need to be coded to field the exception. How the exception is handled is up to the user.

Note 1: In this example, three of the eight breakpoint/watchpoint register sets are being used. Any three contiguous sets could be used to define this breakpoint; the first three were used in this case (0–2). In addition, the last event in the chain, in this case the Watchpoint Block Size and Break/Watch Mode Control Register_2, must be set to enable trace. This indicates the end of the chain. All three of the events must occur in order for the trigger to occur. The function routine_3 could be entered numerous times, but until the write to variable test_1 followed by the read of variable test_2 occurs, there will be no trigger.

Note 2: In this example, the exception #12 occurs when the simple buffer is full.

- Example 3—Sets a chained breakpoint. Makes the first event the instruction address of a function named routine_3 in Context_4, followed by a Context_4 write within a 128-byte RAM block (base at variable named test_1, data value must be 0x12345678), followed by the instruction address of a function named “routine_4” in Context_4. This captures the call sequence that occurred leading up to, rather than after, the event.

- Implementation—Sets the debug registers as shown in Table 9-16.

Table 9-16. Example 3 of Debug-Feature Register Implementation

Debug Feature Register Name	Hex Value	Notes
Debug Control Register	0x00000001	Make S/W BKPT instruction an NOP.
Watchpoint Block Size and Break/Watch Mode Control Register_5	0x0000D230	Enabled, breakpoint type, Context_4 aware, mode is chained; block size is N/A for a breakpoint.
Breakpoint Address/Watchpoint Base Address Register_5	&routine_3	This is the address of function routine_3.
Watchpoint Data Register_5	0x00000000	This is N/A for a breakpoint.
Watchpoint Data Mask Register_5	0x00000000	This is N/A for a breakpoint.
Watchpoint Block Size and Break/Watch Mode Control Register_6	0x0000B277	Enabled, watchpoint type, Context_4 aware, mode is chained, memory write to a block size of 128 for a range of addresses.
Breakpoint Address/Watchpoint Base Address Register_6	&test_1 (mask w/size)	This is the base address in memory of a 128-byte block of interest. Boundary mask with block size is needed.
Watchpoint Data Register_6	0x12345678	Data value of 0x12345678.
Watchpoint Data Mask Register_6	0xFFFFFFFF	Enables all mask bits, data value must now be 0x12345678
Watchpoint Block Size and Break/Watch Mode Control Register_7	0x0000D220	Enables, breakpoint type, Context_4 aware, mode is breakpoint; block size is N/A for a breakpoint.
Breakpoint Address/Watchpoint Base Address Register_7	&routine_4	This is the address in memory of function routine_4.
Watchpoint Data Register_7	0x00000000	This is N/A for a breakpoint.
Watchpoint Data Mask Register_7	0x00000000	This is N/A for a breakpoint.
Debug Trace Buffer Control Register	0x0000C446	Tracing initially enabled, will be disabled explicitly by trapx to Context_0, clear full flag, call mode, circular buffer; buffer size is 2048 bytes to capture at least 512 calls/branches.
Base Memory Write Address Register	Address of trace buffer, for example 0xXXXXX800	Put the address in memory where the user wants the trace buffer to reside. In this case, it must be on 2048-byte boundary.

For this example, the setup is complete. The Context_4 BKPT exception hander (vector #12) would need to be coded to perform a trapx to the Master Context in order to disable tracing as quickly as possible, thus preserving the trace buffer, which should now contain the call sequence leading up to the trigger. A switch to Context_0 will indicate a key mark in the buffer. From there, back up to reconstruct the sequence of events. The trapx handler for the Master Context could be code to do whatever else is appropriate, which is left to the user.

Note 1: In this example, three of the eight breakpoint/watchpoint register sets are being used. Any three contiguous sets could be used to define this breakpoint; the last three were used in this case (5–7). In addition, the last event in the chain, in this case the Watchpoint Block Size and Break/Watch Mode Control Register_7, must be set to breakpoint mode. This indicates the end of the chain. All three of the events must occur in order for the trigger to occur.

Note 2: In this example, the exception #12 occurs when the last watchpoint is encountered.

10. Power Control

10.1 Overview

A discussion of the power saving options of the fido chip is provided in this section. In certain applications, the ability to invoke reduced power modes is desirable and fido supports three power saving modes. Clock signals can be selectively controlled to manage power while retaining memory and register contents. Power-saving modes are entered either automatically or programmatically by using the STOP or LPSTOP instructions from the Master Context. In addition, two registers provide additional power control while the system is running. These registers are discussed later in this section (see Sections 10.3.1, Power-On Reset Register, and 10.3.2, Clock Mask Register).

10.2 Power-Saving Modes

The three power-saving modes are as follows:

- AutoSleep—This mode is automatic in fido. Any time all contexts are set to the NOT_RDY state this mode is automatically entered. The CPU will be fully capable, but no memory access will be made; all registers, the ALU, etc., will not be transitioning but will be preserved. Contexts go NOT_RDY by executing the SLEEP instruction. An interrupt will make the associated context RDY and operation continues as usual. The CPU clock is running in this mode.

- Idle—This mode will save more power than AutoSleep. To enter this mode, the Master Context will issue a STOP instruction. The entire CPU will freeze in place (i.e., all contexts will stop and wait). This mode must be used carefully as it is probably best if anything that is running is prepared for this to happen (i.e., all other contexts are either NOT_RDY or not doing anything critical). The Master Context can explicitly set all contexts to the NOT_RDY state, setting PCs if and stack pointers if necessary, etc. Care must be taken to have the user's application in a known state before restart. Here the CPU will hold its place but the CPU clock will stop so more power is saved. An interrupt is needed to exit this mode.

- Hard Sleep—This mode saves the most power possible, halting the entire machine. This state will be entered into by the programmer via the LPSTOP instruction from the Master Context. The entire clock tree (CPU and peripherals) is halted, along with the oscillator circuit. RAM contents and registers are maintained. The RESET_N pin must be asserted and released to bring the processor out of this state and re-start the clock. Software can view the Power-On Reset Register to determine whether it is a hard reset or recovering from hard sleep. As with the Idle mode, software should perform any cleanup work necessary before entering this mode (such as stopping UIC programs or SDRAM).

Table 10-1 summarizes the fido1100 power saving modes.

Table 10-1. Summary of fido1100 Power-Saving Modes

Element	AutoSleep Mode	Idle Mode	Hard Sleep Mode
Invoked via	SLEEP Instruction	STOP Instruction[1]	LPSTOP Instruction[1]
Which Context	All Contexts	Context_0	Context_0
Exited by	Interrupt	Interrupt	Reset
CPU Clock	Running	Halted	Halted
Master Clock	Running	Running	Halted
External Mem Clk	Running	Running	Halted
External xtal1	Running	Running	Halted
Execution Unit	Running (Idle)	Stopped (Waiting for Interrupt)	Halted
All Contexts	NOT_RDY	NOT_RDY	Stopped (State Unchanged)
Interrupts	Operating	Operating	Halted
Internal Registers	Data Retained	Data Retained	Data Retained
Internal SRAM	Data Retained	Data Retained	Data Retained
Internal RREM	Data Retained	Data Retained	Data Retained
PMU RX Mem	Data Retained	Data Retained	Data Retained
PMU TX Mem	Data Retained	Data Retained	Data Retained
PMU MacFilter Mem	Data Retained	Data Retained	Data Retained
UIC RX & TX Mem	Data Retained	Data Retained	Data Retained
UIC Program Mem	Data Retained	Data Retained	Data Retained
SDRAM	Refresh Running	Refresh Running	Refresh Halted
AtoD	Running	Running	Halted
DMA	Running	Running	Halted
TCU	Running	Running	Halted
SYSTEM TIMERS	Running	Running	Halted
WDT	Running	Running	Halted
DEBUG	Running	Running	Halted
IBIU	Running	Running	Halted
EBIU	Running	Running	Halted
CMU	Running	Running	Halted
PMU	Running	Running	Halted
UIC Program State	Running	Running	Halted (State Retained)
JTAG	Running	Limited Functions[2]	Limited Functions[2]

[1]If a STOP or LPSTOP instruction is executed in contexts 1–4, it is treated as a sleep instruction. Both instructions take a parameter to set to the context-core-status register, which can be useful in critical code sections.
[2]JTAG will not be able read/write registers or memory because this function requires the Execution Engine. This is only a concern when the debugger is connected.

The sections below describe the effects of power-saving modes and how to recover from them.

AutoSleep Power Mode—(All Contexts Sleeping) Effects and Recovery:

- Master Clock is enabled, and all peripherals are clocked (as set by the Clock Mask Register).
- All contexts are in NOT_RDY state.
- Execution Engine is running (receiving clocks). However, because all contexts are NOT_RDY (sleeping), no instructions are being processed.
- CMU is running and will process interrupts.
- An interrupt to any context will exit AutoSleep Power Mode (assuming the interrupt has priority to be processed).
  - The interrupt will move that context from NOT_RDY state to RDY state.
  - The CMU will re-activate the context.

Idle Power Mode—(STOP Instruction executed by Context_0) Effects and Recovery:

- Master Clock is enabled, and all peripherals are clocked (as set by the Clock Mask Register)
- All contexts will change to NOT_RDY (if currently RDY), or will remain halted when the STOP Instruction was executed by Context_0
- Status Register of Context_0 modified by STOP instruction
- Execution Engine stopped (no clock)
- CMU is running and will process interrupts
- An interrupt to any context will exit Idle Mode (assuming the interrupt has priority to be processed)
  - The Execution Engine will start running (receiving clocks)
  - The interrupt will move that context from NOT_RDY state to RDY state
  - The CMU will re-activate the context

Hard Sleep Power Mode—(LPSTOP Instruction executed by Context_0) Effects and Recovery:

- Master Clock is disabled and all peripherals are halted (no clock)

- All contexts retain their current state when the LPSTOP Instruction was executed by Context_0
- Status Register of Context_0 modified by LPSTOP instruction
- Execution Engine stopped (no clock)
- CMU is halted, no interrupts are processed
- UICs will stop on their current instruction (no clock). (They should have been placed into a controlled shutdown state, see Chapter 7, Peripheral Management Unit.)
- Major Reset Recovery
  - Caused by RESET_N pin in conjunction with POR bit set (see Section 10.3.1, Power-On Reset Register)
  - Context_0
    - SR set to 0x2700
    - VBR set to 0x0
    - PC set to Vector 0
    - SSP set to Vector 0
    - Control Reg set to 0x1F02 (Priority 7, RDY state)
  - Contexts 1–4
    - SR set to 0x2700
    - VBR set to 0x0
    - PC not changed
    - SSP not changed
    - Control Reg set to 0x1800 (Priority 0, HALTED state)
  - Context_0 will start running at address defined by Vector 0
    - All other contexts will not run because their state is HALTED
    - Context_0 can examine the POR bit of the Power-On Reset Register to identify the type of reset
  - UICs
    - Full reset on all registers (both POR and reset for the trigger), see Chapter 7, Peripheral Management Unit, for UICs
  - Clock Mask Register is reset
    - Clocks masked from all peripherals
- Minor Reset Recovery
  - Caused by RESET_N pin in conjunction with POR bit clear (see Power-On Reset Register)

- Context_0
  - SR set to 0x2700
  - VBR set to 0x0
  - PC set to Vector 0
  - SSP set to Vector 0
  - Control Reg set to 0x1F02 (Priority 7, READY state)

- Contexts 1–4
  - SR Not Changed
  - VBR Not Changed
  - PC Not Changed
  - SSP Not Changed
  - Control Reg Not Changed

- Context_0 will start running at Address defined by Vector 0
  - All other contexts will be held off, but their state will remain unchanged from when the LPSTOP instruction was run. Once Context_0 goes NOT_RDY, then the next highest priority ready context will start running where it left off before the LPSTOP instruction was executed.
  - Context_0 can examine the POR bit of the Power-On Reset Register to identify the type of reset.

- UICs
  - Minimal Reset on all registers (reset is the only trigger), see Chapter 7, Peripheral Management Unit, for UICs
  - Instruction Counters not reset
  - UIC IC will pick up where it left off when clock reasserted

- The Clock Mask Register is not affected

- Reset via POR (Power-On Reset) is same as Major Reset. POR bit in the Power-On Reset Register is set and WDT bit is cleared.

- Reset via JTAG is same as Major Reset. POR bit in the Power-On Reset Register is set and WDT bit is cleared.

- Reset via Watchdog is same as Major Reset. POR bit in the Power-On Reset Register is unaffected and WDT bit is set.

- Reset via RESET_N pin is either a Major or Minor reset as defined above. POR bit in the Power-On Reset Register is unaffected and WDT bit is cleared.

10.3 Detailed Register Definition

In the above description of power management options, the Power-On Reset Register and Clock Mask Register were discussed. A detailed definition of these two registers as well as their application in managing the power-saving features of fido is provided in this section.

Separately from the power management modes, the UICs and other peripherals can be put to sleep as needed to conserve power. Sleeping and waking UICs (once they have been operational) is procedural, so refer to Chapter 7, Peripheral Management Unit, for those specifics.

Typically, the application design would be defined, and the Master Context would know which peripherals are going to be in use, and would turn on those peripherals early in the power up sequence.

- The Power-On Reset Register is used to determine the nature of a reset (either watchdog timer or power-on reset). Software can also write to this register to control the type of hardware reset processing that is performed.

- The Clock Mask Register is used to reduce power consumption while the system is running by turning off unused peripherals.

10.3.1 Power-On Reset Register

This register is access-controlled and writable by only the Master Context (Context_0). It serves two purposes:

- Indicates to software whether the most recent reset was a cold power-up or a soft watchdog timer reset and to control resets.

- Indicates to hardware whether to perform a full reset or soft reset, as controlled by software.

Table 10-2. Power-On Reset Register

31–2	1	0
Reserved	WDT	
–	RO	RW

Note: Reset value is 0x00000001.

- Bits [31–2]—Reserved

- Bit [1]—Watchdog timeout flag. This flag is set to one by the watchdog circuit if it generates a reset event. It is set to zero if any other type of reset occurs. It is read only to software.
  - 1—Most recent reset event was a WDT timeout
  - 0—Most recent reset event was not a WDT timeout, basically anything else

- Bit [0]—Power On Reset (POR) Flag. This flag is set to one by the power-on reset hardware. It can be read and written by software to force hard reset processing (does not generate a hard reset, only controls hardware initialization).
  - 1—If set to one at reset, then the hardware sets all registers to their default values
  - 0—If set to zero, the hardware does not set registers to default values

Given the POR flag, if a system does not need to distinguish between power-on reset and any other reset, then the software need do nothing with the register; it will always read "1" and all resets will result in registers being put in their power-on reset state.

Since the POR flag is software writable, it allows the user to determine hardware initialization on a WDT reset.

10.3.2 Clock Mask Register

This register is access-controlled and writable by only the Master Context (Context_0). It is the key to power consumption control while the system is running. The Master Context should set this register as desired early in the power-up sequence (see Table 10-3). It has several features:

- Turns off clock to peripherals (TCUs, System Timers, SDRAM, AtoD, PMU, and UICs)
- Re-enables clock to peripherals
- Provides status to indicate that sleep state has been reached
- Commands the peripheral to resume normal operation

Note: Clocks to the timers can be disabled; however, the watchdog timer cannot be affected by this register.

Table 10-3. Clock Mask Register

31–14	13	12	11	10	9	8	7–4	3	2	1	0
Reserved	TCU1	TCU0	System Timers	SDRAM	AtoD	PMU	Reserved	UIC3	UIC2	UIC1	UIC0
–	RW	RW	RW	RW	RW	RW	–	RW	RW	RW	RW

Note: Reset value at POR = 0x3F0F, all peripherals are OFF (all bits are inverted polarity).

- Bits [31–14]—Reserved

- Bit [13]—TCU1 Power control (1 → power off, 0 → power on)

- Bit [12]—TCU0 Power control (1 → power off, 0 → power on)
- Bit [11]—System Timer Power control (1 → power off, 0 → power on)
- Bit [10]—SDRAM Power control (1 → power off, 0 → power on)
- Bit [9]—AtoD Power control (1 → power off, 0 → power on)
- Bit [8]—PMU Power control (1 → power off, 0 → power on)
- Bits [7–4]—Reserved
- Bit [3]—UIC3 Power control (1 → power off, 0 → power on)
- Bit [2]—UIC2 Power control (1 → power off, 0 → power on)
- Bit [1]—UIC1 Power control (1 → power off, 0 → power on)
- Bit [0]—UIC0 Power control (1 → power off, 0 → power on)

Note: The external RESET_N input (non-POR) does not change this register.

11. Access-Controlled Registers

11.1 Overview

One of fido's key features for software developers to be aware of is that many registers are access-controlled or have fields within registers that are access-controlled. This simply means that only the Master Context (Context_0) can write to these registers or to those fields within these registers.

Furthermore, there is one more register, the Configuration Access Control Register, which has a lock bit to prevent even the Master Context from writing to it. This design restriction was implemented for safety critical applications where guarantees have to be made that certain setups after initialization can never be changed. If the user application does not require this feature, the user must merely be aware that access-controlled registers are writable from only the Master Context.

If any context other than Context_0 writes to an access-controlled register, it will fail quietly. The write simply does not take place and no fault or exception is generated. The same is true if Context_0 writes when the lock bit in the Configuration Access Control Register is set. Reads of access-controlled registers can be done from any context (see Chapter 12, Register Map Reference, for a list of access-controlled registers).

11.2 Configuration Access Control Register

This register has a single bit field which controls access to the Access-Controlled registers. When access is enabled (Lock = 0), they are writable only by the Master Context. If access is explicitly disabled (Lock=1) by the Lock bit of this register, then the affected control registers are not writable by any context (see Chapter 12, Register Map Reference). After a reset, the default value of the register has access enabled (see Table 11-1).

Table 11-1. Configuration Access Control Register

31–1	0
reserved	Lock
–	RW

Note: Reset value is 0x00000000.

- Bits [31–1]—Reserved
- Bit [0]—Lock bit
  - 0—Write access is allowed to configuration register space (for Context_0)
  - 1—Write access disabled to configuration register space

This Register is read/written only via the MOVEC instruction. The Configuration Access Control Register is located at address 0xFFE for the MOVEC instruction.

Access-controlled registers can be read by any context at any time. Only the writes are restricted—Master Context only—subject to the Lock bit. The Master Context can set or reset the Lock bit at will, using only the MOVEC instruction.

Although the initialization of access-controlled registers needs to be done by the Master Context during the power-up sequence, the restriction is not difficult.

12. Register Map Reference

Table 12-1. Interrupt 'N' Control Registers (where N=0..7)

31–14	13	12	11	10	9–8	7–5	4–3	2–0
Reserved	Enable	Status	Polarity	Level	Reserved	Priority	Reserved	Context
–	RW	R	RW	RW	–	RW	–	RW

Note: Reset value is 0x00000000.

- Bits [31–14]—Reserved
- Bit [13]—Enable (0=interrupt channel disabled, 1=interrupt channel enabled)
- Bit [12]—Status, read this bit to determine interrupt channel status
  - 0—No interrupt pending
  - 1—Interrupt is pending

Note: Reading this bit acknowledges the interrupt and clears the bit.

- Bit [11]—Polarity of interrupt signal
  - 0—Low level for level sensitive or falling edge for edge sensitive
  - 1—High level for level sensitive or rising edge for edge sensitive
- Bit [10]—Type of interrupt signal
  - 0—Level sensitive
  - 1—Edge sensitive
- Bits [9–8]—Reserved
- Bits [7–5]—Priority of interrupt channel

Note: Sets the priority of the interrupt channel to be assigned, where zero is the lowest and seven is the highest, relative to the SR for the associated context.

- Bits [4–3]—Reserved
- Bits [2–0]—Associated context for this interrupt

Note: This field defines associated context for that interrupt channel. Only this context can be conditionally awakened by this interrupt (based on priority scheme). Context values are 0–4.

Table 12-2. Faulted Context Register

31–8	7–0
Reserved	Context ID
–	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bits [7–0]—Faulted context ID (a value from 0 to 4)

Table 12-3. Context Control Register

Context Control Register						
31–16	15	14–13	12–11	10–8	7–2	1–0
Reserved	Reserved	Interrupt Mode	Reserved	Priority	Reserved	State
–	R	RW	R	RW	–	RW

Note: Reset value for Context_0 is 0x1F02; for other contexts is 0x1800.

- Bits [31–16]—Reserved
- Bit [15]—Reserved, always reads 0
- Bits [14–13]—Defines the type of response the context makes to a pending interrupt
  - 00 → Standard mode. Operates like a standard microprocessor. If interrupt is of higher than or equal priority than the current interrupt mask in context status register, then the current state of the context is stacked and execution is begun at the interrupt vector.
  - 01 → Fast mode. All interrupts are masked while the context is in the Ready state, when a context is in the NOT READY state, an interrupt of priority higher than or equal to the mask in the status register for that context will cause a vector fetch and execution begins. Nothing is stacked.
  - 11 → Fast single-threaded mode. All interrupts are masked while the context is in the Ready state, when a context is in the NOT READY state, an interrupt of priority higher than or equal to the mask in the status register for that context will cause execution to begin at the current program counter location for that interrupt. Nothing is stacked—no vector fetch is performed.
- Bits [12–11] → Reserved, always read back 1 (future expansion for priority)
- Bits [10–8] → Priority of the context (0–7) where 7 is highest, 0 is lowest
- Bits [7–2] → Reserved, always reads 0

- Bits [1–0] → Execution state of the context
  - 00 → HALTED. The context will not execute when in the halted state. Only a write to this register can move a context out of this state.
  - 01 → NOT READY. The context can be made ready by any interrupt/exception targeted to it.
  - 10 → READY. The context is ready to run and will be placed into execution when it is the highest-priority ready context.

Table 12-4. Context Timer Enable Register

31	30–21	20	19	18	17	16	15–5	4	3	2	1	0
ENI	Reserved	Mode4	Mode3	Mode2	Mode1	Mode0	Reserved	EN4	EN3	EN2	EN1	EN0
RW	–	RW	RW	RW	RW	RW	–	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bit [31]—Write of “1” enables the Context Idle Timer, write of 0 disables, a read returns its status.
- Bits [30–21]—Reserved.
- Bit [20]—Context_4 timer mode control, write of 0 specifies standard context timer mode, write of 1 specifies time slice mode, a read returns current mode.
- Bit [19]—Context_3 timer mode control, write of 0 specifies standard context timer mode, write of 1 specifies time slice mode, a read returns current mode.
- Bit [18]—Context_2 timer mode control, write of 0 specifies standard context timer mode, write of 1 specifies time slice mode, a read returns current mode.
- Bit [17]—Context_1 timer mode control, write of 0 specifies standard context timer mode, write of 1 specifies time slice mode, a read returns current mode.
- Bit [16]—Context_0 timer mode control, write of 0 specifies standard context timer mode, write of 1 specifies time-slice mode, a read returns current mode.

Note: Concerning the mode bits [20–16]: When a timer is in standard mode, a context timer > the maximum time will generate a priority 6 overtime fault to the Master Context. When a timer is in time-slice mode, a context timer > the maximum time will fault to the local context which will remain in a RDY state. It is anticipated the fault handler for this context will be responsible for scheduling activity.

- Bits [15–5]—Reserved.
- Bit [4]—Context_4 timer enables, write of 1 enables, write of 0 disables, a read returns current status.
- Bit [3]—Context_3 timer enables, write of 1 enables, write of 0 disables, a read returns current status.
- Bit [2]—Context_2 timer enables, write of 1 enables, write of 0 disables, a read returns current status.
- Bit [1]—Context_1 timer enables, write of 1 enables, write of 0 disables, a read returns current status.
- Bit [0]—Context_0 timer enables, write of 1 enables, write of 0 disables, a read returns current status.

Table 12-5. Context Timer Counter Register

31–24	23–0
Reserved	Count
–	RW

Note: Reset value is 0x00000000.

- Bits [31–24]—Reserved
- Bits [23–0]—Count → When active, the counter is incremented every 16 external clock cycles, for example:
  - 66-MHz base frequency
    - Counter LSB = 240 nsec
    - Counter Period (max) = 4.03 seconds
  - 50-MHz base frequency
    - Counter LSB = 320 nsec
    - Counter Period (max) = 5.36 seconds

Table 12-6. Context Maximum Time Register

31–24	23–9	8–0
Reserved	Max Value	Reserved
–	RW	–

Note: Reset value is 0x00000000.

- Bits [31–24]—Reserved

- Bits [23–9]—Max Value , for example:
  - System clock speed 50 MHz
    - Minimum timeout 163.8 µsec
    - Maximum timeout 5.36 seconds
  - System clock speed 66 MHz
    - Minimum timeout 124.1 µsec
    - Maximum timeout 4.03 seconds
- Bits [8–0] → Reserved, always read back 0

Table 12-7. Context Timer Clear Register

31	30–5	4	3	2	1	0
CLRI	Reserved	Clr4	Clr3	Clr2	Clr1	Clr0
RW	–	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bit [31]—Write of 1 clears the Context Idle Timer, always reads 0
- Bits [30–5]—Reserved
- Bit [4]—Write of 1 clears Context_4 Timer Register, always reads 0
- Bit [3]—Write of 1 clears Context_3 Timer Register, always reads 0
- Bit [2]—Write of 1 clears Context_2 Timer Register, always reads 0
- Bit [1]—Write of 1 clears Context_1 Timer Register, always reads 0
- Bit [0]—Write of 1 clears Context_0 Timer Register, always reads 0

Table 12-8. Context Idle Timer Register

31–24	23–0
Reserved	Count
–	R

Note: Reset value is 0x00000000.

- Bits [31–24]—Reserved
- Bits [23–0]—Count → When active, the counter is incremented every 16 external clock cycles, for example:
  - 66-Mhz base frequency
    - Counter LSB = 240 nsec

    - Counter Period (max) = 4.03 seconds

- 50-Mhz base frequency
    - Counter LSB = 320 nsec
    - Counter Period (max) = 5.36 seconds

Table 12-9. Context Claim Priority Inheritance Register

31–3	2–0
Reserved	Priority
–	R

Table 12-10. Pending Contexts Register

31–5	4	3	2	1	0
Reserved	Ctx4 pend	Ctx3 pend	Ctx2 pend	Ctx1 pend	Reserved
–	R	R	R	R	–

Table 12-11. Claim Register

31–12	11–0
Reserved	Object ID
–	RW

Note: Reset value is 0x00000000.

- Bits [31–12]—Reserved
- Bits [11–0]—Object ID

Table 12-12. Software Interrupt Control Register

31–14	13	12	11–8	7–5	4–0
Reserved	Enable	Status	Reserved	Priority	Reserved
–	RW	R	–	RW	–

Note: Reset value is 0x00000000.

- Bits [31–14]—Reserved
- Bit [13]—Interrupt enable bit for this context (1=enabled, 0=disabled)
- Bit [12]—Interrupt status bit for this context (read this bit to determine status), 1=interrupt pending, a read will clear the bit, acknowledging the interrupt

- Bits [11–8]—Reserved
- Bits [7–5]—This field allows the priority of the interrupt channel to be assigned, 0 is lowest and 7 is highest
- Bits [4–0]—Reserved

Table 12-13. Software Interrupt Actuation Register

31–0
Signature Data
RW

Table 12-14. Relocatable RAM Control Register

31–11	10–1	0
Base Address	Reserved	EN
RW	–	RW

Note: Reset value is 0x00000000.

- Bits [31–11]—Base address of memory block (on a 2-Kbyte boundary) that will be overlaid to RREM.

Note: This base address value must not overlap any fido1100 internal memory (User SRAM, register space, etc.).

- Bits [10–1]—Reserved, set to 0
- Bit [0]—Enable bit for RREM block (0=disabled, 1=enabled)

Note: The Relocatable RAM memory can be loaded like any other until any of the enable flags are set. Once any block is enabled, the only path to access all blocks is through fetching instructions (the memory can only be used for code, not for operand space). Thus all blocks to be used must be set up prior to enabling any block.

Table 12-15. Configuration Access Control Register

31–1	0
Reserved	Lock
–	RW

Note: Reset value is 0x00000000.

- Bits [31–1]—Reserved

- Bit [0]—Lock bit
  - 0 → Write access is allowed to configuration register space (for Master Context only)
  - 1 → Write access disabled to configuration register space

Table 12-16. Memory Base Offset Register

31–20	19–0
Offset	Reserved
RW	–

Note: Reset value is 0x00100000.

- Bits [31–20]—Forms the upper 12 bits of address decode

Note: The value of this field should never be set to 0 as it will cause a conflict with User SRAM to overlay the flash address area.

- Bits [19–0] – Reserved and are set to 0

Table 12-17. MPU Block Control Base Register

31–6	5–0
Base Address	Reserved
RW	–

Note: Reset value is 0x00000000.

- Bits [31–6]—Base Address (addresses bits 31–6 of the protected block)

Note: Must be on a 64-byte boundary

- Bits [5–0]—Reserved

Table 12-18. MPU Block Control Attributes Register

31–7	6–2	1	0
Reserved	Size	RO	EN
–	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–7]—Reserved

- Bits [6–2]—Block size (see Table 5-16)

- Bit [1]—Read-Only Control bit
  - 0 → Block can be written and read
  - 1 → Block is read only

- Bit [0]—Block Enable bit
  - 0 → Block is disabled, has no effect on address space
  - 1 → Block enabled, address rules apply to contexts as defined in CTXx_MPU_Allocation registers

Table 12-19. CTX MPU Allocation Register

31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
–	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved

- Bit [15]—Block 15 application bit
  - 0 → block does not apply to this context
  - 1 → block applies to this context

- Bit [14]—Block 14 application bit
  - 0 → block does not apply to this context
  - 1 → block applies to this context

- Bits [13–0] Block 13 to Block 0 individual application bits
  - 0 → block does not apply to this context
  - 1 → block applies to this context

Table 12-20. External Bus Chip Select Control Register

31–12	11	10	9	8	7–6	5	4	3–0
Base Address	Reserved	Mode Select	Output Enable	SDRAM Enable	Width	Byte Enable	Reserved	Size
RW	–	RW	RW	RW	RW	RW	–	RW

Note: Reset value is CS0 =0x00000205 or 0x00000245 (256 Kbytes, 8/16-bit-wide devices dependent on Size input at reset). CS1–CS7 = 0x00000000.

- Bits [31–12]—Base Address → Specifies the beginning address for the CS block. Must be a multiple of bank size (bank size is specified by bits 3..0)

Note: If the SDRAM Enable bit is 1, then only the upper 12 bits [31–20] are valid, and must match the value set in Bits [11–0] of the appropriate SDRAM External Bank register. The remaining bits of this field [19–12] must be set to "0."

- Bit [11]—Reserved

- Bit [10]—Mode Select (0 → use chip select output, 1 → use address output)

Note: This field only applies to muxed chip-select pins:
CS7 muxed with A27 (hardware signal A27_CS7_N)
CS6 muxed with A28 (hardware signal A28_CS6_N)
CS5 muxed with A29 (hardware signal A29_CS5_N)
CS4 muxed with A30 (hardware signal A30_CS4_N)

- Bit [9]—Output Enable
  - 0—Disable chip select, select upper address bits as outputs
  - 1—Enable chip select, de-select upper address bits as outputs

- Bit [8]—SDRAM Enable
  - 0—Associated memory is not SDRAM.
  - 1—Associated memory is SDRAM.

Note: When this flag is set all other fields (except Base Address, Mode Select, Output Enable, and Size) in this register, as well as the External Bus Chip Select Timing Register, are invalid. This enables the SDRAM Control Registers.

- Bits [7–6]—Width → Specifies the bus width of the attached peripheral. Determines the byte lane of the bus used to access the external device (via the BE_N byte enable signals):
  - 00—8-bit device, all data transferred as bytes on D[7–0]
  - 01—16-bit device, all data transferred as words on D[15–0]. Unaligned word accesses require two bus cycles
  - 10—Reserved
  - 11—Reserved

Note: Size input (sampled at reset) sets the Size field for CS0. This field cannot be written to on CS0. It is latched at reset and cannot be overwritten

- Bit [5]—Byte Enable → Controls byte enable signal behavior when this chip select is active
  - 0—Byte enable signals active for reads and write cycles
  - 1—Byte enable signals become byte write enables (OR-ed with RW_N)

- Bit [4]—Reserved

- Bits [3–0]—Size → Specifies the total range covered by this CS:
  - 0000—8 Kbytes
  - 0001—16 Kbytes
  - 0010—32 Kbytes
  - 0011—64 Kbytes
  - 0100—128 Kbytes
  - 0101—256 Kbytes
  - 0110—512 Kbytes
  - 0111—1 Mbyte
  - 1000—2 Mbytes
  - 1001—2 Mbytes
  - 1010—8 Mbytes
  - 1011—16 Mbytes
  - 1100—32 Mbytes
  - 1101—64 Mbytes
  - 1110—128 Mbytes
  - 1111—256 Mbytes

Table 12-21. Chip Select Timing Register

31–27	26–22	21	20–19	18–16	15–14	13–12	11–10	9–8	7–6	5–4	3–2	1–0
TwWAIT	TrWait	RDY_ENABLE	Reserved	THLD	Reserved	TCS	Reserved	TOE	Reserved	TWEF	Reserved	TWER
RW	RW	RW	–	RW	R	RW	–	RW	–	RW	–	RW

Note: The default settings for the Chip Select Timing Register at POR or external reset are as follows: CS0 loaded to 0x31811031 (suitable for most FLASH ROM; assumes CS0 will be used for RESET vector at address 0x00000000):

TwWAIT—6 clocks
TrWAIT—6 clocks
THLD—1 clock
TCS—1 clock
TOE—0, coincident with CSn_N
TWEF—3 clocks
TWER—1 clock
RDY_ENABLE—0, disable external ready.

CS1–CS7 unaffected by external reset (value retained).
CS1–CS7 set to 0x31811031 as above by POR reset.

- Bits [31–27]—TwWAIT → Depends on RDY_ENABLE (Bit [21])
  - If RDY_ENABLE=0, TwWAIT specifies the width of the chip select active period for the write cycle, allows for 0–31 resulting in a wait time of 1–32 clocks.
  - If RDY_ENABLE=1, TwWAIT specifies the wait time before the RDY_N line is first sampled for the write cycle. This provides a max wait time of 484nS at 66 MHz, anything greater than this will require the external RDY_N line and external logic.

- Bits [26–22]—TrWAIT → Depends on RDY_ENABLE (Bit [21])
  - If RDY_ENABLE=0, specifies the width of the chip select active period for the read cycle, allows for 0–31 resulting in a wait time of 1–32 clocks.
  - If RDY_ENABLE=1, specifies the wait time before the RDY_N line is first sampled for the read cycle. This provides a max wait time of 484nS at 66 MHz. Anything greater than this will require the external RDY_N line and external logic.

- Bit [21]—Ready Enable → Use is described above

- Bits [20–19]—Reserved

- Bits [18–16]—THLD → Specifies the time between when the CSn_N and BEn_N signals go inactive (hi) and the address is removed. Value is 0–7 clocks.

- Bits [15–14]—Reserved

- Bits [13–12]—TCS → Specifies the time between when the address bus is driven and the CSn_N and BEn_N signals go active (low). Value is 0–3 clocks.

- Bits [11–10]—Reserved

- Bits [9–8]—TOE → Specifies the time between when the CSn_N and BEn_N signals go active (low) and the OE signal goes active (low). Value is 0–3 clocks.

- Bits [7–6]—Reserved

- Bits [5–4]—TWEF → Specifies the time between when the CSn_N and BEn_N signals go active (low) and the WE_N signal goes active (low). Value is 0–3 clocks.

- Bits [3–2]—Reserved

- Bits [1–0]—TWER → Specifies the time between when the WE_N signal goes inactive (hi) and the CSn_N and BEn_N signals go inactive (hi). Value is 0–3 clocks.

Table 12-22. External Bus Priority Register

31–3	2–0
Reserved	Priority
Reserved	Priority

Note: Reset value is 0x00000000.

- Bits [31–3]—Reserved
- Bits [2–0]—Priority → The priority the external bus is assigned, 0 is the lowest, 7 the highest.

Table 12-23. SDRAM Timing Parameter 0 Register

31–20	19–16	15	14–12	11–8	7–6	5–4	3–2	1–0
Reserved	TRP	Reserved	TRCD	TRF	Reserved	TWR	Reserved	TCL
–	RW	–	RW	RW	–	RW	–	RW

Note: Reset value is 0x00A22602.

- Bits [31–20]—Reserved
- Bits [19–16]—TRP → Pre-charge cycle time. This parameter specifies the cycles needed by the pre-charge command. That is, the next valid SDRAM command can be issued after the time specified in this parameter.
- Bit [15]—Reserved
- Bits [14–12]—TRCD → RAS-to-CAS delay. This parameter specifies the minimum period between active command and the following read/write command. The maximum allowed. Value is 3.
- Bits [11–8]—TRF → Auto-refresh cycle time. This parameter specifies the time needed by SDRAM to execute auto-refresh command. That is, the next valid SDRAM command can be issued after the time specified in this parameter.

Note: The minimum value for this field is 3. If 1 or 2 is put here, it will cause "double refreshes." Although not harmful, they can be distracting.

- Bits [7–6]—Reserved
- Bits [5–4]—TWR → Write-recovery time. This parameter specifies the period between pre-charge and the last valid write data and the period between the last read data out and write command.

- Bits [3–2]—Reserved
- Bits [1–0]—TCL → CAS-latency. This parameter specifies the time between read command and the first data out. Due to limitation of the read pipeline, the allowed CAS latency is 2 or 3.
  - 00—Illegal
  - 01—Illegal
  - 10—CAS=2
  - 11—CAS=3

Table 12-24. SDRAM Timing Parameter 1 Register

31–24	23–20	19–16	15–0
Reserved	INI_PREC	INI_REFT	REF_INTV
–	RW	RW	RW

Note: Reset value is 0x00480820.

- Bits [31–24]—Reserved
- Bits [23–20]—INI_PREC → Initial pre-charge times. The default value of this field is 4.
- Bits [19–16]—INI_REFT → Initial refresh times. The default value of this field is 8.
- Bits [15–0]—REF_INTV → Refresh interval. One refresh command should be issued if the refresh counter equals the refresh interval.

Table 12-25. SDRAM Configuration 0 Register

31–17	16	15–14	13–12	11	10–8	7–6	5–4	3–0
Reserved	MA2T	Reserved	DDW	Reserved	DSZ	Reserved	MBW	BNKSIZE
–	RW	–	RW	–	RW	–	RW	RW

Note: Reset value is 0x00001226.

- Bits [31–17]—Reserved
- Bit [16]—MA2T → Double Memory Address Cycle Enable. This register is used to control if the address should be validated before the SDRAM command is issued. This bit should always be set to “0” for the fido1100.
- Bits [15–14]—Reserved

- Bits [13–12]—DDW → SDRAM Data Width. This register indicates the data width of each individual SDRAM Module. This indicates the width of a single SDRAM device attached to fido1100.
  – 00—×4 device
  – 01—×8 device
  – 10—×16 device
  – 11—Reserved

- Bit [11]—Reserved

- Bits [10–8]—DSZ → SDRAM Size. This register indicates the size of each individual SDRAM Module (in bits). This indicates the size of a single SDRAM device attached to the fido1100:
  – 000—16 Mbit
  – 001—64 Mbit
  – 010—128 Mbit
  – 011—256 Mbit
  – 100—512 Mbit
  – 101—Reserved
  – 110—Reserved
  – 111—Reserved

- Bits [7–6]—Reserved

- Bits [5–4]—MBW → Memory Bus Width. This register indicates the bus size of external memory bus. This is width of the fido1100 external data bus connected to the SDRAM devices:
  – 00—Memory data width is 8
  – 01—Memory data width is 16
  – 10—Reserved
  – 11—Reserved

- Bits [3–0]—BNKSIZE → Bank Size. The following encoding shows the size of bank (in bytes). Bank sizes other than the following values may cause an unexpected error. This identifies the total size of the memories attached to a single Chip Select:
  – 0000—Reserved
  – 0001—2 Mbyte
  – 0010—4 Mbyte
  – 0011—8 Mbyte
  – 0100—16 Mbyte
  – 0101—32 Mbyte
  – 0110—64 Mbyte
  – 0111—128 Mbyte

- 1000—256 Mbyte
- 1001—Reserved
- 1010—Reserved
- 1011—Reserved
- 1100—Reserved
- 1101—Reserved
- 1110—Reserved
- 1111—Reserved

Table 12-26. SDRAM Configuration 1 Register

31-5	4	3	2	1	0
Reserved	IPREC	IREF	ISMR	PWDN	SREF
–	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–5]—Reserved
- Bit [4]—IPREC → Initial pre-charge start flag. If IPREC is set to “1,” the SDRAM controller will start pre-charging if SDRAM stays in the IDLE state. This flag will be cleared to zero if the executed pre-charge command is equal to the specified initial pre-charge count. Writing 0 to this bit has no affect.
- Bit [3]—IREF → Initial refresh start flag. If IREF is set to “1,” refresh controller will start sending refresh command to control engine until initial refresh time. This flag will be cleared if the executed refresh command is equal to the specified initial refresh time. Writing 0 to this bit has no affect.
- Bit [2]—ISMR → Start set-mode-register. If ISMR is set to “1,” refresh controller will send a set-mode-register command to control engine. This bit will be cleared if set-mode-register command is done. Writing 0 to this bit has no affect.
- Bit [1]—PWDN → Power-down operation mode. If this parameter is set to “1,” SDRAM controller will pull CKE low to suspend the clock while SDRAM controller is in IDLE state. That is, all queued write buffers in SDRAM controller are cleared and all SDRAM banks are pre-charged.
  - The power-down command will cause the controller to clear the CKE line during a no op command. CKE will remain low while in power-down mode.
  - After entering power-down mode, the PWDN bit will remain set until a write to the register clears it.
  - While in power-down mode, the controller will continue to perform refreshes and read/write operations will also be allowed. CKE will go high to perform the operation, and then will go low to put memory back into power-down mode.

- Bit [0]—SREF → Self-refresh mode. If this parameter is set to "1," SDRAM controller will send self-refresh-entry command to SDRAM while controller is in IDLE state (all write buffers queued are cleared and all SDRAM banks are pre-charged). After entering into self-refresh state, this bit will be cleared to 0. Setting this bit to 0 has no effect.
  - The self-refresh command will cause the controller to clear the CKE line during a refresh command. CKE will remain low while in self-refresh mode.
  - While in self-refresh mode, the controller will not make any bus requests nor perform any refresh commands. However data in the SDRAM memory will be retained (the SDRAM memory will be performing internal self-refreshes).
  - Any read or write operation will cause the controller to leave self-refresh mode. CKE will go back high, and the controller will resume refresh operations.

Table 12-27. SDRAM External Bank Configuration Register

31–13	12	11–0
Reserved	BNK_EN	BNK_BASE
–	RW	RW

Note: Reset value is:
SDRAM_Ext_Bank_0 → 0x1800
SDRAM_Ext_Bank_1 → 0x0820
SDRAM_Ext_Bank_2 → 0x0840
SDRAM_Ext_Bank_3 → 0x0860
SDRAM_Ext_Bank_4 → 0x0880
SDRAM_Ext_Bank_5 → 0x08A0
SDRAM_Ext_Bank_6 → 0x08C0
SDRAM_Ext_Bank_7 → 0x08E0
The default values of these registers puts the Bank base address in the "little-endian" memory space.

- Bits [31–13]—Reserved

- Bit [12]—BNK_EN → Bank enable flag.
  - 0—Bank is disabled
  - 1—Bank is enabled

Note 1: External Bank 0 is enabled by default. All others are disabled

Note 2: The SDRAM Enable flag in the associated External Bus Chip Select Register must be set.

- Bits [11–0]—BNK_BASE → 12-bit base address of external bank. This field is equivalent to Bits [31–20] of the fido1100 address bus.

Table 12-28. PMUChxy_Control

31–19	18	17	16	15–13	12–11	10–8	7	6	5	4	3	2	1–0
Reserved	rcv packet size en	uic int en	pmu int en	int priority	Reserved	int context	channel en	xmit start	Reserved	xmt int enable	rcv int enable	error int enable	mode
–	RW	RW	RW	RW	RW	RW	RW	RW	–	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–19]—Reserved

- Bit [18]—Receive Packet Size Enable (rcv packet size en) → Determines whether the size of the received packet is written to the beginning of the packet buffer by hardware.
  - 0—Size not written to buffer, full buffer is used for received content.
  - 1—First long word in buffer is the size of the received packet.

- Bit [17]—UIC Interrupt Enable (uic int en) → Mask to allow UIC to interrupt directly the CPU via the PMU interrupt. The PMU interrupt (in proxy for the UIC) will be fired per the settings of the context and priority interrupt settings.

Note: This field applies only to the primary-channel control registers. It has no affect on secondary-channel registers.

- Bit [16]—PMU Interrupt Enable (pmu int en) → Interrupt enable for PMU Channel. The PMU interrupt will be fired using the settings of the Primary Channel Context and Priority interrupt settings.

Note: The UIC Interrupt Enable is independent of the PMU Interrupt Enable but requires the PMU Interrupt is enabled for operation.

- Bits [15–13]—Interrupt Priority (int priority) → Interrupt priority for the assigned context. Priorities range from zero through seven (000–111).

Note: This field applies only to the primary-channel control registers. It has no affect on secondary-channel registers.

- Bits [12–11] → Reserved

- Bits [10–8]—Interrupt Context (int context) → Assigns the PMU channel and its associated UIC to a CPU context
  - 000—Context_0
  - 001—Context_1
  - 010—Context_2

- 011—Context_3
- 100—Context_4
- 101–111—Reserved

Note: This field applies only to the primary-channel control register. It has no affect on secondary-channel registers.

- Bit [7]—Channel Enable (channel en) → Enables this PMU channel

- Bit [6]—Transmit Start (xmit start) → Starts transmit of data from PMU to UIC
  - 0—No affect
  - 1—Start transmit (if enabled)

Note: Asserting this bit will clear the Transmit Complete (xmit complete) flag in the PMU status register.

- Bit [5]—Reserved

- Bit [4]—Transmit Interrupt Enable (xmit int enable) → Enables PMU Interrupt to fire when PMU has completed transferring data to UIC

- Bit [3]—Receive Interrupt Enable (rcv int enable) → Enables PMU Interrupt to fire when PMU has received a full block of data from the UIC

- Bit [2]—Error Interrupt Enable (error int enable) → Enables PMU Interrupt to fire when UIC indicates an error

- Bits [1–0]—PMU Mode (mode)
  - 00—FIFO Mode → Dual-Port Memory operates as a FIFO (CPU should read/write from Data Registers of PMU)
  - 01—Random Access Mode → Dual-Port Memory is random access (CPU can directly access Dual-Port Memory in PMU)
  - 10–11—Not Used

Table 12-29. PMUChxy_Status

31–11	10	9	8	7	6–4	3	2	1	0
Reserved	rcv FIFO empty	xmit FIFO full	Reserved	interrupt flag	Reserved	Rx overflow	uic error	rcv complete	xmit complete
–	R	R	–	R	–	R	R	R	R

Note: Reset value is 0x00000400.

- Bits [31–11]—Reserved

- Bit [10]—Receive FIFO Empty (rcv FIFO empty) → Indicates that the receive FIFO buffer is empty (PMUChxy_RcvFBufRdPtr equals PMUChxy_RcvFBufWrPtr)
- Bit [9]—Transmit FIFO Full (xmit FIFO full) → Indicates that the transmit FIFO buffer has no space available (PMUChxy_XmitFBufRdPtr is one location greater than PMUChxy_XmitFBufWrPtr)
- Bit [8]—Reserved
- Bit [7]—PMU Interrupt Flag (interrupt flag) → Indicates the PMU has fired an interrupt (cleared on read)

Note: Serves as interrupt acknowledge for PMU-generated interrupt and must read UIC status registers first to clear interrupt conditions originating there.

- Bits [6–4]—Reserved
- Bit [3]—Receive Overflow (rx overflow) → Write Frame Buffer pointer has passed the Read Frame Buffer pointer for the receive FIFO
- Bit [2]—UIC Error (uic error) → Indicates an error has occurred in the UIC (cleared on read)
- Bit [1]—Receive Complete (rcv complete) → The PMU has received a packet of data from the UIC (cleared on read)
- Bit [0]—Transmit Complete (xmit complete) → The PMU has completed transmitting data to the UIC (cleared on read and on Transmit Start [xmit start])

Table 12-30. PMUChxy_PckXmitSize

31–12	11–0
Reserved	tx_size
–	RW

Note: Reset value is 0x00000000.

- Bits [31–12]—Reserved
- Bits [11–0]—Transmit Size (tx_size) → Size in bytes

Table 12-31. PDMAChxy_PckRcvSize

31–13	12–0	
Reserved	rx_size	0
–	RW	R

Note: Reset value is 0x00000000.

- Bits [31–13]—Reserved
- Bits [12–0]—Receive Size (rx_size) → Size in bytes (rounded up to next even value)

Table 12-32. PMUChxy_RcvFBufStart

31–13	12–0		
Reserved	rcv_fb_start	0	0
–	RW	R	R

Note: Reset value is 0x00000000.

- Bits [31–13]—Reserved
- Bits [12–0]—Receive Frame Buffer Start (rcv_fb_start) → Start Address of the Receive Frame Buffer for this channel based on a long-word (32-bit) boundary.

Table 12-33. PMUChxy_RcvFBufEnd

31–13	12–0		
Reserved	rcv_fb_end	1	1
–	RW	R	R

Note: Reset value is 0x00000011.

- Bits [31–13]—Reserved
- Bits [12–0]—Receive Frame Buffer End (rcv_fb_end) → End Address of the Receive Frame Buffer for this channel based on a long-word (32-bit) boundary.

Table 12-34. PMUChxy_XmitFBufStart

31–12	11–0		
Reserved	xmt_fb_start	0	0
–	RW	R	R

Note: Reset value is 0x00000000.

- Bits [31–12]—Reserved

- Bits [11–0]—Transmit Frame Buffer Start (xmt_fb_start) → Start Address of the Transmit Frame Buffer for this channel based on a long-word (32-bit) boundary.

Table 12-35. PMUChxy_XmitFBufEnd

31–12	11–0		
Reserved	xmt_fb_end	1	1
–	RW	R	R

Note: Reset value is 0x00000011.

- Bits [31–12]—Reserved

- Bits [11–0]—Transmit Frame Buffer End (xmt_fb_end) → End address of the Transmit Frame Buffer for this channel based on a long word (32 bit) boundary.

Table 12-36. PMUChxy_RcvFBufWrPtr

31–13	12–0	
Reserved	rcv_fb_wr_ptr	0
–	RW	R

Note: Reset value is 0x00000000.

- Bits [31–13]—Reserved

- Bits [12–0]—Receive Frame Buffer Write Pointer (rcv_fb_wr_ptr) → Current write address of the Receive Frame Buffer for this channel based on a word-aligned (16-bit) boundary.

Table 12-37. PMUChxy_RcvFBufRdPtr

31–13	12–0		
Reserved	rcv_fb_rd_ptr	0	0
–	RW	R	R

Note: Reset value is 0x00000000

- Bits [31–13]—Reserved

- Bits [12–0]—rcv_fb_rd_ptr → Address in memory of rcv frame buffer (FIFO) for reading data from buffer (FIFO) (CPU reading received data inserted by UIC), value in bytes, long-word aligned

Table 12-38. PMUChxy_XmitFBufRdPtr

31–12	11–0	
Reserved	xmt_fb_rd_ptr	0
–	RW	R

Note: Reset value is 0x00000000.

- Bits [31–12]—Reserved
- Bits [11–0]—Transmit Frame Buffer Read Pointer (xmt_fb_rd_ptr) → Current read address of the Transmit Frame Buffer for this channel based on a word-aligned (16-bit) boundary.

Table 12-39. PMUChxy_XmitFBufWrPtr

31–12	11–0		
Reserved	xmt_fb_wr_ptr	0	0
–	RW	R	R

Note: Reset value is 0x00000000.

- Bits [31–13]—Reserved
- Bits [11–0]—Transmit Frame Buffer Write Pointer (xmt_fb_wr_ptr) → Current write address of the Transmit Frame Buffer for this channel based on a long-word-aligned (32-bit) boundary.

Table 12-40. PMUChxy_Xmit_Data

31–0
xmit data FIFO head
W

Note: Reset value is 0x00000000.

- Bits [31–0]—32-bit long word from PMU Tx FIFO.

Note: Each read by the UIC from the PMU will increment the Transmit Frame Buffer Read Pointer (PMU_Chxy_XmitFBufRdPtr) by two bytes, indicating 16 bits have been read.

Table 12-41. PMUChxy_Rcv_Data

31–0
rcv data FIFO head
R

Note: Reset value is 0x00000000.

- Bits [31–0]—32-bit long word from PMU Rx FIFO.

Note: Each write by the UIC to the PMU will increment the Receive Frame Buffer Write Pointer (PMUChxy_RcvFBufWrPtr) by two bytes, indicating 16 bits have been written.

Table 12-42. GPIO_DIR_A

31–8	7	6	5	4	3	2	1	0
Reserved	GPIO7 Dir	GPIO6 Dir	GPIO5 Dir	GPIO4 Dir	GPIO3 Dir	GPIO2 Dir	GPIO1 Dir	GPIO0 Dir
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x000000FF.

- Bits [31–8]—Reserved
- Bit [7]—GPIO7 Dir
  - 0—Pin set as output
  - 1—Pin set as input
- Bit [6]—GPIO6 Dir
  - 0—Pin set as output
  - 1—Pin set as input
- Bit [5]—GPIO5 Dir
  - 0—Pin set as output
  - 1—Pin set as input
- Bit [4]—GPIO4 Dir
  - 0—Pin set as output
  - 1—Pin set as input
- Bit [3]—GPIO3 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [2]—GPIO2 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [1]—GPIO1 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [0]—GPIO0 Dir
  - 0—Pin set as output
  - 1—Pin set as input

Table 12-43. GPIO_DIR_B

31–8	7	6	5	4	3	2	1	0
Reserved	GPIO15 Dir	GPIO14 Dir	GPIO13 Dir	GPIO12 Dir	GPIO11 Dir	GPIO10 Dir	GPIO9 Dir	GPIO8 Dir
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x000000FF.

- Bits [31–8]—Reserved

- Bit [7]—GPIO15 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [6]—GPIO14 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [5]—GPIO13 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [4]—GPIO12 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [3]—GPIO11 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [2]—GPIO10 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [1]—GPIO09 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [0]—GPIO08 Dir
  - 0—Pin set as output
  - 1—Pin set as input

Table 12-44. PIO_DIR_C

31–5	4–2			1	0
Reserved	UICOP			GPI017 Dir	GPI016 Dir
–	RW	R	R	RW	RW

Note: Reset value is 0x03.

- Bits [31–5]—Reserved

- Bits [4–2]—UICOP (UIC other purposes)

- Bit [1]—GPIO17 Dir
  - 0—Pin set as output
  - 1—Pin set as input

- Bit [0]—GPIO16 Dir
  - 0—Pin set as output
  - 1—Pin set as input

Note: To avoid indeterminate behavior, a read/modify/write operation is recommended for Port C registers (i.e., read the register, modify the value using OR/AND, then write the modified value to the register).

Table 12-45. GPIO_INV_A

31–8	7	6	5	4	3	2	1	0
Reserved	GPIO7 Invert	GPIO6 Invert	GPIO5 Invert	GPIO4 Invert	GPIO3 Invert	GPIO2 Invert	GPIO1 Invert	GPIO0 Invert
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved

- Bit [7]—GPIO7 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [6]—GPIO6 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [5]—GPIO5 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [4]—GPIO4 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [3]—GPIO3 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [2]—GPIO2 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [1]—GPIO1 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

- Bit [0]—GPIO0 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

Table 12-46. GPIO_INV_B

31—8	7	6	5	4	3	2	1	0
Reserved	GPIO15 Invert	GPIO14 Invert	GPIO13 Invert	GPIO12 Invert	GPIO11 Invert	GPIO10 Invert	GPIO9 Invert	GPIO8 Invert
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved

- Bit [7]—GPIO15 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted
- Bit [6]—GPIO14 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted
- Bit [5]—GPIO13 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted
- Bit [4]—GPIO12 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted
- Bit [3]—GPIO11 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted
- Bit [2]—GPIO10 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted
- Bit [1]—GPIO9 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted
- Bit [0]—GPIO8 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

Note: The lower two bits of this register operate as GPIO Inversion bits. The remaining bits have other purposes (Port C has only two GPIO pins).

Table 12-47. GPIO_INV_C

31–8	7–2	1	0
Reserved	UICOP	GPIO17 Invert	GPIO16 Invert
–	R	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bits [7–2]—UICOP (UIC other purposes)
- Bit [1]—GPIO17 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted
- Bit [0]—GPIO16 Invert
  - 0—Pin set as not inverted
  - 1—Pin set as inverted

Table 12-48. GPIO_Data_A

31–8	7	6	5	4	3	2	1	0
Reserved	GPIO7 Data	GPIO6 Data	GPIO5 Data	GPIO4 Data	GPIO3 Data	GPIO2 Data	GPIO1 Data	GPIO0 Data
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bits [31–8]—Reserved
- Bit [7]—GPIO7 Pin Data
- Bit [6]—GPIO6 Pin Data
- Bit [5]—GPIO5 Pin Data
- Bit [4]—GPIO4 Pin Data
- Bit [3]—GPIO3 Pin Data
- Bit [2]—GPIO2 Pin Data
- Bit [1]—GPIO1 Pin Data
- Bit [0]—GPIO0 Pin Data

Table 12-49. GPIO_Data_B

31–8	7	6	5	4	3	2	1	0
Reserved	GPIO15 Data	GPIO14 Data	GPIO13 Data	GPIO12 Data	GPIO11 Data	GPIO10 Data	GPIO9 Data	GPIO8 Data
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bit [31–8]—Reserved
- Bit [7]—GPIO15 Pin Data
- Bit [6]—GPIO14 Pin Data
- Bit [5]—GPIO13 Pin Data
- Bit [4]—GPIO12 Pin Data
- Bit [3]—GPIO11 Pin Data
- Bit [2]—GPIO10 Pin Data
- Bit [1]—GPIO9 Pin Data
- Bit [0]—GPIO8 Pin Data

Note: The lower two bits of this register operate as GPIO Data bits. The remaining bits have other purposes (Port C has only two GPIO pins).

Table 12-50. GPIO_Data_C

31–8	7–2	1	0
Reserved	UICOP	GPIO17 Data	GPIO16 Data
–	R	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bits [7–2]—UICOP (UIC other purposes)
- Bit [1]—GPIO17 Pin Data
- Bit [0]—GPIO16 Pin Data

Note: If using the GPIO pins with pull-up resistors (i.e., open collector or open drain interfaces), set the following configuration using the registers associated with the desired GPIO pin. Set the inversion to OFF (clear bit in GPIO_INV_x), the data to zero (clear bit in GPIO_DATA_x), and the direction to output (set bit in GPIO_DIR_x).

Table 12-51. ConfigRegA—Program Control

31–8	7	6	5	4–3	2	1	0
Reserved	TXE	TXEN	TXNB	THC	SSR	PGC	PGM
–	RW	RW	RW	RW	R	R	RW

Note: Reset value is 0x00000001.

- Bits [31–8]—Reserved

- Bit [7]—Transmit/Receive Register (TxReg/RxReg) Endian formats (TXE)
  - 0—TxReg/RxReg is set as little-Endian and does not reverse bit order on a byte (reset value)
  - 1—TxReg/RxReg are big-Endian and reverses bit order on a byte

- Bit [6]—Transmit/Receive Register (TxReg/RxReg) Enables (TXEN)
  - 0—UIC controls the TxReg/RxReg enable internally (reset value)
  - 1—TxReg enable is controlled by an external pin (set in UIC ConfigRegC)

- Bit [5]—Transmit/Receive Register (TxReg/RxReg) Nibble Mode (TXNB)
  - 0—TxReg/RxReg transmits and receives one bit at a time (reset value)
  - 1—TxReg/RxReg transmits and receives one nibble at a time

- Bits [4–3]—UIC Thread Count (THC)
  - 11—Four threads, program memory is segmented into four banks
  - 10—Reserved
  - 01—Two threads with program memory segmented into two banks using threads one and four only
  - 00—One thread with program memory non segmented using thread one only

- Bit [2]—UIC is executing (SSR) → Set by UIC Configuration Manager hardware
  - 0—UIC is idle (reset value)
  - 1—UIC is executing programs after being programmed and CPU has cleared Program Mode (bit zero)

- Bit [1]—Programming Complete (PGC) → Set by UIC Configuration Manager hardware
  - 0—UIC program has not been loaded (reset value)
  - 1—UIC program and configuration have been loaded

- Bit [0]—Program Mode (PGM)
  - 0—Normal Operation
  - 1—Programming Mode (Reset value)

Table 12-52. ConfigRegD—Master Control Register

31–8	7	6	5	4–3	2	1	0
Reserved	HDLC	ALT	Edge	CRC Size	NRZI	RMAS	PYJO
–	RW	RW	RW	RW	RW	R	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved

- Bit [7]—High Level Data Link Mode (HDLC) (when using UIC HDLC firmware)
  - 0—Bit Stuffing for Transmit (TxReg) and Receive (RxReg) registers not used
  - 1—Transmit (TxReg) and Receive (RxReg) registers use bit stuffing, recognizing 0x01111110 as a frame delimiter, seven or more “1”s as an abort sequence.

- Bit [6]—Alternate Transmit/Receive Register (TxReg/RxReg) pin mapping (ALT). Bit is exclusive with UIC Configuration Register C Bit [0]
  - 0—UIC Pins 0 and 1 will be tied to GPIO_DATA_A register Bit [0] and Bit [1], respectively
  - 1—UIC Pins 0 and 1 will be tied to Transmit (TxReg) and Receive (RxReg) registers, respectively (primarily used for HDLC mapping)

- Bit [5]—Clock Edge Setting (Edge)
  - 0—Transmit (TxReg) and Receive (RxReg) registers use rising edge of clock
  - 1—Transmit (TxReg) and Receive (RxReg) registers use falling edge of clock

- Bits [4–3]—CRC Size Setting (CRC Size)
  - 00—CRC uses 8 bits
  - 01—CRC uses 16 bits
  - 10—CRC uses 32 bits
  - 11—Reserved

- Bit [2]—Non Return To Zero Inverted mode (NRZI)
  - 0—Transmit (TxReg) and Receive (RxReg) registers use straight encoding
  - 1—Transmit (TxReg) and Receive (RxReg) registers use a zero representation as a change in wire state, and a data 1 as no change

- Bit [1]—Read Me A Story (RMAS)
  - 0—UIC still preparing to enter sleep mode after PYJO = 1
  - 1—UIC has entered sleep mode

- Bit [0]—Put Your Jammies On (PYJO) (Initiate UIC sleep mode)
  - 0—UIC is in RUN mode
  - 1—Command UIC to enter sleep mode

Table 12-53. ConfigRegK—UIC Firmware ID

31–8	7–4	3–0
Reserved	ID	Version
–	R	R

- Bits [31–8]—Reserved
- Bits [7–4]—ID (see Table 7-27)
- Bits [3–0]—Versions can range from 0x0 to 0xF

Table 12-54. ConfigRegL—UIC Programming Checksum

31–8	7–0
Reserved	Checksum
–	R

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bits [7–0]—Checksum value

Table 12-55. INTERRUPT_STATUS_REG

31–8	7	6	5	4	3	2	1	0
Reserved	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bits [7–0]—UIC definable interrupt flags.

Table 12-56. INTERRUPT_MASK_REG

31–8	7–0							
Reserved	INT MASK7	INT MASK6	INT MASK5	INT MASK4	INT MASK3	INT MASK2	INT MASK1	INT MASK0
–	RW							

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bits [7–0]—Masks bits for each UIC definable interrupt flag.

Table 12-57. DMAChx_Control

31	30	29	28	27	26–25	24	23–22	21	20–17	16	15–13	12–11	10–8	7	6–4	3–2	1–0
EN	EOP	MPU Fault	Bus Fault	SDRAM Dest	Dest Addr Control	SDRAM Src	Source Addr Control	PMU Dir	Transfer Control	Ack Enable	Xfer Priority	Reserved	Context	Int En	Int Priority	Burst Size	Xfer Size
RW	R	R	RW	RW	RW	RW	RW	RW	RW	RW	RW	–	RW	RW	R/W	RW	RW

Note: Reset value is 0.00000000.

- Bit [31]—DMA Channel Enable (EN)
  - 0—DMA channel is disabled
  - 1—DMA channel is enabled
- Bit [30]—End of process (EOP) → Set when DMA has completed transferring data and indicates an interrupt has been generated by the DMA (cleared on read)
- Bit [29]—Memory Protection Fault (MPU) Fault → Set if an MPU Fault occurs during data transfer (cleared on read)
- Bit [28]—Bus Fault → Set if a Bus Fault occurs during transfer (cleared on read)
- Bit [27]—SDRAM Destination (SDRAM Dest)
  - 0—DMA destination is not in SDRAM
  - 1—DMA destination is in SDRAM
- Bits [26–25]—DMA Destination Address Control (Dest Addr Control)
  - 00—Increment destination address on transfer
  - 01—Decrement destination address on transfer
  - 10—Hold destination address on transfer
  - 11—PMU Transmit FIFO write → Enables bypass mode transfers to the PMU Transmit FIFO Head and holds destination address on transfer

Note 1: For PMU Transmit FIFO writes in bypass mode, the DMA Destination Address Register (DMAChx_Destination) holds a value from zero to seven specifying which PMU Transmit Buffer FIFO will be written. Only the lower three bits of the Destination Address register will be evaluated when PMU Transmit FIFO write mode is selected.

Note 2: The source address for PMU Transmit FIFO writes in bypass mode must be in one of the following fido1100 internal memory spaces:
- *Internal URAM*
- *PMU Receive FIFO Head*
- *PMU Receive Buffer in Random Access mode*

- Bit [24]—SDRAM Source (SDRAM Src)
  - 0—DMA destination is not in SDRAM
  - 1—DMA destination is in SDRAM

- Bits [23–22]—Source Address Control (Source Addr Control)
  - 00—Increment source address on transfer
  - 01—Decrement source address on transfer
  - 10—Hold source address on transfer
  - 11—PMU Receive FIFO Read → Enables bypass mode transfers to the PMU Transmit FIFO Head and holds source address on transfer

Note 1: For PMU Receive FIFO reads in bypass mode, the DMA Source Address Register (DMAChx_Source) holds a value from zero to seven specifying which PMU Receive buffer FIFO will be read. Only the lower 3 bits of the Source Address register will be evaluated when PMU Receive FIFO Read mode is selected.

Note 2: The destination address for PMU Receive FIFO reads in bypass mode must be in one of the following fido1100 internal memory spaces:
- *Internal URAM*
- *PMU Transmit FIFO Head*
- *PMU Transmit Buffer in Random Access mode*

- Bit [21]—PMU Direction (PMU Dir) → Determines which PMU status bit of the selected PMU channel (set with the Transfer Control bits) to monitor for PMU FIFO data transfers to and from the DMA. This only applies if the Transfer Control bits have been set to binary 1000 through 1111.
  - 0—Read from PMU, transfer occurs when the PMU Receive FIFO Empty (Recv FIFO Empty) flag is zero in the PMU Channel Status (PMUChxy_Status) register
  - 1—Write to PMU, transfer occurs when the PMU Transmit FIFO Full (Xmit FIFO Full) flag is zero in the PMU Channel Status (PMUChxy_Status) register

- Bits [20–17]—DMA Transfer Control (Transfer Control)
  - 0000—Transfer when ready (Trigger → data transfer runs to completion)
  - 0001—Timer Counter 0 (Trigger → data transfer runs to completion)
  - 0010—Timer Counter 1 (Trigger → data transfer runs to completion)
  - 0011—Reserved

- 0100—Reserved
- 0101—Reserved
- 0110—Reserved
- 0111—External DMA Request (Throttle → data transfer only occurs when active)
- 1000—PMU Channel 0A ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)
- 1001—PMU Channel 0B ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)
- 1010—PMU Channel 1A ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)
- 1011—PMU Channel 1B ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)
- 1100—PMU Channel 2A ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)
- 1101—PMU Channel 2B ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)
- 1110—PMU Channel 3A ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)
- 1111—PMU Channel 3B ready Direction set by PMU Dir bit (Throttle → data transfer only occurs when active)

Note: The use of the term "Throttle" refers to the speed at which data is transferred by the DMA. Depending upon the amount of empty space present in memory (e.g., PMU FIFO buffer), the DMA will transfer at a rate until the memory is full at which point data transfers will stop (throttle down). As soon as memory becomes available, the DMA will again start to transfer data (throttle up).

- Bit [16]—Acknowledge Enable (ACK Enable) → When set, enables DMA Acknowledge/Interrupt (ACK/INT) muxed pin as DMA ACK output.

- Bits [15–13]—DMA Transfer Priority (Xfer Priority) → Sets the priority of DMA transfers with respect to CPU context priorities. Priority levels range between zero and seven (000–111). Only writable through the master context (Context_0)

- Bits [12–11]—Reserved

- Bits [10–8]—DMA Context Assignment (Context) → Sets the context the DMA interrupt is associated with. Only writable through the master context (Context_0)
  - 000..100—Context zero through four
  - 101..111—Reserved

- Bit [7]—Interrupt Enable (Int En) → Enables the DMA interrupt when the DMA Count (DMAChx_Count) register reaches zero
  - 0—DMA interrupt is not enabled
  - 1—DMA interrupt is enabled

- Bits [6–4]—Interrupt Priority (Int Priority) → Interrupt priority level for the assigned context. Priority levels range between zero and seven (000–111)

- Bits [3–2]—DMA transfer data burst size (Burst Size) → Number of transfers DMA should do before allowing a CPU access to occur
  - 00—Single transfer
  - 01—4 transfer burst
  - 10—32 transfer burst
  - 11—128 transfer burst

- Bits [1–0]—DMA Transfer Size (Xfer Size) → Set the data size of each data transfer
  - 00—Byte
  - 01—Word (16-bits)
  - 10—Long word (32-bits)
  - 11—Reserved

Table 12-58. DMAChx_Source

31–0
Source address
RW

Note: Reset value is 0x00000000.

- Bits [31–0]—Source address for DMA data or, in bypass mode, a number 0–7 indicating which PMU FIFO to read from.

Table 12-59. DMAChx_Destination

31–0
Destination address
RW

Note: Reset value is 0x00000000.

- Bits [31–0]—Destination address for DMA data or, in bypass mode, a number 0–7 indicating which PMU FIFO to read from.

Table 12-60. DMAChx_Count

31–0
count
RW

Note: Reset value is 0x00000000.

- Bits [31–0]—DMA transfer count.

Note: This is not a byte count. This register holds the number of transfers per Bits [0–1] of the DMAChx_Control register.

Table 12-61. MAC Filter Mode Configuration Register

31–25	24	23	22–15	14–13	12–11	10–9	8–7	6–4	3–1	0
Reserved	Reset read pointer in MAC filter RAM	Reset write pointer in MAC filter RAM	Reserved	UIC_3_ mode	UIC_2_ mode	UIC_1_ mode	UIC_0_ mode	P1 Mode	P0 Mode	Partition
–	RW	RW	–	RW	RW	RW	RW	RW	RW	RW

Note: Reset address is 0x00000000.

- Bits [31–25]—Reserved
- Bit [24]—Reset read pointer in MAC filter RAM → When set, resets the read pointer to the beginning of the table (cleared automatically).
- Bit [23]—Reset write pointer in MAC filter RAM → When set, resets the write pointer to the beginning of the table (cleared automatically).
- Bits [22–15]—Reserved
- Bits [14–13]—UIC_3_mode → Sets the MAC filtering mode for UIC 3
  - 00—No Filtering
  - 01—Reserved
  - 10—MAC Filtering uses HASH Table Partition 0
  - 11—MAC Filtering uses HASH Table Partition 1
- Bits [12–11]—UIC_2_mode → Sets the MAC filtering mode for UIC 2
  - 00—No Filtering
  - 01—Reserved
  - 10—MAC Filtering uses HASH Table Partition 0
  - 11—MAC Filtering uses HASH Table Partition 1

- Bits [10–9]—UIC_1_mode → Sets the MAC filtering mode for UIC 1
  - 00—No Filtering
  - 01—Reserved
  - 10—MAC Filtering uses HASH Table Partition 0
  - 11—MAC Filtering uses HASH Table Partition 1

- Bits [8–7]—UIC_0_mode → Sets the MAC filtering mode for UIC 0
  - 00—No Filtering
  - 01—Reserved
  - 10—MAC Filtering uses Table Partition 0
  - 11—MAC Filtering uses Table Partition 1

- Bits [6–4]—Partition 1 mode → Sets the mode for how RAM Partition Table 1 is used
  - 000—Destination address filtering mode
  - 001—Destination address filtering with broadcast override (broadcast messages are not filtered) mode
  - 010—Accept all broadcast messages mode
  - 011—Source address filtering mode
  - 100—Hash filtering mode
  - 101—Reserved
  - 110—Reserved
  - 111—Reserved

- Bits [3–1]—Partition 0 mode → Sets the mode for how RAM Partition Table 0 is used
  - 000—Destination address filtering mode
  - 001—Destination address with broadcast override (broadcast messages are not filtered) mode
  - 010—Accept all broadcast messages mode
  - 011—Source address filtering mode
  - 100—Hash filtering mode
  - 101—Reserved
  - 110—Reserved
  - 111—Reserved

- Bit [0]—Partition table setup
  - 0—Single partition → A single 256 entry partition shared by all UICs
  - 1—Dual partition → Two independent 128 entry partition tables

Table 12-62. MAC_Filter_Data_Write

31–16	15–0
Reserved	Write data
–	W

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved
- Bits [15–0]—MAC filter data. This data is written 16 bits at a time. Three writes are required to set a single 48-bit filter location.

Table 12-63. MAC_Filter_Data_Read

31–16	15–0
Reserved	Read data
–	R

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved
- Bits [15–0]—MAC filter data. This data reads 16 bits at a time. Three reads are required to set a single 48-bit filter location.

Table 12-64. System Timer Control Register

31–5	4–1	0
Reserved	Bit Rate Select	En
–	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–5]—Reserved
- Bits [4–1]—Bit Rate Select → Selects which bit of the System Timer feeds the five-bit System Interrupt Counter
  - 0000—Bit [0] (Divides by 2)
  - 0001—Bit [1] (Divides by 4)
  - 0010—Bit [2] (Divides by 8)
  - 0011—Bit [3] (Divides by 16)
  - 0100—Bit [4] (Divides by 32)
  - 0101—Bit [5] (Divides by 64)
  - 0110—Bit [6] (Divides by 128)
  - 0111—Bit [7] (Divides by 256)
  - 1000—Bit [8] (Divides by 512)

  - 1001—Bit [9] (Divides by 1024)
  - 1010–1111—Reserved

- Bit [0]—System Timer Enable (En) → Enables the System Timer circuitry
  - 0—System Timer disabled
  - 1—System Timer enabled

Table 12-65. System Timer Prescale Register

31–16	15–0
Reserved	Prescale Value
–	RW

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved

- Bits [15–0]—Prescale Value → Reloads value when Prescale counter reaches zero
  - Equation 8-3 calculates the prescale value for a desired System Timer Interrupt_0 period.

$$Prescale = \frac{(Desired\ Period\ in\ Seconds)\,(System\ Clock)}{2\,(Bitrate)}$$ **Equation 12-1**

  - Equation 8-4 presents an example of a System Timer Interrupt_0 with a one second period using a system clock of 12 MHz and a bit rate select value of 1024.

$$Prescale = \frac{1 \times 12{,}000{,}000}{2 \times 1024} = 5859 = 0x16E3$$ **Equation 12-2**

Table 12-66. System Timer Interrupt Control Register

31–12	11	10	9	8	7–6	5–3	2–0
Reserved	Clear CT	Mode	Int Flag	Enable	Reserved	Context	Priority
–	RW	RW	RW	RW	–	RW	RW

Note: Reset value is 0x00000000. x = 0 to 4.

- Bits [31–12]—Reserved

- Bit [11]—Clear Context Timers (Clear CT) → Determines if the interrupt clears the context timer registers.

- Bit [10]—System Timer Interrupt Mode (Mode) → Determines how the associated System Timer interrupt is handled
  - 0—System timer interrupt acts as a standard interrupt request to the assigned context

- 1—Sets the state of the specified context from not ready to ready. If the context is halted or already ready, then this interrupt has no effect.

- Bit [9]—Interrupt Flag (Int Flag) → If this bit is set, it indicates the System Timer interrupt has occurred (cleared on read).

- Bit [8]—Enable → Enables the interrupt
  - 0—Disables the interrupt
  - 1—Enables the interrupt

- Bits [7–6]—Reserved

- Bits [5–3]—Sets which context the System Timer interrupt is associated with
  - 000—Context_0
  - 001—Context_1
  - 010—Context_2
  - 011—Context_3
  - 100—Context_4
  - 101–111—Reserved

- Bits [2–0]—Interrupt Priority (Priority) → Interrupt priority for the assigned interrupt. Interrupt priority range is zero through seven (000–111), where seven is the highest priority.

Table 12-67. Watchdog Timer Control Register

31–3	2	1	0
Reserved	Int En	En	Reserved
–	RW	RW	–

Note: Reset value is 0x00000000.

- Bits [31–3]—Reserved

- Bit [2]—Interrupt Enable (Int En)
  - 0—Disables the WDT warning interrupt
  - 1—Enables the WDT warning interrupt

- Bit [1]—Enable (En)
  - 0—WDT is disabled
  - 1—Enables the WDT. Once enabled, the WDT can only be disabled by a power on reset (POR).

- Bit [0]—Reserved

Note: This register is only writeable by the Master Context (Context_0) when the lock bit in the Configuration Access Control Register is deasserted. It is readable by any context.

Table 12-68. Watchdog Timer Reload Register

31–16	15–0				
Reserved	Reload Count	1	1	1	1
–	RW	R	R	R	R

Note: Reset value is 0x0000000F.

- Bits [31–16]—Reserved
- Bits [15–0]—Reload Count → Countdown value for the Watchdog Timer counter. The least significant 4 bits of the Reload Count are forced to be ones.

Notes: This register is writeable by only the Master Context (Context_0) when the lock bit in the Configuration Access Control Register is deasserted. It is readable by any context.

The Watchdog Timer Reload Register can be written only if the Enable bit (En) of the Watchdog Timer Control Register is zero.

A Minor Reset reloads the Watchdog Timer but does not disable it.

Table 12-69. TCUx_Status

31–16	15	14	13	12	11	10	9	8	7	6–0
Reserved	CH0_IC	CH1_IC	CH2_IC	CH3_IC	CH0_OC	CH1_OC	CH2_OC	CH3_OC	TE	Reserved
–	RW	RW	RW	RW	RW	RW	RW	RW	RW	–

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved
- Bit [15]—Channel 0 Input Capture Flag (CH0_IC) → When set, indicates an event occurred and the value from the Timer Counter register has been captured to the TCU Channel 0 Input Capture register.
- Bit [14]—Channel 1 Input Capture Flag (CH1_IC) → When set, indicates an event occurred and the value from the Timer Counter register has been captured to the TCU Channel 1 Input Capture register.

- Bit [13]—Channel 2 Input Capture Flag (CH2_IC) → When set, indicates an event occurred and the value from the Timer Counter register has been captured to the TCU Channel 2 Input Capture register.
- Bit [12]—Channel 3 Input Capture Flag (CH3_IC) → Is set when the value in the TCU Channel 3 Output Compare register is equal to the value of the Timer Counter register.
- Bit [11]—Channel 0 Output Compare Flag (CH0_OC) → Is set when the value in the TCU Channel 0 Output Compare register is equal to the value of the Timer Counter register.
- Bit [10]—Channel 1 Output Compare Flag (CH1_OC) → Is set when the value in the TCU Channel 1 Output Compare register is equal to the value of the Timer Counter register.
- Bit [9]—Channel 2 Output Compare Flag (CH2_OC) → Is set when the value in the TCU Channel 2 Output Compare register is equal to the value of the Timer Counter register.
- Bit [8]—Channel 3 Output Compare Flag (CH0_IC) → Is set when the value in the TCU Channel 3 Output Compare register is equal to the value of the Timer Counter register.
- Bit [7]—Termination Event Flag (TE) → When set, indicates the value in the Timer Counter register has reached a termination point.
- Bits [6–0]—Reserved

Table 12-70. TCUx_Mode

31	30–23	22–20	19–17	16	15	14	13–12	11	10–7	6–5	4–0
TC IRQ En	Reserved	Context	Priority	TE IRQ En	Up/ Dn	En	Auto Reload	Reload	Prescaler	CLK Mode	Reserved
RW	–	RW	RW	RW	RW	RW	RW	RW	RW	RW	–

- Bit [31]—Timer Counter Master Interrupt Enable (TC IRQ En) → Enables all interrupts for the Timer Counter Unit.
- Bits [30–23]—Reserved
- Bits [22–20]—Context → Sets which context the Timer Counter Unit interrupt is associated with
  - 000—Context_0
  - 001—Context_1
  - 010—Context_2
  - 011—Context_3

- 100—Context_4
- 101–111—Reserved

- Bits [19–17]—Interrupt Priority (IRQ Priority) → Interrupt priority for the assigned interrupt. Priorities range from zero to seven (000–111).

- Bit [16]—Termination Event Interrupt Enable (TE IRQ En) → Will fire an interrupt if TC_IRQEN is set and a Termination Event occurs.

- Bit [15]—Up/Down (Up/Dn) → Determines the direction of count.
  - 0 − Counts down
  - 1 − Counts up

- Bit [14]—TCU Enable (En) → When set, enables the counter to begin counting on the next valid clock edge. Counter is not reloaded upon enabling.

- Bits [13–12]—Auto Reload → Determines the configuration of the Timer Counter auto reload functionality
  - 00—Disables auto reload. Timer will count up/down to full count and stop on Termination Value clearing the En bit.
  - 01—Disables auto reload. Timer will free run by counting up/down to full count without stopping or reloading.
  - 10—Enables auto reload. Timer counts up/down to the Termination Value. On reaching it (rollover), the Timer reload value will be automatically reloaded into the Timer Counter. ICM preload functionality also is enabled.
  - 11—Reserved

- Bit [11]—Reload → When enabled, forces the Timer reload value to be loaded into the Timer Counter (Self Clearing).

- Bits [10–7]—Prescaler → Sets what clock divider value to use. The prescaler counter is reset each time the Timer Counter is reloaded (see Table 8-11).

Table 12-71. TCUx_Counter

31–16	15–0
Reserved	Count Value
–	RW

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved

- Bits [15–0]—Count Value → Contains the current count value

Table 12-72. TCUx_Chy_Mode

31–16	15–14	13	12	11–10	9	8	7	6	5–4	3	2	1	0
Reserved	ICM	ICM IRQ En	ICM Pload	OCM	OCM IRQ En	PM	PWM	PWM Level	Reserved	Pin Ctrl	ICM SRC	COC	Reserved
–	RW	RW	RW	RW	RW	RW	RW	RW	–	RW	RW	RW	–

- Bits [31–16]—Reserved
- Bits [15–14]—Input Capture Mode (ICM) → Determines which clock edge the data is captured on (see Table 8-15)
- Bit [13]—Input Capture Mode Interrupt Enable (ICM IRQ En) → When set, enables the Input Capture Mode Interrupt when the input capture condition occurs.
- Bit [12]—Input Capture Mode Preload Enable (ICM Pload) → When set, enables the counter to preload the next count value upon the occurrence of an input capture condition. The preload will only occur if the Autoreload has been enabled in the register.
- Bits [11–10]—Output Compare Mode (OCM) → Sets the level of output upon a compare (see Table 8-16).
- Bit [9]—Output Compare Mode Interrupt Enable (OCM IRQ En) → When set, enables the Output Compare Mode Interrupt when the output compare condition occurs.
- Bit [8]—Pulse Width Measurement (PWM) → When set, enables the counter to count only when the corresponding TCU input capture input matches the selected level.
- Bit [7]—Pulse Width Measurement Level (PWM Level) → When cleared, counter will count when the corresponding TCU input capture input is low. When set, the counter will count when the corresponding TCU input capture input is high. Pulse Width Measurement must be enabled.
- Bit [6]—Pulse Mode (PM) → When set, enables the output compare to toggle each time the counter rolls over.
- Bits [5–4]—Reserved
- Bit [3]—Pin Control (Pin Ctrl) → When cleared, sets the associated hardware pin for input capture. When set, enables the associated hardware pin for output compare.
- Bit [2]—Input Capture Source (ICM Src) → When cleared, selects the corresponding pin (subject to the setting of the Pin Crtl bit) as the trigger for input capture mode. When set, selects the internal signal as the trigger for input capture mode.

Note: If using one of the UICs as a source as an Input Capture Source, the Input Capture Mode (ICM) bits must be set for the rising edge of the clock.

- Bit [1]—Clear on Capture (COC) → When set, enables the counter to reset when an output compare occurs.

- Bit [0]—Reserved

Table 12-73. TCUx_Chy_Input_Capture

31–16	15–0
Reserved	Capture Value
–	R

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved

- Bits [15–0]—Capture Value → Holds the value captured by the Timer Counter Register upon an event occurrence.

Table 12-74. TCUx_Chy_Output_Compare

TCUx_Chy_Output_Compare	
31–16	15–0
Reserved	Compare Value
–	R

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved

- Bits [15–0]—Compare Value → the value to be compared with the value in the Timer Counter Register to create an event.

Table 12-75. ADC_ControlRegister

31–14	13–11	10–8	7	6	5	4	3	2–0
Reserved	Context	IRQ Priority	En	Scan	Mult	CD	IRQ Enable	Channel Select
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–14]—Reserved

- Bits [13–11]—Context → Sets which context the ADC interrupt is associated with

- 000—Context_0
- 001—Context_1
- 010—Context_2
- 011—Context_3
- 100—Context_4
- 101–111—Reserved

- Bits [10–8]—Interrupt Priority (IRQ Priority) → Interrupt priority for the ADC interrupt. Priorities range from zero to seven (000–111).

- Bit [7]—ADC Enable (En) → Enables and powers the ADC circuitry

- Bit [6]—Scan Mode (Scan) → Determines the scan mode of the ADC channel inputs

  - 0—Scan is disabled allowing single-channel conversion if in single mode. If in multiple mode, each channel is converted once

  - 1—Scan mode is enabled allowing single continuous channel conversion if in single mode. If in multiple mode, each channel is converted continuously in order zero to seven (upon rollover from channel seven to zero, data in result registers will be overwritten if it has not already been read).

- Bit [5]—Multiple Channel Conversion (Mult) → Enables multiple channel functionality for ADC channels zero through seven
  - 0—Single channel conversion operation
  - 1—Multiple conversion operation (channels zero to seven)

- Bit [4]—Conversion Done (CD) → Indicates the ADC conversion has completed (cleared on read)

- Bit [3]—Interrupt Enable (IRQ En) → Enables the ADC interrupt upon completion of an ADC conversion

- Bits [2–0]—Channel Select → Selects the channel in single-conversion-mode operation. These bits are ignored in multiple-conversion operation.
  - 000—Channel 0
  - 001—Channel 1
  - 010—Channel 2
  - 011—Channel 3
  - 100—Channel 4
  - 101—Channel 5
  - 110—Channel 6
  - 111—Channel 7

Table 12-76. ADC_StartRegister

31–1	0
Reserved	Start
–	RW

Note: Reset value is 0x00000000.

- Bits [31–1]—Reserved
- Bit [0]—Start ADC Conversion (Start) → Starts the ADC conversion process

Table 12-77. ADC Data Available Register

31–8	7	6	5	4	3	2	1	0
Reserved	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
–	RW	RW	RW	RW	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–8]—Reserved
- Bit [7]—Channel 7 Data Available Flag (CH7) → Indicates an ADC conversion has completed and data is available in the channel 7 result register.
- Bit [6]—Channel 6 Data Available Flag (CH6) → Indicates an ADC conversion has completed and data is available in the channel 6 result register.
- Bit [5]—Channel 5 Data Available Flag (CH5) → Indicates an ADC conversion has completed and data is available in the channel 5 result register.
- Bit [4]—Channel 4 Data Available Flag (CH4) → Indicates an ADC conversion has completed and data is available in the channel 4 result register.
- Bit [3]—Channel 3 Data Available Flag (CH3) → Indicates an ADC conversion has completed and data is available in the channel 3 result register.
- Bit [2]—Channel 2 Data Available Flag (CH2) → Indicates an ADC conversion has completed and data is available in the channel 2 result register.
- Bit [1]—Channel 1 Data Available Flag (CH1) → Indicates an ADC conversion has completed and data is available in the channel 1 result register.
- Bit [0]—Channel 0 Data Available Flag (CH0) → Indicates an ADC conversion has completed and data is available in the channel 0 result register.

Table 12-78. ADC Channel × Data Register

31–10	9–0
Reserved	Result
–	RW

Note: Reset value is 0x00000000.

- Bits [31–10]—Reserved
- Bits [9–0]—ADC Data Result (Result) → The result of the ADC converted data for a particular channel

Table 12-79. Debug Control Register Definition

31–5	4	3–1	0
Reserved	H/W Breakpoint Mode	Reserved	S/W BKPT instruction ctrl
RW	–	RW	–

Note: Reset value is 0x00000000.

- Bits [31–5]—Reserved
- Bit [4]—H/W Breakpoint mode bit, only applies if the debugger is connected

Note 1: When the debugger is not connected, this bit should be left at its default setting of logic 0 to prevent indeterminate behavior.

Note 2: When the debugger is not connected, exception #12 is raised to the running context when an S/W BKPT or H/W breakpoint/watchpoint is encountered.

- Bits [3–1]—Reserved
- Bit [0]—S/W Breakpoint instruction control bit
  - 0 → BKPT instruction executes as a NOP; instruction execution uninterrupted
  - 1 → BKPT instruction will generate exception #12 to the local context (no HALT occurs—what happens next is up to the exception handler, which is application-dependent)

Note 3: If the debugger is connected, the BKPT instruction will not generate exception #12 but rather will HALT either just the running context or the entire CPU, depending upon the breakpoint mode bit in the JTAG Debug Control Register (which is not software-accessible).

Table 12-80. Watchpoint Block Size and Break/Watch Mode Control Register

31–16	15	14–13	12	11–10	9–7	6	5–4	3–0
Reserved	Enable	R/W Control	Context Aware	Reserved	Context	Type	Mode	Block Size
–	RW	RW	RW	–	RW	RW	RW	RW

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved
- Bit [15]—Enable (0 → disable, 1 → enable)
- Bits [14–13]—Read/Write trigger control
  - 00 → Trigger on read or write
  - 01 → Trigger on write
  - 10 → Trigger on read
  - 11 → Reserved
- Bit [12]—Context Aware control
  - 0 → Trigger in any context
  - 1 → Trigger in specified context only (as defined in context field in bits 9–7)
- Bits [11–10]—Reserved
- Bits [9–7]—Context specific trigger control → this field specifies which context (0–4) in which the event must occur to cause a trigger

Note: If Bit [12] is logic 0, this field is ignored.

- Bit [6]—Type (0 → Breakpoint, 1 → Watchpoint)
- Bits [5–4]—Mode
  - 00 → Reserved
  - 01 → Enable trace (turn on tracing when event triggers)
  - 10 → Breakpoint (raise exception on event trigger)

Note: It will HALT if the debugger is connected.

  - 11 → Chained (enable next breakpoint/watchpoint in chain)

Note: The last event in a chain must be breakpoint or enable trace.

- Bits [3–0]—Block size (for watchpoints only), will watch memory in this block size
  - 0000 → 1 byte

- 0001 → 2 bytes
- 0010 → 4 bytes
- 0011 → 8 bytes
- 0100 → 16 bytes
- 0101 → 32 bytes
- 0110 → 64 bytes
- 0111 → 128 bytes
- 1000 → 256 bytes
- 1001 → 512 bytes
- 1010 → 1024 bytes
- 1011 → 2048 bytes
- 1100 → 4096 bytes
- 1101 → 8192 bytes
- 1110 → 16384 bytes
- 1111 → 32768 bytes

Table 12-81. Breakpoint Address/Watchpoint Base Address Register

31–0
Base Address
RW

Note: Reset value is 0x00000000.

- Bits [31–0]—Instruction address or base data address for the breakpoint/watchpoint

Note: For a watchpoint, the base address is subject to a boundary based on the block-size field in the Watchpoint Block Size and Break/Watch Mode Control Register.

Table 12-82. Watchpoint Data Register

31–0
Data
RW

Note: Reset value is 0x00000000.

- Bits [31–0]—Data value for watchpoint

Table 12-83. Watchpoint Data Mask Register

31–0
Data Mask
RW

Note: Reset value is 0x00000000.

- Bits [31–0]—Data mask for watchpoint

Table 12-84. Debug Trace Buffer Control Register

31–16	15	14	13–12	11–10	9–8	7–6	5–4	3–0
Reserved	Enable	Full Flag	Reserved	Trace Mode	Reserved	Buffer Mode	Reserved	Buffer Size
–	RW	RW	–	RW	–	RW	–	RW

Note: Reset value is 0x00000000.

- Bits [31–16]—Reserved

- Bit [15]—Trace enable (enabled explicitly by Context_0 or break/watch point; disabled explicitly by Context_0 or when buffer full)
  - 0 → Disable
  - 1 → Enable

- Bit [14]—Full flag
  - In simple buffer trace mode, the full flag bit is set to 1 when the trace buffer fills. The full flag bit is cleared when Context_0 writes a 1 to this bit.
  - If the full flag bit is set when a break/watch point is evaluated as "true," a new trace will not be started.

- Bits [13–12]—Reserved

- Bits [11–10]—Trace mode
  - 00 → Reserved
  - 01 → Call tracing mode
  - 10 → Branch tracing mode
  - 11 → Data tracing mode

- Bits [9–8]—Reserved

- Bits [7–6]—Specifies trace buffer mode
  - 00 → Single address mode
  - 01 → Circular buffer mode
  - 10 → Simple buffer mode

- Bits [5–4]—Reserved

- Bits [3–0]—Buffer size (used in the case of the buffer modes)
  - 0000 → 32 bytes
  - 0001 → 64 bytes
  - 0010 → 128 bytes

- 0011 → 256 bytes
- 0100 → 512 bytes
- 0101 → 1 Kbyte
- 0110 → 2 Kbytes
- 0111 → 4 Kbytes
- 1000 → 8 Kbytes
- 1001 → 16 Kbytes
- 1010 → 32 Kbytes
- 1011 → 64 Kbytes
- 1100 → 128 Kbytes
- 1101 → 256 Kbytes
- 1110 → 512 Kbytes
- 1111 → 1 Mbyte

Table 12-85. Trace Buffer Base Address Register

31–0
Base Address of Trace Buffer
RW

Note: Reset value is 0x00000000.

- Bits [31–0]—Address in memory of trace buffer

Note: This base address must be on a boundary defined by the buffer size field of the Debug Trace Buffer Control Register.

Table 12-86. Power-On Reset Register

31–2	1	0
Reserved	WDT	
–	RO	RW

Note: Reset value is 0x00000001.

- Bits [31–2]—Reserved

- Bit [1]—Watchdog timeout flag. This flag is set to one by the watchdog circuit if it generates a reset event. It is set to zero if any other type of reset occurs. It is read only to software.
  - 1—Most recent reset event was a WDT timeout
  - 0—Most recent reset event was not a WDT timeout, basically anything else

- Bit [0]—Power On Reset (POR) Flag. This flag is set to one by the power-on reset hardware. It can be read and written by software to force hard reset processing (does not generate a hard reset, only controls hardware initialization).
  - 1—If set to one at reset, then the hardware sets all registers to their default values
  - 0—If set to zero, the hardware does not set registers to default values

Table 12-87. Clock Mask Register

31–14	13	12	11	10	9	8	7–4	3	2	1	0
Reserved	TCU1	TCU0	System Timers	SDRAM	AtoD	PMU	Reserved	UIC3	UIC2	UIC1	UIC0
–	RW	RW	RW	RW	RW	RW	–	RW	RW	RW	RW

Note: Reset value at POR = 0x3F0F, all peripherals are OFF (all bits are inverted polarity).

- Bits [31–14]—Reserved
- Bit [13]—TCU1 Power control (1 → power off, 0 → power on)
- Bit [12]—TCU0 Power control (1 → power off, 0 → power on)
- Bit [11]—System Timer Power control (1 → power off, 0 → power on)
- Bit [10]—SDRAM Power control (1 → power off, 0 → power on)
- Bit [9]—AtoD Power control (1 → power off, 0 → power on)
- Bit [8]—PMU Power control (1 → power off, 0 → power on)
- Bits [7–4]—Reserved
- Bit [3]—UIC3 Power control (1 → power off, 0 → power on)
- Bit [2]—UIC2 Power control (1 → power off, 0 → power on)
- Bit [1]—UIC1 Power control (1 → power off, 0 → power on)
- Bit [0]—UIC0 Power control (1 → power off, 0 → power on)

Note: The external RESET_N input (non-POR) does not change this register.

13. Revision History

Table 13-1 presents the sequence of revisions to document IA221080723.

Table 13-1. Revision History

Date	Revision	Description	Page(s)
August 19, 2008	00	Initial Release	NA
September 9, 2008	01	Change to Table 6-1: IO RDY_N, HOLDREQ_N, AND HOLDGNT_N changed from active high to active low.	128
September 18, 2008	02	"TsWAIT" changed to "TwWAIT".	133, 265
October 8, 2008	03	Updated the RESET_N and RESET_OUT_N signal names.	Throughout document
July 20, 2009	04	Corrected the Major Reset Value and Minor Reset Value in Table 5-20; Added a note to paragraph 4.8.7 regarding systems using an external bus master; Added a bullet under paragraph 4.6.5.10 regarding MPU block setup to point to non-existent memory; Corrected Bit descriptions in Table 8-16; Updated description for Bit [2] on page 217 and added a note regarding the TCU input capture function.	55, 64, 97, 217
November 30, 2009	05	Deleted references to the SKIP Assembler User Guide and UIC Programmer Guide, since those are not officially released documents at this point.	Throughout the document
May 6, 2010	06	Revised Control Register descriptions for Bits 9 and 10.	132

14. For Additional Information

The fido1100 Instruction Set Reference Guide as well as other helpful tools and files are available. For example, the GDB debugger supports both profiling and tracing of executing code. When building a prototype using the fido1100 communications controller, a file in the "Samples" directory can assist in customizing the software for a specific design.

The Innovasic Support Team is continually planning and creating tools for your use. Visit http://www.Innovasic.com for up-to-date documentation and software. Our goal is to provide timely, complete, accurate, useful, and easy-to-understand information. Please feel free to contact our experts at Innovasic at any time with suggestions, comments, or questions.

Innovasic Support Team
3737 Princeton NE
Suite 130
Albuquerque, NM 87107

(505) 883-5263
Fax: (505) 883-5477
Toll Free: (888) 824-4184
E-mail: support@innovasic.com
Website: http://www.Innovasic.com