TMS320VC547x CPU and Peripherals Reference Guide

Literature Number: SPRU038
December 2001

Printed on Recycled Paper

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that products or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products.
www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Preface

Read This First

About This Manual

This user's guide serves as a reference for the Texas Instruments TMS320VC547x low-power, enhanced-architecture, dual-core digital signal processor (DSP), and is intended to assist hardware and software engineers in developing applications using this device. It describes both cores—the TMS320C54x™ DSP CPU and the ARM7TDMI™ microcontroller unit (MCU)—and their peripherals, together with the memory and peripheral interface associated with each core.

Throughout this book, the TMS320VC547x dual-core device is referred to as the VC547x. Where the ARM7TDMI MCU core is referred to separately, the alphanumeric designation is shortened to ARM™. Information about the two processor cores in relevant chapters is provided separately, rather than combining similar features where applicable.

Notational Conventions

This book uses the following conventions.

- Instruction Sets
  - The TMS320VC547x DSP CPU can use either of two forms of the instruction set: a mnemonic form or an algebraic form. This book uses the mnemonic form of the instruction set. For information about the mnemonic form of the instruction set, see *TMS320C54x DSP Reference Set, Volume 2: Mnemonic Instruction Set*. For information about the algebraic form of the instruction set, see *TMS320C54x DSP Reference Set, Volume 3: Algebraic Instruction Set*. These references are both listed in the section titled *Related Documentation From Texas Instruments.*

- The TMS320VC547x MCU CPU uses its own instruction set. For information about the MCU's instruction set, see *TMS470R1x User's Guide*, also shown in *Related Documentation From Texas Instruments.*

- Program listings and program examples are shown in a `special typeface`.

  Here is a segment of a program listing:

```
STL    A,*AR1+      ;Int_RAM(I)=0
RSBX   INTM         ;Globally enable interrupts
B      MAIN_PG      ;Return to foreground program
```

- Square brackets, [and], identify an optional parameter. If you use an optional parameter, specify the information within the brackets; do not type the brackets themselves.

Related Documentation From Texas Instruments

The following books provide related documentation for the TMS320VC547x. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477-8924. When ordering, please identify the book by its title and literature number. Many of these documents are located on the Internet at http://www.ti.com.

TMS320C54x DSP Reference Set, Volume 2: Mnemonic Instruction Set (literature number SPRU172) describes the TMS320C54x digital signal processor mnemonic instructions individually. Also includes a summary of instruction set classes and cycles.

TMS320C54x DSP Reference Set, Volume 3: Algebraic Instruction Set (literature number SPRU179) describes the TMS320C54x digital signal processor algebraic instructions individually. Also includes a summary of instruction set classes and cycles.

TMS470R1x User's Guide (literature number SPNU134) describes the TMS470R1x RISC microcontroller, its architecture (including registers), the ICEBreaker™ module, interfaces (memory, coprocessor, and debugger), 16-bit and 32-bit instruction sets, and electrical specifications.

TMS320C54x DSKplus User's Guide (literature number SPRU191) describes the TMS320C54x digital signal processor starter kit (DSK), which allows you to execute custom C54x™ code in real time and debug it line by line. Covered are installation procedures, a description of the debugger and the assembler, customized applications, and initialization routines.

TMS320C54x Assembly Language Tools User's Guide (literature number SPRU102) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the C54x generation of devices.

TMS320C54x C Source Debugger User's Guide (literature number SPRU099) tells you how to invoke the C54x emulator, evaluation module, and simulator versions of the C source debugger interface. This book discusses various aspects of the debugger interface, including window management, command entry, code execution, data management, and breakpoints. It also includes a tutorial that introduces basic debugger functionality.

TMS320C54x Code Generation Tools Getting Started Guide (literature number SPRU147) describes how to install the TMS320C54x assembly language tools and the C compiler for the C54x devices. The installation for MS-DOS™, OS/2™, SunOS™, Solaris™, and HP-UX™ 9.0x systems is covered.

TMS320C54x Evaluation Module Technical Reference (literature number SPRU135) describes the C54x evaluation module, its features, design details and external interfaces.

TMS320C54x Optimizing C/C++ Compiler User's Guide (literature number SPRU103) describes the C54x C compiler. This C compiler accepts ANSI standard C source code and produces TMS320™ assembly language source code for the C54x generation of devices.

TMS320C54x Simulator Getting Started Guide (literature number SPRU137) describes how to install the TMS320C54x simulator and the C source debugger for the C54x. The installation for MS-DOS™, PC-DOS™, SunOS™, Solaris™, and HP-UX™ systems is covered.

TMS320 Third-Party Support Reference Guide (literature number SPRU052) alphabetically lists over 100 third parties that provide various products that serve the family of TMS320 digital signal processors. A myriad of products and applications are offered—software and hardware development tools, speech recognition, image processing, noise cancellation, modems, etc.

TMS320 DSP Development Support Reference Guide (literature number SPRU011) describes the TMS320™ DSP family of digital signal processors and the tools that support these devices. Included are code-generation tools (compilers, assemblers, linkers, etc.) and system integration and debug tools (simulators, emulators, evaluation modules, etc.). Also covered are available documentation, seminars, the university program, and factory repair and exchange.

Trademarks

Contents

Figures

Tables

Chapter 1

Introduction

This chapter introduces the TMS320VC547x—a dual-core device consisting of a programmable digital signal processor (DSP) and a microcontroller unit (MCU).

This introduction includes a general description of the VC547x and a list of its key features.

For specific information about the 54x DSP core or the RISC MCU core, see the appropriate user guide listed under *Related Documentation from Texas Instruments* in the Preface of this book.

Topic **Page**

1.1 General Description of the VC547x

The VC547x devices (VC5470 and VC5471) integrate a TMS320C54x™ DSP subsystem with its program and data memories (all RAM) and an ARM7™ RISC microcontroller core with emulation facilities. On the VC5471, an integrated Ethernet 10/100 Base-T interface is supported for connection to the Ethernet physical layer (PHY) via a media-independent interface (MII).

1.2 Key Features of the VC547x

- Dual-CPU processor integrating a TMS320C54x DSP and an ARM7TDMI RISC MCU
- DSP: 100-MHz, low-power, TMS320C54x 16-bit core with 72K x 16-bit on-chip RAM [16K x 16-bit dual-access RAM (DARAM) and 56K x 16-bit single-access RAM (SARAM)]
- DSP on-chip peripherals
  - Two high-speed, full-duplex multichannel buffered serial ports (McBSPs) allowing the DSP core to interface directly with codecs and other devices in the system
  - A six-channel direct memory access (DMA) controller enabling six independent block transfers with no intervention from the CPU
  - ARM port interface (API) shared-memory interface for efficient information exchange between the ARM and the DSP CPUs
  - Software-programmable wait-state generator capable of extending external bus cycles by up to 14 machine cycles
  - External memory interface (EMIF)
  - One independent software-programmable hardware timer for control operations
- RISC: 47.5-MHz, ARM7TDMI microcontroller core with 16K bytes of on-chip SARAM
- RISC peripherals
  - VC5471 only: Ethernet interface module with 10/100-Mbps IEEE 802.3 Ethernet media-access controller (MAC)
  - Universal asynchronous receiver/transmitter (UART) compatible with NS 16C750-compliant devices
  - NS 16C750-compliant UART/IrDA interface allowing the connection through an infrared transmitter to any external data peripherals using the slow infrared (SIR) protocol

- Serial port interface
- Thirty-six general-purpose I/O pins configurable in read or write mode by internal registers
- Inter-integrated circuit (I^2C) interface connecting ARM to I^2C-compliant devices made by Texas Instruments
- Three on-chip MCU timers (one watchdog timer and two general-purpose timers)
- Interrupt handler managing prioritized and maskable interrupts for both internal modules and external devices
- Memory interface between ARM CPU and its internal peripherals, external Flash and SRAM memories
- Synchronous dynamic random-access memory (SDRAM) memory interface between the ARM CPU and external SDRAM memories
- Clock management module controlling clock generation and activity for the DSP, MCU, and peripherals

❑ On-chip scan-based emulation logic, IEEE Std 1149.1† (JTAG) boundary scan logic

† IEEE Standard 1149.1–1990, IEEE Standard Test-Access Port

❑ ICECrusher module for emulation of both the DSP and RISC CPUs

❑ Smart power management and low-power modes:

- DSP low-power mode
- ARM low-power mode
- Peripherals low-power mode

❑ 10-ns single-cycle, fixed-point instruction execution time (100 MIPS) for 3.3-V power supply (1.8-V core)

❑ Provided in a 257-pin MicroStar BGA™ package (GHK Suffix)

Chapter 2

Architecture

This chapter provides an overview of the architectural structure of the dual-core TMS320VC547x device.

For specific information about the 54x DSP core or the RISC MCU core (such as bus structure, CPU, or pipeline operations), see the appropriate user guide listed under *Related Documentation from Texas Instruments* in the Preface of this book.

Topic **Page**

2.1 Functional Overview of the VC547x

The VC547x architecture is based on a dual-processor core plus some application peripherals that are memory-mapped in the ARM memory space.

The VC547x consists of the following modules:

DSP Subsystem

- ❑ TMS320C54x DSP core with 72K words (16-bit) of data/program RAM.

DSP Peripherals:

- ❑ Two multichannel buffered serial ports (McBSPs)
- ❑ Direct memory access (DMA) controller
- ❑ ARM port interface (API)
- ❑ Programmable wait-state generator
- ❑ External memory interface (EMIF)
- ❑ Timer
- ❑ Phase-locked loop (PLL)

ARM7 RISC Microcontroller

- ❑ ARM7TDMI CPU core (32/16-bit RISC processor)
- ❑ ARM ICECrusher for emulation purposes

ARM Peripherals:

- ❑ ARM memory interface for external SRAM, flash or ROM, and SDRAM.
- ❑ On-chip 16K-byte (32 x 4096) zero wait-state SRAM.
- ❑ ARM general-purpose I/Os (GPIOs) with 8 x 8 keyboard interface.
- ❑ Three timers (two generic, one watchdog)
- ❑ UART 16C750 interface (UART_IRDA) with
  - ■ IRDA control capabilities (SIR)
  - ■ supports only the software flow control (UART mode)

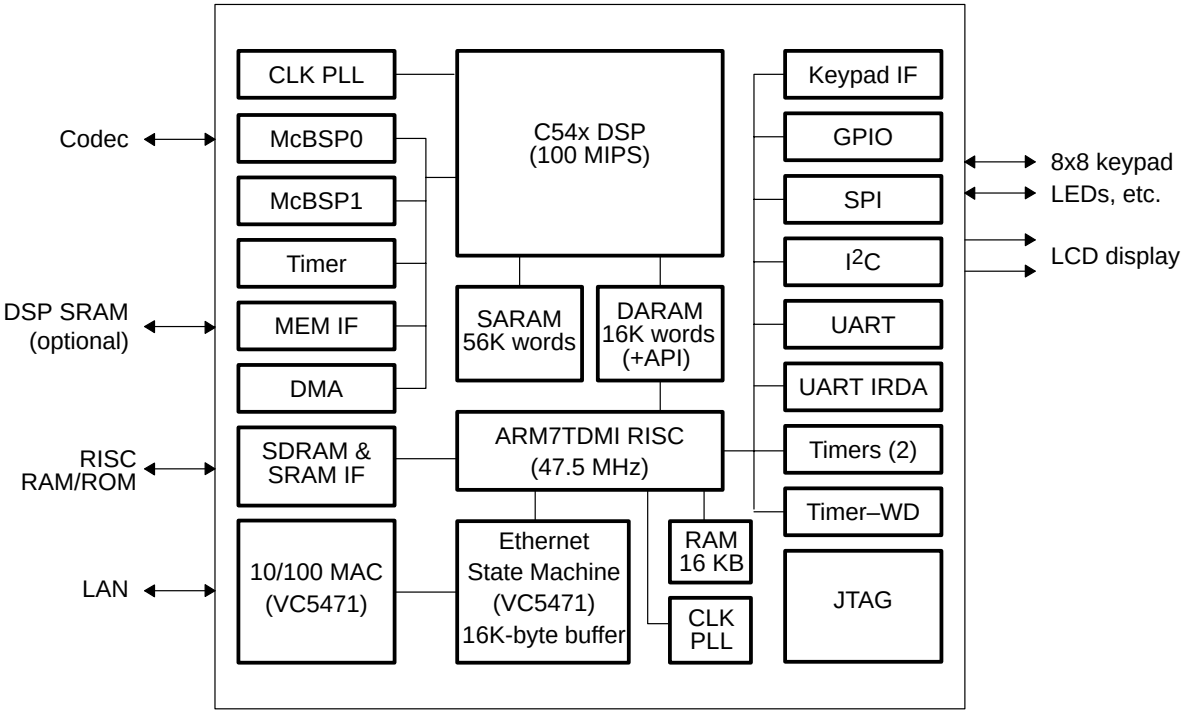
- ❏ UART 16C750 interface (UART) with
  - ■ hardware flow protocol (DCD, CTS/RTS)
  - ■ autobaud function
- ❏ ARM interrupt handler (INTH)
- ❏ Clock generator and control (CLKM)
- ❏ Phase-locked loop (PLL)
- ❏ Ethernet 10/100Base-T Interface (EIM) (on VC5471)
- ❏ I^2C *master-only* interface (I2C)
- ❏ Serial peripheral interface (SPI)

Other peripherals:

- ❏ JTAG TAP controller

2.2 Functional Block Diagram of the VC547x

Figure 2–1. TMS320VC547x Functional Block Diagram



2.3 DSP Subsystem Overview (TMS320C54x DSP Core)

The DSP subsystem is based on the TMS320C54x DSP core, and is completely code-compatible with other C54x products.

C54x devices are fixed-point digital signal processors (DSPs) in Texas Instruments' TMS320 family. The C54x CPU, with its advanced modified Harvard architecture, features minimized power consumption and a high degree of parallelism.

This processor has one program memory bus and three data memory buses. It also provides an arithmetic logic unit (ALU) that has a high degree of parallelism, application-specific hardware logic, on-chip memory, and additional on-chip peripherals. The basis of the operational flexibility and speed of this DSP is a highly specialized instruction set.

Separate program and data spaces allow simultaneous access to program instructions and data, providing the high degree of parallelism. Two read operations and one write operation can be performed in a single cycle. Instructions with parallel store and application-specific instructions can fully utilize this architecture. In addition, data can be transferred between data and program spaces. Such parallelism supports a powerful set of arithmetic, logic, and bit-manipulation operations that can all be performed in a single machine cycle. In addition, this processor includes the control mechanisms to manage interrupts, repeated operations, and function calls.

2.3.1 Features

The DSP core includes the following features:

- Low-power C54x DSP CPU
- Software-programmable wait-state generator with bank-switching wait-state logic
- External memory interface
  - Program space
  - Data space
  - I/O space
- Scan-based emulation logic

2.3.2 DSP CPU Core Associations

The DSP CPU core is associated with an ARM port interface (API), a programmable phase-locked loop (PLL), an interrupt handler, a parallel interface XIO, a timer, 72K words of RAM, two multichannel buffered serial ports (McBSPs), and a JTAG interface.

The maximum DSP cycle frequency is programmable up to 100 MHz.

2.4 DSP Memory Space

The C54x DSP uses an enhanced Harvard architecture. This architecture has multiple memory spaces and four parallel buses that allow you to access both program and data simultaneously. Each of the four buses accesses different memory spaces for different aspects of the DSP operation. These are:

- The program bus (PB) reads from program memory space, which contains the instructions to be executed.
- The write data bus (EB) writes into data memory space, which stores data used by the instructions and tables eventually used in execution. It also writes into I/O memory space.
- The two read data buses (CB and DB) read data from the data memory space and the DB data bus accesses I/O memory space. The I/O memory space interfaces to external memory-mapped peripherals and can serve as extra data storage space.
- The DSP subsystem includes 72K words of on-chip RAM as well as an extensive external memory range, which can be used to interface to a variety of memory types or peripherals.

2.4.1 On-Chip RAM

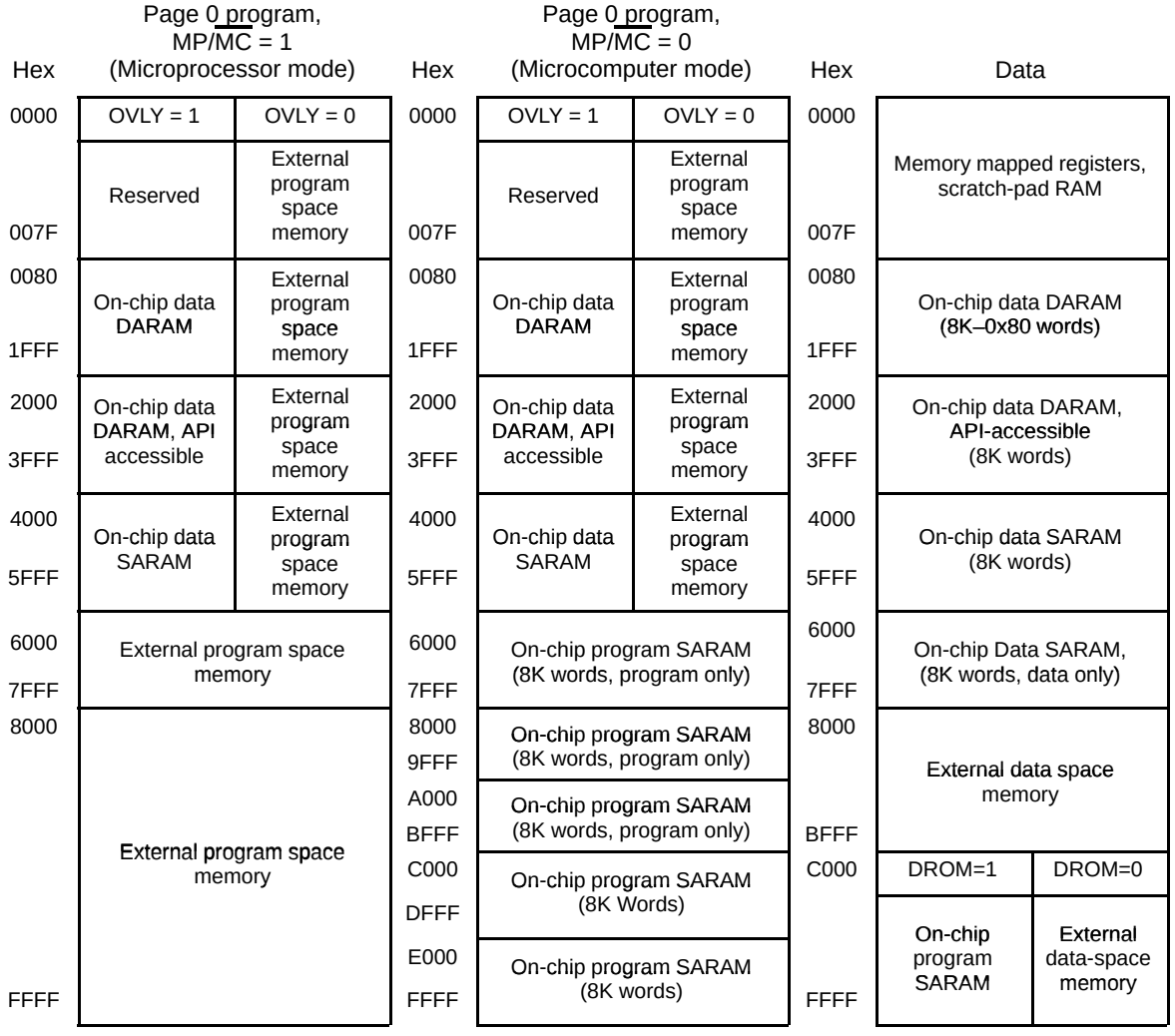
The DSP subsystem features 72K x 16 bits of on-chip RAM (two blocks of 8K x16-bit DARAM and seven blocks of 8K x 16-bit SARAM). The DSP CPU can perform two accesses to a DARAM in one machine cycle (two reads in one cycle, or a read and a write in one cycle). It can also perform multiple accesses to separate memory blocks in one machine cycle.

After reset, the lower address range of the program space is mapped to external memory, the lower address range of the data space is mapped with on-chip RAM blocks. However, the OVLY bit in the PMST register can be used to map these RAM blocks into both program and data space.

2.4.2 Normal Mode DSP Memory Map

The “Normal Mode” DSP subsystem provides the memory map shown in Figure 2–2. This is the memory map that applies when the API Boot Mode feature is not enabled. The Normal Mode memory map applies any time that BSCR[4] (ABMDIS) is 1, or when DSP_REG[9] (DSP_APIBN) port is high.

Figure 2–2. DSP Subsystem Memory Map for DSP Accesses (When DSP_APIBN = 1 or ABMDIS = 1)



2.4.3 API Boot Mode

Since there is no ROM at all in the C547x, there is no built-in boot-load program. All DSP code must come from external sources – either a Flash or ROM via the DSP's XIO pins, or code uploaded via the ARM's API interface to DSP RAM.

Under normal DSP reset conditions, the DSP begins operation either from internal RAM code (when in microcomputer mode) or from external memory on the XIO bus (microprocessor mode). A new mode is made possible by the presence of the API module and the MCU that controls it, called API Boot mode. When the DSP subsystem is in API boot mode, the upper 2K words of the 8K-word API DARAM is aliased so that it is found both in DSP data space at 0x3800–0x3FFF and in DSP program space at 0xF800 to 0xFFFF.

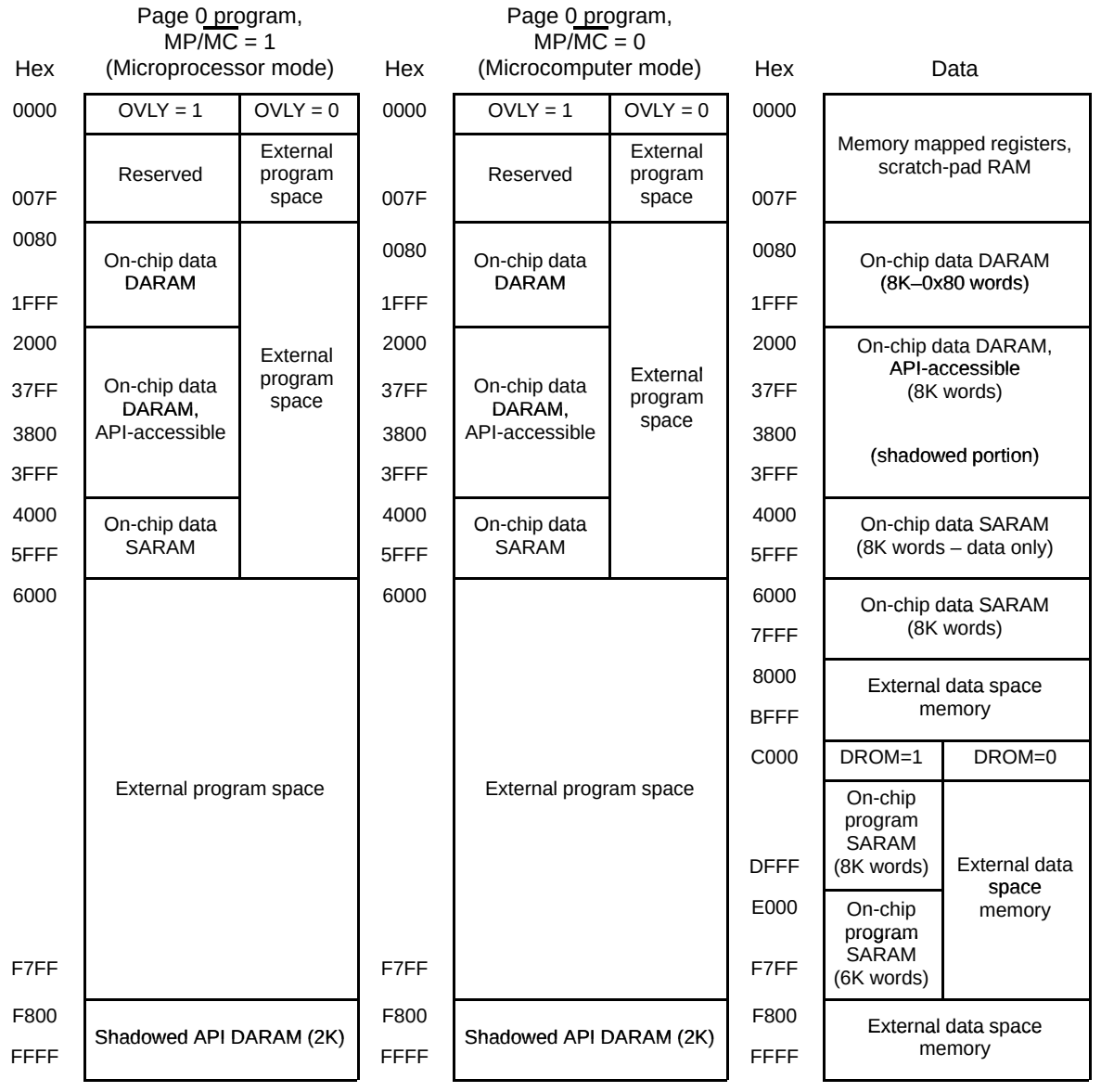
To make use of this mode, the MCU directly controls the DSP's reset and API boot mode signals. It holds the DSP in reset, enables API boot mode, and loads the DSP initialization code via the API. Once it has loaded the code to the appropriate location in the API DARAM, the MCU releases the DSP from reset, and the DSP begins executing the code. It is recommended that the MCU change the API boot mode input signal only when the DSP is held in reset.

Note: The microprocessor or microcomputer mode can be important in API boot mode.

2.4.4 API Boot Mode DSP Memory Map

The Memory Map, when the API Boot Mode feature is enabled, is shown in Figure 2–3. API Boot Mode is enabled when the DSP_REG[9] (DSP_APIBN) port is low and BSCR[4] (ABMDIS) is 0.

Figure 2–3. API Boot Mode DSP Subsystem Memory Map for DSP Accesses (When DSP_APIBN = 0 and ABMDIS = 0)†



† When DSP_REG[9] (DSP_APIBN) = 0 and BSCR[4] (ABMDIS) = 0, 2K words of the API DARAM are remapped to program-space, regardless of DSP_REG[10] (MP/$\overline{MC}$) value. All other internal program-space RAMs are disabled in program space. Overlayable data-space RAMs may be dual-mapped to program-space via OVLY.

2.4.5 Extended Program Memory

The DSP subsystem includes a memory-paging scheme to extend the number of addressable program space locations from 64K to 1M words. The four extended address pins (A16 to A19) are used to address 15 pages of program memory. Each page includes 64K addressable locations. The extended program addresses are supported by the following eight instructions:

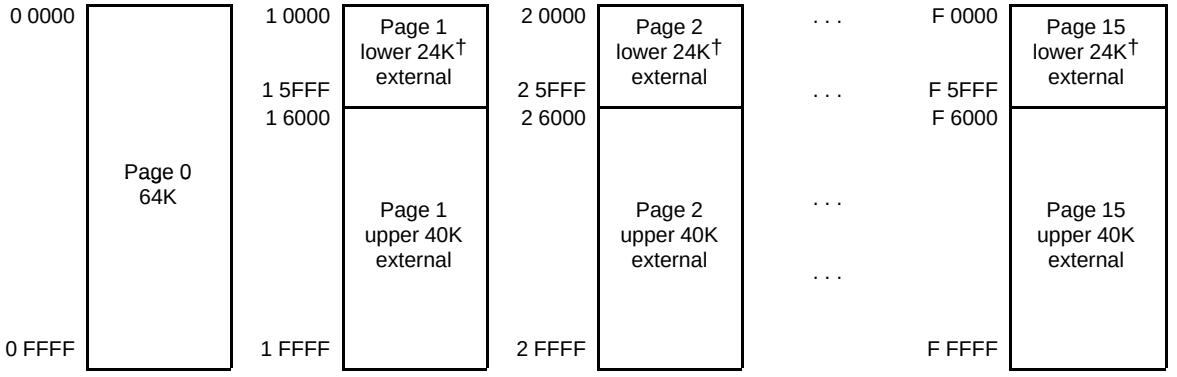
- FB[D] – Far branch
- FBACC[D] – Far branch to the location specified by the value in accumulator A or accumulator B
- FCALA[D] – Far call to the location specified by the value in accumulator A or accumulator B
- FCALL[D] – Far call
- FRET[D] – Far return
- FRETE[D] – Far return with interrupts enabled
- READA – Read program memory addressed by accumulator A and store in data memory
- WRITA – Write data to program memory addressed by accumulator A

For more information on these instructions, please refer to the *TMS320C54x DSP Reference Set, Volume 2: Mnemonic Instruction Set*, (literature number SPRU172).

When the OVLY bit is set, each page of program memory is made up of two parts: a common block of 24K words maximum and a unique block of 40K words minimum (see Figure 2–4). The common block is shared by all pages, and each unique block is accessible only through its assigned page.

The value of the program counter extension register (XPC) defines the page selection. At a hardware reset, the XPC is initialized to 0.

Figure 2–4. DSP Extended Program Memory Map



† Accesses to the lower 24K words of pages 1 through 15 are to external program memory only when the OVLY bit is cleared to 0. If the OVLY bit is set to 1, on-chip RAM is mapped to 0x0 to 0x5FFF of pages 1 through 15. Note external address pins have been provided to support 15 external pages.

2.4.6 Relocatable Interrupt Vector Table

The reset, interrupt, and trap vectors are addressed in program space. These vectors are soft—meaning that the processor, when taking the trap, loads the program counter (PC) with the trap address and executes the code at the vector location. Four words are reserved at each vector location to accommodate a delayed branch instruction—either two 1-word instructions or one 2-word instruction—which allows branching to the appropriate interrupt service routine with minimal overhead.

At device reset, the reset, interrupt, and trap vectors are mapped to address FF80h in program space. However, these vectors can be remapped to the beginning of any 128-word page in program space after device reset. This is done by loading the interrupt vector pointer (IPTR) bits in the PMST register with the appropriate 128-word page boundary address. After loading IPTR, any user interrupt or trap vector is mapped to the new 128-word page.

Note: The hardware reset (RS) vector cannot be remapped because a hardware reset loads the IPTR with 1s. Therefore, the reset vector is always fetched at location FF80h in program space.

2.5 DSP Registers

This section lists the DSP subsystem registers that are available in the VC547x. For a description of all registers except BSCR, see either the *TMS320VC5409 Fixed-Point Digital Signal Processor* data sheet (literature number SPRS082) or the *TMS320VC5402 Fixed-Point Digital Signal Processor* data sheet (literature number SPRS079). For a description of BSCR, see section 3.5.4, *Bank-Switching Control Register*.

The DSP subsystem peripheral mapping is shown in Table 2–1.

Table 2–1. DSP Peripheral Memory-Mapped Registers

Register	Address	Description	Type
DRR20	20h	Data receive register 2	McBSP #0
DRR10	21h	Data receive register 1	McBSP #0
DXR20	22h	Data transmit register 2	McBSP #0
DXR10	23h	Data transmit register 1	McBSP #0
TIM	24h	Timer register	Timer
PRD	25h	Timer period counter	Timer
TCR	26h	Timer control register	Timer
–	27h	Reserved	
SWWSR	28h	Software wait-state register	External Bus
BSCR	29h	Bank-switching control register	External Bus
–	2Ah	Reserved	
SWCR	2Bh	Software wait-state control register	External Bus
–	2Ch–37h	Reserved	
SPSA0	38h	McBSP0 subbank address register	McBSP #0
SPSD0	39h	McBSP0 subbank data register	McBSP #0
–	3Ah–3Bh	Reserved	
GPIOCR	3C	General-purpose I/O pins control register	GPIO
GPIOSR	3D	General-purpose I/O pins status register	GPIO
–	3E–3F	Reserved	

Table 2–1. DSP Peripheral Memory-Mapped Registers (Continued)

Register	Address	Description	Type
DRR21	40h	Data receive register 1	McBSP #1
DRR11	41h	Data receive register 2	McBSP #1
DXR21	42h	Data transmit register 1	McBSP #1
DXR11	43h	Data transmit register 2	McBSP #1
–	44h–47h	Reserved	
SPSA1	48h	McBSP1 subbank address register	McBSP #1
SPSD1	49h	McBSP1 subbank data register	McBSP #1
–	4Ah–53h	Reserved	
DMPREC	54h	DMA channel priority and enable control register	DMA
DMSA	55h	DMA subbank address register	DMA
DMSDI	56h	DMA subbank data register with autoincrement	DMA
DMSDN	57h	DMA subbank data register	DMA
CLKMD	58h	Clock mode register	PLL
–	59h–5Fh	Reserved	

2.6 DSP Subsystem Peripherals

The sections that follow describe the peripherals provided by the DSP subsystem.

2.6.1 Multichannel Buffered Serial Ports (McBSP0 and McBSP1)

The VC547x has two high-speed, full-duplex multichannel buffered serial ports (McBSPs) that allow direct interface to other C54x devices, codecs, and other devices in a system. The McBSPs used in the VC547x are based on the standard serial port interface found on other TMS320C54x devices. The two McBSPs that are accessible to the DSP are named McBSP0 and McBSP1.

The McBSPs provide:

- ❏ Full-duplex communication
- ❏ Double-buffered data registers that allow a continuous data stream
- ❏ Independent framing and clocking for receive and transmit

Capabilities

In addition, the McBSPs have the following capabilities:

- ❏ Direct interface to:
  - ■ T1/E1 framers
  - ■ MVIP switching-compatible and ST-BUS-compliant device
  - ■ IOM-2-compliant devices
  - ■ AC97-compliant devices
  - ■ Some IIS-compliant devices
  - ■ Serial peripheral interface (SPI) devices
- ❏ Multichannel transmit and receive of up to 32 channels
- ❏ Direct interface to industry-standard codecs, analog interface chips (AICs), and other serially connected A/D and D/A devices
- ❏ A wide selection of data sizes, including 8, 12, 16, 20, 24, and 32 bits
- ❏ μ-law and A-law companding

- External shift clock or an internal, programmable-frequency shift clock for data transfer
- Autobuffering capability through the six-channel DMA controller
- Programmable polarity for both frame synchronization and data clocks

External Interface

The McBSPs consist of separate transmit and receive channels that operate independently. The external interface of each McBSP consists of the following pins:

- BCLKX: Transmit reference clock
- BDX: Transmit data
- BFSX: Transmit frame synchronization
- BCLKR: Receive reference clock
- BDR: Receive data
- BFSR: Receive frame synchronization

Transmitter/Receiver Pins

The six pins listed are functionally equivalent to the pins of previous serial port interface pins in the C54x generation of DSPs. On the transmitter, transmit frame synchronization and clocking are indicated by the BFSX and BCLKX pins, respectively. The CPU or DMA can initiate transmission of data by writing to the data transmit register (DXR). Data written to DXR is shifted out on the BDX pin through a transmit shift register (XSR). This structure allows DXR to be loaded with the next word to be sent while the transmission of the current word is in progress.

On the receiver, receive frame synchronization and clocking are indicated by the BFSR and BCLKR pins, respectively. The CPU or DMA can read received data from the data receive register (DRR). Data received on the BDR pin is shifted into a receive shift register (RSR) and then buffered in the receive buffer register (RBR). If the DRR is empty, the RBR contents are copied into the DRR. If not, the RBR holds the data until the DRR is available. This structure allows storage of the two previous words while the reception of the current word is in progress.

Data Movement

The CPU and DMA can move data to and from the McBSPs and can synchronize transfers based on McBSP interrupts, event signals, and status flags. The DMA is capable of handling data movement between the McBSPs and memory with no intervention from the CPU.

Programmable Functions

In addition to the standard serial port functions, the McBSP provides programmable clock and frame sync generation. Among the programmable functions are:

- ❏ Frame synchronization pulse width
- ❏ Frame period
- ❏ Frame synchronization delay
- ❏ Clock reference (internal vs. external)
- ❏ Clock division
- ❏ Clock and frame sync polarity

The on-chip companding hardware allows compression and expansion of data in either μ-law or A-law format. When companding is used, transmit data is encoded according to specified companding law and received data is decoded to 2s-complement format.

Multiple Channel Selection

The McBSPs allow multiple channels to be independently selected for the transmitter and receiver. When multiple channels are selected, each frame represents a time-division multiplexed (TDM) data stream. In using TDM data streams, the CPU may only need to process a few of them. Thus, to save memory and bus bandwidth, multichannel selection allows independent enabling of particular channels for transmission and reception. Up to 32 channels can be enabled.

Clock-Stop Mode

The clock-stop mode (CLKSTP) in the McBSP provides compatibility with the serial peripheral interface (SPI) protocol. Clock-stop mode works with only single-phase frames and one word per frame. The word sizes supported by the McBSP are programmable for 8-, 12-, 16-, 20-, 24-, or 32-bit operation. When the McBSP is configured to operate in SPI mode, both the transmitter and the receiver operate together as a master or as a slave.

The McBSP is fully static and operates at arbitrarily low clock frequencies. The maximum frequency is CPU clock frequency divided by two.

2.6.2 Direct Memory Access Controller (DMAC)

The DSP subsystem includes a six-channel DMA controller, for performing data transfers independent of the CPU. The DMA controller has access to off-chip memory: it can move data from external DSP memory to internal memory (or internal to external). The primary function of the DMA controller is to manage data transfers between on-chip memory and the serial ports.

The DMA Controller includes the following features:

- ❏ Operates independently of the CPU
- ❏ Six independent channels (DMA controller can keep track of the context of six independent block transfers)
- ❏ Higher priority for memory accesses than CPU
- ❏ Each channel has an independently programmable priority level
- ❏ Each channel's source and destination address registers include configurable indexing modes. The address can remain constant, be post-incremented/decremented, or be adjusted by a programmable value.
- ❏ Each read or write transfer may be initialized by selected events (internally only)
- ❏ Each DMA channel can interrupt the CPU upon completion of a half or entire block transfer
- ❏ Supports doubleword transfers; i.e., a 32-bit transfer of two 16-bit words (internally only)
- ❏ The DMA cannot access the API DARAM

2.6.3 ARM Port Interface (API)

Information is exchanged between the ARM and the DSP through the on-chip shared API memory. The API memory is an 8K × 16-bit word DARAM (dual-access RAM) block. The API memory can also be used by the DSP as general-purpose data or program DARAM. DARAM is memory that can be accessed twice in the same clock cycle. In the VC547x, only the DSP memory has DARAM.

The API has two modes of operation that are selected by the API control register (APIC): shared-access mode (SAM) and host-only mode (HOM). In SAM, both the DSP and the ARM can access the API memory, and asynchronous host accesses from the ARM are resynchronized internally. If the DSP and the ARM try to perform an access at the same time, the ARM has access priority and the DSP waits one cycle. SAM can be run when the DSP is in IDLE1 mode. In HOM, only the ARM can access the API memory. When HOM is selected, API memory blocks are disabled for DSP access, no value can be read from or written to those memory locations.

SAM is the default configuration when the DSP exits from a reset phase. SAM is normally selected whenever the DSP is in normal operating mode or in IDLE1. HOM is normally selected before the DSP is placed in IDLE2 or IDLE3.

2.6.4 Software-Programmable Wait-State Generator

The software wait-state generator extends external bus cycles by up to fourteen machine cycles to interface with slower off-chip memory and I/O devices. Devices that require more than fourteen wait-states can be interfaced using the hardware DSP_READY line. When all external accesses are configured for zero wait states, the internal clocks to the wait-state generator are automatically disabled. Disabling the wait-state generator clocks reduces the power consumption of the DSP subsystem.

The software wait-state register (SWWSR) controls the operation of the wait-state generator. The 14 least significant bits (LSBs) of the SWWSR specify the number of wait states (0 to 7) to be inserted for external memory accesses to five separate address ranges. This allows a different number of wait states for each of the five address ranges. In addition, the software wait-state multiplier (SWSM) bit of the software wait-state control register (SWCR) defines a multiplication factor of 1 or 2 for the number of wait states. At reset, the wait-state generator is initialized to provide seven wait states on all external memory accesses.

2.6.5 External Memory Interface

The DSP external memory interface provides access to three separate off-chip memory spaces—program space, data space, and I/O space. The interface consists of a 16-bit address bus, a 4-bit extended program space address bus, a 16-bit bidirectional data bus, three space select signals, and various control signals.

The space select signals—program space select ($\overline{PS}$), data space select ($\overline{DS}$), and I/O space select ($\overline{IS}$)—determine which external memory space is accessed. The control signals of the external memory interface—R/$\overline{W}$, READY, $\overline{MSTRB}$, and $\overline{IOSTRB}$—are used to control the flow of data to and from external memory devices.

2.6.6 Hardware Timer

The DSP subsystem features one independent software-programmable timer. Three memory-mapped registers (MMRs) control the operation of the timer. These registers are: the timer control register (TCR), the timer period register (PRD), and the timer counter register (TIM). The timer resolution is the CPU clock rate of the DSP. The timer has a 20-bit dynamic range. The timer consists of a programmable 16-bit main counter (TIM), and a programmable 4-bit prescalar. The main counter is driven by the 4-bit prescalar, which decrements by one at every CPU clock. Once the prescalar reaches zero, the 16-bit counter decrements by one. When the 16-bit counter decrements to zero, a maskable interrupt (TINT) is generated, and the counter is reloaded with the period value defined in the PRD. The timer can be stopped, restarted, reset, or disabled via the bits of the timer control register (TCR).

2.7 ARM Core Overview (ARM7TDMIE)

What does ARM7TDMIE stand for?

ARM™: Advanced RISC Machines

7: Version number of the architecture

T: Thumb™: 32-bit-wide instruction words, 16-bit-wide memory

D: Debug: two breakpoints to stop the CPU (both hardware and software)

M: Multiplier: Has one multiplier

I: Interface: JTAG (Joint Test Action Group). Archaic name for serial scan standard prior to IEEE sanctioning as IEEE 1149.1

E: Emulation extension. Texas Instruments (TI) calls it ICECrusher™

2.7.1 ARM7TDMI Overview

The ARM7TDMI processor core is a member of the ARM7™ Thumb™ family. It is a low-power 32-bit RISC processor incorporating the Thumb 16-bit compressed instruction set. This microprocessor works in 32-bit or 16-bit instructions and on 32-, 16- or 8-bit data. The excellent code density achieved with Thumb leads to system cost reductions by reducing the required memory size and achieving 32-bit system performance from 16-bit-wide memories.

Thumb is a unique architectural strategy employed by the ARM7TDMI processor. It makes the ARM microprocessor ideally suited to high-volume applications with memory restrictions, or applications where code density is an issue.

The key idea behind Thumb is that of a super-reduced instruction set. Essentially, the ARM7TDMI processor has two instruction sets:

- ❏ The standard 32-bit ARM set
- ❏ A 16-bit Thumb set

The Thumb set's 16-bit instruction length allows it to approach twice the density of standard ARM code while retaining most of the ARM's performance advantage over a traditional 16-bit processor using 16-bit registers. This is possible because Thumb code operates on the same 32-bit register set as ARM code. Thumb code is able to occupy as little as 65% of the code size of ARM, and provide up to 160% of the performance of an equivalent ARM processor connected to a 16-bit memory system. This enables the development of applications with increased functionality and performance while maintaining competitive system cost and power consumption.

The ARM7TDMI ICEBreaker™ module provides integrated on-chip debug support for the ARM7TDMI core. Used with ARM's software development toolkit and the Multi-ICE™ interface unit, the ICEBreaker logic allows source-level debug, code download, instruction and data breakpoints even when the ARM processor is integrated within a larger chip. (Note: The name ICEBreaker has changed and is now known as the EmbeddedICE™ macrocell.)

2.7.2 ARM7TDMIE

ARM7TDMIE refers to the module formed by combining the ARM7TDMI (Thumb) processor module and the ICECrusher emulation module. Combining the ARM and ICECrusher modules provides debug support which is an enhanced version of the emulation support found on the ARM7TDMI module in its ICEBreaker block. The functions of the ICECrusher module are accessed through the JTAG port of the Thumb module; the emulation options and modes are selected by writing into ARM internal shadow registers.

2.7.3 ARM7TDMIE Emulation Features

The ARM7TDMIE core emulation capability is a combination of the individual ARM7TDMI and ICECrusher module capabilities.

The overall debug features are:

- ❏ Single processor and multiprocessor debug
- ❏ High-level language and assembly debug (run, halt, step...)
- ❏ Real-time (CPU continuously running) or non-real-time (CPU stopped) debug options
- ❏ Combined 32- and 16-bit modes for ARM processor
- ❏ Endianness transparency
- ❏ Unlimited breakpoints via op-code replacement (software breakpoint)
- ❏ Two hardware breakpoints (one configurable as software breakpoint) with maskable cycle type, address and data compare
- ❏ Two external breakpoint events
- ❏ Internal events generate external triggers
- ❏ Benchmarking/profiling capability

2.8 ARM Memory Space

ARM memory space is shared between the external memory interface, the SDRAM interface, and the ARM peripherals (see Table 2–2).

The memory interface provides five chip-select signals, while the SDRAM interface provides one chip-select signal.

Table 2–2. ARM Memory Space

Name	Start Address	Stop Address	Size in Bytes	Data Access
$\overline{\text{CS0}}$	0000:0000	007F:FFFF	8M	8/16/32
$\overline{\text{CS1}}$	0080:0000	00FF:FFFF	8M	8/16/32
$\overline{\text{CS2}}$	0100:0000	017F:FFFF	8M	8/16/32
$\overline{\text{CS3}}$	0180:0000	01FF:FFFF	8M	8/16/32
CS4	0200:0000	027F:FFFF	8M	8/16/32
Reserved	0280:0000	0FFF:FFFF		
$\overline{\text{SDRAM\_CS}}$	1000:0000	11FF:FFFF	32M	8/16/32
Reserved	1200:0000	FFBF:FFFF		
Internal SRAM	FFC0:0000	FFC0:3FFF	16K	8/16/32
Reserved	FFC0:4000	FFCF:FFFF		
EIM SRAM (VC5471) Reserved (VC5470)	FFD0:0000	FFD0:3FFF	16K	8/16/32
Reserved	FFD0:4000	FFDF:FFFF		
API RAM	FFE0:0000	FFE0:3FFF	16K	16/32
Reserved	FFE0:4000	FFF3:FFFF		
API registers	FFE4:0000	FFE4:0001	2	16
Reserved	FFE4:0002	FFFE:FFFF		
EIM (VC5471) Reserved (VC5470)	FFFF:0000	FFFF:01FF	512	32
Reserved	FFFF:0200	FFFF:07FF		
UART_IRDA	FFFF:0800	FFFF:0FFF	2K	32
UART	FFFF:1000	FFFF:17FF	2K	32

Table 2–2. ARM Memory Space (Continued)

Name	Start Address	Stop Address	Size in Bytes	Data Access
I2C	FFFF:1800	FFFF:1FFF	2K	32
SPI	FFFF:2000	FFFF:27FF	2K	32
GPIO	FFFF:2800	FFFF:28FF	256	32
KBGPIO	FFFF:2900	FFFF:29FF	256	32
TIMER0	FFFF:2A00	FFFF:2AFF	256	32
TIMER1	FFFF:2B00	FFFF:2BFF	256	32
TIMER2	FFFF:2C00	FFFF:2CFF	256	32
INTH	FFFF:2D00	FFFF:2DFF	256	32
MEMINT	FFFF:2E00	FFFF:2EFF	256	32
CLKM	FFFF:2F00	FFFF:2FFF	256	32
SDRAMIF	FFFF:3000	FFFF:30FF	256	32
ARM_PLL	FFFF:3200	FFFF:32FF	256	32
Reserved	FFFF:3300	FFFF:FFFF		

2.9 ARM Registers

As noted in Table 2–3, not all of the ARM peripheral memory-mapped registers have the same base address.

Table 2–3. ARM Peripheral Memory-Mapped Registers

Register	Base Address	Offset	Description	Type
APIC	0xFFE4:0000	0x0000	API Control Register	API
EIM_SCR (VC5471)	0xFFFF:0000	0x0000	EIM Ethernet State Machine (ESM) Control Register	EIM
EIM_SR (VC5471)	0xFFFF:0000	0x0004	EIM Status Register	EIM
EIM_CPUTX_BAR (VC5471)	0xFFFF:0000	0x0008	CPU TX Descriptors Base Address Register	EIM
EIM_CPURX_BAR (VC5471)	0xFFFF:0000	0x000C	CPU RX Descriptors Base Address Register	EIM
EIM_BSR (VC5471)	0xFFFF:0000	0x0010	Packet Buffer Size Register	EIM
EIM_CPUFCR (VC5471)	0xFFFF:0000	0x0014	CPU Filtering Control Register	EIM
EIM_CPUDAR1 (VC5471)	0xFFFF:0000	0x0018	CPU Destination Address Register 1	EIM
EIM_CPUDAR0 (VC5471)	0xFFFF:0000	0x001C	CPU Destination Address Register 0	EIM
EIM_MFV1 (VC5471)	0xFFFF:0000	0x0020	Multicast Filter Valid Mask Register 1	EIM
EIM_MFV0 (VC5471)	0xFFFF:0000	0x0024	Multicast Filter Valid Mask Register 0	EIM
EIM_MFM1 (VC5471)	0xFFFF:0000	0x0028	Multicast Filter Mask Register 1	EIM
EIM_MFM0 (VC5471)	0xFFFF:0000	0x002C	Multicast Filter Mask Register 0	EIM
EIM_RXTR (VC5471)	0xFFFF:0000	0x0030	RX Threshold Register	EIM
EIM_CPURXRDY (VC5471)	0xFFFF:0000	0x0034	RX CPU Ready Register	EIM
EIM_IER (VC5471)	0xFFFF:0000	0x0038	EIM Interrupt Enable Register	EIM
EIM_ENET0_TX (VC5471)	0xFFFF:0000	0x003C	Pointer to next free descriptor in ENET0 TX queue	EIM
EIM_ENET0_RX (VC5471)	0xFFFF:0000	0x0040	Pointer to next free descriptor in ENET0 RX queue	EIM
Reserved	0xFFFF:0000	0x0044–0x0048		

Table 2–3. ARM Peripheral Memory-Mapped Registers (Continued)

Register	Base Address	Offset	Description	Type
EIM_CPU_TX (VC5471)	0xFFFF:0000	0x004C	Pointer to next free descriptor in CPU TX queue	EIM
EIM_CPU_RX (VC5471)	0xFFFF:0000	0x0050	Pointer to next free descriptor in CPU RX queue	EIM
EIM_MODE_E0 (VC5471)	0xFFFF:0000	0x0100	ENET0 Mode Register	EIM ENET0
EIM_BOSEED_E0 (VC5471)	0xFFFF:0000	0x0104	Random Backoff Seed Register	EIM ENET0
EIM_BOCNT_E0 (VC5471)	0xFFFF:0000	0x0108	Random Backoff Count Register (reads the current random backoff)	EIM ENET0
EIM_FLWPAU_E0 (VC5471)	0xFFFF:0000	0x010C	Flow Control Pause Count Register (pause count value to be transmitted as a flow control frame)	EIM ENET0
EIM_FLWCNT_E0 (VC5471)	0xFFFF:0000	0x0110	ENET0 Flow Control Register	EIM ENET0
EIM_VTYPE_E0 (VC5471)	0xFFFF:0000	0x0114	Virtual LAN tagged frame compare value	EIM ENET0
EIM_SYSERR_E0 (VC5471)	0xFFFF:0000	0x0118	System Error Interrupt Status Register	EIM ENET0
EIM_TXBUF_RDY_E0 (VC5471)	0xFFFF:0000	0x011C	There is no register associated with the Transmit Descriptor Buffer Ready address. Writing to this address is used to prompt the ENET0 DMA controller to start processing the next transmit descriptor.	EIM ENET0
EIM_TDBA_E0 (VC5471)	0xFFFF:0000	0x0120	Transmit Descriptor Base Address Register	EIM ENET0
EIM_RDBA_E0 (VC5471)	0xFFFF:0000	0x0124	Receive Descriptor Base Address Register	EIM ENET0
EIM_PAR1_E0 (VC5471)	0xFFFF:0000	0x0128	Destination Physical Address Register bits 47:32	EIM ENET0
EIM_PAR0_E0 (VC5471)	0xFFFF:0000	0x012C	Destination Physical Address Register bits 31:0	EIM ENET0
EIM_LAR1_E0 (VC5471)	0xFFFF:0000	0x0130	Logical Hash Filter Address Register bits 63:32	EIM ENET0
EIM_LAR0_E0 (VC5471)	0xFFFF:0000	0x0134	Logical Hash Filter Address Register bits 31:0	EIM ENET0
EIM_ADD_E0 (VC5471)	0xFFFF:0000	0x0138	Address Mode Enable Register	EIM ENET0
EIM_DRPC_E0 (VC5471)	0xFFFF:0000	0x013C	Descriptor Ring Poll Interval Count Register	EIM ENET0

Table 2–3. ARM Peripheral Memory-Mapped Registers (Continued)

Register	Base Address	Offset	Description	Type
EIM_RAA_E0 (VC5471)	0xFFFF:0000	0x0140	Read Abort Address Register	EIM ENET0
Reserved	0xFFFF:0000	0x0200–0x0240		
UART_IRDA_RHR	0xFFFF:0800	0x0000	Receive Holding Register	UART IRDA
UART_IRDA_THR	0xFFFF:0800	0x0004	Transmit Holding Register	UART IRDA
UART_IRDA_FCR	0xFFFF:0800	0x0008	FIFO Control Register	UART IRDA
UART_IRDA_SCR	0xFFFF:0800	0x000C	Status Control Register	UART IRDA
UART_IRDA_LCR	0xFFFF:0800	0x0010	Line Control Register	UART IRDA
UART_IRDA_LSR (UART mode)	0xFFFF:0800	0x0014	Line Status Register (UART mode)	UART IRDA
UART_IRDA_LSR (SIR mode)	0xFFFF:0800	0x0014	Line Status Register (SIR mode)	UART IRDA
UART_IRDA_SSR	0xFFFF:0800	0x0018	Supplementary Status Register	UART IRDA
UART_IRDA_MCR (UART mode only)	0xFFFF:0800	0x001C	Modem Control Register	UART IRDA
UART_IRDA_MSR	0xFFFF:0800	0x0020	Modem Status Register	UART IRDA
UART_IRDA_IER (UART mode)	0xFFFF:0800	0x0024	Interrupt Enable Register (UART mode)	UART IRDA
UART_IRDA_IER (SIR mode)	0xFFFF:0800	0x0024	Interrupt Enable Register (SIR mode)	UART IRDA
UART_IRDA_ISR (UART mode)	0xFFFF:0800	0x0028	Interrupt Status Register (UART mode)	UART IRDA
UART_IRDA_ISR (SIR mode)	0xFFFF:0800	0x0028	Interrupt Status Register (SIR mode)	UART IRDA
UART_IRDA_EFR	0xFFFF:0800	0x002C	Enhanced Feature Register	UART IRDA
UART_IRDA_XON1	0xFFFF:0800	0x0030	XON1 Character Register	UART IRDA
UART_IRDA_XON2	0xFFFF:0800	0x0034	XON2 Character Register	UART IRDA
UART_IRDA_XOFF1	0xFFFF:0800	0x0038	XOFF1 Character Register	UART IRDA
UART_IRDA_XOFF2	0xFFFF:0800	0x003C	XOFF2 Character Register	UART IRDA
UART_IRDA_SPR	0xFFFF:0800	0x0040	Scratch-pad Register	UART IRDA
UART_IRDA_DIV_115K	0xFFFF:0800	0x0044	Divisor for 115 k-baud generation	UART IRDA

Table 2–3. ARM Peripheral Memory-Mapped Registers (Continued)

Register	Base Address	Offset	Description	Type
UART_IRDA_DIV_BITRATE	0xFFFF:0800	0x0048	Divisor for baud rate generation	UART IRDA
UART_IRDA_TCR	0xFFFF:0800	0x004C	Transmission Control Register	UART IRDA
UART_IRDA_TLR	0xFFFF:0800	0x0050	Trigger Level Register	UART IRDA
UART_IRDA_MDR1	0xFFFF:0800	0x0054	Mode Definition Register 1	UART IRDA
UART_IRDA_MDR2	0xFFFF:0800	0x0058	Mode Definition Register 2	UART IRDA
UART_IRDA_TXFLL	0xFFFF:0800	0x005C	Transmit Frame Length Register (LSB)	UART IRDA
UART_IRDA_TXFLH	0xFFFF:0800	0x0060	Transmit Frame Length Register (MSB)	UART IRDA
UART_IRDA_RXFLL	0xFFFF:0800	0x0064	Received Frame Length Register (LSB)	UART IRDA
UART_IRDA_RXFLH	0xFFFF:0800	0x0068	Received Frame Length Register (MSB)	UART IRDA
UART_IRDA_SFLSR	0xFFFF:0800	0x006C	Status FIFO Line Status Register	UART IRDA
UART_IRDA_SFREGL	0xFFFF:0800	0x0070	Status FIFO Register (LS part of the received frame)	UART IRDA
UART_IRDA_SFREGH	0xFFFF:0800	0x0074	Status FIFO Register (MS part of the received frame)	UART IRDA
UART_IRDA_BLR	0xFFFF:0800	0x0078	Begin of File Length Register	UART IRDA
UART_IRDA_PULSE_WIDTH	0xFFFF:0800	0x007C	Pulse Width Register	UART IRDA
UART_IRDA_ACREG	0xFFFF:0800	0x0080	Auxiliary Control Register	UART IRDA
UART_IRDA_PULSE_START	0xFFFF:0800	0x0084	Starting time of the pulse (offset within the pulse period) expressed in number of ARM/*div_115k* clock cycles.	UART IRDA
UART_IRDA_RX_W_PTR	0xFFFF:0800	0x0088	RX FIFO write pointer	UART IRDA
UART_IRDA_RX_R_PTR	0xFFFF:0800	0x008C	RX FIFO read pointer	UART IRDA
UART_IRDA_TX_W_PTR	0xFFFF:0800	0x0090	TX FIFO write pointer	UART IRDA
UART_IRDA_TX_R_PTR	0xFFFF:0800	0x0094	TX FIFO read pointer	UART IRDA
UART_IRDA_ STATUS_W_PTR	0xFFFF:0800	0x0098	Write pointer of status FIFO	UART IRDA
UART_IRDA_ STATUS_R_PTR	0xFFFF:0800	0x009C	Read pointer of status FIFO	UART IRDA

Table 2–3. ARM Peripheral Memory-Mapped Registers (Continued)

Register	Base Address	Offset	Description	Type
UART_IRDA_RESUME	0xFFFF:0800	0x00A0	Resume Register (Dummy read to restart the transmission or reception)	UART IRDA
UART_IRDA_MUX	0xFFFF:0800	0x00A4	Selects the UART_IRDA output pin multiplexing	UART IRDA
UART_RHR	0xFFFF:1000	0x0000	Receive Holding Register	UART Modem
UART_THR	0xFFFF:1000	0x0004	Transmit Holding Register	UART Modem
UART_FCR	0xFFFF:1000	0x0008	FIFO Control Register	UART Modem
UART_SCR	0xFFFF:1000	0x000C	Status Control Register	UART Modem
UART_LCR	0xFFFF:1000	0x0010	Line Control Register	UART Modem
UART_LSR	0xFFFF:1000	0x0014	Line Status Register	UART Modem
UART_SSR	0xFFFF:1000	0x0018	Supplementary Status Register	UART Modem
UART_MCR	0xFFFF:1000	0x001C	Modem Control Register	UART Modem
UART_MSR	0xFFFF:1000	0x0020	Modem Status Register	UART Modem
UART_IER	0xFFFF:1000	0x0024	Interrupt Enable Register	UART Modem
UART_ISR	0xFFFF:1000	0x0028	Interrupt Status Register	UART Modem
UART_EFR	0xFFFF:1000	0x002C	Enhanced Feature Register	UART Modem
UART_XON1	0xFFFF:1000	0x0030	XON1 Character Register	UART Modem
UART_XON2	0xFFFF:1000	0x0034	XON2 Character Register	UART Modem
UART_XOFF1	0xFFFF:1000	0x0038	XOFF1 Character Register	UART Modem
UART_XOFF2	0xFFFF:1000	0x003C	XOFF2 Character Register	UART Modem
UART_SPR	0xFFFF:1000	0x0040	Scratch-pad Register	UART Modem
UART_DIV_115K	0xFFFF:1000	0x0044	Divisor for 115 kbauds generation	UART Modem
UART_DIV_BITRATE	0xFFFF:1000	0x0048	Divisor for baud rate generation	UART Modem
UART_TCR	0xFFFF:1000	0x004C	Transmission Control Register	UART Modem
UART_TLR	0xFFFF:1000	0x0050	Trigger Level Register	UART Modem
UART_MDR	0xFFFF:1000	0x0054	Mode Definition Register	UART Modem
UART_UASR	0xFFFF:1000	0x0058	UART Autobauding Status Register	UART Modem
UART_RDPTR_URX	0xFFFF:1000	0x005C	RX FIFO Read Pointer Register	UART Modem

Table 2–3. ARM Peripheral Memory-Mapped Registers (Continued)

Register	Base Address	Offset	Description	Type
UART_WRPTR_URX	0xFFFF:1000	0x0060	RX FIFO Write Pointer Register	UART Modem
UART_RDPTR_UTX	0xFFFF:1000	0x0064	TX FIFO Read Pointer Register	UART Modem
UART_WRPTR_UTX	0xFFFF:1000	0x0068	TX FIFO Write Pointer Register	UART Modem
DEVICE_REG	0xFFFF:1800	0x0000	Device Register (Device identification code for I^2C slave device)	I^2C
ADDRESS_REG	0xFFFF:1800	0x0004	Address Register (I^2C slave device internal register address)	I^2C
DATA_WRITE	0xFFFF:1800	0x0008	Data Write Register (Data to write on I^2C bus)	I^2C
DATA_READ	0xFFFF:1800	0x000C	Data Read Register (Data to read on I^2C bus)	I^2C
CMD_REG	0xFFFF:1800	0x0010	Command Register	I^2C
CONF_FIFO	0xFFFF:1800	0x0014	Configuration FIFO Register	I^2C
CONF_CLK	0xFFFF:1800	0x0018	Configuration Clock Register	I^2C
CONF_CLK_REF	0xFFFF:1800	0x001C	Configuration Clock Functional Reference Register	I^2C
STATUS_FIFO_REG	0xFFFF:1800	0x0020	Status FIFO Register	I^2C
STATUS_ACTIVITY_REG	0xFFFF:1800	0x0024	Status Activity Register	I^2C
SPI_SET	0xFFFF:2000	0x0000	Set Up SPI (Register dedicated to the configuration of the serial port)	SPI
SPI_CTRL	0xFFFF:2000	0x0004	Control SPI (SPI control register)	SPI
SPI_STATUS	0xFFFF:2000	0x0008	Status Register	SPI
SPI_TX	0xFFFF:2000	0x000C	Transmit Register	SPI
SPI_RX	0xFFFF:2000	0x0010	Receive Register	SPI
GPIO_IO	0xFFFF:2800	0x0000	I/O bits: Writeable when I/O is configured as an output; reads value on I/O pin when I/O is configured as an input	GPIO
GPIO_CIO	0xFFFF:2800	0x0004	GPIO configuration register	GPIO
GPIO_IRQA	0xFFFF:2800	0x0008	In conjunction with GPIO_IRQB determines the behavior when GPIO pins configured as input IRQ	GPIO

Table 2–3. ARM Peripheral Memory-Mapped Registers (Continued)

Register	Base Address	Offset	Description	Type
GPIO_IRQB	0xFFFF:2800	0x000C	In conjunction with GPIO_IRQA determines the behavior when GPIO pins configured as input IRQ	GPIO
GPIO_DDIO	0xFFFF:2800	0x0010	Delta Detect Register (detects changes in the I/O pins)	GPIO
GPIO_EN	0xFFFF:2800	0x0014	Selects register for muxed GPIOs	GPIO
KBGPIO_IO	0xFFFF:2900	0x0000	Keyboard I/O bits: Writeable when KBGPIO is configured as an output; reads value on I/O pin when KBGPIO is configured as an input	KBGPIO
KBGPIO_CIO	0xFFFF:2900	0x0004	KBGPIO configuration register	KBGPIO
KBGPIO_IRQA	0xFFFF:2900	0x0008	In conjunction with KBGPIO_IRQB determines the behavior when KBGPIO pins configured as input IRQ	KBGPIO
KBGPIO_IRQB	0xFFFF:2900	0x000C	In conjunction with KBGPIO_IRQA determines the behavior when KBGPIO pins configured as input IRQ	KBGPIO
KBGPIO_DDIO	0xFFFF:2900	0x0010	Delta Detect Register (detects changes in the KBGPIO pins)	KBGPIO
KBGPIO_EN	0xFFFF:2900	0x0014	Selects register for muxed KBGPIOs	KBGPIO
CNTL_TIMER0	0xFFFF:2A00	0x0000	Control TIMER0 Register	TIMER0
VALUE_TIM0	0xFFFF:2A00	0x0004	Current value of TIMER0 register	TIMER0
CNTL_TIMER1	0xFFFF:2B00	0x0000	Control TIMER1 Register	TIMER1
VALUE_TIM1	0xFFFF:2B00	0x0004	Current value of TIMER1 register	TIMER1
CNTL_TIMER2	0xFFFF:2C00	0x0000	Control TIMER2 Register	TIMER2
VALUE_TIM2	0xFFFF:2C00	0x0004	Current value of TIMER2 register	TIMER2
IT_REG	0xFFFF:2D00	0x0000	Interrupt Register (It stores an incoming interrupt)	INTH
MASK_IT_REG	0xFFFF:2D00	0x0004	Mask Interrupt Register	INTH
SRC_IRQ_REG	0xFFFF:2D00	0x0008	Source IRQ Register (indicates the active IRQ interrupt)	INTH
SRC_FIQ_REG	0xFFFF:2D00	0x000C	Source FIQ Register (indicates the active FIQ interrupt)	INTH

Table 2–3. ARM Peripheral Memory-Mapped Registers (Continued)

Register	Base Address	Offset	Description	Type
SRC_IRQ_BIN_REG	0xFFFF:2D00	0x0010	Source IRQ (binary coded) Register (indicates the active interrupt – in order to save software processing time, this register indicates the interrupt # having requested a MCU action.)	INTH
INT_CTRL_REG	0xFFFF:2D00	0x0018	Interrupt control Register	INTH
ILR_IRQ_0	0xFFFF:2D00	0x001C	Interrupt Level Register0 (defines interrupt priority level for the corresponding interrupt)	INTH
ILR_IRQ_1	0xFFFF:2D00	0x0020	Interrupt Level Register1	INTH
ILR_IRQ_2	0xFFFF:2D00	0x0024	Interrupt Level Register2	INTH
ILR_IRQ_3	0xFFFF:2D00	0x0028	Interrupt Level Register3	INTH
ILR_IRQ_4	0xFFFF:2D00	0x002C	Interrupt Level Register4	INTH
ILR_IRQ_5	0xFFFF:2D00	0x0030	Interrupt Level Register5	INTH
ILR_IRQ_6	0xFFFF:2D00	0x0034	Interrupt Level Register6	INTH
ILR_IRQ_7	0xFFFF:2D00	0x0038	Interrupt Level Register7	INTH
ILR_IRQ_8	0xFFFF:2D00	0x003C	Interrupt Level Register8	INTH
ILR_IRQ_9	0xFFFF:2D00	0x0040	Interrupt Level Register9	INTH
ILR_IRQ_10	0xFFFF:2D00	0x0044	Interrupt Level Register10	INTH
ILR_IRQ_11	0xFFFF:2D00	0x0048	Interrupt Level Register11	INTH
ILR_IRQ_12	0xFFFF:2D00	0x004C	Interrupt Level Register12	INTH
ILR_IRQ_13	0xFFFF:2D00	0x0050	Interrupt Level Register13	INTH
ILR_IRQ_14	0xFFFF:2D00	0x0054	Interrupt Level Register14	INTH
ILR_IRQ_15	0xFFFF:2D00	0x0058	Interrupt Level Register15	INTH
IRQ_SLEEP_REG	0xFFFF:2D00	0x005C	IRQ sleep register	INTH
CS0_REG	0xFFFF:2E00	0x0000	External memory control register for CS0 memory range	MEMINT
CS1_REG	0xFFFF:2E00	0x0004	External memory control register for CS1 memory range	MEMINT
CS2_REG	0xFFFF:2E00	0x0008	External memory control register for CS2 memory range	MEMINT

Table 2–3. ARM Peripheral Memory-Mapped Registers (Continued)

Register	Base Address	Offset	Description	Type
CS3_REG	0xFFFF:2E00	0x000C	External memory control register for CS3 memory range	MEMINT
CS4_REG	0xFFFF:2E00	0x0010	External memory control register for CS4 memory range	MEMINT
API_REG	0xFFFF:2E00	0x0014	ARM port interface wait-state configuration register	MEMINT
SDRAM_REG	0xFFFF:2E00	0x0018	SDRAM data bus size control register	MEMINT
BS_CONFIG	0xFFFF:2E00	0x001C	Bank-switching configuration register (indicates the number of dummy cycles for insertion when changing chip select)	MEMINT
CLKM_REG	0xFFFF:2F00	0x0000	Clock configuration register (individually enables/disables clocking for ARM and its peripherals)	CLKM
DSP_REG	0xFFFF:2F00	0x0004	DSP PLL Register	CLKM
WAKEUP_REG	0xFFFF:2F00	0x0008	Interrupt Clock Wakeup Register (keeps a copy of the normal operation configuration; upon waking up, values in WAKEUP_REG will get copied into CLKM_REG)	CLKM
AUDIO_CLK	0xFFFF:2F00	0x000C	Defines the divisor for generating the AudioCLK output from the ARM_PLL output.	CLKM
CLKM_CNTL_RESET	0xFFFF:2F00	0x0010	Reset control register (defines the reset signal to the DSP and RESET_OUT pin).	CLKM
WATCHDOG_STATUS	0xFFFF:2F00	0x0014	Indicates whether a watchdog timeout has occurred.	CLKM
RESET_REG	0xFFFF:2F00	0x0018	Reset register (defines the reset signal to the ARM peripherals)	CLKM
LOW_POWER_REG	0xFFFF:2F00	0x001C	Indicates whether the ARM is clocking at the ARM PLL rate or at REFCLK/512.	CLKM
LOW_POWER_REG_VALUE	0xFFFF:2F00	0x0020	Low-power value register	CLKM
SDRAM_CONFIG	0xFFFF:3000	0x0000	SDRAM configuration register	SDRAMIF
SDRAM_REF_COUNT	0xFFFF:3000	0x0004	SDRAM refresh counter register	SDRAMIF
SDRAM_CNTL	0xFFFF:3000	0x0008	SDRAM control register	SDRAMIF

Table 2–3. ARM Peripheral Memory-Mapped Registers (Continued)

Register	Base Address	Offset	Description	Type
SDRAM_INIT_CONF	0xFFFF:3000	0x000C	SDRAM initialization refresh counter register	SDRAMIF
ARM_PLL_REG	0xFFFF:3200	0x0000	ARM PLL configuration register	ARM_PLL

2.10 ARM Peripherals

2.10.1 ARM Memory Interface (MEMINT)

The ARM memory interface handles:

- External ARM memory access management. It performs:
  - ARM **read and write** access size adaptation to the memory width (from 8-bit up to 32-bit)
  - ARM access duration management (wait state insertion) to enable the connection of slow memory devices
  - Memory control signals generation (chip-selects, write strobe generation, ...) with five chip-select signals, each corresponding to an address range of eight megabytes
- ARM-to-API memory access management:
  - ARM access size adaptation for **API read and write access**. A 32-bit API READ transaction is transformed into two 16-bit read accesses. A 32-bit API WRITE access is transformed into two 16-bit write transactions.
  - Address signal timing adaptation to be compliant with the API interface requirement
- ARM $\overline{\text{WAIT}}$ and access control flags generation, such as byte-latch, etc. (This function is transparent to the user.)

The memory interface is designed to work with the ARM processor in Little-Endian or Big-Endian mode depending on the chip input, BIGEND. Each chip-select memory range can be configured in either Little- or Big-Endian mode.

2.10.2 SDRAM Memory Interface (SDRAMIF)

The SDRAM interface module effectively sits between the ARM processor and the SDRAM controller and functions as an isolation module between the two. It belongs to the memory interface since it interfaces with the MEMINIT and generates signals that are of memory interface responsibilities. The main features of the SDRAM memory interface are:

- Operates with the ARM memory interface (MEMINT) so that SDRAM memories can be used on the same board with Flash and/or SRAM.
- Supports 32-bit-wide and 16-bit wide SDRAM interfaces (32-bit data configuration can be obtained by connecting with a 32-bit-wide SDRAM or by connecting two 16-bit SDRAMs in parallel)

- ❏ 256M-byte chip-enabled space
- ❏ Operates at the ARM clock speed
- ❏ Very flexible programming of SDRAM timing parameters
- ❏ Supports four open pages of SDRAM

2.10.3 Interrupt Handler (INTH)

The interrupt handler provides up to 16 prioritized and maskable interrupts (IRQ0–15) to the ARM core. It receives interrupts from both internal modules and external chip environment via the GPIO pins. Each incoming interrupt can be individually masked using dedicated configuration registers.

An Interrupt Level Register is associated with each incoming interrupt to define a priority to the corresponding interrupt. If several interrupts have the same priority level, they are sent in a predefined order.

Each interrupt can be routed to one of the two input interrupts of the ARM core: fast interrupt request (FIQ) and low-priority interrupt request (IRQ).

For details of how the on-chip peripherals are associated with the IRQ lines, see Figure 4–1 on page 4-5.

2.10.4 ARM General-Purpose I/O (GPIO)

Thirty-six general-purpose I/Os (GPIOs), configurable in read or write mode by ARM memory-mapped registers, are provided. Some of the GPIOs are shared with other signals. Each GPIO is associated with six configuration/status bits whose description is given in Table 2–4 and Table 2–5.

Table 2–4. GPIO Control/Status Bits

Bit Name	Description
io	I/O bit. Writeable when I/O is configured as an output (*cio* = 0). Reads value on I/O pin when I/O is configured as an input (*cio* = 1)
cio	Configure I/O 0: output 1: input (default)
gpio_irqA	See Table 2–5
gpio_irqB	See Table 2–5
ddio	Delta detect bit. If gpio configured as output (*cio*=0), always reads as 0. If gpio configured as input (*cio*=1), reads a 1 if *io* has changed since *ddio* was last cleared.
gpio_en	Selects register for muxed GPIOs. 0: other signal (default) 1: gpio Non-shared GPIOs are always available at the I/O pin independently of the value of *gpio_en.*

Table 2–5. GPIO_IRQ Bit Definitions

gpio_irqA	gpio_irqB	Function
0	0	Disable IRQ
0	1	An IRQ is generated on the rising edge.
1	0	An IRQ is generated on the falling edge.
1	1	An IRQ is generated on the state change.

The configuration/status bits are accessible through 12 memory-mapped registers: GPIO_IO, KBGPIO_IO, GPIO_CIO, KBGPIO_CIO, GPIO_IRQA, KBGPIO_IRQA, GPIO_IRQB, KBGPIO_IRQB, GPIO_DDIO, KBGPIO_DDIO, GPIO_EN, and KBGPIO_EN.

The 36 GPIOs are divided into two groups: GPIO(19:0) and KBGPIO(15:0). KBGPIOs are keyboard GPIO pins. GPIO(19:0) and KBGPIO(15:0) are basically the same except that some of the keyboard GPIO pins have pullup resistors.

KBGPIO(15:8) have on-chip pullup resistors connected to their input/output pins. KBGPIO(15:0) can be used as normal GPIO pins or be configured for connection of a 8 x 8 keyboard matrix.

2.10.5 Timers (TIMERs)

The VC547x implements three 16-bit timers configurable either in *auto-reload* or in *one-shot* modes. The timers generate interrupts to the ARM when equal to zero.

The first timer (TIMER0) is configured by default as a watchdog for the micro-controller unit (MCU). If this functionality is not required, a specific sequence must be written into a dedicated register in order to configure the watchdog as a general-purpose timer.

The two others (TIMER1 and TIMER2) are general-purpose timers. TIMER2 has a dedicated output pin (TIMER_OUT) which generates a low pulse when a TIMER2 interrupt occurs.

2.10.6 IRDA Universal Asynchronous Receiver/Transmitter 16C750 (UART-IRDA)

This UART interface is compatible with 16C750-compliant devices. It includes the slow infrared (SIR) protocol in order to be connected with an infrared transmitter to any external data peripherals with an IrDA-compliant data interface.

The IR function can be disabled and the UART connected through a standard wired interface.

This UART can be linked to an external PC for concurrent debugging purposes (software flow control only).

The UART IRDA module integrates two 64-word (9 and 11 bits) receive and transmit FIFOs and one 8-word (16 bits) status FIFO with programmable trigger levels. The baud rate is internally generated from a programmable divisor. Transmission parity can be even, odd, or without parity, and the number of stop bits is 1, 1.5, or 2.

The receiver can detect break, idle, framing, and parity errors as well as FIFO overflow. The transmitter can detect FIFO underflow.

All modem operations are controllable via a software interface. In IrDA mode, this upgraded UART 16C750 includes the following additional features:

- ❏ IrDA 1.0 SIR support: allows serial communication at baud rates up to 115.2 Kbit/s
- ❏ Pulse shaping, and pulse recovering: A zero is signaled by sending a single infrared pulse. A one is signaled by not sending any pulse.

- ❏ The device operation, in IrDA 1.0 SIR, is similar to the operation in UART mode. The main difference is that the data transfer operations are normally performed in half-duplex, and the modem control and status signals are not used
- ❏ Frame formatting: addition of variable beginning-of-frame (xBOF) characters and end-of-frame (EOF) characters
- ❏ Uplink/downlink cyclic redundancy check (CRC) generation/detection
- ❏ Asynchronous transparency (automatic insertion of break character)
- ❏ 8-character status FIFO available to monitor frame length and frame errors
- ❏ Variable frame length for RX and TX IrDA frame

Note: MIR and FIR are not implemented.

2.10.7 Universal Asynchronous Receiver/Transmitter 16C750 (UART-Modem)

This UART 16C750 interface is compatible with the NS 16C750 device and is devoted to the connection of a PC-based software debugger tool through a standard wired interface.

The UART Modem module integrates two 64-word (9 and 11 bits) receive and transmit FIFOs with programmable trigger levels. The baud rate is internally generated from a programmable divisor.

Transmission parity can be even, odd, or without parity, and the number of stop bits is 1, 1.5, or 2.

The receiver can detect break, idle, framing, and parity errors as well as FIFO overflow. The transmitter can detect FIFO underflow.

All operations are controllable either via a software interface or by using hardware flow control signals.

This upgraded UART 16C750 includes the following additional features:

- ❏ Hardware flow control (DCD, RTS/CTS)
- ❏ Auto-bauding with the possibility to match the baud rate from 1200 to 115.2 Kbits/s.

2.10.8 Serial Peripheral Interface (SPI)

The serial interface is a bidirectional 3-line interface dedicated to the transfer of data to and from external devices offering a 3-line serial interface.

The SPI interface is full-duplex and is configurable from 1 to 32 bits, providing three enable signals programmable either as positive/negative active or edge-/level-sensitive.

This serial port is based on a looped shift-register, thus allowing both transmit (PISO) and receive (SIPO) modes. The serial port is fully controlled by the ARM Memory Interface (data write, data read, and activation of serialization operations).

The serial clock period (T_{CLKX_SPI}) is derived from the SPI_clock:

$$T_{CLKX\_SPI} = \frac{T_{SPI\_clock}}{4 * PTV} = \frac{T_{SPI\_clock}}{4 * [(1)(2)(4)(8)16]}$$

PTV is a programmable value of a prescale clock divider in register SPI_SET and can take the following values: 1, 2, 4, 8, or 16. For example, if SPI_clock is equal to 47.5 MHz, CLKX_SPI can be set to 11.875, 5.9375, 2.96875, 1.484375, or 0.7421875 MHz.

2.10.9 Ethernet Interface Module (EIM) (VC5471)

The Ethernet interface module provides a straightforward and effective method of integrating IEEE802.3/Ethernet MAC functionality onto a processor I/O subsystem. The Ethernet interface module provides:

- ❏ Memory-mapped interface for configuration
- ❏ Ring buffer chaining
- ❏ RX and TX status reporting
- ❏ 10- and 100-Mbit/s data rate
- ❏ Internal and external loopback
- ❏ IEEE802.3/Ethernet MAC with MII for Ethernet connection
- ❏ Address filtering (Unicast, Multicast, Broadcast, or Promiscuous mode)

2.10.10 Master Inter-Integrated Circuit (I^2C) Interface

The Master I^2C Interface Module provides an interface between the ARM bus and the I^2C bus. It is through the Master I^2C Interface Module that the ARM bus controls the external peripheral devices on the I^2C bus. The Master I^2C Interface Module is compatible with TI I^2C-compliant devices and implements the serial I^2C-bus protocol.

The Master I2C Interface Module supports I2C Master-Only mode with:

- A 7-bit address device
- An 8-bit subaddress
- Master write to slave receiver in single or multiple mode (data loop)
- A 16-byte transmit FIFO
- Master simple read to slave receiver
- Read Combined cycle
- A 3-bit programmable prescale internal clock divider and 7-bit programmable SCL clock divider to support a wide range of input clock frequencies. The I2C SCL clock frequency are:
  - I2C Standard Mode: 100 kHz
  - I2C Fast Mode: 400 kHz
- 3-bit programmable spike filter to provide noise filtering on the I2C input signal

The module does not support:

- I2C bus 10-bit addressing
- I2C bus CBUS compatibility
- Multi-Master I2C
- Clock synchronization as a handshake: slave devices are NOT allowed to hold the SCL line LOW to force the master (VC547x) into a wait state until the slave is ready for the next transfer.

2.10.11 Clock Management (CLKM)

This module is in charge of the control of the clock activity for the DSP, MCU, and peripherals. It includes configuration registers for DSP and MCU clock frequencies programming.

CLKM also manages the reset of all modules connected to the MCU.

The clock management scheme is shown in Chapter 5.

2.11 General-Purpose Peripherals

The Joint Test Action Group (JTAG) port is basically used for two purposes: TI debug tools and ARM/DSP testing with boundary scan. The JTAG interface of the chip can be selected in either one of the following two ways:

1) To access the two processors' on-chip emulators with a pseudo IEEE JTAG protocol for emulation purposes. A PC or Workstation can be connected to the interface to set the bi-emulation mode with the ARM core linked to the DSP core. The ICECrusher module supports the synchronization of the two cores.

   EMU0/1 pins are driven internally by the hardware breakpoints, which are set by the TI debug tools. This remains high until a breakpoint is hit, at which point, the tools can be set up to pull the EMU0/1 signals low. The TI debugger takes care of putting them back to their respective levels.

   The devices can also be set up to react to sensing a low EMU0/1 by setting these parameters within the debug tools. The reaction that occurs within the tools is to break the core for debug purposes. In this perspective, JTAG functions exactly as a DEBUG port.

2) To dialog with an embedded TAP controller with the instruction set supports internal SCAN modes used for ARM testing, DSP testing, and logic testing. JTAG boundary scan and JTAG bypass mode are also supported.

   Note that the JTAG port is not intended to be used as general-purpose I/O (GPIO) ports.

2.12 Clock Frequencies

2.12.1 DSP Clock

The C54x DSP core clock frequency is derived from the input clock REFCLK and the frequency ratio multiplier k (where k is a function of n/m), as defined in Table 2–6.

Table 2–6. C54x DSP Core Clock Frequency

DSP clock = k(REFCLK)					
n	**m**	**k**	**Note About k**	**Minimum PLL Input Frequency**	**Maximum PLL Input Timing**
1 to 15	1	k = n/1	Integer Values (from 1 to 15)	5 MHz	200 ns
1, 3, 5, 7, 9, 11, 13, 15	2	k = n/2	Noninteger Values (from 0.5 to 7.5 with 1.0 granularity)	10 MHz	100 ns
1, 3, 5, 7, 9, 11, 13, 15	4	k = n/4	Noninteger Values (from 0.25 to 3.75 with 0.5 granularity)	20 MHz	50 ns

The DSP clock frequency is dynamically selected in the DSP register CLKMD (0x0058), supporting a maximum output clock frequency of 100 MHz. In addition, the ARM can provide control of the DSP PLL through the DSP_REG (0xFFFF:2F04).

2.12.2 ARM Clock

The ARM core clock frequency is derived from the input clock REFCLK and the frequency ratio multiplier k (where k is a function of n/m), as defined in Table 2–7.

Table 2–7. ARM Core Clock Frequency

ARM clock = k(REFCLK)					
n	**m**	**k**	**Note About k**	**Minimum PLL Input Frequency**	**Maximum PLL Input Timing**
1 to 15	1	k = n/1	Integer Values (from 1 to 15)	5 MHz	200 ns
1, 3, 5, 7, 9, 11, 13, 15	2	k = n/2	Noninteger Values (from 0.5 to 7.5 with 1.0 granularity)	10 MHz	100 ns
1, 3, 5, 7, 9, 11, 13, 15	4	k = n/4	Noninteger Values (from 0.25 to 3.75 with 0.5 granularity)	20 MHz	50 ns

The ARM clock frequency is dynamically selected in the ARM register ARM_PLL_REG (0xFFFF:3200), supporting a maximum output clock frequency of 47.5 MHz. In addition, the ARM clock may be put into a low-power mode as defined by the LOW_POWER_REG (0xFFFF:2F1C) and LOW_POWER_REG_VALUE (0xFFFF:2F20) register set.

2.12.3 Audio Clock

The AUDIO_CLK frequency is equal to:

$$f_{AUDIO\_CLK} = f_{REFCLK} / 2(N+1)$$

where N is a 11-bit programmable register N = (0...2047).

If the REFCLK frequency is equal to 24.576 MHz, the clock management scheme allows the generation of one of the following audio related frequencies: 8 kHz (N = 1535), 16 kHz (N = 767), 2.048 MHz (N = 5), 3.072 MHz (N = 3), 4.096 MHz (N = 2). Such audio frequency (AUDIO_CLK) is available at an output pin and can be used to clock external audio codecs.

2.13 Power-Down Modes

2.13.1 DSP Power-Down Modes

The DSP subsystem has three power-down modes, which are activated by the IDLE1, IDLE2, and IDLE3 instructions. In these modes, the C54x DSP core enters a dormant state and dissipates considerably less power than in normal operation.

The IDLE1 mode halts all DSP CPU activities except the DSP system clock. Because the system clock remains applied to the DSP subsystem peripheral modules, the DSP peripheral circuits continue operating and the DSP_CLKOUT pin remains active. Thus, peripherals such as serial ports and timers can take the CPU out of its power-down state.

The IDLE2 mode halts the DSP subsystem peripherals as well as the DSP core. Because the DSP subsystem peripherals are stopped in this mode, they cannot be used to generate the interrupt to wake up the C54x as with IDLE1. However, power is significantly reduced because the device is completely stopped. To terminate IDLE2, activate any of the external interrupt pins ($\overline{\text{RESET}}$ and $\overline{\text{DSP\_INT0}}$) with a 10-ns minimum pulse.

The IDLE3 mode functions like IDLE2 but it also halts the DSP phase-locked loop (PLL) circuit. IDLE3 is used for a complete shutdown of the C54x. This mode reduces power dissipation more than IDLE2. Furthermore, the IDLE3 state allows you to reconfigure the DSP PLL externally if the system requires the C54x to operate at a lower speed to save power. To terminate IDLE3, activate any of the external interrupt pins ($\overline{\text{RESET}}$ and $\overline{\text{DSP\_INT0}}$) with a 10-ns minimum pulse.

2.13.2 ARM Power-Down Modes

The ARM subsystem has two types of power-down modes, which are activated through register manipulation. In these modes, the ARM subsystem enters a dormant state and dissipates considerably less power than in normal operation.

The first power-down mode is a low-power mode involving the manipulation of the ARM subsystem clock rate. By significantly reducing the ARM subsystem clock frequency through the use of the LOW_POWER_REG (0xFFFF:2F1C) and the LOW_POWER_VALUE (0xFFFF:2F20) register set, a substantial reduction in power can be achieved while the whole ARM subsystem continues to operate at a reduced MIP rate. The ARM is capable of entering and exiting this low-power mode through software control.

The second power-down option is a sleep mode involving the cessation of clocks to portions of the ARM subsystem. By curtailing clocks driven to portions of the ARM subsystem through the use of the CLKM_REG (0xFFFF:2F00) and WAKEUP_REG (0xFFFF:2F08) register set, a substantial reduction in power can be achieved in portions of the ARM subsystem that are not being used. The ARM is capable of entering this sleep mode through software control, but requires an interrupt to exit.

2.14 Interrupt Management

2.14.1 DSP Interrupts

The DSP subsystem interrupts are mapped as shown in Table 2–8.

Table 2–8. DSP Interrupt Mapping

Name	DEC Address	HEX Address	Priority	Function
RS, SINTR	0	00	1	
Reserved	4	04	2	Reserved
SINT17	8	08	—	Software interrupt #17
SINT18	12	0C	—	Software interrupt #18
SINT19	16	10	—	Software interrupt #19
SINT20	20	14	—	Software interrupt #20
SINT21	24	18	—	Software interrupt #21
SINT22	28	1C	—	Software interrupt #22
SINT23	32	20	—	Software interrupt #23
SINT24	36	24	—	Software interrupt #24
SINT25	40	28	—	Software interrupt #25
SINT26	44	2C	—	Software interrupt #26
SINT27	48	30	—	Software interrupt #27
SINT28	52	34	—	Software interrupt #28
SINT29	56	38	—	Software interrupt #29
SINT30	60	3C	—	Software interrupt #30
INT0, SINT0	64	40	3	External user interrupt
Reserved	68	44	4	Reserved
Reserved	72	48	5	Reserved
TINT, SINT3	76	4C	6	Timer interrupt
BRINT0, SINT4	80	50	7	McBSP #0 receive interrupt (default)

Table 2–8. DSP Interrupt Mapping (Continued)

Name	DEC Address	HEX Address	Priority	Function
BXINT0, SINT5	84	54	8	McBSP #0 transmit interrupt (default)
Reserved	88	58	9	Reserved
Reserved	92	5C	10	Reserved
Reserved	96	60	11	Reserved
AINT, SINT9	100	64	12	API interrupt
BRINT1, DMAC2, SINT10	104	68	13	McBSP #1 receive interrupt (default)
BXINT1, DMAC3, SINT11	108	6C	14	McBSP #1 transmit interrupt (default)
DMAC4, SINT12	112	70	15	DMA channel 4 interrupt (default)
DMAC5, SINT13	116	74	16	DMA channel 5 interrupt (default)
Reserved	120–127	78–7F	—	Reserved

2.14.2 MCU Interrupts

For information on handling ARM interrupts, see Chapter 4, *Interrupt Handler.*

Chapter 3

Memory Interface (MEMINT)

This chapter explains the function of the TMS320VC547x Memory Interface (MEMINT), discusses the system and API buses and the external memory interface, provides an overview of SDRAM, and describes the SDRAM interface (SDRAM IF) and its internal and external controls.

MEMINT is associated with the ARM™ microcontroller unit (MCU) of the dual-core (MCU + DSP) TMS320VC547x device.

Topic **Page**

3.1 Memory Interface (MEMINT) Function

The VC547x memory interface (MEMINT) is used to interface to internal ARM memory as well as to external ARM memories by providing the necessary control signals, as well as address and data management, to the available internal and external buses. In addition to interfacing memories to the ARM processor, the memory interface is also responsible for interfacing ARM peripherals to the ARM processor.

Table 3–1 lists some of the terms used throughout this chapter and their meanings.

Table 3–1. MEMINT Terminology

Term	Meaning
Memory	Implies ARM memory
Address	Implies address on any of the ARM buses (internal or external)
Data	Implies data on any of the ARM data buses
System	Implies internal

The ARM memory interface provides the following functionality:

- I**nterface to internal address and internal data buses:** This involves accesses to the on-chip SRAM and peripherals via the internal/system bus.
- **Interface to API address and data buses:** This involves the API bus that internally connects the ARM subsystem (ARMSS) with the DSP subsystem (DSPSS).
- **Interface to synchronous DRAM controller:** The SDRAM Controller is a separate module that exists independently of the Memory Interface module, MEMINT. Both MEMINT and the SDRAM controller are used to generate the necessary address, data, and control signals while performing an access to SDRAM.
- **Interface to external address and external data buses:** This involves external accesses while accessing Flash (ROM) or SRAM.

3.2 System (Internal) Bus

The system (or internal) bus implemented on the ARMSS is a 32-bit-wide bus that supports 8-, 16- and 32-bit accesses. All peripherals on the system bus are 32-bit devices. For this reason, the two lower address bits of the ARM processor address are never used while performing accesses to the peripheral registers.

Transactions on the system bus by the ARM processor is always done with **zero wait-state** internal access.

Table 3–2 shows the ARM internal memory and ARM peripherals that are accessed through the system/internal bus via the memory interface.

Table 3–2. ARM Accesses Through the System/Internal Bus via MEMINT

Start Address	Stop Address	Module Selected
FFC0–0000	FFC0–3FFF	SYSTEM RAM (Internal)
FFD0–0000	FFD0–3FFF	VC5470: Reserved VC5471: EIM RAM (Internal)
FFFF–0000	FFFF–07FF	VC5470: Reserved VC5471: EIM
FFFF–0800	FFFF–0FFF	UART_IRDA
FFFF–1000	FFFF–17FF	UART_MODEM
FFFF–1800	FFFF–1FFF	I2C
FFFF–2000	FFFF–27FF	SPI
FFFF–2800	FFFF–28FF	GPIO
FFFF–2900	FFFF–29FF	KBGPIO
FFFF–2A00	FFFF–2AFF	TIMER0
FFFF–2B00	FFFF–2BFF	TIMER1
FFFF–2C00	FFFF–2CFF	TIMER2
FFFF–2D00	FFFF–2DFF	INTH
FFFF–2E00	FFFF–2EFF	MEMINT
FFFF–2F00	FFFF–2FFF	CLKM
FFFF–3000	FFFF–30FF	SDRAM_IF
FFFF–3200	FFFF–32FF	ARM_PLL

3.3 API Bus Interface

The API bus is an independent bus that is not shared with the system/internal bus. Unlike the internal bus, the API bus is a 16-bit bus. All 32-bit transactions are divided into two 16-bit API transactions. 8-bit transactions on the API bus are not allowed; however, there is no mechanism in place that will inhibit you from performing an 8-bit transaction. If an 8-bit transaction is exercised via the API bus, it is transformed into a 16-bit transaction by duplicating the 8-bit data. This will cause unwanted behavior especially while attempting to write a byte onto a memory location, resulting in a 16-bit write.

Operating Speeds

The ARM and DSP cores within the VC547x device can operate at different speeds. For this reason, a specific wait-state implementation is needed for ensuring proper communication between the two subsystems. Since the speed for both cores can be programmed independently of each other's speed settings, the API wait state should also be programmed accordingly, in order to synchronize accesses to the shared API memory by the two subsystems. The API interface also utilizes the wait-state generator for inserting dummy cycles while performing a back-to-back access, and ensuring correct synchronization of signals between the ARM subsystem and the DSP subsystem.

The API_REG register, part of the MEMINT register file, holds the necessary programmable fields in order to generate the desired wait state.

For API_REG register definitions, see Section 3.5.2, *ARM Port Interface Wait-State Configuration Register*, on page 3-13.

Computing the Wait State

The total number of wait states generated is computed from the parameters described in API_REG register. The formula for computing the desired wait state depends on the type of access that is required (16 vs 32 bits).

For a 16-bit transaction, the desired wait state to be generated ($\overline{WAIT}$) is computed using the following equation:

$$\overline{WAIT} = API\_WS + API\_CS - 1$$

For a long access 32-bit transaction, the desired wait state ($\overline{WAIT}$) is generated using the following equation:

$$\overline{WAIT} = 2 * API\_WS + API\_CS + API\_BS$$

Examples for 16- and 32-bit accesses that will compute API wait states for a given requirement are shown in Example 3–1 and Example 3–2.

If we assume the case where the DSPSS is functioning at a speed which is twice the frequency of the ARMSS, the API_REG register parameters could be programmed with a large degree of safety for successful API transaction in SAM. The following values will satisfy this request:

API_WS = 3, API_CS = 2, and API_BS = 1

For API_REG register definitions, see Section 3.5.2, *ARM Port Interface Wait-State Configuration Register*, on page 3-13.

Example 3–1. 16-Bit Transaction

The following example demonstrates the waveform diagram that is generated by programming the API_REG register with the values shown below for a 16-bit transaction:

Type of transaction desired: 16-bit

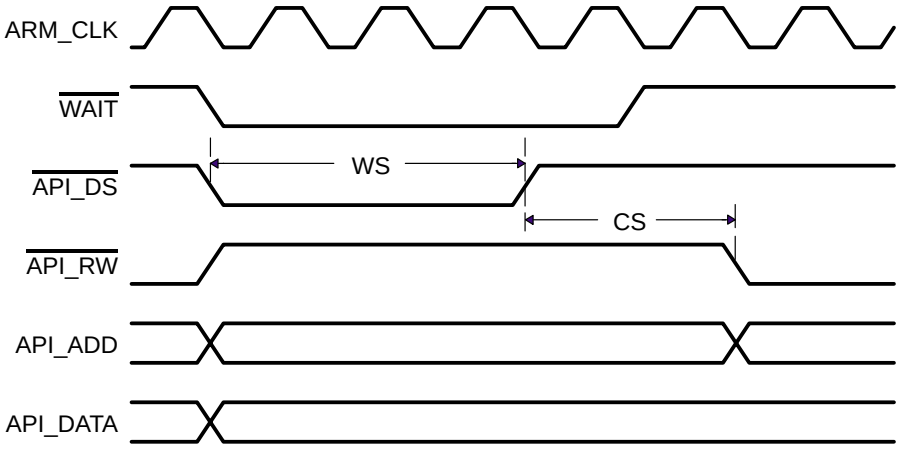
API_WS = 3
API_CS = 2
API_BS = 1

$\Rightarrow \overline{WAIT} = API\_WS + API\_CS - 1$

$= 3 + 2 - 1$

$= 4$

Figure 3–1. 16-Bit API Write Access With API_WS = 3, API_CS = 2, API_BS = 1



Note: The API_BS value has no effect on the wait state since we are interested only in a 16-bit access.

Example 3–2. 32-Bit Transaction

The following example demonstrates the waveform diagram that is generated by programming the API_REG register with the values shown below for a 32-bit transaction:

Type of transaction desired: 32-bit

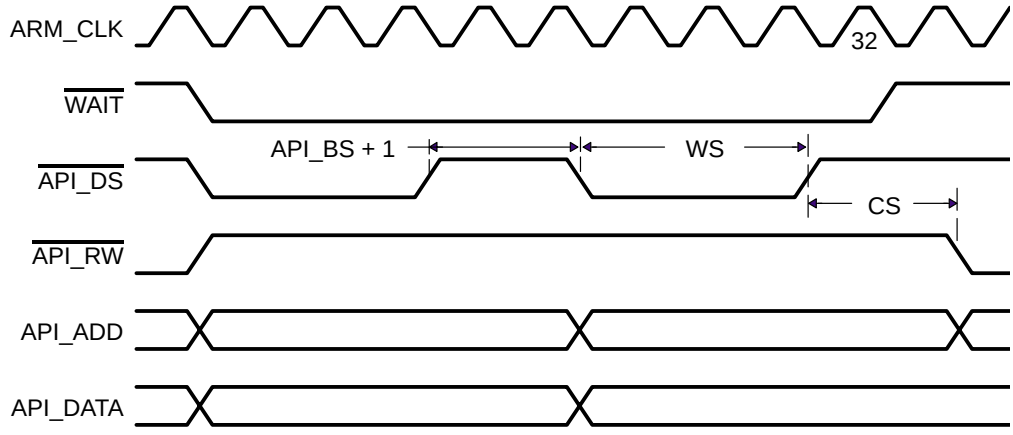
API_WS = 3
API_CS = 2
API_BS = 1

$\Rightarrow \overline{WAIT} = 2 * API\_WS + API\_CS + API\_BS$

$= 2 * 3 + 2 + 1$

$= 9$

Figure 3–2. 32-Bit API Write Access with API_WS = 3, API_CS = 2, API_BS = 1



It is VERY important to make sure that the user code correctly programs these parameters. As mentioned previously, in order to program the necessary fields correctly, the application should know the frequency of the ARM subsystem and the DSP subsystem.

It is also important to carefully choose the values set for the parameters since different combinations of values for the same parameters will yield identical wait-state values, but with different waveforms. One simple rule that will help when assigning values to the parameters is to make sure that the value chosen for API_WS is greater than that for API_CS and API_BS (when performing a 32-bit access).

3.4 External Memory Interface

In addition to managing the interface to internal memory and peripherals (via the use of the API bus and the system/internal bus), the memory interface also provides management for external accesses.

3.4.1 ROM (Flash) and SRAM

External memory devices supported are ROM (Flash), SRAM, and SDRAM. The memory interface provides support for 8-, 16-, and 32-bit-wide ROM (Flash) and SRAM memories. The only exception arises during power-up conditions when the device has to know the data width of the attached memory. Due to the assumption that an 8-bit-wide memory will result in a large peformance decrease, the current implementation can only differentiate between a 16-bit or a 32-bit-wide access during power up. See the section titled *Special Note About $\overline{CS0}$* under Figure 3–3 on page 3-11.

Table 3–3 shows the signals that are used to manage the external memory interface.

Table 3–3. ROM (Flash) and SRAM Memory Interface Signals

Bit	Signal
ADD[22:00]	ARM Address Bus
DATA[31:00]	ARM Data Bus
$\overline{CS0}$	ARM Chip Select
$\overline{CS1}$	ARM Chip Select
$\overline{CS2}$	ARM Chip Select
$\overline{CS3}$	ARM Chip Select
CS4	ARM Chip Select
R/$\overline{W}$	Read/Write Direction Control Signal
$\overline{BE[3:0]}$	External Byte Enable for Flash or SRAM access

Features

The memory interface supports the following features:

- ❑ Generating a programmable number of wait states
- ❑ Generating a programmable number of dummy cycles after an access takes place
- ❑ Selection of endianism (big or little)
- ❑ Write access
- ❑ Automatic insertion of a single wait state during a write while accessing a zero wait-state memory
- ❑ Handling of different memory device sizes

The above parameters are unique for each selector; i.e., each chip-select signal ($\overline{CS0}$, $\overline{CS1}$, $\overline{CS2}$, CS4, and $\overline{SDRAM\_CS}$). Each chip-select signal has its own set of programmable parameters that is used to control the respective device, allowing versatility. The exception happens only with the SDRAM chip-select parameters.

For interfacing with SDRAM, the $\overline{SDRAM\_CS}$ control parameters available are: Device Size (16- vs 32-bit-wide data access), Dummy Cycle Insertion, and Endianism Selector.

One register (from the MEMINT register file shown in Table 3–4) for each chip-select signal holds the programmable fields that are used to identify the type of device attached to the VC547x.

3.5 Memory Interface (MEMINT) Registers

Base address (hex): FFFF:2E00

Register width: 32 bits

Table 3–4. MEMINT Registers

Register	Description	Offset Address
CS0_REG	External Memory Control Register for $\overline{CS0}$ memory range	0x0000
CS1_REG	External Memory Control Register for $\overline{CS1}$ memory range	0x0004
CS2_REG	External Memory Control Register for $\overline{CS2}$ memory range	0x0008
CS3_REG	External Memory Control Register for $\overline{CS3}$ memory range	0x000C
CS4_REG	External Memory Control Register for CS4 memory range	0x0010
API_REG	ARM Port Interface Wait-State Configuration Register	0x0014
SDRAM_REG	SDRAM Data Bus Size Control Register	0x0018
BS_CONFIG	Bank-Switching Configuration Register	0x001C

3.5.1 External Memory Control Register for $\overline{CS0}$–$\overline{CS3}$, CS4 Memory Range

The external memory control registers (CS0_REG–CS4_REG) are five independent programmable registers that are used to customize the interface between the VC547x ARMSS and its external memory (Flash or SRAM) by allowing the user the flexibility to attach memories with different characteristics. The bit/field functions of each external memory control register are as follows:

WS[4:0] holds the values for the additional wait states to be asserted (0 to 31).

DVS[6:5] holds the width of the SRAM or Flash memory. The two-bit field is sufficient to differentiate among the three supported devices (8-, 16-, or 32-bit).

WE[7] differentiates between SRAM and Flash. If external memory is SRAM, WRITE is enabled (Read/Write capability is selected). If external memory is Flash, WRITE is disabled (Read-Only capability is selected).

BIGEND[8] configures the external memory as big- or little-endian.

DC[11:9] inserts additional cycles (up to 7) in order to avoid bus contention during bank-switching. It works in conjunction with the BS_CONFIG register. See Section 3.5.6, *Bank-Switching Configuration Register*, on page 3-19 for more information.

WS1[12] dictates if an automatic additional wait state is needed while writing to a zero wait-state external SRAM or external Flash. Note that this applies only during Write, not Read.

Figure 3–3. External Memory Control Register for $\overline{CS0}$–$\overline{CS3}$, CS4 Memory Range (CS0_REG–CS4_REG)

Address (hex): Base = 0xFFFF:2E00, Offset = 0x0000 (CS0_REG)
Offset = 0x0004 (CS1_REG)
Offset = 0x0008 (CS2_REG)
Offset = 0x000C (CS3_REG)
Offset = 0x0010 (CS4_REG)

31–16
Reserved

15–13	12	11–9	8	7	6–5	4–0
Reserved	WS1	DC	BIGEND	WE	DVS	WS
	RW-0	RW-1	RW-0	RW-1	RW-01	RW-1

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–13 **Reserved**.

Bit 12 **WS1.** Wait state to be inserted on zero-wait-state device during write access.

0 No wait state inserted
1 One wait state inserted

Bits 11–9 **DC**. Number of dummy cycles to be inserted during bank-switching.

Bit 8 **BIGEND**. Endianism.

0 Little endian
1 Big endian

Bit 7 **WE**. Write Enable.

0 Write disabled
1 Write enabled

Bits 6–5 **DVS**. Device Size.

00	Reserved (CS0_REG only). See ROMSIZE pin description.
00	8-bit-wide bus (CS1_REG–CS4_REG). See note 3
01	16-bit-wide bus. See note 1
10	32-bit-wide bus. See note 2
11	Reserved

Bits 4–0 **WS**. Number of wait states.

Notes:

1) 16-bit-wide SDRAM memory can only be made up of a single 16-bit-wide memory. No multiple 8-bit-wide memory support is available.
2) 32-bit-wide SDRAM memory can be made up of two 16-bit-wide memories. No single 32-bit-wide memory support is available. The exception to this is the 64-Mbit 32-bit-wide memory, which is supported.
3) $\overline{CS0}$–$\overline{CS3}$ ⇒ Active Low
   CS4 ⇒ Active High

Special Note About $\overline{CS0}$

The ARM Reset Vector is always at address 0x00000000. Consulting Table 3–7 indicates that this address is mapped onto external memory. It is most likely that a Flash memory (SRAM is usually used in place of Flash on development boards) is attached to the VC547x device at this particular space, and Chip Select 0 is the signal that will be used to control the attached memory at this location. Some of the programmable parameters/fields (endianism, size) of this Chip Select register need to be known during power-up time in order for the ARM core to access the attached memory (Flash) with the correct physical settings.

During Power Up/Reset, the Memory Interface automatically inserts the maximum allowable wait states. This is necessary because at boot time, we do not know the speed of the external device. The default is to assume the slowest device. The other unknown is the size of the attached memory (most likely Flash). This information is provided to the ARMSS by sampling the multiplexed pin ($\overline{CS3}$/ROMSIZE16) during power-up time. This pin should be pulled high or low, depending the size of the attached memory.

If $\overline{CS3}$/ROMSIZE16 is sampled high, the attached memory is a 32-bit-wide memory; if $\overline{CS3}$/ROMSIZE16 is sampled low, the attached memory is a 16-bit-wide memory. Similarly, sampling another multiplexed pin, CS4/BIGEND, identifies the endianism of the memory. If CS4/BIGEND is sampled high, the attached memory device, controlled by the $\overline{CS0}$ signal, is operating in Big-Endian mode. If CS4/BIGEND is sampled low, the attached memory (Flash) is operating in Little-Endian mode.This will allow the ARMSS to correctly fetch program code from the attached memory and perform correct execution.

3.5.2 ARM Port Interface Wait-State Configuration Register

Figure 3–4. ARM Port Interface Wait-State Configuration Register (API_REG)

Address (hex): Base = 0xFFFF:2E00, Offset = 0x0014

31–10	9–6	5–4	3–0
Reserved	API_CS	API_BS	API_WS
	RW-0	RW-0	RW-0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–10 **Reserved**. Read as zeros.

Bits 9–6 **API_CS**. Hold Time. Number of clock cycles API_NRW is valid (i.e., high) after release of $\overline{API\_DS}$.

Bits 5–4 **API_BS**. One clock cycle less than the total number of clock cycles $\overline{API\_DS}$ is inactive (i.e., high). Used in back-to-back accesses or in 32-bit accesses.†

Bits 3–0 **API_WS**. Number of clock cycles $\overline{API\_DS}$ is maintained (i.e., low).

† The API_BS parameter (field) is useful only for back-to-back accesses (while performing 32-bit transactions). There is a minimum of one cycle insertion that is added by the system automatically. For this reason, one clock-cycle value is always subtracted from the desired value. Due to design limitations, values other than 0 and 1 are not allowed. System may hang if other values are used.

3.5.3 API Control Register

This ARM register is only 16 bits and *must* be declared as a *short* variable.

Figure 3–5. API Control Register (APIC)

Address (hex): Base = 0xFFE4:0000, Offset = 0x0000

15–4	3	2	1	0
Reserved	HINT	DSPINT	APIMODE	Reserved
	R-0	W-0	R-0	

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 15–4 **Reserved**. Read zeros.

Bit 3 **HINT**. Host Interrupt. Read Only. A write does not modify this bit. This bit is the mirror of the HINT bit (bit 3 of the DSP BSCR register [0x0029]), which must be toggled from 0 to 1, and back to 0 in order for the DSP to create an ARM interrupt.

Bit 2 **DSPINT**. DSP Interrupt. Write Only. Always read as zero. Writing a 1 generates a DSP AINT/SINT9 interrupt. The bit is automatically cleared by the hardware after the interrupt is generated.

Bit 1 **APIMODE**. API Mode. Read Only. A write does not modify this bit. This bit is the mirror of the APIMODE bit (bit 2 of the DSP BSCR register [0x0029]).

0	Shared-Access Mode. Both the DSP and the ARM may access the API RAM.
1	Host-Only Mode. The ARM may access the API RAM, but the DSP may not.

Bit 0 **Reserved**. Always returns 0.

3.5.3.1 DSP and ARM Required Interrupt Methodology

In order for the interrupts to work between the DSP and the ARM, be sure the ARM register CPSR has the I and F flag bits cleared in User Mode.

To interrupt the ARM processor from the DSP, *pulse* the HINT bit (bit 3 of the DSP register BSCR [0x0029]).

```
BSCR |= BSCR_HINT;
for (i=0; i<HINT_PAUSE; i++)
{
// pause
}
BSCR &= ~BSCR_HINT;
```

To interrupt the DSP processor from the ARM, *write a 1* into the DSPINT bit (bit 2 of the ARM register APIC [0xFFE4:0000]).

The ARM register APIC *must* be declared as a *short* variable.

```
volatile u16 *APIC = (u16*)0xFFE40000; // Set bit 2 to
                                       // interrupt the DSP

*APIC |= (u16)0x4;                     // Interrupt DSP
```

3.5.4 Bank-Switching Control Register

The following register description is for a DSP subsystem register and should not be confused with an ARM subsystem register. Please refer to Table 2–1 for a complete list of DSP subsystem registers.

Figure 3–6. Bank-Switching Control Register (BSCR)

DSP Address (hex) = 0029

15–12	11	10–5	4	3	2	1	0
BNKCMP	PS-DS	Reserved	ABMDIS	HINT	APIMODE	Reserved	EXIO
RW-1111	RW-1	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 15–12 **BNKCMP**. Bank Compare. Determines the external memory bank size. BNKCMP is used to mask the four MSBs of an address. For example, if BNKCMP = 1111b, the four MSBs (bits 12–15) are compared, resulting in a bank size of 4K words. Bank sizes of 4K words to 64K words are allowed. Table 3–5 shows the relationship between BNKCMP and the address range.

Table 3–5. Relationship Between BNKCMP and Bank Size

BNKCMP				MSBs to Compare	Bank Size (16-Bit Words)
Bit 15	Bit 14	Bit 13	Bit 12		
0	0	0	0	None	64K
1	0	0	0	15	32K
1	1	0	0	15–14	16K
1	1	1	0	15–13	8K
1	1	1	1	15–12	4K

Bit 11 **PS-DS**. Program Read/Data Read Access. Inserts an extra cycle between consecutive accesses of program read and data read, or data read and program read.

0 No extra cycles are inserted by this feature

1 One extra cycle is inserted between consecutive data and program reads.

Bits 10–5 **Reserved**. These bits are reserved and will retain written values.

Bit 4 **ABMDIS**. API Boot Mode Disable. This bit will force the DSP out of API Boot Mode. This bit has no effect when DSP_APIBN port is high. When DSP_APIBN is low, a 0 in this bit enables API Boot Mode and a 1 in this bit disables API Boot Mode. Note that a DSP subsystem reset will clear this bit and thus re-enable the DSP_APIBN port.

0 API Boot Mode is under the control of the DSP_APIBN input port

1 DSP_APIBN input port is ignored and the DSP subsystem is kept out of API Boot Mode.

Bit 3 **HINT**. Host Processor Interrupt. This bit enables/disables an interrupt to the MCU from the DSP. At reset, the HINT bit is cleared. To send a proper interrrupt to the MCU, the DSP must write a 1 to HINT, delay an appropriate time, then write a 0 to HINT to create a pulse of an appropriate duration on the interrupt signal to the MCU.

0 Interrupt signal to the MCU is not active

1 Interrupt signal to the MCU is active

Bit 2 **APIMODE**. Host-Only Mode/Shared-Access Mode (HOM/SAM) Enable. This bit enables/disables the API Host-Only Mode. During reset, the APIMODE bit is set (HOM is automatically selected); after reset, the APIMODE bit is cleared (SAM is automatically selected).

0 The Shared-Access Mode (SAM) is enabled. Both the DSP and the MCU may access the API RAM.

1 The Host-Only Mode (HOM) is enabled. The MCU may access the API RAM; the DSP may not access the API RAM. DSP writes to the API RAM are ignored, and DSP reads from the API RAM return undefined values. DSP IDLE modes have no effect on MCU accesses to the API memory when in Host-Only Mode.

Bit 1 **Reserved**.

Bit 0 **EXIO**. External Bus Interface Off. The EXIO bit controls the external-bus-off function.

0 The external bus interface functions as usual.

1 The address bus, data bus, and control signals become inactive after completing the current bus cycle. Table 3–6 lists the states of the external interface signals when the interface is disabled. Note that the DROM, MP/$\overline{\text{MC}}$, and OVLY bits in the PMST and the HM bit of ST1 cannot be modified when the interface is disabled.

Table 3–6. State of Signals When External Bus Interface is Disabled (EXIO = 1)

Signal	State
A[19:0]	Previous State
D[15:0]	High-Impedance
$\overline{PS}$, $\overline{DS}$, $\overline{IS}$	High Level
$\overline{MSTRB}$, $\overline{IOSTRB}$	High Level
R/$\overline{W}$	High Level
$\overline{MSC}$	High Level
$\overline{IAQ}$	High Level

3.5.4.1 DSP and ARM Required Interrupt Methodology

In order for the interrupts to work between the DSP and the ARM, be sure the ARM register CPSR has the I and F flag bits cleared in User Mode.

To interrupt the ARM processor from the DSP, *pulse* the HINT bit (bit 3 of the DSP register BSCR [0x0029]).

```
BSCR |= BSCR_HINT;
for (i=0; i<HINT_PAUSE; i++)
{
// pause
}
BSCR &= ~BSCR_HINT;
```

To interrupt the DSP processor from the ARM, *write a 1* into the DSPINT bit (bit 2 of the ARM register APIC [0xFFE4:0000]).

The ARM register APIC *must* be declared as a *short* variable.

```
volatile u16 *APIC = (u16*)0xFFE40000; // Set bit 2 to
                                       // interrupt the DSP

*APIC |= (u16)0x4;                     // Interrupt DSP
```

3.5.5 SDRAM Data Bus Size Control Register

SDRAM_REG is a programmable register that is used to customize the interface between the ARMSS and the external SDRAM.

DVS[1:0] holds the SDRAM Device-Data bus-width (can select memories between 16 and 32 bits wide).

DC[3:2] inserts additional cycles (up to 3) in order to avoid bus contention during bank-switching. It works in conjunction with the BS_CONFIG register. See Section 3.5.6, *Bank-Switching Configuration Register*, on page 3-19 for more information.

BIGEND[4] configures the external memory as big- or little-endian.

Figure 3–7. SDRAM Data Bus Size Control Register (SDRAM_REG)

Address (hex): Base = 0xFFFF:2E00, Offset = 0x0018

31–3	4	3–2	1–0
Reserved	BIGEND	DC	DVS
	RW-0	RW-0	RW-01

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–3 **Reserved**.

Bit 4 **BIGEND**. Endianism

0 Little endian

1 Big endian

Bits 3–2 **DC**. Number of dummy cycles to be inserted during bank-switching.

Bits 1–0 **DVS**. SDRAM device (bus) size; i.e., data width

00	Reserved
01	16-bit-wide data, see note 1
10	32-bit-wide data, see note 2
11	Reserved

Notes:

1) 16-bit-wide SDRAM memory can only be made up of a single 16-bit-wide memory. No multiple 8-bit-wide memory support is available.
2) 32-bit-wide SDRAM memory can be made up of two 16-bit-wide memories. No single 32-bit-wide memory support is available. The exception to this is the 64-Mbit 32-bit-wide memory. The 64-Mbit memory, which is 32 bits wide, is supported.
3) $\overline{CS0}$–$\overline{CS3} \Rightarrow$ Active Low
   CS4 $\Rightarrow$ Active High

3.5.6 Bank-Switching Configuration Register

Bus contention usually exists when performing a back-to-back access on two different banks or two different types of memories operating at different speeds.

One of the fields (DC) that is found within the Chip Select registers (CS[4:0]_REG, and SDRAM_REG) holds the number of dummy cycles that are to be inserted during bank-switching. When two different chip selects are used to control two different devices or banks of a memory, the BS_CONFIG register is used to enable/disable the insertion of the dummy cycles, selectively, while memory (device) access changes from one memory (device) to another memory (device). This gives the user much more flexibility by inserting additional dummy cycles when performing back-to-back access during bank-switching, thus eliminating the need for additional logic.

Figure 3–8. Bank-Switching Configuration Register (BS_CONFIG)

Address (hex): Base = 0xFFFF:2E00, Offset = 0x001C

31–30	29–25	24–20	19–15
Reserved	SDRAM matrix	CS4 matrix	CS3 matrix
	RW-0	RW-0	RW-0

14–10	9–5	4–0
CS2 matrix	CS1 matrix	CS0 matrix
RW-0	RW-0	RW-0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–30 **Reserved**. Always return 0.

Bits 29–25 **SDRAM matrix**. Insert dummy cycles per SDRAM_REG (FFFF:2E18) DC[3:2] when switching from the SDRAM bank to the following defined banks:

Bit 29: CS4 bank
Bit 28: CS3 bank
Bit 27: CS2 bank
Bit 26: CS1 bank
Bit 25: CS0 bank

Bits 24–20 **CS4 matrix**. Insert dummy cycles per CS4_REG (FFFF:2E10) DC[11:9] when switching from the CS4 bank to the following defined banks:

Bit 24: Reserved
Bit 23: CS3 bank
Bit 22: CS2 bank
Bit 21: CS1 bank
Bit 20: CS0 bank

Bits 19–15 **CS3 matrix**. Insert dummy cycles per CS3_REG (FFFF:2E0C) DC[11:9] when switching from the CS3 bank to the following defined banks:

Bit 19: Reserved
Bit 18: CS4 bank
Bit 17: CS2 bank
Bit 16: CS1 bank
Bit 15: CS0 bank

Bits 14–10 **CS2 matrix**. Insert dummy cycles per CS2_REG (FFFF:2E08) DC[11:9] when switching from the CS2 bank to the following defined banks:

Bit 14: Reserved
Bit 13: CS4 bank
Bit 12: CS3 bank
Bit 11: CS1 bank
Bit 10: CS0 bank

Bits 9–5 **CS1 matrix**. Insert dummy cycles per CS1_REG (FFFF:2E04) DC[11:9] when switching from the CS1 bank to the following defined banks:

Bit 9: Reserved
Bit 8: CS4 bank
Bit 7: CS3 bank
Bit 6: CS2 bank
Bit 5: CS0 bank

Bits 4–0 **CS0 matrix**. Insert dummy cycles per CS0_REG (FFFF:2E00) DC[11:9] when switching from the CS0 bank to the following defined banks:

Bit 4: Reserved
Bit 3: CS4 bank
Bit 2: CS3 bank
Bit 1: CS2 bank
Bit 0: CS1 bank

Chip-Select Sample Configurations

Example 3–3 on page 3-22 shows sample configurations for all six chip selects.

To better understand how this works, consider the explanation of the first row in Example 3–3 on page 3-22: Assume that you made access to the memory location that is controlled by $\overline{CS0}$ and your next access is to another memory location or device that is controlled by any of the other chip selects ($\overline{CS1}$, $\overline{CS2}$, $\overline{CS3}$, or CS4). You may have different requirements for different devices/memories since the speeds of each device/memory could be different. Bits [4:0] of BS_CONFIG register would handle all possible cases for additional cycle insertion when access moves from a device/memory controlled by Chip Select 0 to a device/memory controlled by another chip select. A single bit for each chip select will either enable or disable the additional cycle insertion before enabling the next access to devices controlled by the other chip selects. A value of one enables and a value of zero disables the dummy cycle insertion.

Referring to the bit definition of the BS_CONFIG register in the first row of Example 3–3 (for Chip Select 0 matrix), and assuming that the last access performed was at a location that is within the Chip Select 0 space and the next access to be made is to a device/memory location controlled by Chip Selects 2 or 4, then the additional cycles equal to the amount programmed onto the DC field of the corresponding chip-select register is inserted before allowing the next access. This way, no two devices will be driving the bus at the same time.

Example 3–3. Sample Configurations for Chip Selects

Configuration	Description
BS_CONFIG[4:0] = 01010	Insert dummy cycle when switching from CS0 to CS2 or CS4
BS_CONFIG[9:5] = 00011	Insert dummy cycle when switching from CS1 to CS0 or CS2
BS_CONFIG[14:10] = 00000	Never insert dummy cycle when switching from CS2
BS_CONFIG[19:15] = 00000	Never insert dummy cycle when switching from CS3
BS_CONFIG[24:20] = 01101	Insert dummy cycles when switching from CS4 to CS0 or CS2 or CS3
BS_CONFIG[29:25] = 00001	Insert dummy cycle when switching from SDRAM to CS0

Note that during insertion of dummy cycles, NO chip select will be active in order to make sure that there is no device driving the bus.

3.6 ARM Memory Space

The ARM memory space is shared between internal memory, the External Memory Interface, the SDRAM Interface, and the ARM peripherals (see Table 3–7). The Memory Interface provides five chip-select signals in addition to the SDRAM interface that has its own chip-select signal.

Table 3–7. ARM Memory Space

	Start Address	Stop Address	Allocated Size in Bytes	Data Access
$\overline{\text{CS0}}$	0000:0000	007F:FFFF	8M	8/16/32
$\overline{\text{CS1}}$	0080:0000	00FF:FFFF	8M	8/16/32
$\overline{\text{CS2}}$	0100:0000	017F:FFFF	8M	8/16/32
$\overline{\text{CS3}}$	0180:0000	01FF:FFFF	8M	8/16/32
CS4	0200:0000	027F:FFFF	8M	8/16/32
Reserved	0280:0000	0FFF:FFFF		
$\overline{\text{SDRAM\_CS}}$	1000:0000	11FF:FFFF	32M	8/16/32
Reserved	1200:0000	FFBF:FFFF		
Internal SRAM	FFC0:0000	FFC0:3FFF	16K	8/16/32
Reserved	FFC0:4000	FFCF:FFFF		
EIM SRAM (VC5471) Reserved (VC5470)	FFD0:0000	FFD0:3FFF	16K	8/16/32
Reserved	FFD0:4000	FFDF:FFFF		
API RAM	FFE0:0000	FFE0:3FFF	16K	16/32
Reserved	FFE0:4000	FFF3:FFFF		
API registers	FFE4:0000	FFE4:0001	2	16
Reserved	FFE4:0002	FFFE:FFFF		
EIM (VC5471) Reserved (VC5470)	FFFF:0000	FFFF:07FF	2K	32
UART_IRDA	FFFF:0800	FFFF:0FFF	2K	32
UART	FFFF:1000	FFFF:17FF	2K	32
I2C	FFFF:1800	FFFF:1FFF	2K	32
SPI	FFFF:2000	FFFF:27FF	2K	32

Table 3–7. ARM Memory Space (Continued)

	Start Address	Stop Address	Allocated Size in Bytes	Data Access
GPIO	FFFF:2800	FFFF:28FF	256	32
KBGPIO	FFFF:2900	FFFF:29FF	256	32
TIMER0	FFFF:2A00	FFFF:2AFF	256	32
TIMER1	FFFF:2B00	FFFF:2BFF	256	32
TIMER2	FFFF:2C00	FFFF:2CFF	256	32
INTH	FFFF:2D00	FFFF:2DFF	256	32
MEMINT	FFFF:2E00	FFFF:2EFF	256	32
CLKM	FFFF:2F00	FFFF:2FFF	256	32
SDRAMIF	FFFF:3000	FFFF:30FF	256	32
ARM_PLL	FFFF:3200	FFFF:32FF	256	32
Reserved	FFFF:3300	FFFF:FFFF		

Note: As shown in Table 3–7, each chip-select signal controls a specific range of memory.

3.7 SDRAM

Synchronous dynamic random-access memories (SDRAMs) are DRAMs that have all of the conventional controls and data fully synchronized on a clock input.

The SDRAM interface provides support for 16- and 32-bit configurations. Although 8-bit data accesses are allowed, connecting a single 8-bit SDRAM externally is not allowed. 16-bit configurations (for 8-, 16-, and 32-bit data accesses via a 16-bit data bus) can only be obtained by using a 16-bit-wide SDRAM. 32-bit configurations (for 8-, 16-, and 32-bit accesses via a 32-bit bus) can only be obtained by placing two 16-bit-wide SDRAMs in parallel. The one exception to this is the 64-Mbit, 32-bit-wide SDRAM memory. Due to cost savings, the 64-Mbit, 32-bit-wide SDRAM is the only 32-bit-wide SDRAM that is supported. When needed, other sizes can be made up of multiple 16-bit-wide SDRAM memories.

In order to differentiate between the two types of buses used (16-bit vs 32-bit), the SDRAM interface memory register should be programmed accordingly, reflecting the real device size used.

3.7.1 Introduction

SDRAM's synchronous design provides a simple user interface compared to standard DRAMs, which are basically asynchronous memories. SDRAM operations are determined by commands sampled on an active edge of the SDRAM clock.

Other differences between standard DRAMs and SDRAMs are the Burst mode and the Mode register. The Burst mode is a high-speed access mode that uses an internal address generator to supply the first column address. The following addresses are internally generated. The mode register is used to store valid operational parameters like burst length and CAS latency.

3.7.2 SDRAM IF Overview

The SDRAM interface (IF) receives all the parameters it needs from an external controller, which also manages external requests, refresh, clock gating, and the data bus.

The main features of the SDRAM memory interface are:

- It operates as part of the ARM memory interface (MEMINT) allowing SDRAM memories to be used on the same board with Flash and/or SRAM.
- Supports 32-bit-wide and 16-bit-data-wide SDRAM interfaces
- Up to 256M-bit chip-enabled (CE) space
- Operates at the ARM clock speed
- Flexible programming of SDRAM parameters
- Supports up to four open pages of SDRAM

The SDRAM interface module is part of the memory interface. It effectively sits between the ARM processor and an SDRAM controller. The SDRAM interface receives and interprets read/write commands from the ARMSS and sends them with correct timing to the SDRAM controller, which in turn generates the correct combination on I/O pins. Connecting SDRAMs to the VC547x only requires programming the appropriate SDRAM interface parameters.

Programming the SDRAM IF

The SDRAM interface is programmed through several dedicated registers. SDRAM_REG of the memory interface is used to select the appropriate SDRAM device size and the endianness used while reading and writing data. Inside the SDRAM interface submodule itself, CONFIG, REFRESH, CONTROL, and INIT_CONF registers are used for programming other SDRAM parameters. The SDRAM interface provides support for both 16- and 32-bit configurations.

The 16-bit data configuration can be obtained from a single 16-bit-wide memory. The 32-bit data configuration can be obtained by connecting two 16-bit-wide memories in parallel. The Memory Interface register, SDRAM_REG, should also be programmed accordingly to reflect the real device data width used externally (32- or 16-bit). The sizes of the SDRAM supported are 16M bits, 64M bits, 128M bits, and 256M bits. This device information needs to be programmed in the CONFIG register of the SDRAM interface submodule.

Utilization of SDRAM

Utilization of SDRAM inside the VC547x requires initialization of the SDRAM interface. The VC547x SDRAM interface has limited burst access capabilities. For all 32-bit-wide accesses, the SDRAM interface will burst-terminate after each doubleword. If appropriately programmed, the SDRAM interface will perform a 2-word burst for all 16-bit-wide accesses.

Access begins with the registration of an ACTIVE command, which is then followed by a READ or WRITE command. The address bits registered coincident with the ACTIVE command are used to select the bank and row to be accessed. SDRAM_BA[1:0] (multiplexed with ADD[19:18]) are used to select the bank and SDRAM_A[12:0]† (multiplexed with ADD[12:0]) are used to select the ROW. The address bits SDRAM_A[9:0]‡ registered coincident with the READ or WRITE command are used to select the starting column location for the burst access. SDRAM_BA[1:0] select a particular bank that the READ or WRITE is going to take place.

† SDRAM_A[12:0] is used for 256M bits.
SDRAM_A[11:0] is used for 64 and 128M bits.
SDRAM_A[10:0] is used for 16M bits.

‡ Column width depends on the physical data width of the device.

Precharge

Autoprecharge does not apply in full-page burst mode. Before opening a different ROW, a PRECHARGE command is used to deactivate the open row in a particular bank or all banks. SDRAM_A10 is used to indicate the type of precharge that takes place. If SDRAM_A10 is LOW then the current bank that is being accessed will be precharged. If SDRAM_A10 is HIGH, then all BANKs will be precharged. Once a bank is precharged, it will not be available until the precharge time, TRP, is completed. All this is transparent to the user since the SDRAM controller will handle the necessary control signal generation. The user is responsible for programming the SDRAM Interface registers with the correct values so that proper control signals are generated accordingly.

Initialization

SDRAM must be initialized prior to normal operation. After applying power and stable clock, the initialization process begins by programming the SDRAM_REG register† (most importantly data width). Then, the SDRAM interface registers are programmed and the initialization state machine is kick-started (by writing a 1 to the SDRAM_INIT field of the SDRAM_CNTL register) to make the SDRAM ready for operation. A minimum of 100-µs delay is required to make sure that initialization procedure has taken place properly. This is achieved by programming the INIT_NOP_MAX_CNT field of the SDRAM_INIT_CONF register with cycle count that would at least make up a 100-µs time. See section 3.9 for registers definitions. The SDRAM controller will set the READY field within the SDRAM_CNTL register as an indication for the user that the SDRAM is ready for operation.

† The SDRAM_REG register could also be the last register of the SDRAM to program. You must only make sure that it is programmed appropriately before reading or writing from the SDRAM.

The following steps are needed for initializing the SDRAM interface:

- ❏ Programming of SDRAM_REG in the memory interface module
- ❏ Programming of SDRAM parameters in the SDRAMIF submodule
- ❏ Write a 1 to the SDRAM_INIT bit of the SDRAM_CNTL register
- ❏ Wait until a 1 is read inside the READY bit of the SDRAM_CNTL register.

See section 3.5.5 for register definitions of the SDRAM_REG register and see section 3.9 for register definitions of the SDRAM Interface Registers.

3.7.3 Supported Devices

The SDRAM IF has been designed to support 16M-bit, 64M-bit, 128M-bit, and 256M-bit SDRAM memories.

3.8 SDRAM Interface

The TMS320VC547x SDRAM interface supports 2-bank 16M-bit SDRAM and 4-bank 64M-/128M-/256M-bit SDRAM, providing an interface to high-speed and high-density memory.

3.9 SDRAM IF Registers

Base address (hex): FFFF:3000

Bit width: 32 bits

Table 3–8. SDRAM IF Registers

Register	Description	Offset Address
SDRAM_CONFIG	SDRAM Configuration Register	00h
SDRAM_REF_COUNT	SDRAM Refresh Counter Register	04h
SDRAM_CNTL	SDRAM Control Register	08h
SDRAM_INIT_CONF	SDRAM Initialization Refresh Counter Register	0Ch

SDRAM_REG

$\overline{\text{SDRAM\_CS}}$ is one of the SDRAM control signals used to select SDRAM memories when the VC547x device is required to perform accesses. SDRAM_REG is the register used to customize the behavior of the $\overline{\text{SDRAM\_CS}}$ signal while interfacing to different types of SDRAM memories. The SDRAM_REG register, part of the MEMINT register file, is used to indicate the data width of the SDRAM data bus as well as the endianism type of access that is used during READ and WRITE operations.

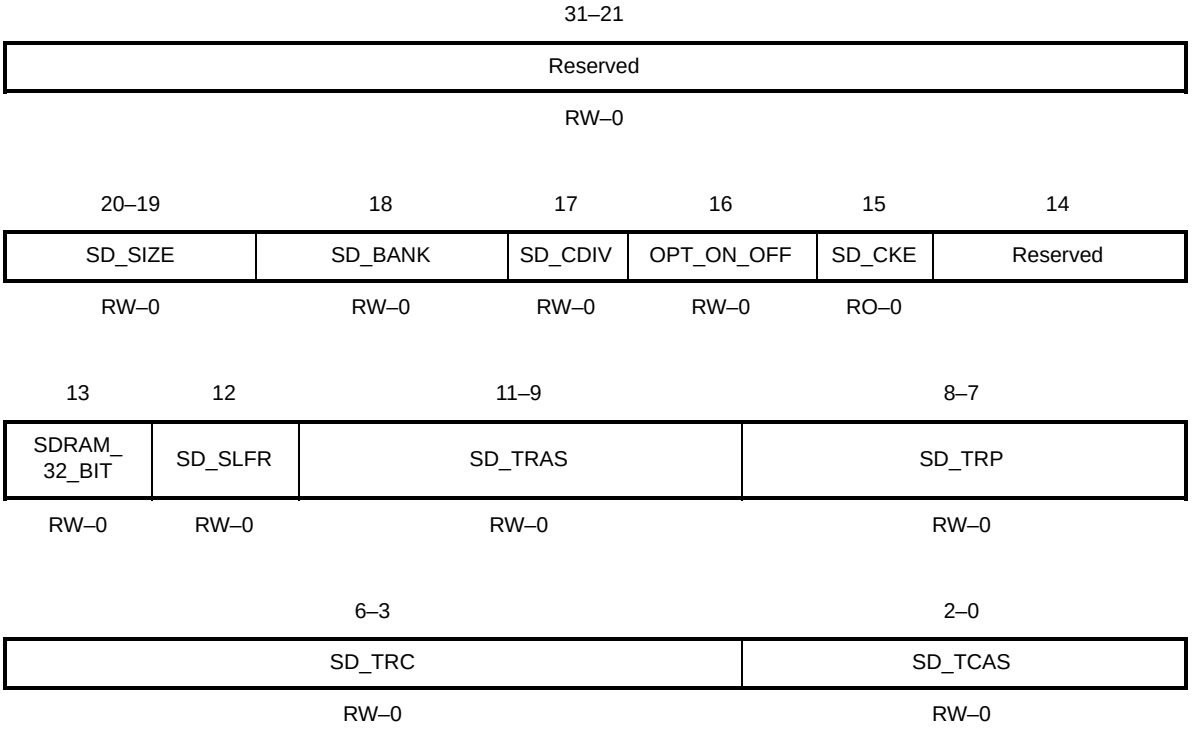
SDRAM_REG is a programmable register that is used to customize the interface between the ARMSS and the external SDRAM.

For SDRAM_REG definitions, see Section 3.5.5, *SDRAM Data Bus Size Control Register*, on page 3-18.

3.9.1 SDRAM Configuration Register

Figure 3–9. SDRAM Configuration Register (SDRAM_CONFIG)

Base address = 0xFFFF:3000, Offset Address = 0x0000



Note: R = Read access; W = Write access; O = Read only; value following dash (–) = value after reset

Bits 31–21 **Reserved**. These bits are reserved (read as zeros).

Bits 20–19 **SD_SIZE**. Size of the SDRAM memory.

00	16M bits (size of row 11)
01	64M bits (size of row 12)
10	128M bits (size of row 12)
11	256M bits (size of row 13)

Bit 18 **SD_BANK**. Number of banks.

0	Two banks (for 16M bits)
1	Four banks (for 64M bits, 128M bits, and 256M bits)

Bit 17 **SD_CDIV**. Clock divider

0 No division

1 SDRAM_CLK = SDRAM_CLK_IN / 2

Bit 16 **OPT_ON_OFF**. SDRAM Controller optimization flag. Should be set to OFF for normal SDRAM operation. ON is a debug feature.

0 Optimization off

1 Optimization on

Bit 15 **SD_CKE**. Clock Enable. This read-only bit monitors the signal GPIO5/SDRAM_CKE as defined below.

0 Clock disabled

1 Clock enabled

Bit 14 **Reserved**.

Bit 13 **SDRAM_32_BIT**. Selects between 32-bit-wide or 16-bit-wide memories.

0 16-bit-wide memories are used

1 A single 32-bit-wide memory is used (only applies for 64-Mbit device)

Bit 12 **SD_SLFR**. Self-refresh request.

When in self-refresh mode, the SDRAM retains data without external clocking. Once self-refresh mode is engaged, the SDRAM provides its own internal clocking, causing it to perform its own refresh cycles.

0 Exit self-refresh mode

1 Enter self-refresh mode

To enter self-refresh mode:

1. Configure GPIO5/SDRAM_CKE for CKE mode. (GPIO_EN [FFFF:2814] bit 5, GPIO_EN_5 = 0)
2. Configure GPIO5/SDRAM_CKE for output mode. (GPIO_CIO [FFFF:2804] bit 5, GPIO_CIO_5 = 0)
3. Put the SDRAM into self-refresh mode (SDRAM_CONFIG [FFFF:3000] bit 12, SD_SLFR = 1). When activated, this bit will drive the GPIO5/SDRAM_CKE line low and force the autorefresh logic to produce one more refresh cycle. This sequence will put the SDRAM into self-refresh mode.

To exit self-refresh mode:

1. The SDRAM is still in self-refresh mode until the ARM produces the first autorefresh. Properly set up the clock delay time before the first autorefresh is issued after the ARM exits self-refresh mode. (SDRAM_CONFIG [FFFF:3000] bits[11:9], SD_TRAS)
2. Force the ARM to exit self-refresh mode and restart the autorefresh process. (SDRAM_CONFIG [FFFF:3000] bit 12, SD_SLFR = 0)

Bits 11–9 **SD_TRAS**. RAS latency in SDRAM clock cycles.

The minimum amount of time that the SDRAM should remain in self-refresh mode (once it has entered this mode). When the SDRAM is in self-refresh mode, the SDRAM is capable of retaining data even if the rest of the system is down without external clocking (minimum 42 ns, which gives 2 cycles at 47.5 MHz).

000	2 SDRAM clock cycles
001	3 SDRAM clock cycles
010	4 SDRAM clock cycles
011	5 SDRAM clock cycles
100	6 SDRAM clock cycles
101	7 SDRAM clock cycles
110	8 SDRAM clock cycles
111	9 SDRAM clock cycles

Bits 8–7 **SD_TRP**. TRP latency in SDRAM clock cycles.

Delay (latency), in clock cycles, for availability of a next row access within a bank(s) once the row has been deactivated, i.e., precharged. (minimum 21.05 ns , which gives 1 cycle at 47.5 MHz)

00	2 SDRAM clock cycles
01	3 SDRAM clock cycles
10	4 SDRAM clock cycles
11	5 SDRAM clock cycles

Bits 6–3 **SD_TRC**. TRC latency in SDRAM clock cycles.

The minimum time interval between successive ACTIVE commands to the same bank (minimum 70 ns , which gives 4 cycles at 47.5 MHz)

0000	2 SDRAM clock cycles
0001	3 SDRAM clock cycles
0010	4 SDRAM clock cycles
0011	5 SDRAM clock cycles
0100	6 SDRAM clock cycles
0101	7 SDRAM clock cycles
0110	8 SDRAM clock cycles
0111	9 SDRAM clock cycles
1000	10 SDRAM clock cycles
1001	11 SDRAM clock cycles
1010	12 SDRAM clock cycles
1011	13 SDRAM clock cycles
1100	14 SDRAM clock cycles
1101	15 SDRAM clock cycles
1110	16 SDRAM clock cycles
1111	17 SDRAM clock cycles

Bits 2–0 **SD_TCAS**. CAS latency in SDRAM clock cycles.

Specifies the READ to data-out delay.

001	CAS latency = 1
010	CAS latency = 2
011	CAS latency = 3

Delay (latency), in clock cycles, between the registration of a READ command and the availability of the first piece of output data.

Note: In order to maximize random short read performances, it is recommended that the lowest allowed CAS latency value be used.

3.9.2 SDRAM Refresh Counter Register

Figure 3–10. SDRAM Refresh Counter Register (SDRAM_REF_COUNT)

Base address = 0xFFFF:3000, Offset Address = 0x0004

31–19	18–16	15–0
Reserved	DIVIDER	REFRESH_COUNTER
RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–19 **Reserved**. These bits are reserved (read as zeros).

Bits 18–16 **DIVIDER**. SDRAM refresh counter predivider.

000	divide by 1
001	divide by 2
010	divide by 4
011	divide by 8
100	divide by 16
101	divide by 32
110	divide by 64
111	divide by 128

Since the resolution of the REFRESH_COUNTER field is large, the pre-divider can be used (but not useful) for the VC547x device. This field is better left in its default value (zero) so that the SDRAM clock is not scaled.

Bits 15–0 **REFRESH_COUNTER**. Refresh counter value in SDRAM clock cycles.

This value is loaded when the refresh down counter goes through zero or when it starts.

The value that is loaded here is determined based on the type and the speed of the SDRAM used. The formula to calculate this value is:

TREF / (RowTotal * SDRAM_CLK Period)

TREF is the required refresh time interval (usually in milliseconds) to refresh a single bank.

RowTotal is the total number of rows within all banks.

SDRAM_CLK Period is one SDRAM clock period value.

For a 16M-bit SDRAM that requires a refresh every 64 ms, with the SDRAM clk equal to the CPU clock running at 47.5 MHz (21.05 ns):

Since a 16M-bit SDRAM has 2 banks and the row resolution is 11 bits, this implies that we have a total of $2 * 2^{11} = 4096$ Rows.

$$\text{REFRESH\_COUNTER Value} = 64*10^{-3} / (4096 * 21.05*10^{-9}) = 742.28$$

Any value less than 742.28 (e.g., 700) will be a good value to use for this scenario.

3.9.3 SDRAM Control Register

Figure 3–11. SDRAM Control Register (SDRAM_CNTL)

Base address = 0xFFFF:3000, Offset Address = 0x0008

31–2	1	0
Reserved	READY	SDRAM_INIT
RW–0	R-0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–2 **Reserved**. These bits are reserved (read as zeros).

Bit 1 **READY**. This bit is used by the Initialization State Machine to indicate to the user when the initialization has completed and the SDRAM is ready for Read/Write operation.

0 SDRAM not ready

1 SDRAM ready

Bit 0 **SDRAM_INIT**. SDRAM initialization bit. This bit is used to kick-start the Initialization State Machine to start initializing the SDRAM once all register programming has taken place.

0 Do not initialize SDRAM

1 Initialize SDRAM (self-clearing bit)

3.9.4 SDRAM Initialization Refresh Counter Register

Figure 3–12. SDRAM Initialization Refresh Counter Register (SDRAM_INIT_CONF)

Base address = 0xFFFF:3000, Offset Address = 0x000C

31–18	17–14	13–0
Reserved	INIT_REF_MAX_CNT	INIT_NOP_MAX_CNT
RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–18 **Reserved**. These bits are reserved (read as zeros).

Bits 17–14 **INIT_REF_MAX_CNT**.

A counter used to store the total number of autorefresh cycles that must be performed once the SDRAM is in IDLE state, i.e., past the 100-µs delay (during initialization phase) and all bank precharge has taken place. Any value above five will be satisfactory.

Bits 13–0 **INIT_NOP_MAX_CNT**.

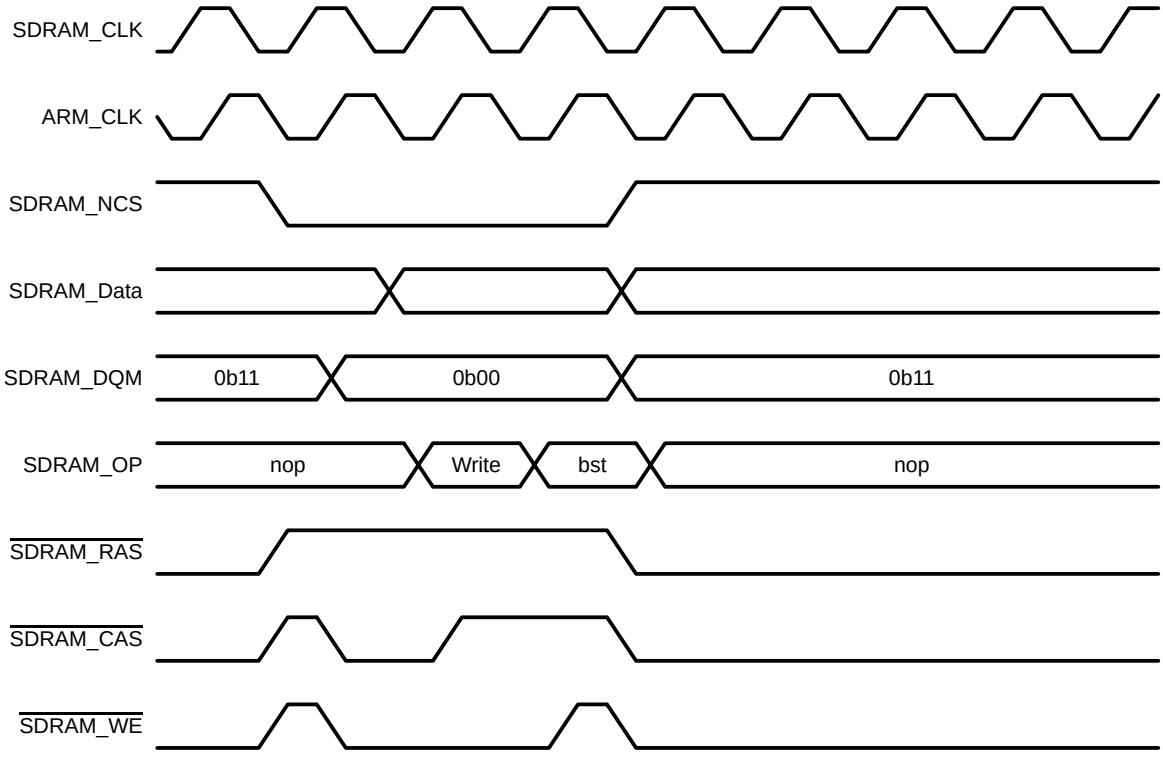
Clock cycle count that is equivalent to a minimum of 100-µs time. During this time, the Initialization State Machine applies the NOP command to the SDRAM, as is required by SDRAMs.

3.10 Waveforms

3.10.1 Waveforms of Read/Write Operations With Rows Enabled/Disabled

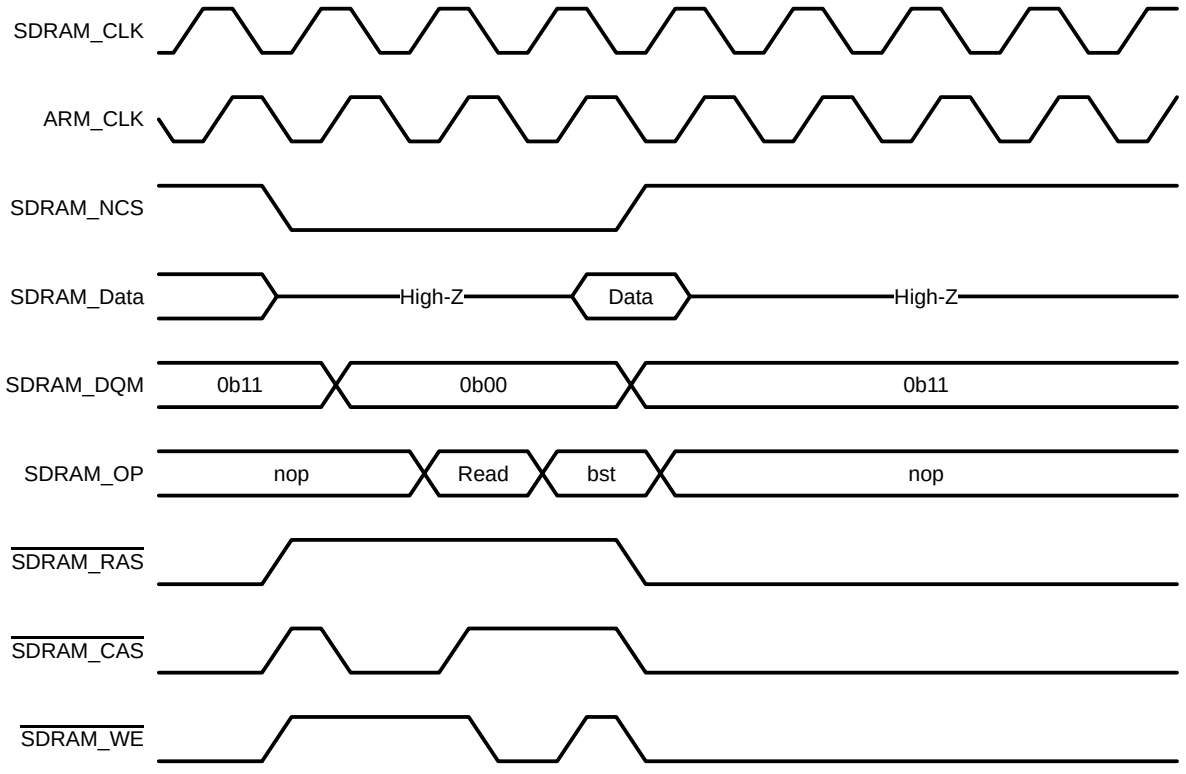
3.10.1.1 Write Waveform With Bank Already Activated

Figure 3–13. Write Operation With Row Already Enabled – Parameters: tcas = 2, trc = 4, trp = 1



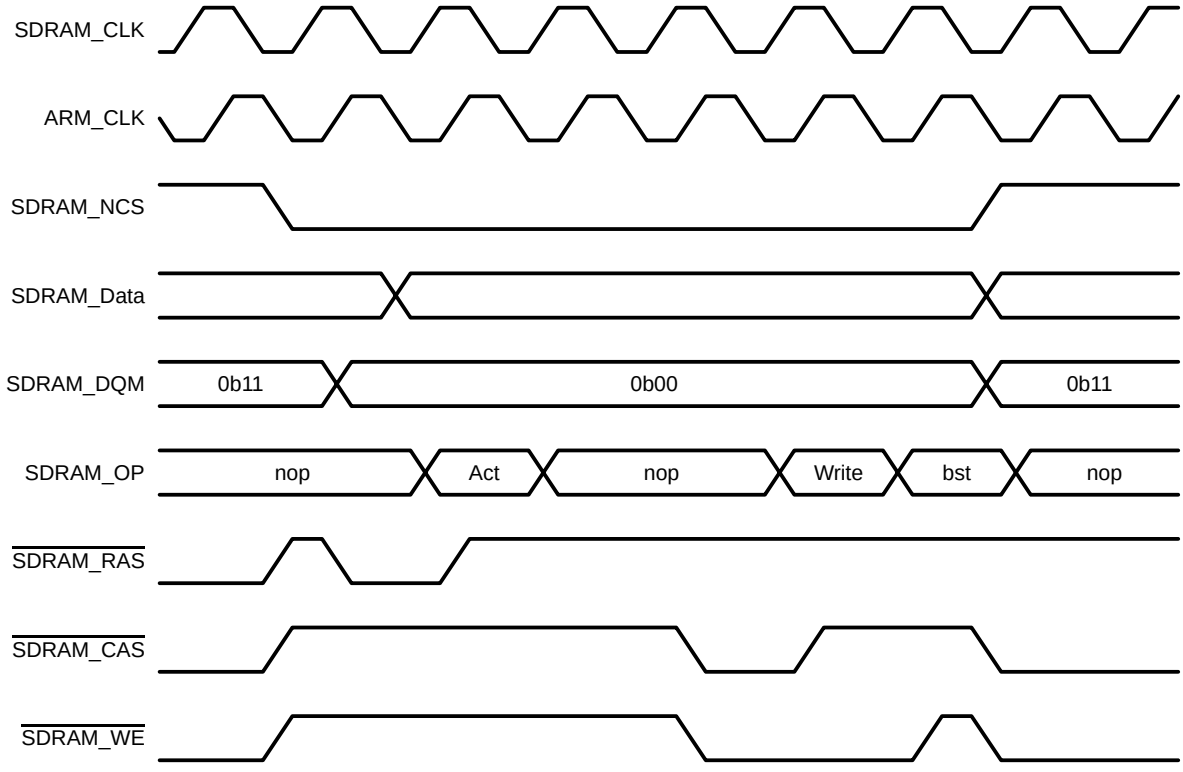
3.10.1.2 Read Waveform With Bank Already Activated

Figure 3–14. Read Operation With Row Already Activated – Parameters: tcas = 2, trc = 4, trp = 1



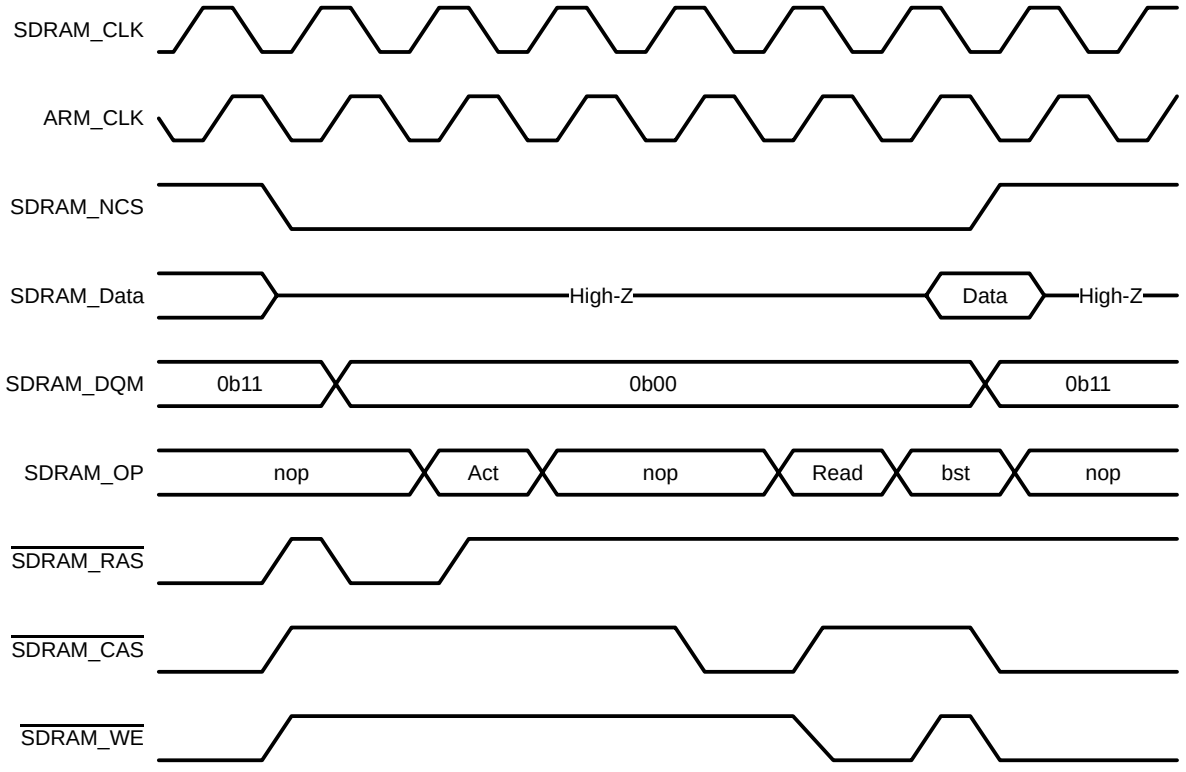
3.10.1.3 Write Waveform With Bank Not Activated

Figure 3–15. Write Operation With Row Disabled – Parameters: tcas = 2, trc = 4, trp = 1



3.10.1.4 Read Waveform With Bank Not Activated

Figure 3–16. Read Operation With Row Disabled – Parameters: tcas = 2, trc = 4



3.10.2 Waveforms With External Transactions (8-, 16-, and 32-Bit Devices)

The following waveforms show external transactions for 8-, 16-, and 32-bit devices with different write, wait-state, and endianness parameters.

Figure 3–17. 8-Bit Device Transaction in Little Endian

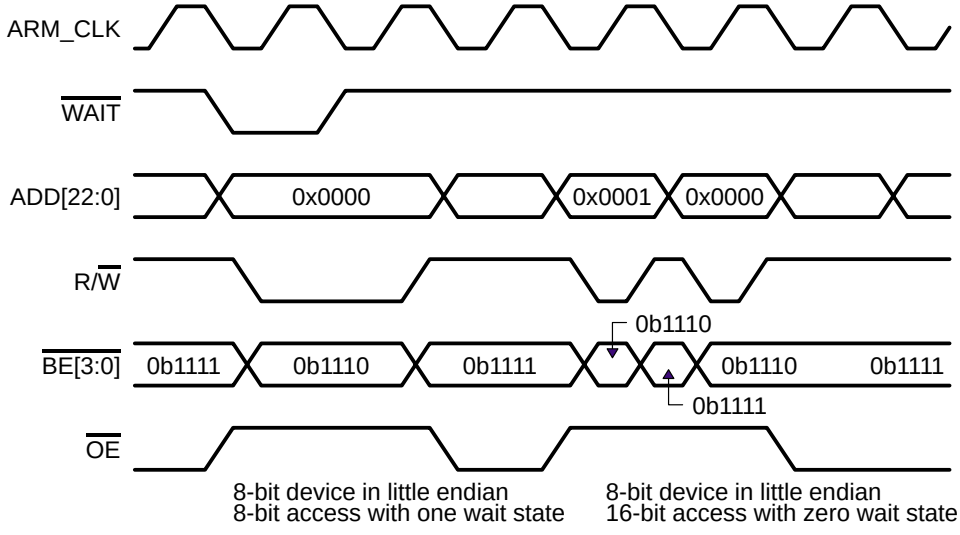


Figure 3–18. 32-Bit Write Access on 8-Bit Big-Endian Device With One Wait State

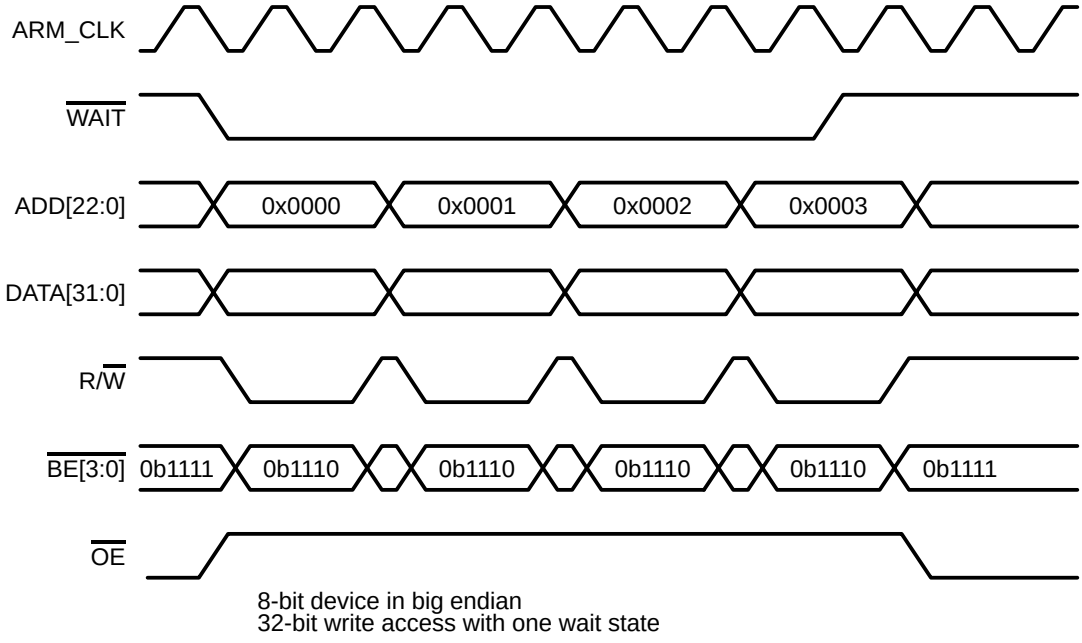


Figure 3–19. 32-Bit Write Access on 8-Bit Little-Endian Device With One Wait State

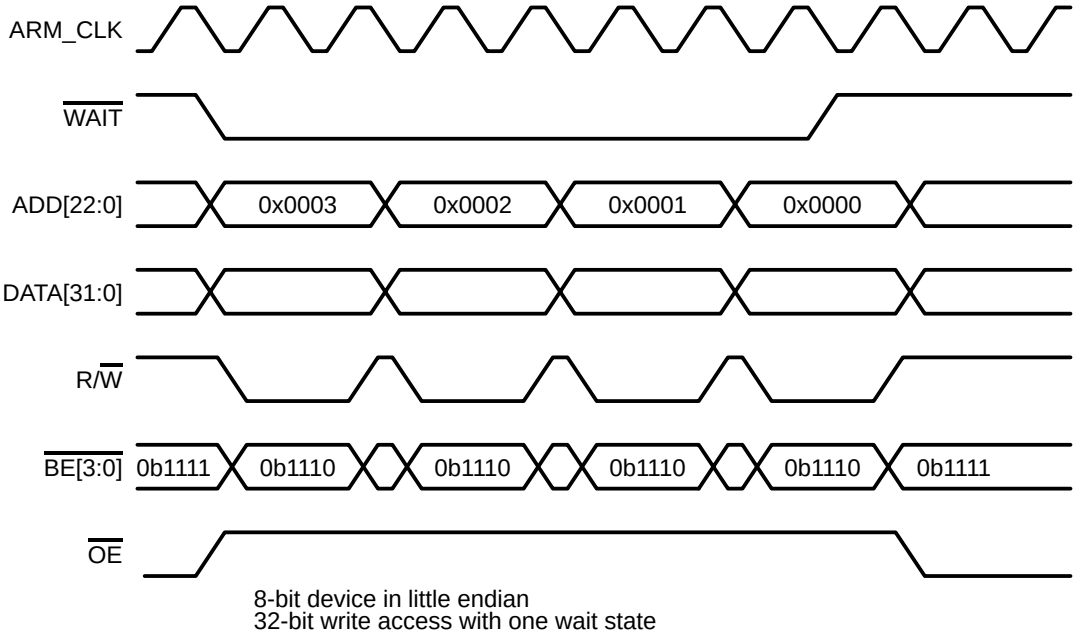
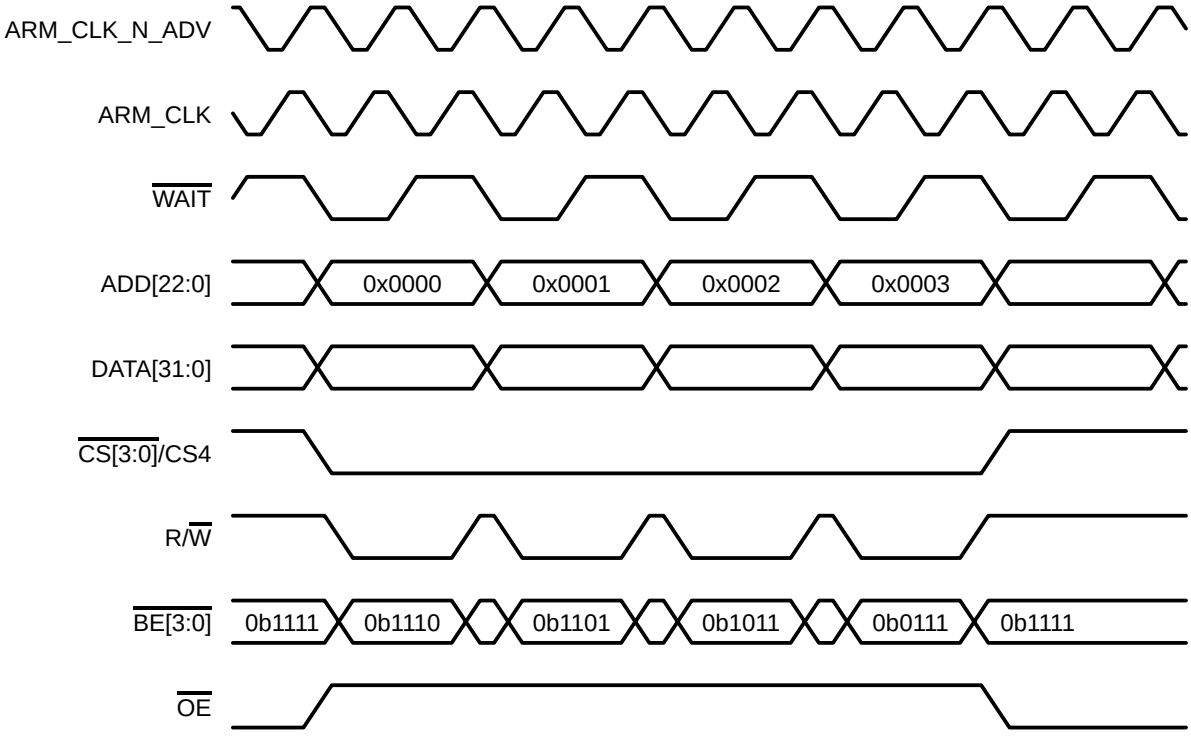
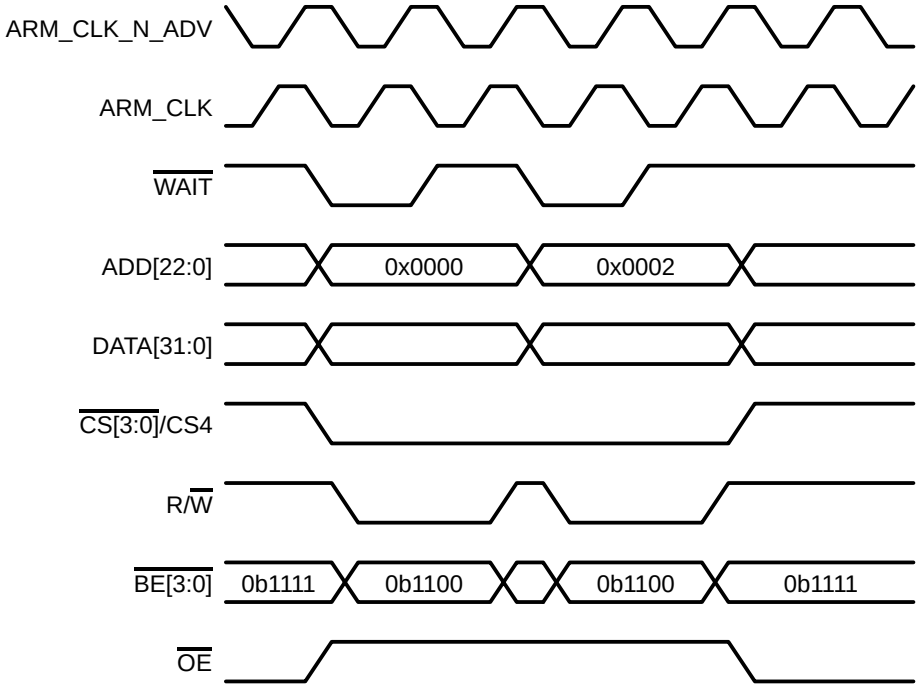


Figure 3–20. 8-Bit Accesses on 32-Bit Device



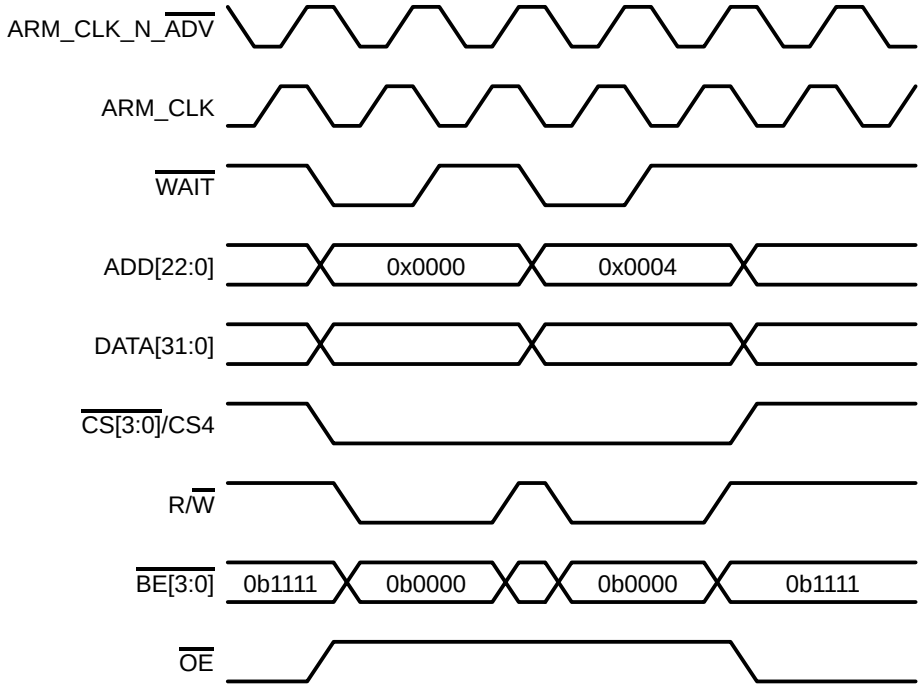
Four 8-bit write accesses on consecutive addresses on 32-bit device with one wait state

Figure 3–21. 16-Bit accesses on 32-Bit Device



Two 32-bit write accesses on consecutive addresses
on 32-bit device with one wait state

Figure 3–22. 32-Bit Accesses on 32-Bit Device



Two 32-bit write accesses on consecutive addresses on 32-bit device with one wait state

Chapter 4

Interrupt Handler

This chapter provides a functional description of the Interrupt Handler (INTH) in the TMS320VC547x DSP device, describes the ARM™ microcontroller unit (MCU) interrupt requests, and shows the MCU accessible registers.

The Interrupt Handler is associated with the ARM™ microcontroller unit (MCU) of the dual-core (MCU + DSP) VC547x device.

Topic **Page**

4.1 Functional Description

The ARM7 owns two interrupt lines: $\overline{\text{IRQ}}$ (low-priority interrupt request) and $\overline{\text{FIQ}}$ (fast interrupt request). To the ARM processor, the FIQ interrupt is of higher priority than the IRQ interrupt, and the IRQ interrupt is automatically masked when entering the FIQ handler (by hardware).

Passing Interrupts to the ARM Processor

In the VC547x, sources of interrupts do not go directly to the ARM processor; instead, they go through the interrupt handler, which is fully programmable and provides up to 16 prioritized and maskable interrupts (IRQ0–15) to the ARM core. Therefore, it is the software's responsibility to configure the interrupt handler for correct passing of interrupts to the ARM processor.

The ARM core receives interrupts from internal modules and from the external chip environment. External interrupts are received by the interrupt handler via the general-purpose I/O (GPIO) pins. Each incoming interrupt can be individually masked using the Mask Interrupt register. One Interrupt Level Register (ILR) is associated with each incoming interrupt. Each interrupt can be individually configured for falling edge enable of an input interrupt line of the ARM core by setting the appropriate bit in the corresponding ILR. All on-chip peripherals are associated with the irq_lines (see Figure 4–1). (The input pins to the interrupt handler are all named irq_lines, but indeed, you have to set up the interrupt handler to define where each input line goes. You can route it to either IRQ or FIQ and define the priority and the active level. It is inside the interrupt handler that you decide, by software, if an interrupt should be an FIQ or an IRQ.)

Management of FIQ and IRQ Interrupts

The management of the FIQ and IRQ interrupts is done in parallel inside the interrupt handler module. In addition, IRQ and FIQ outputs can be reset by software using dedicated bits in the Control register.

The ILR is also responsible for assigning a priority to the corresponding interrupt. If several interrupts have the same priority level, they are sent in a predefined order. (See section 4.3.7, *Interrupt Level Registers (Read/Write)*.)

In addition, the ARM clock must time this module. The latency from an incoming interrupt to the output interrupt generation depends on the number of interrupts arriving at the same time.

- If there is only one, the latency is five ARM clock cycles.
- If all interrupts become active at the same time and are routed to the same output interrupt, the latency can reach 3 + N $\times$ 2 ARM cycles (where N = number of incoming interrupts).

A part of the power management function is also implemented in this module.

4.2 MCU Interrupts

In the VC547x device, the IRQ0–IRQ15 INTH input lines are associated with the on-chip peripherals as shown in Table 4–1 and Figure 4–1. The ARM7 core has two interrupt requests: $\overline{\text{IRQ}}$ and $\overline{\text{FIQ}}$. The INTH input interrupts (IRQ0–IRQ15) can be individually routed to either of the two ARM interrupt requests by setting the appropriate bit in the configuration register.

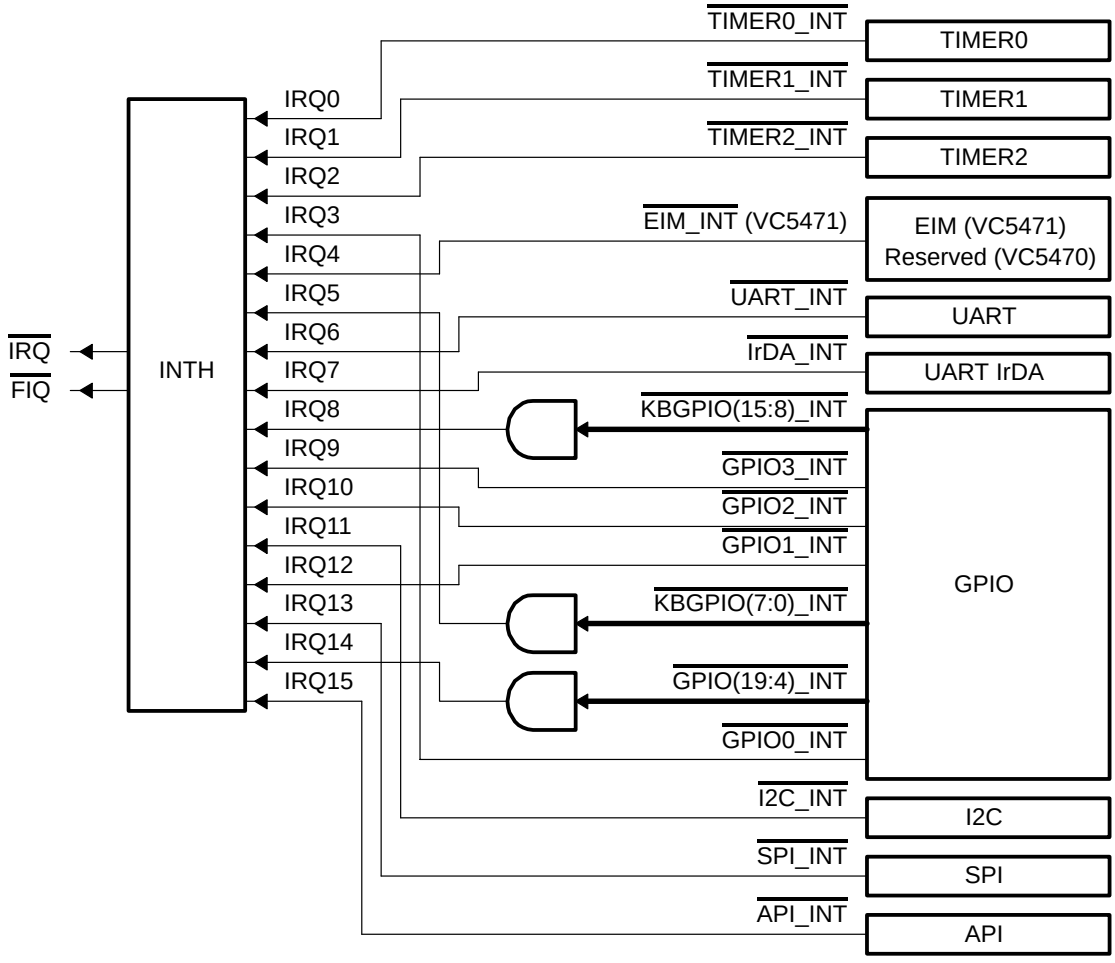
Table 4–1. ARM Peripherals Interrupt Mapping

IRQ Request	Interrupt
IRQ0	Watchdog TIMER0 interrupt
IRQ1	TIMER1 interrupt
IRQ2	TIMER2 interrupt
IRQ3	GPIO0 interrupt
IRQ4	Ethernet interface interrupts (VC5471) 1. transmit interrupt 2. receive interrupt 3. system error Reserved (VC5470)
IRQ5	KBGPIO[7:0] interrupts

Table 4–1. ARM Peripherals Interrupt Mapping (Continued)

IRQ Request	Interrupt
IRQ6	UART interrupts 1. error on receive line 2. receive timeout 3. received character 4. character to transmit 5. status change 6. received XOFF/special character detected 7. CTS/RTS deactivation 8. CTS/RxD activity detection (only in OFF mode)
IRQ7	UART_IRDA interrupts 1. error on receive line 2. receive timeout 3. received character 4. character to transmit 5. status change 6. received XOFF/special character detected 7. CTS/RTS deactivation
IRQ8	KBGPIO[15:8] interrupts
IRQ9	GPIO3 interrupt
IRQ10	GPIO2 interrupt
IRQ11	I2C interrupts
IRQ12	GPIO1 interrupt
IRQ13	SPI interrupts 1. received data 2. data to transmit
IRQ14	GPIO[19:4] interrupts
IRQ15	API interrupt

Figure 4–1. ARM Peripheral Interrupt Mapping Diagram



4.2.1 Internal Registers

Internal registers are not accessible by the MCU. One group is dedicated for processing IRQs and the other for processing FIQs.

1) Processed IRQ (P_IRQ) – Processed FIQ (P_FIQ): Each time an Interrupt has been processed (to determine if it is the next interrupt to be sent to the MCU), the corresponding bit is set to 1.

2) Next IRQ (N_IRQ) – Next FIQ (N_FIQ): Holds the next incoming interrupt to be sent to IRQ or FIQ.

4.2.2 Interrupt Sequence

This section describes the sequence in which interrupts occur. Because the IRQ and FIQ treatments are exactly identical, the following sequence only describes the IRQ interrupt:

- One or several incoming interrupts go down, setting the corresponding bit or bits in the Interrupt register.
- At this time, there are two possible cases:
  - **There is only one incoming interrupt that is active:** If IRQ is not already active, the interrupt handler sends an IRQ.
  - **There are several incoming interrupts that are active:** In this case, the interrupt handler must determine which is the new interrupt to be serviced. To do this, it compares the priority level of an interrupt with the one held in a dedicated internal register (N_IRQ) and stores the one having the highest priority in N_IRQ. It continues this operation until all active interrupts have been processed. If IRQ is not already active, the interrupt handler sends an IRQ.
- When an IRQ is sent, the Source IRQ register is updated (indicating the interrupt contained in N_IRQ) and the priority resolver is reset (and restarted if necessary)
- To know which incoming interrupt has requested an MCU action, the software must read the Source IRQ register. (Only one bit of this register can be active at any time; this bit indicates the active IRQ interrupt.) After that, the software runs the corresponding subroutine.
- To finish this sequence, MCU software must set a dedicated bit (NEW_IRQ_AGR) in the Control register in order to reset the IRQ output and the Source IRQ register, and thus, to allow a new IRQ generation.

4.3 ARM Memory-Mapped Registers

Base address (hex): FFFF:2D00

Register width: 32 bits

All of these registers are controlled directly by the internal VC547x system bus.

Table 4–2. ARM Memory-Mapped Registers

Register	Description	Offset Address
IT_REG	Interrupt Register	00h
MASK_IT_REG	Mask Interrupt Register	04h
SRC_IRQ_REG	Source IRQ Register	08h
SRC_FIQ_REG	Source FIQ Register	0Ch
Reserved		10h
INT_CTRL_REG	Interrupt Control Register	18h
ILR_IRQ_0	Interrupt Level Register 0	1Ch
ILR_IRQ_1	Interrupt Level Register 1	20h
ILR_IRQ_2	Interrupt Level Register 2	24h
ILR_IRQ_3	Interrupt Level Register 3	28h
ILR_IRQ_4	Interrupt Level Register 4	2Ch
ILR_IRQ_5	Interrupt Level Register 5	30h
ILR_IRQ_6	Interrupt Level Register 6	34h
ILR_IRQ_7	Interrupt Level Register 7	38h
ILR_IRQ_8	Interrupt Level Register 8	3Ch
ILR_IRQ_9	Interrupt Level Register 9	40h
ILR_IRQ_10	Interrupt Level Register 10	44h
ILR_IRQ_11	Interrupt Level Register 11	48h
ILR_IRQ_12	Interrupt Level Register 12	4Ch
ILR_IRQ_13	Interrupt Level Register 13	50h
ILR_IRQ_14	Interrupt Level Register 14	54h

Table 4–2. ARM Memory-Mapped Registers (Continued)

Register	Description	Offset Address
ILR_IRQ_15	Interrupt Level Register 15	58h
IRQ_SLEEP_REG	IRQ Sleep Register	5Ch

4.3.1 Interrupt Register

In case of an edge-sensitive interrupt, the Interrupt register stores an incoming interrupt. When the MCU accesses SRC_IRQ_REG or SRC_FIQ_REG register, the bit corresponding to the interrupt which has requested MCU action is reset.

The MCU can also clear each bit individually. To do this, the MCU must write a zero to the corresponding bits at the IT_REG address (the other bits will keep their previous value). This can be used just before the MCU unmasks some interrupts, and thus, “forgets” some interrupt occurrences.

The MCU can read this register. For an incoming edge-sensitive interrupt, the read value corresponds to the value held in the storage element.

Figure 4–2. Interrupt Register (IT_REG)

Base address = 0xFFFF:2D00, Offset Address = 0x0000

31–16
Reserved

15	14	13	12	11	10	9	8
IRQ15	IRQ14	IRQ13	IRQ12	IRQ11	IRQ10	IRQ9	IRQ8
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**.

Bit 15 **IRQ15.** It stores an incoming interrupt.

Read access:

– Edge-sensitive interrupt: Value held in the storage element

Write access:

0 IRQ15 is cleared

1 IRQ15 keeps its previous value

⇓ ⇓

Bit 0 **IRQ0.** It stores an incoming interrupt.

Read access:

– Edge-sensitive interrupt: Value held in the storage element

Write access:

0 IRQ0 is cleared

1 IRQ0 keeps its previous value

4.3.2 Mask Interrupt Register

Each incoming interrupt can be individually masked by this register.

MASK_IT_REG operates after IT_REG. This means that occurrences of incoming interrupts are always stored in IT_REG.

Figure 4–3. Mask Interrupt Register (MASK_IT_REG)

Base address = 0xFFFF:2D00, Offset Address = 0x0004

31–16
Reserved

15	14	13	12	11	10	9	8
IRQ_15_ MSK	IRQ_14_ MSK	IRQ_13_ MSK	IRQ_12_ MSK	IRQ_11_ MSK	IRQ_10_ MSK	IRQ_9_ MSK	IRQ_8_ MSK
RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1

7	6	5	4	3	2	1	0
IRQ_7_MSK	IRQ_6_MSK	IRQ_5_MSK	IRQ_4_MSK	IRQ_3_MSK	IRQ_2_MSK	IRQ_1_MSK	IRQ_0_MSK
RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**.

Bit 15 **IRQ_15_MSK.** Disable IRQ_15 interrupt

⇓ ⇓

Bit 0 **IRQ_0_MSK.** Disable IRQ_0 interrupt

4.3.3 Source IRQ Register

Indicates the active IRQ interrupt. Only one bit of this register is active at a given time.

Figure 4–4. Source IRQ Register (SRC_IRQ_REG)

Base address = 0xFFFF:2D00, Offset Address = 0x0008

31–16
Reserved
R-0

15	14	13	12	11	10	9	8
IRQ15_ IRQ	IRQ14_ IRQ	IRQ13_ IRQ	IRQ12_ IRQ	IRQ11_ IRQ	IRQ10_ IRQ	IRQ9_ IRQ	IRQ8_ IRQ
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
IRQ7_IRQ	IRQ6_IRQ	IRQ5_IRQ	IRQ4_IRQ	IRQ3_IRQ	IRQ2_IRQ	IRQ1_IRQ	IRQ0_IRQ
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Note: R = Read access; value following dash (–) = value after reset

Bits 31–16 **Reserved**.

Bit 15 **IRQ15_IRQ**. Indicates the active IRQ interrupt.

0 IRQ15 is not the active interrupt
1 IRQ15 is the active interrupt

⇓ ⇓

Bit 0 **IRQ0_IRQ**. Indicates the active IRQ interrupt.

0 IRQ0 is not the active interrupt
1 IRQ0 is the active interrupt

4.3.4 Source FIQ Register

Indicates the active FIQ interrupt. Only one bit of this register is active at a given time.

Figure 4–5. Source FIQ Register (SRC_FIQ_REG)

Base address = 0xFFFF:2D00, Offset Address = 0x000C

31–16
Reserved
R-0

15	14	13	12	11	10	9	8
IRQ15_ FIQ	IRQ14_ FIQ	IRQ13_ FIQ	IRQ12_ FIQ	IRQ11_ FIQ	IRQ10_ FIQ	IRQ9_ FIQ	IRQ8_ FIQ
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
IRQ7_FIQ	IRQ6_FIQ	IRQ5_FIQ	IRQ4_FIQ	IRQ3_FIQ	IRQ2_FIQ	IRQ1_FIQ	IRQ0_FIQ
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Note: R = Read access; value following dash (–) = value after reset

Bits 31–16 **Reserved**.

Bit 15 **IRQ15_FIQ**. Indicates the active FIQ interrupt.

0 IRQ15 is not the active interrupt
1 IRQ15 is the active interrupt

⇓ ⇓

Bit 0 **IRQ0_FIQ**. Indicates the active FIQ interrupt.

0 IRQ0 is not the active interrupt
1 IRQ0 is the active interrupt

4.3.5 Interrupt Control Register

Figure 4–6. Interrupt Control Register (INT_CTRL_REG)

Base address = 0xFFFF:2D00, Offset Address = 0x0018

31–2	1	0
Reserved	NEW_FIQ_AGR	NEW_IRQ_AGR
	RW-0	RW-0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–2 **Reserved**.

Bit 1 **NEW_FIQ_AGR**. New FIQ agreement.
Reset FIQ output. Clear source FIQ register. Enables a new FIQ generation

Reset by internal logic.

Bit 0 **NEW_IRQ_AGR**. New IRQ agreement.
Reset IRQ output. Clear source IRQ register. Enables a new IRQ generation

Reset by internal logic.

Note: IRQ (FIQ) output and SRC_IRQ_REG and SRC_IRQ_BIN_REG (SRC_FIQ_REG) registers are reset only if the bit in the Interrupt register (IT_REG) corresponding to the interrupt having requested MCU action is already cleared or masked.

For an edge-sensitive interrupt, the Interrupt register bit is deactivated when reading the SRC_IRQ_REG or SRC_IRQ_BIN_REG (SRC_FIQ_REG) registers.

4.3.6 IRQ Sleep Register

Figure 4–7. IRQ Sleep Register (IRQ_SLEEP_REG)

Base address = 0xFFFF:2D00, Offset Address = 0x005C

31–1	0
Reserved	IRQ_SLEEP
	RW-0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–1 **Reserved**.

Bit 0 **IRQ_SLEEP**. When set to 1, stops the state machine. Should be set to 0 for normal usage

4.3.7 Interrupt Level Registers (Read/Write)

There is one ILR_IRQ register per incoming IRQ interrupt.

Base address = 0xFFFF:2D00

Table 4–3. Offset Addresses of Interrupt Level Registers 0–15 (ILR_IRQ_0 – ILR_IRQ_15)

Offset Address (Hex)	Name	Corresponding Interrupt
0x001C	ILR_IRQ_0	irq_0
0x0020	ILR_IRQ_1	irq_1
0x0024	ILR_IRQ_2	irq_2
0x0028	ILR_IRQ_3	irq_3
0x002C	ILR_IRQ_4	irq_4
0x0030	ILR_IRQ_5	irq_5
0x0034	ILR_IRQ_6	irq_6
0x0038	ILR_IRQ_7	irq_7
0x003C	ILR_IRQ_8	irq_8
0x0040	ILR_IRQ_9	irq_9
0x0044	ILR_IRQ_10	irq_10
0x0048	ILR_IRQ_11	irq_11
0x004C	ILR_IRQ_12	irq_12
0x0050	ILR_IRQ_13	irq_13
0x0054	ILR_IRQ_14	irq_14
0x0058	ILR_IRQ_15	irq_15

Since all ILR_IRQ registers are the same, only ILR_IRQ_0 is shown (see Figure 4–8).

4.3.8 Interrupt Level Register 0

Figure 4–8. Interrupt Level Register 0 (ILR_IRQ_0)

Base address = 0xFFFF:2D00, Offset Address = 0x001C (see Table 4–3 for other offset addresses)

31–6	5	4–1	0
Reserved	SENSE_EDGE	PRIORITY	FIQ
	RW-0	RW-0	RW-0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–6 **Reserved**.

Bit 5 **SENSE_EDGE**.

0 The corresponding interrupt is disabled.

1 The corresponding interrupt is falling-edge sensitive enabled.

Bits 4–1 **PRIORITY**. Defines the priority level when the corresponding interrupt is routed to IRQ.

0 Is the highest priority level

N–1 Is the lowest priority level

Bit 0 **FIQ**.

0 The corresponding interrupt is routed to IRQ.

1 The corresponding interrupt is routed to FIQ.

Predefined Order in Case of Identical Priority Level

Assuming that all interrupts have the same priority level and are active at the same moment, the order of servicing will be: IRQ_15 , IRQ_14, …. IRQ_0.

Chapter 5

Clock Management Module

This chapter provides an overview of the clock management module, describes the three modes of clock operation for both the DSP and ARM subsystems, shows the clock module registers, and discusses the phase-locked loop (PLL) clock source.

Topic **Page**

5.1 Clock Management Module Overview

The ARM subsystem clock management module in the VC547x device is in charge of controlling clock activity for the DSP, MCU, and peripherals. It includes configuration registers for DSP and MCU clock frequency programming. The clock module also manages the reset of all modules connected to the MCU.

Specifically, the clock module is responsible for managing the clock used inside the ARM subsystem. This module also manages the different reset signals found inside the ARM subsystem as well as the DSP clock generator during a start-up or reset condition. Once the ARM subsystem clock module initializes the DSP clock generator operation, the DSP can change its clock frequency by reprogramming its own CLKMD register. Both the ARM and DSP PLLs are software-programmable resulting in a high level of flexibility to generate independent clock frequencies used by both subsystems. The ARM subsystem clock module is software controllable through a set of registers that allow application software to selectively stop any clock, reset a module, and control the DSP PLL, as well as the boot mode of the DSP, through control of the MPNMC and APIBN signals.

Both the ARM PLL and the DSP PLL are software-programmable. Immediately following reset or power-up, the content of the respective registers (PLL_REG_ARM and CLKMD_DSP) for both subsystems depends upon the status of the programmable ports on their respective PLLs. For the ARM subsystem, these ports are hardwired so that programmable capability of the default mode is not available. For the DSP subsystem, the programmable ports to the DSP PLL are connected to the output of a register that is controlled by ARM. This allows the DSP PLL default values to be programmable under control of the ARM subsystem; and consequently, the ARM subsystem initializes the DSP clock generator via software. The six ports that are connected to each PLL initialize the PLLMUL, the PLLNDIV, and the PLLON/OFF fields of the these registers.

5.1.1 Clock Operation Modes

The three possible modes of clock operation for both subsystems are:

- ❏ PLL mode (sometimes known as Normal mode)
- ❏ Divide (DIV) mode
- ❏ Low-Power mode

The normal mode makes use of the PLL and eventually of the voltage controlled oscillator (VCO) to generate the desired clock frequency output. The Divide mode does not use the PLL or the VCO, but divides down the input clock signal (REFCLK) by 2. This mode of operation should not be used for the normal mode of operation.

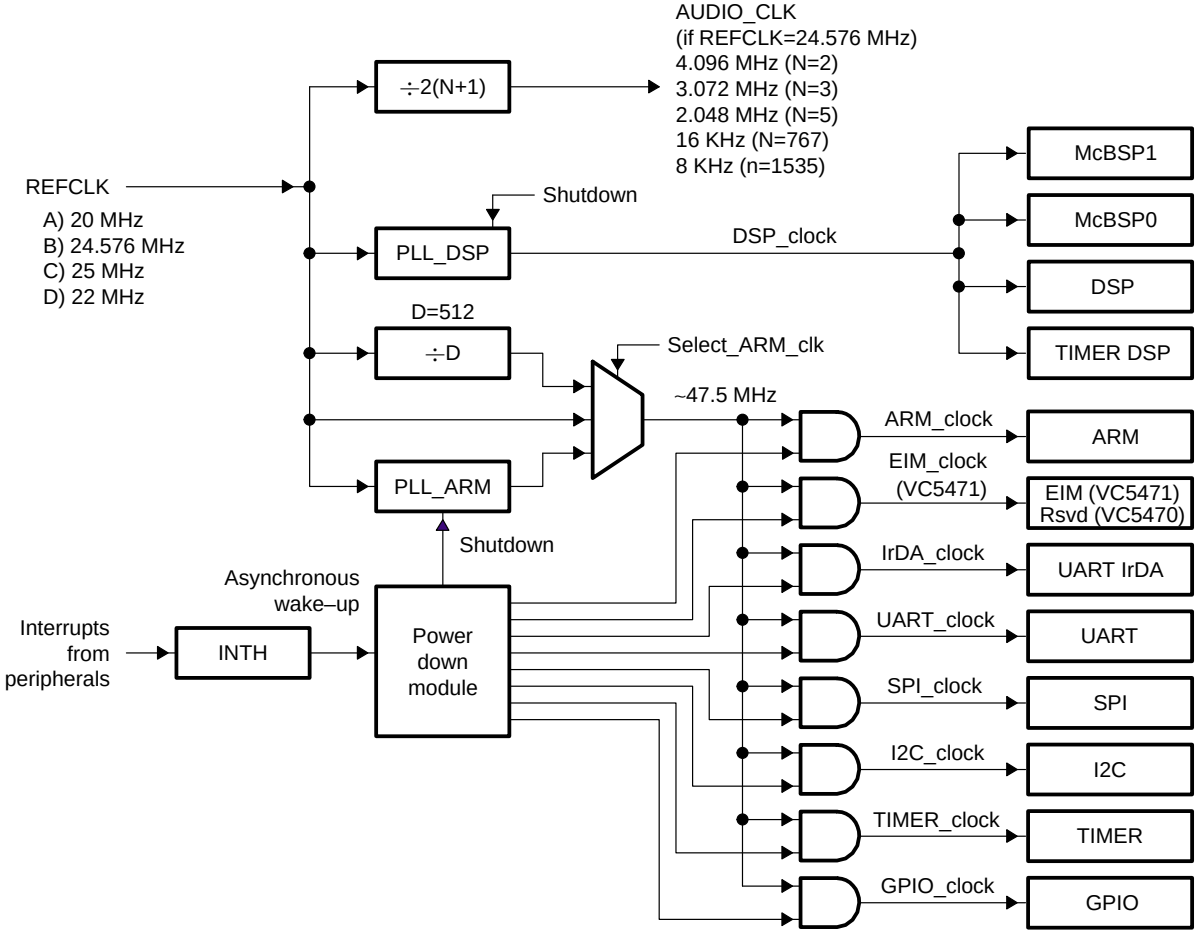
In Low-Power mode, the input clock (REFCLK) is divided by the content of the LOW_POWER_REG_VALUE register to generate clock output. The values that are allowed in the LOW_POWER_REG_VALUE register are ($512 \leq x \leq 1023$), and 0. When the value 0 is used, the clock is stopped. For this reason, values between 512 and 1023 should be used when a clock signal with a divide by 512 frequency is required by an application.

It is possible to read and write all described registers, which are aligned on 32-bit addresses in the VC547x's memory map. All unused bits are read as 0 by the software.

5.1.2 Features Controlled by the Clock Management Module

- ❏ Control of DSP PLL signals
- ❏ Control of DSP boot mode
- ❏ Control of DSP reset
- ❏ Control of stop bit for all ARM subsystem modules
- ❏ Control of RESET for all ARM subsystem modules
- ❏ Control of WAKEUP for all ARM subsystem modules
- ❏ Control of external NRESET signal
- ❏ Control of AUDIO clock generation
- ❏ Control of AUDIO clock stop mechanism
- ❏ Control of input source clock
- ❏ Control of low-power clock
- ❏ Control of the PLL VOLTAGE CONTROLLED OSCILLATOR (PLL-VCO)

Figure 5–1. Clock Management Module



5.2 Clock Module Register Tables

5.2.1 CLKM Module Registers

Base address (hex): FFFF:2F00

Register width: 32 bits

Table 5–1. Clock Module (CLKM) Registers

Register	Description	Offset Address
CLKM_REG	Clock Configuration Register	00h
DSP_REG	DSP Phase-Locked Loop Register	04h
WAKEUP_REG	Interrupt Clock Wakeup Register	08h
AUDIO_CLK	Audio Rate Register	0Ch
CLKM_CNTL_RESET	Reset Control Register	10h
WATCHDOG_STATUS	Watchdog Status Register	14h
RESET_REG	Reset Register	18h
LOW_POWER_REG	Low-Power Mode Register	1Ch
LOW_POWER_REG_VALUE	Low-Power Value Register	20h

5.2.2 PLL_REG Register (ARMSS)

Base address (hex): FFFF:3200

Register width: 32 bits

Table 5–2. PLL_REG Register (ARMSS)

Register	Description	Offset Address
ARMSS	PLL_REG Clock Control Register	00h

5.2.3 CLKMD Register (DSPSS)

Address (hex): 0058

Register width: 32 bits

Table 5–3. CLKMD Register (DSPSS)

Register	Description	Address
DSPSS	CLKMD Clock Control Register	0058h

5.3 DSP Subsystem Control

Within the VC547x device, the ARM subsystem (ARMSS) has a master role to the DSP subsystem (DSPSS). It sees the DSP subsystem as a peripheral/slave and has the power to control much of its functionality. Two responsibilities of the ARM subsystem are initializing the DSP PLL during power-up time and reset, and controlling the DSP Boot mode. The ARMSS makes use of one of its memory-mapped register, DSP_REG, to control these tasks. The contents of the DSP_REG drives the internal ports (not available externally) connected to the DSPSS with the appropriate state, as it is programmed, resulting in a flexible/programmable control. Figure 5–2 illustrates the DSP_REG register.

5.3.1 DSP Phase-Locked Loop Register

Figure 5–2. DSP Phase-Locked Loop Register (DSP_REG)

Address (hex): Base = 0xFFFF:2F00, Offset = 0x0004

31–11	10	9	8
Reserved	DSP_MPNMC	DSP_APIBN	DSP_PLL_SHUTOFF
	RW–0	RW–0	RW–0

7	6	5	4	3	2	1	0
DSP_PLL_FRRSN	DSP_PLL_FRPLL_DIVN	DSP_PLL_FRPLLONOFF	DSP_PLL_FRDIV0	DSP_PLL_FRDIV1	DSP_PLL_FRDIV2	DSP_PLL_FRDIV3	DSP_PLL_FRDIVN
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–11 **Reserved.**

Bit 10 **DSP_MPNMC**.

0 microcontroller mode
1 microprocessor mode

Bit 9 **DSP_APIBN**.

0 API boot mode
1 not API boot mode

Bit 8 **DSP_PLL_SHUTOFF**.

0 DSP PLL not shut off
1 DSP PLL shut off

Bit 7 **DSP_PLL_FRRSN**.

0 DSP PLL reset active

1 DSP PLL reset inactive

Bit 6 **DSP_PLL_FRPLL_DIVN**. Port value PLLNDIV of DSP PLL module.

Bit 5 **DSP_PLL_FRPLLONOFF**. Port value PLLON/OFF of DSP CLKMD register.

Bit 4 **DSP_PLL_FRDIV0**. Port value PLLMUL B-12 of DSP CLKMD register.

Bit 3 **DSP_PLL_FRDIV1**. Port value PLLMUL B-13 of DSP CLKMD register.

Bit 2 **DSP_PLL_FRDIV2**. Port value PLLMUL B-14 of DSP CLKMD register.

Bit 1 **DSP_PLL_FRDIV3**. Port value PLLMUL B-15 of DSP CLKMD register.

Bit 0 **DSP_PLL_FRDIVN**. Port value PLLDIV B-11 of DSP CLKMD register.

Bits 10 and 9 of the DSP_REG register control the DSP boot mode. When the DSP comes out of reset, it begins executing DSP code from DSP program space at address 0xFF80. Based on the status of DSP_MPNMC and DSP_APIBN, the DSP boot program at this location can be downloaded by the ARMSS or it can already be residing in DSP memory. Table 5–4 shows the available DSP boot modes that can be achieved by programming these two fields appropriately. When choosing the API memory mode, the ARMSS is responsible for downloading the required boot code to the DSP starting at DSP program memory address 0xFF80, and for holding the DSP in reset long enough to finish downloading the DSP program boot code.

Table 5–4. DSP Boot Mode

DSP_MPNMC	DSP_APIBN	DSP Boot Memory
0	0	API memory
0	1	On-chip RAM
1	0	API memory
1	1	External DSP memory

Bit 8, DSP_PLL_SHUTOFF, is used to enable and disable the input clock to DSP PLL. When the DSP PLL is shut off (disabled), the voltage controlled oscillator is turned off. While in this stage, the DSP can operate in DIV or Power-Down mode. Note that the DIV mode is forbidden for normal use, employed only for test purposes and PLL lock-up time stage, and should be avoided.

Bits 7 to 0 are used to hold the state of the attached internal pins to the DSP PLL. This also programs the DSP CLKMD register fields with default values. These fields initialize the PLLMUL, PLLCOUNT, and PLLON/OFFF fields of the CLKMD registers. For a more detailed description of the CLKMD register and its field definitions, see section 5.5.2, *CLKMD Clock Control Register (DSPSS).*

Bit 6, DSP_PLL_FRPLL_DIVN, will select if the DSP clock generator operates in DIV mode or PLL mode. During Reset DIV mode is chosen.

Bit 5, DSP_PLL_FRPLLONOFF, will select if the voltage controlled oscillator (VCO) is turned on or off. During reset, the VCO is turned off.

Bits 4 to 1, DSP_PLL_FRDIV0,1,2,3, will initialize one of the required fields (used in conjunction with PLLDIV and PLLNDIV) to define the frequency multiplier to the input clock (REFCLK). Since DSP_PLL_FRPLL_DIVN (bit 6) is 0 during reset, any value between 0 and 14 results in a Divide-by-Two mode of operation.

Note: The only other available option for this field is a divide-by-4 option (REFCLK/4), and this can be chosen by programming this field value to 15.

The state of **bit-field 0** initializes one of the required fields (the others are PLLMUL and PLLNDIV) to define the frequency multiplier to the input clock. During reset, the state of this value does not affect the operation of the DSP clock generator when bit 6, DSP_PLL_FRPLL_DIVN, is set to zero configuring the DSP PLL to operate in DIV mode.

> **Note:** The state of this field determines if the DSP frequency multiplier, *k*, is to be an integer or non-integer value when the DSP clock generator is configured to function in PLL mode (which is also known as the normal mode).

As shown in the DSP_REG definition, upon reset (from the RESET values shown), the DSP subsystem is ready to boot from API memory space with its PLL module held in reset, its analog part disabled, and its frequency multiplier programmed in Divide-by-Two mode.

The DSP_REG register is used to program the DSP PLL during reset or power-up time. In order to control the duration of time that the DSP is held in reset and also to control the external reset signal to external peripherals, the ARMSS makes use of another memory-mapped register, CLKM_CNTL_RESET.

5.3.2 Reset Control Register

The CLKM_CNTL_RESET register is used to control DSP reset as well as the external reset signal, $\overline{\text{RESET}}$, to external peripherals. The CLKM_CNTL_RESET register resets the DSP via an internal DSP pin connected to the ARM subsystem. This register has only two fields, each with a single bit. The CLKM_CNTL_RESET register is illustrated in Figure 5–3.

Figure 5–3. Reset Control Register (CLKM_CNTL_RESET)

Address (hex): Base = 0xFFFF:2F00, Offset = 0x0010

31–2	1	0
Reserved	EXTERNAL_ NRESET	DSP_ NRESET
	RW–0	RW–1

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–2 **Reserved**.

Bit 1 **EXTERNAL_NRESET**.

0 $\overline{\text{RESET\_OUT}}$ is 1
1 $\overline{\text{RESET\_OUT}}$ is 0

Bit 0 **DSP_NRESET**.

0 DSP released from reset
1 DSP held in reset

By resetting/setting the state of DSP_NRESET, the DSP can be released/held in reset, respectively.

The EXTERNAL_NRESET is used to generate an appropriate reset signal to external peripherals via the external pin, $\overline{\text{RESET\_OUT}}$.

Note: The status of the DSP_NRESET signal is set upon power-up, reset, or during a watchdog reset, while the EXTERNAL_NRESET signal is only reset during power-up or reset.

5.4 ARM Subsystem Control

The DSP is one of the slave devices that is controlled by the ARMSS and is seen as a peripheral. The ARMSS has other peripherals that are under its control that also require clock sources. The ARMSS supplies these peripherals with clock sources and controls the operation of the clocks independently via two additional ARM memory-mapped registers, CLKM_REG and WAKEUP_REG. Figure 5–4 shows CLKM_REG and Figure 5–5 shows WAKEUP_REG.

5.4.1 Clock Configuration Register

Figure 5–4. Clock Configuration Register (CLKM_REG)

Address (hex): Base = 0xFFFF:2F00, Offset = 0x0000

31–13	12	11	10	9	8
Reserved	ARM_ CLK_STOP	SPI_ CLK_STOP	I2C_ CLK_STOP	UART_MODEM_ CLK_STOP	UART_IRDA_ CLK_STOP
	RW–0	RW–0	RW–0	RW–0	RW–0

7	6	5	4	3	2	1	0
WATCHDOG_ CLK_STOP	TIMER1_ CLK_STOP	TIMER2_ CLK_STOP	EIM_ CLK_STOP (VC5471) Reserved (VC5470)	IRQ_ CLK_STOP	ARMIO_ CLK_STOP	SDRAM_ CLK_STOP	AUDIO_ CLK_STOP
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–13 **Reserved**. Always return 0.

Bit 12 **ARM_CLK_STOP**. Operate the ARM logic clock as defined below.

0 Block clock active
1 Block clock stopped

Bit 11 **SPI_CLK_STOP**. Operate the Serial Port Interface logic clock as defined below.

0 Block clock active
1 Block clock stopped

Bit 10 **I2C_CLK_STOP**. Operate the I^2C logic clock as defined below.

0 Block clock active
1 Block clock stopped

Bit 9 **UART_MODEM_CLK_STOP**. Operate the UART Modem logic clock as defined below.

0 Block clock active

1 Block clock stopped

Bit 8 **UART_IRDA_CLK_STOP**. Operate the UART IrDA logic clock as defined below.

0 Block clock active

1 Block clock stopped

Bit 7 **WATCHDOG_CLK_STOP**. Operate the Timer 0 logic (configured as a watchdog timer) clock as defined below.

0 Block clock active

1 Block clock stopped

Bit 6 **TIMER1_CLK_STOP**. Operate the Timer 1 logic clock as defined below.

0 Block clock active

1 Block clock stopped

Bit 5 **TIMER2_CLK_STOP**. Operate the Timer 2 logic clock as defined below.

0 Block clock active

1 Block clock stopped

Bit 4 **EIM_CLK_STOP** (VC5471). Operate the Ethernet Interface logic clock as defined below.

0 Block clock active

1 Block clock stopped

Reserved (VC5470).

Bit 3 **IRQ_CLK_STOP**. Operate the Interrupt logic clock as defined below.

0 Block clock active

1 Block clock stopped

Bit 2 **ARMIO_CLK_STOP**. Operate the ARM I/O logic clock as defined below.

0 Block clock active

1 Block clock stopped

Bit 1 **SDRAM_CLK_STOP**. Operate the SDRAM logic clock as defined below.

0 Block clock active

1 Block clock stopped

Bit 0 **AUDIO_CLK_STOP**. Operate the Audio Rate logic clock as defined below.

0 Block clock active

1 Block clock stopped

5.4.2 Interrupt Clock Wakeup Register

The contents of the WAKEUP_REG is transferred inside the CLKM_REG each time a wake-up condition occurs. A wake-up condition in the VC547x device is triggered each time an interrupt is detected. If an application decides not to use a specific module, it has to stop the clock by writing the corresponding bit in the CLKM_REG as well as in the WAKEUP_REG register. By not doing so and after the first interrupt detection, the WAKEUP_REG register is transferred inside the CLKM_REG, thus re-enabling the module clock.

Since the ARMSS has total control of its peripherals, it has to have the ability to control its peripheral reset mechanism independently. It is also possible for any ARMSS peripheral to control the reset signals for other peripherals, with the exception of the processors ARMSS and DSPSS. This is accomplished via another ARM memory-mapped register, RESET_REG. Figure 5–5 shows WAKEUP_REG.

Figure 5–5. Interrupt Clock Wakeup Register (WAKEUP_REG)

Address (hex): Base = 0xFFFF:2F00, Offset = 0x0008

31–13	12	11	10	9	8
Reserved	ARM_ WAKEUP	SPI_ WAKEUP	I2C_ WAKEUP	UART_MODEM_ WAKEUP	UART_IRDA_ WAKEUP
	RW–0	RW–0	RW–0	RW–0	RW–0

7	6	5	4	3	2	1	0
WATCHDOG_ WAKEUP	TIMER1_ WAKEUP	TIMER2_ WAKEUP	EIM_ WAKEUP (VC5471) Reserved (VC5470)	IRQ_ WAKEUP	ARMIO_ WAKEUP	SDRAM_ WAKEUP	AUDIO_ WAKEUP
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–13 **Reserved**. Always return 0.

Bit 12 **ARM_WAKEUP**. An ARM interrupt received will force a write of the contents of this bit into ARM_CLK_STOP (bit 12 of CLKM_REG [FFFF:2F00]).

Bit 11 **SPI_WAKEUP**. An ARM interrupt received will force a write of the contents of this bit into SPI_CLK_STOP (bit 11 of CLKM_REG [FFFF:2F00]).

Bit 10 **I2C_WAKEUP**. An ARM interrupt received will force a write of the contents of this bit into I2C_CLK_STOP (bit 10 of CLKM_REG [FFFF:2F00]).

Bit 9 **UART_MODEM_WAKEUP**. An ARM interrupt received will force a write of the contents of this bit into UART_MODEM_CLK_STOP (bit 9 of CLKM_REG [FFFF:2F00]).

Bit 8 **UART_IRDA_WAKEUP**. An ARM interrupt received will force a write of the contents of this bit into UART_IRDA_CLK_STOP (bit 8 of CLKM_REG [FFFF:2F00]).

Bit 7 **WATCHDOG_WAKEUP**. An ARM interrupt received will force a write of the contents of this bit into WATCHDOG_CLK_STOP (bit 7 of CLKM_REG [FFFF:2F00]).

Bit 6 **TIMER1_WAKEUP**. An ARM interrupt received will force a write of the contents of this bit into TIMER1_CLK_STOP (bit 6 of CLKM_REG [FFFF:2F00]).

Bit 5 **TIMER2_WAKEUP**. An ARM interrupt received will force a write of the contents of this bit into TIMER2_CLK_STOP (bit 5 of CLKM_REG [FFFF:2F00]).

Bit 4 **EIM_WAKEUP** (VC5471). An ARM interrupt received will force a write of the contents of this bit into EIM_CLK_STOP (bit 4 of CLKM_REG [FFFF:2F00]).

Reserved (VC5470).

Bit 3 **IRQ_WAKEUP**. An ARM interrupt received will force a write of the contents of this bit into IRQ_CLK_STOP (bit 3 of CLKM_REG [FFFF:2F00]).

Bit 2 **ARMIO_WAKEUP**. An ARM interrupt received will force a write of the contents of this bit into ARMIO_CLK_STOP (bit 2 of CLKM_REG [FFFF:2F00]).

Bit 1 **SDRAM_WAKEUP**. An ARM interrupt received will force a write of the contents of this bit into SDRAM_CLK_STOP (bit 1 of CLKM_REG [FFFF:2F00]).

Bit 0 **AUDIO_WAKEUP**. An ARM interrupt received will force a write of the contents of this bit into AUDIO_CLK_STOP (bit 0 of CLKM_REG [FFFF:2F00]).

5.4.3 Reset Register

Rules for proper reset:

- ❑ Block clocks must be operational as defined in CLKM_REG (FFFF:2F00) and WAKEUP_REG (FFFF:2F08).
- ❑ The reset is clock synchronous and must be acitve for at least one clock cycle and is recommended to be active for eight clock cycles.

Figure 5–6. Reset Register (RESET_REG)

Address (hex): Base = 0xFFFF:2F00, Offset = 0x0018

31–11	10	9	8
Reserved	SPI_ RESET	I2C_ RESET	UART_MODEM_ RESET
	RW–1	RW–1	RW–1

7	6	5	4	3	2	1	0
UART_IRDA_ RESET	WATCHDOG_ RESET	TIMER1_ RESET	TIMER2_ RESET	EIM_ RESET (VC5471) Reserved (VC5470)	IRQ_ RESET	ARMIO_ RESET	SDRAM_ RESET
RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–11 **Reserved**. Always return 0.

Bit 10 **SPI_RESET**. Reset for the Serial Port Interface logic including registers defined as type SPI.

0 Block reset active
1 Block operational

Bit 9 **I2C_RESET**. Reset for the I^2C logic including registers defined as type I2C.

0 Block reset active
1 Block operational

Bit 8 **UART_MODEM_RESET**. Reset for the UART Modem logic including registers defined as type UART Modem.

0 Block reset active

1 Block operational

Bit 7 **UART_IRDA_RESET**. Reset for the UART IrDA logic including registers defined as type UART IRDA.

0 Block reset active

1 Block operational

Bit 6 **WATCHDOG_RESET**. Reset for the Timer 0 logic (configured as a watchdog timer) including registers defined as type TIMER0.

0 Block reset active

1 Block operational

Bit 5 **TIMER1_RESET**. Reset for the Timer 1 logic including registers defined as type TIMER1.

0 Block reset active

1 Block operational

Bit 4 **TIMER2_RESET**. Reset for the Timer 2 logic including registers defined as type TIMER2.

0 Block reset active

1 Block operational

Bit 3 **EIM_RESET** (VC5471). Reset for the Ethernet Interface logic including registers defined as type EIM and EIM ENET0.

0 Block reset active

1 Block operational

Reserved (VC5470).

Bit 2 **IRQ_RESET**. Reset for the Interrupt logic including registers defined as type INTH.

0 Block reset active

1 Block operational

Bit 1 **ARMIO_RESET**. Reset for the ARM I/O logic including registers defined as type GPIO and KBGPIO.

0 Block reset active

1 Block operational

Bit 0 **SDRAM_RESET**. Reset for the SDRAM logic including registers defined as type SDRAMIF.

0 Block reset active

1 Block operational

The ARMSS clock module is also responsible for the generation and control of the AUDIO_CLK frequency, which is also generated from the external reference clock (REFCLK).

5.4.4 Audio Rate Register

Figure 5–7. Audio Rate Register (AUDIO_CLK)

Address (hex): Base = 0xFFFF:2F00, Offset = 0x000C

31–12	11–0
Reserved	AUDIO_CLK_CMP
	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–12 **Reserved**.

Bits 11–0 **AUDIO_CLK_CMP**. Provides the comparison value for AUDIO_CLK generation. The clock reference is the input clock REFCLK.

The AUDIO_CLK register provides a way to set the AUDIO output frequency by setting the comparison value that toggles the output. The clock that is used is always the input clock (REFCLK) of the VC547x device. The output waveform of the Audio Clock has a duty cycle of 50% provided the input clock, REFCLK, has a duty cycle of 50%.

The formula to compute the output frequency is

freq_audio_clk = (ref_clk_freq / (audio_clk_cmp + 1) x 2) where

freq_audio_clk is the Audio Clock frequency to be generated,

ref_clk_freq is the Input Clock (REFCLK) frequency, and

audio_clk_cmp is a divisor value (between 0 and 4095), from the AUDIO_CLK register.

When not in use, the Audio Clock can be disabled by clearing the appropriate bit values found in the CLKM_REG and WAKEUP_REG registers.

5.4.5 Watchdog Status Register

The clock module also hosts a watchdog status register, WATCHDOG_STATUS, which is used to identify the module that caused a reset. The watchdog timer or the ARMSS can reset the VC547x device. The WATCHDOG_STATUS register is an ARM memory-mapped register that keeps track of the module that caused the last reset until it is read. When the register content is read to learn the reset initiator (ARMSS or watchdog), the WATCHDOG_STATUS register is cleared automatically. The reset value for this register during power-up time is zero, which indicates that the reset was caused by the ARMSS.

Figure 5–8. Watchdog Status Register (WATCHDOG_STATUS)

Address (hex): Base = 0xFFFF:2F00, Offset = 0x0014

31–1	0
Reserved	WATCHDOG_STATUS
	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–1 **Reserved**.

Bit 0 **WATCHDOG_STATUS**.

0 no reset from watchdog

1 reset from watchdog

5.4.6 Low-Power Mode Register

The LOW_POWER_REG register is used to control the additional clock mode operation of the ARMSS. In the Low-Power mode, the clock generated is a scaled version of the external input clock (REFCLK) where the frequency of the generated clock is extremely low. The highest frequency clock generated in this mode is always REFCLK frequency / 512 . The duty cycle of this generated clock can be modified by using a value between 512 and 1023. With a value of 512, the duty cycle is 1/512. With a value of 1023, it is 50 percent.

LOW_POWER_REG is used to select between the normal (PLL) mode, divide (DIV) mode, and Low-Power mode. The normal mode is defined as the mode where we use the output of the subsystem PLL module (applies for both the ARMSS and DSPSS) as input to the clock module. Divide-by-Two is the mode where the input clock to the clock module is the input clock REFCLK divided by two. The Low-Power mode is defined as the mode where the input of the clock module is the low-power clock, which is also derived from the input clock (REFCLK) scaled by the value stored in the ARM memory-mapped register (LOW_POWER_VALUE_REG). Note that the DSPSS only functions in the Normal/PLL mode and in the DIV mode. Figure 5–9 illustrates the LOW_POWER_REG register.

Figure 5–9. Low-Power Mode Register (LOW_POWER_REG)

Address (hex): Base = 0xFFFF:2F00, Offset = 0x001C

31–2	1	0
Reserved	SWITCH_TO_DIV_BY_2	SWITCH_TO_LOW_POWER
	RW–1	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–2 **Reserved**.

Bit 1 **SWITCH_TO_DIV_BY_2**.

0 normal mode
1 use Divide-by-Two mode

Bit 0 **SWITCH_TO_LOW_POWER**.

0 normal mode
1 switch to low-power mode

Recall that there is another register—CLKMD for DSP PLL, and PLL_REG for the ARMSS—that controls the mode of operations (between Normal/PLL mode and DIV mode) and their voltage controlled oscillators. The Low-Power mode applies to the ARMSS. Care must be taken when changing from one mode to another since control overlap exists between the two ARMSS clock module control registers.

In order to change mode from Normal/PLL mode to DIV mode within the ARMSS the corresponding programming of fields is done in the PLL_REG register not in the LOW_POWER_REG register. It is **not supported** to switch from normal mode to DIV mode using the LOW_POWER_REG.

Upon reset, the default mode is the Divide-by-Two mode. This mode is merely provided for production test purposes and should not be used in the normal mode of operations. Standard operation is to go from Divide-by-Two mode to normal mode after setting the PLL in its operational mode (by programming the respective subsystem PLL control registers). If the application needs to go in a Divide-by-Two mode, it has to use the PLL Divide-by-Two mode.

By setting 1 to the `SWITCH_TO_LOW_POWER` field (bit 0) in the LOW_POWER_REG register, the ARMSS clock module will switch its current clock source mode to the Low-Power clock mode. It is perfectly allowed to go from normal mode to Low-Power mode and from Low-Power mode to normal mode via this field. However, it is **not supported** to switch from Low-Power mode to Divide-by-Two mode using the LOW_POWER_REG register.

5.4.7 Low-Power Register Value Register

As explained above, the Low-Power clock mode is a scaled frequency of the input reference clock (REFCLK) with the REFCLK frequency divided by the content of the memory-mapped register, LOW_POWER_REG_VALUE.

Figure 5–10. Low-Power Register Value Register (LOW_POWER_REG_VALUE)

Address (hex): Base = 0xFFFF:2F00, Offset = 0x0020

31–10	9–0
Reserved	LOW_POWER_CNT
	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–10 **Reserved**.

Bits 9–0 **LOW_POWER_CNT**. Low-power counter value.

The LOW_POWER_REG_VALUE is used to define the divider value that generates the low-power clock for the Low-Power mode. It is not allowed to program a value less than the binary value 1000000000. This means that the minimum frequency division factor is 512. In the event a value less than 512 is put into the LOW_POWER_REG_VALUE, the low-power clock is stopped or disabled and there is no clock source for the ARMSS. The maximum division factor is 1023. The division factor is used to obtain the clock frequency in the Low-Power mode.

The formula used is **freq_low_power = freq_refclk / low_power_cnt** where

- freq_low_power is the clock frequency in Low-Power mode,
- freq_refclk is the input reference clock (REFCLK), and
- low_power_cnt is the division factor set to obtain the desired low-power frequency.

The duty cycle of this clock is not kept the same as the input clock REFCLK.

5.5 Phase-Locked Loop (PLL)

Both subsystems (ARMSS and DSPSS) make use of independent PLLs to generate their clock source from a common external clock source applied to the VC547x device via REFCLK pin. The PLL_REG register of the ARMSS and the CLKMD register of the DSPSS together provide a way to control and generate independent final clock sources to be used by the subsystems. Note that the ARMSS has an additional Low-Power clock mode of operation which is controlled by another additional register, LOW_POWER_REG.

The PLL in the ARM subsystem provides an interface between the ARM processor and the PLL voltage controlled oscillator (PLL-VCO). This control is done using the PLL_REG register.

5.5.1 PLL_REG Register (ARMSS)

PLL_REG is one of the clock control registers within the ARMSS located in the ARM_PLL memory space.

Figure 5–11. PLL Clock Control Register (PLL_REG) – ARMSS

Address (hex): Base = 0xFFFF:3200, Offset = 0x0000

31–16	15–12	11
Reserved	PLLMUL	PLLDIV
	RW–0	RW–0

10–3	2	1	0
PLLCOUNT	PLLON_OFF	PLLNDIV	STATUS
RW–1	RW–0	RW–0	R–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**.

Bits 15–12 **PLLMUL**. Defines the PLL factor k.

Bit 11 **PLLDIV**. In conjunction with pllndiv and k, defines the PLL dividing factor m.

Bits 10–3 **PLLCOUNT.** A counter that starts decrementing from its preset value as soon as PLLNDIV is set to 1.

Bit 2 **PLLON_OFF**. In conjunction with PLLNDIV, enables the PLL analog part (VCO).

Bit 1 **PLLNDIV**. Sets the PLL/DIV mode.

0 Divider (DIV) mode

1 PLL mode

Bit 0 **STATUS.** (Read-only).

0 DIV mode: the output clock is the input clock multiplied by the PLL

1 PLL mode: the output clock is the input clock multiplied by the PLL multiplication ratio

Note: Writing to PLLCOUNT, PLLON/OFF, PLLDIV, and PLLMUL fields is possible only when the STATUS bit is set to zero (DIV mode).

PLLMUL[3:0] in conjunction with PLLDIV and PLLNDIV defines the frequency multiplier, *k*. As you will observe in the following paragraph, *k* is the final computed value that will be used as the multiplying value in order to get the final output clock.

PLLDIV, in conjunction with PLLMUL[3:0] and PLLNDIV, defines the PLL frequency dividing factor. When PLLNDIV is set to zero (i.e., clock operation is in divide mode), the setting of PLLDIV does not play a role in calculating the value of *k*. When PLLNDIV is set to one (i.e., clock operation is in PLL mode), the value of PLLDIV affects the multiplier value, *k*, causing it to be either an integer value or a non-integer value, depending on values programmed in the fields of PLLMUL[3:0] and PLLDIV. Table 5–5 shows the conditions for the above statements to be met as well as the restriction imposed on the values to be programmed in the above fields.

Table 5–5. Conditions Affecting PLL Frequency Dividing Factor

PLLNDIV	PLLDIV	PLLMUL	Multiplier “k”	Note on “k”
0 (Div Mode)	X (don’t care)	0–14	0.5	0.5
0 (Div Mode)	X (don’t care)	15	0.25	0.25
1 (PLL Mode)	0	0 – 14	PLLMUL + 1	Int Values (1 – 15)
1 (PLL Mode)	0	15	1	Single Int Value: 1 (Bypass Mode)
1 (PLL Mode)	1	0 or Even	(PLLMUL + 1) / 2	Non-Int values in increment of 1.0. [0.5 – 7.5]
1 (PLL Mode)	1	Odd	PLLMUL / 4	Non-Int values in increment of 0.50 [0.25 – 3.75]

PLLCOUNT[7:0] specifies the number of input clock cycles—REFCLK—(in increments of 16 cycles) for the PLL lock timer to count before the PLL begins clocking the processor after the PLL is started. This programmable lock timer provides a convenient method of automatically delaying clocking of the device by the PLL until lock is achieved.

The PLL lock timer is a counter loaded from the PLLCOUNT[7:0] field. The lock time is activated when the clock generator operating mode is switched from DIV mode to PLL mode. The timer can be preset to any value from 0 to 255, and its input clock is REFCLK divided by 16. The resulting lock-up delay can therefore be set from 0 to 255 x 16 REFCLK cycles. The PLL requires a maximum of 30 μs for lock-up time (for input frequencies between 5 MHz and 140 MHz) and appropriate values that generate this amount or greater (which is a function of the input clock frequency REFCLK) should be programmed in the PLLCOUNT [7:0]. During this lock up period, the clock generator continues to operate in DIV mode. When the PLL lock time has decremented to 0, the respective PLL will be the source of the final output clock that is used by the particular subsystem.

PLLON/OFF enables or disables the voltage controlled oscillator (VCO). The VCO is required to run while the clock generator is operating in PLL mode; this is not a requirement while the clock generator is operating in DIV mode.

Table 5–6. VCO Operating States

PLLON/OFF	PLLNDIV	VCO
0	0	Off
0	1	On
1	0	On
1	1	On

PLLNDIV determines whether the clock generator works in PLL mode or in DIV mode, thus defining the frequency multiplier *k* in conjunction with PLLMUL and PLLDIV.

PLLNDIV = 0: Divider (DIV) mode

PLLNDIV = 1: PLL mode

PLLSTATUS indicates the mode that the clock generator is operating in

STATUS = 0 : Divider (DIV) mode

STATUS = 1: PLL mode

5.5.2 CLKMD Clock Control Register (DSPSS)

The DSPSS clock control register, CLKMD, has an identical use as the ARMSS clock control register, PLL_REG. The only difference between the two is that the register size for the DSPSS is 32 bits wide (of which the high 16 are reserved).

The following register description is for a DSP subsystem register and should not be confused with an ARM subsystem register. Please refer to Table 2–1 for a complete list of DSP subsystem registers.

Figure 5–12. CLKMD Clock Control Register (CLKMD) – DSPSS

DSP Address (hex) = 0058

31–16	15–12	11
Reserved	PLLMUL	PLLDIV
	RW–0	RW–0

10–3	2	1	0
PLLCOUNT	PLLON_OFF	PLLNDIV	STATUS
RW–1	RW–0	RW–0	R–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**.

Bits 15–12 **PLLMUL**. Defines the PLL factor k.

Bit 11 **PLLDIV**. In conjunction with pllndiv and k, defines the PLL dividing factor m.

Bits 10–3 **PLLCOUNT.** A counter that starts decrementing from its preset value as soon as pllindiv is set to 1.

Bit 2 **PLLON_OFF**. In conjunction with pllindiv, enables the PLL analog part (VCO).

Bit 1 **PLLNDIV**. Sets the PLLDIV mode.

0 Divider (DIV) mode

1 PLL mode

Bit 0 **STATUS.** (Read-only).

0 DIV mode: the output clock is the input clock multiplied by the PLL

1 PLL mode: the output clock is the input clock multiplied by the PLL multiplication ratio

Chapter 6

Timer Module

This chapter provides a description of the three timers implemented on the TMS320VC547x device, discusses the watchdog function, shows the timer registers, and gives an overview of programming the timers.

The timers discussed in this chapter are associated with the ARM™ microcontroller unit (MCU) of the dual-core (MCU + DSP) VC547x device.

Topic **Page**

6.1 Timer Module Introduction

The TMS320VC547x implements three 16-bit timers configurable in Auto-Reload or One-Shot mode with *on-the-fly* read capability. The timers are on-chip down counters that generate interrupts to the ARM CPU each time the counter decrements to zero.

The first timer (TIMER0) is configured by default as a watchdog for the micro-controller unit (MCU). If this functionality is not required, a specific sequence must be written into a dedicated register in order to configure the watchdog as a general-purpose timer.

The two other timers (TIMER1 and TIMER2) are general-purpose timers.

As noted in Table 6–1, not all of the Timer module registers have the same base address.

Table 6–1. Timer Module Registers

Name	Base Address (hex)	Offset Address (hex)	Description
CNTL_TIMER0	0xFFFF:2A00	0x0000	TIMER0 Control Register
READ_TIM0	0xFFFF:2A00	0x0004	TIMER0 Current Value Register
CNTL_TIMER1	0xFFFF:2B00	0x0000	TIMER1 Control Register
READ_TIM1	0xFFFF:2B00	0x0004	TIMER1 Current Value Register
CNTL_TIMER2	0xFFFF:2C00	0x0000	TIMER2 Control Register
READ_TIM2	0xFFFF:2C00	0x0004	TIMER2 Current Value Register

6.2 TIMER0

By default, TIMER0 is configured as a watchdog timer. The watchdog is designed to detect user programs stuck in infinite loops resulting in loss of program control, such as *runaway* programs.

On power up, the LOAD_TIM field of the CNTL_TIMER0 register is set to the maximum value (0xFFFF) and the Watchdog mode is enabled but the watchdog is not started. The timer is started by setting the ST bit of CNTL_TIMER0 register to one.

Once the watchdog is started, the user's program must write periodically into the LOAD_TIM field of the CNTL_TIMER0 register before the counter underflows, in order to reload the timer with a new value. Note that in order to avoid undefined results, the PTV, AR, and LOAD_TIM fields (of the CNTL_TIMER register) must not be programmed when the timer is running and it is therefore mandatory that the start bit (ST) be at zero while programming the PTV, AR, and LOAD_TIM bit fields in order to have correct behavior.

Since it is impossible to write a new value if the ST bit is one, the following programming sequence must be followed in order to prevent the watchdog timer from going into an underflow condition:

1) Write the sequence 0xF5 followed by 0xA0 into the WDS field of CNTL_TIMER0 to disable the watchdog mode.

2) Write 0 to the ST bit of CNTL_TIMER0 to stop the timer.

3) Write a new value in the LOAD_TIM field of CNTL_TIMER0 (this new value can be different or it can be the same as the previous value in LOAD_TIM).

4) Write a 1 to the TM bit of CNTL_TIMER0 to switch back to the watchdog mode.

5) Write a 1 to the ST bit of CNTL_TIMER0 to restart the timer.

This programming sequence makes it more difficult to disable the watchdog timer; however, there is no protection in the clock module that would prevent the software application from stopping the watchdog timer's clock. A counter-underflow resets the ARM core and all modules controlled by it, including TIMER0.

In the Watchdog mode, the value of the prescaler field (PTV of CNTL_TIMER register) is fixed at seven. Thus, the time from writing a new value to counter underflow is comprised between:

$256 \times Tclk$ to $16,777,216 \times Tclk$. ($Tint = Tclk \times (LOAD\_TIM + 1) \times 2^{(PTV+1)}$)

For a clock frequency of 928 kHz, the timer interrupt period is 276 µs < t < 18 s.

6.2.1 Disabling the Watchdog Function

Timer0 may be configured as a general-purpose timer by writing a predefined sequence (0xF5 followed by 0xA0) in the WDS field of the CNTL_TIMER0 register.

Receiving 0xF5 initializes a sequence decoder. Once in this state if the next write is different from the 0xA0, the state machine resets the ARM core.

TIMER0 functions exactly like TIMER1 or 2 when it is configured as a general-purpose timer.

6.2.2 Re-Enabling the Watchdog Function

It is possible to come back to the default mode (Watchdog timer) by writing a one in the TM bit of the CNTL_TIMER0 register. In this case, the value loaded into the LOAD_TIM register is set to the maximum value (0xFFFF) as on power-up.

6.2.3 Timer0 Control Register

Figure 6–1. Timer0 Control Register (CNTL_TIMER0)

Address (hex): Base = FFFF:2A00, Offset = 0x0000

31–30	29–22	21	20–5	4	3	2–0
Reserved	WDS	TM	LOAD_TIM	AR	ST	PTV
	RW–0	RW–1	W–1	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–30 **Reserved**.

Bits 29–22 **WDS**. Watchdog disable.
Writing a predefined sequence (0xF5 followed by 0xA0) in this field disables the watchdog functionality.
After having received 0xF5, if the second write access is different from 0xA0, the ARM core is reset

Bit 21 **TM**. Timer mode.

Write access:

0 Writing a 0 in this bit has no effect

1 Switch back TIMER mode to WATCHDOG

Read access:

Status of TIMER mode

0 Timer is used as a general-purpose timer

1 Timer is used as a watchdog timer

Bits 20–5 **LOAD_TIM.** Load timer value.

Bit 4 **AR**. Auto-Reload timer.

0 One-Shot timer

1 Auto-Reload timer

Bit 3 **ST**. Start timer bit. If *ar* = 0, this bit is automatically reset by internal logic when timer is equal to zero.

0 Stop timer

1 Start timer

Bits 2–0 **PTV.** Prescale clock timer value.

000	2
001	4
010	8
011	16
100	32
101	64
110	128
111	256

The Prescaler, along with the load timer value, is used to set up the timer interrupts period. See section 6.3.1, *Timer Interrupt Period*, for more information.

The timer is started by setting the ST bit to one. When the timer is configured as a general-purpose timer, it is stopped by resetting the ST bit to zero. However, TIMER0 cannot be stopped while in the Watchdog mode; ie., writing a zero to bit ST has no effect if the TM bit is set to one. To stop a Watchdog timer, the timer must first be switched to be a general-purpose timer (writing the predefined sequence 0xF5 followed by 0xA0 into the WDS field), then a zero must be written to the ST bit.

If the auto-reload (AR) bit is disabled, the timer decrements from the loaded value to zero and then stops. In the other case (AR = 1), the timer continues by loading a new value from the LOAD_TIM field into the READ_TIM0 register.

6.2.4 Timer0 Current Value Register

Figure 6–2. Timer0 Current Value Register (READ_TIM0)

Address (hex): Base = 0xFFFF:2A00, Offset = 0x0004

31–16	15–0
Reserved	TIMER VALUE
	R–1

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**.

Bits 15–0 **TIMER VALUE**. Current value of timer

The READ_TIM0 register shows the current reading of the timer.

All three timers have on-the-fly read capability.

6.3 TIMER1 and TIMER2

TIMER1 and TIMER2 are both general-purpose timers. General-purpose timers are started by setting a dedicated bit to one (*ST* in *CNTL_TIMER* register) and stopped by resetting this bit.

If the auto-reload bit is disabled (*AR* in *CNTL_TIMER* register), the timer decrements from the loaded value to zero and then stops. In the other case (*AR*=1), the timer continues.

A new value (from the LOAD_TIM field of the CNTL_TIMER register) is loaded into the timer when it passes through zero or when it starts.

When the timer is stopped, the content of the decrementer is not affected so it is possible to read the value of the timer after it stops.

A programmable clock divider reduces the frequency of the clock used by the timer.

An interrupt is produced when the corresponding timer is equal to zero.

6.3.1 Timer Interrupt Period

The timer interrupt period is defined by:

- ❏ The value of the prescaler field (*PTV* of *CNTL_TIMER* register)
- ❏ The value of the load timer field (*LOAD_TIM* of CNTL_TIMER register)

The timer interrupt period is as follows:

PTV	Frequency Divisor
0	2
1	4
2	8
3	16
4	32
5	64
6	128
7	256

$$Tint = Tclk \times (LOAD\_TIM + 1) \times 2^{(PTV+1)}$$

6.3.2 TIMER1 and TIMER2 Control Registers

TIMER1 and TIMER2 registers are exactly the same. For this reason, only one register is shown in Figure 6–3

Figure 6–3. Timer1,2 Control Registers (CNTL_TIMER1,2)

CNTL_TIMER1 – Address (hex): Base = 0xFFFF:2B00, Offset = 0x0000
CNTL_TIMER2 –Address (hex): Base = 0xFFFF:2C00, Offset = 0x0000

31–21	20–5	4	3	2–0
Reserved	LOAD_TIM	AR	ST	PTV
	W–1	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–21 **Reserved**.

Bits 20–5 **LOAD_TIM**. Load timer value.

Bit 4 **AR**. Auto-reload timer.

0	One-Shot timer
1	Auto-Reload timer

Bit 3 **ST**. Start timer bit. If *ar* = 0, this bit is automatically reset by internal logic when timer is equal to zero.

0	Stop timer
1	Start timer

Bits 2–0 **PTV.** Prescale clock timer value.

000	2
001	4
010	8
011	16
100	32
101	64
110	128
111	256

The Prescaler, along with the load timer value, is used to set up the timer interrupts period. See section 6.3.1, *Timer Interrupt Period*, for more information.

General-purpose timers are started by setting the ST bit to one; they are stopped by resetting the ST bit to zero.

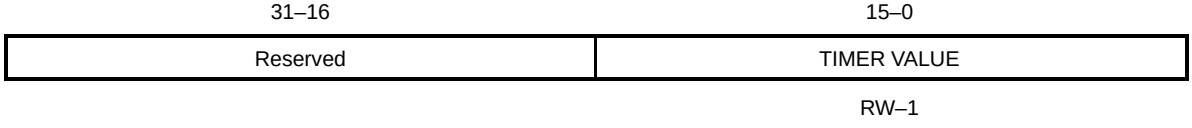
If the auto-reload (AR) bit is disabled, the timer decrements from the loaded value to zero and then stops. In the other case (*AR* = 1), the timer continues by loading a new value from the LOAD_TIM field into the corresponding READ_TIM register.

6.3.3 TIMER1 and TIMER2 Current Value Registers

READ_TIM1 and READ_TIM2 registers are exactly the same. For this reason, only one register is shown in Figure 6–4.

Figure 6–4. Timer1,2 Current Value Registers (READ_TIM1,2)

READ_TIM1 – Address (hex): Base = 0xFFFF:2B00, Offset = 0x0004
READ_TIM2 – Address (hex): Base = 0xFFFF:2C00, Offset = 0x0004



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–30 **Reserved**.

Bits 15–0 **TIMER VALUE**. Current value of timer.

The READ_TIM register shows the current reading of the timer.

All three timers have on-the-fly read capability.

6.4 Programming the Timers

In order to avoid undefined results, the PTV, AR, and LOAD_TIM fields (of CNTL_TIMER register) must not be programmed when the timer is running.

It is mandatory that the start bit (ST) be at zero while programming the LOAD_TIM field in order to have correct behavior.

6.5 Read Timer Operations

The VC547x bus system allows a read of all registers while the timer is running.

Chapter 7

General-Purpose I/O Module (GPIO)

This chapter provides a functional description of the general-purpose I/O module (GPIO) and shows all GPIO and KBGPIO registers.

GPIO is associated with the ARM™ microcontroller unit (MCU) of the dual-core (MCU + DSP) TMS320VC547x device.

Topic **Page**

7.1 Functional Description

This module provides 36 general-purpose input/outputs (I/Os) configurable in read or write mode by internal registers. The 36 general-purpose I/Os are split into 2 groups: GPIO(19:0) and KBGPIO(15:0). KBGPIOs (15:8) have on-chip pullup resistors connected to their input/output pins. KBGPIOs (15:0) can be used as normal GPIO pins or be configured for connection of 8x8 keyboard matrix as shown in Section 7.2.3, *Keyboard Connection*, on page 7-17.

7.1.1 General-Purpose I/O (GPIO)

GPIOs are programmed by the micro controller using two separate chip selects. One is for the GPIO (`cs_gpio_i`) and one is for KBGPIO (`cs_kbd_i`).

For the VC547x device, the following mapping is done:

- `cs_gpio_i` mapped to address 0xFFFF–2800 to 0xFFFF–28FF
- `cs_kb_i` mapped to address 0xFFFF–2900 to 0xFFFF–29FF

The two groups of GPIOs basically share the same functionality except the way interrupts are made available to the interrupt handler. Interrupts can be generated for rising, falling, or state changes, as well as being disabled.

Individual interrupts are available for GPIO0, GPIO1, GPIO2, and GPIO3. Grouped interrupts are available for GPIO (19:4), KBGPIO (15:8), and KBGPIO (7:0).

Some of the GPIOs are shared with other signals. Each GPIO is associated with six configuration/status bits described in Table 7–1 and Table 7–2.

The configuration/status bits are accessible through 12 memory-mapped registers listed in Section 7.2.

Table 7–1. GPIO Control/Status Bits

Bit Name	Description
io	I/O bit: Writeable when I/O is configured as an output (cio = 0) Reads value on I/O pin when I/O is configured as an input (cio = 1)
cio	Configured I/O 0: output 1: input (default)
gpio_irqA	GPIO interrupt configuration. See Table 7–2
gpio_irqB	GPIO interrupt configuration. See Table 7–2
ddio	Delta detect bit: If GPIO configured as output (cio = 0) always read as 0 If GPIO configured as input (cio = 1), reads 1 if io has changed since ddio was last cleared
gpio_en	Selects signal for muxed GPIOs 0: other I/O signal (default) 1: gpio Non-shared GPIOs are always available at the I/O pin independently of the value of gpio_en

Table 7–2. GPIO_IRQ Bit Definitions

gpio_irqB	gpio_irqA	Function
0	0	Disable IRQ
0	1	An IRQ is generated on the rising edge
1	0	An IRQ is generated on the falling edge
1	1	An IRQ is generated on the state change

7.2 GPIO/KBGPIO Registers

There are 12 memory-mapped GPIO/KBGPIO registers.

GPIO Registers

Base address (hex): FFFF:2800

Register width: 32 bits

Table 7–3. GPIO Registers

Register	Description	Offset Address
GPIO_IO	GPIO Input/Output Register	00h
GPIO_CIO	GPIO Configuration Register	04h
GPIO_IRQA	GPIO Interrupt Request Register A	08h
GPIO_IRQB	GPIO Interrupt Request Register B	0Ch
GPIO_DDIO	GPIO Delta Detect Register	10h
GPIO_EN	GPIO Mux Select Register	14h

KBGPIO Registers

Base address (hex): FFFF:2900

Register width: 32 bits

Table 7–4. KBGPIO Registers

Register	Description	Offset Address
KBGPIO_IO	KBGPIO Keyboard Input/Output Register	00h
KBGPIO_CIO	KBGPIO Configuration Register	04h
KBGPIO_IRQA	KBGPIO Interrupt Request Register A	08h
KBGPIO_IRQB	KBGPIO Interrupt Request Register B	0Ch
KBGPIO_DDIO	KBGPIO Delta Detect Register	10h
KBGPIO_EN	KBGPIO Mux Select Register	14h

7.2.1 GPIO Registers

Figure 7–1. GPIO_IO Register

Address (hex): Base = 0xFFFF:2800, Offset = 0x0000

31–20	19	18	17	16
Reserved	GPIO_IO_19	GPIO_IO_18	GPIO_IO_17	GPIO_IO_16
RW–0	RW–0	RW–0	RW–0	RW–0

15	14	13	12	11	10	9	8
GPIO_IO_15	GPIO_IO_14	GPIO_IO_13	GPIO_IO_12	GPIO_IO_11	GPIO_IO_10	GPIO_IO_9	GPIO_IO_8
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

7	6	5	4	3	2	1	0
GPIO_IO_7	GPIO_IO_6	GPIO_IO_5	GPIO_IO_4	GPIO_IO_3	GPIO_IO_2	GPIO_IO_1	GPIO_IO_0
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–20 **Reserved**. These bits are reserved (read as zeros).

Bit 19 **GPIO_IO19**. Writeable when I/O is configured as output (gpio_cio19 is 0). Reads value on GPIO19 pad when I/O is configured as input (gpio_cio19 is 1).

Bits 18–0 **GPIO_IO18:0**. Same as gpio_io19.

Figure 7–2. GPIO_CIO Register

Address (hex): Base = 0xFFFF:2800, Offset = 0x0004

31–20	19	18	17	16
Reserved	GPIO_CIO_19	GPIO_CIO_18	GPIO_CIO_17	GPIO_CIO_16
RW–0	RW–1	RW–1	RW–1	RW–1

15	14	13	12	11	10	9	8
GPIO_CIO_15	GPIO_CIO_14	GPIO_CIO_13	GPIO_CIO_12	GPIO_CIO_11	GPIO_CIO_10	GPIO_CIO_9	GPIO_CIO_8
RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1

7	6	5	4	3	2	1	0
GPIO_CIO_7	GPIO_CIO_6	GPIO_CIO_5	GPIO_CIO_4	GPIO_CIO_3	GPIO_CIO_2	GPIO_CIO_1	GPIO_CIO_0
RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–20 **Reserved**. These bits are reserved (read as zeros).

Bit 19 **GPIO_CIO19**.

0 GPIO19 configured as output
1 GPIO19 configured as input (default)

Bits 18–0 **GPIO_CIO18:0**. Same as gpio_cio19.

Figure 7–3. GPIO_IRQA Register

Address (hex): Base = 0xFFFF:2800, Offset = 0x0008

31–20	19	18	17	16
Reserved	GPIO_IRQA_19	GPIO_IRQA_18	GPIO_IRQA_17	GPIO_IRQA_16
RW–0	RW–0	RW–0	RW–0	RW–0

15	14	13	12	11	10	9	8
GPIO_IRQA_15	GPIO_IRQA_14	GPIO_IRQA_13	GPIO_IRQA_12	GPIO_IRQA_11	GPIO_IRQA_10	GPIO_IRQA_9	GPIO_IRQA_8
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

7	6	5	4	3	2	1	0
GPIO_IRQA_7	GPIO_IRQA_6	GPIO_IRQA_5	GPIO_IRQA_4	GPIO_IRQA_3	GPIO_IRQA_2	GPIO_IRQA_1	GPIO_IRQA_0
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

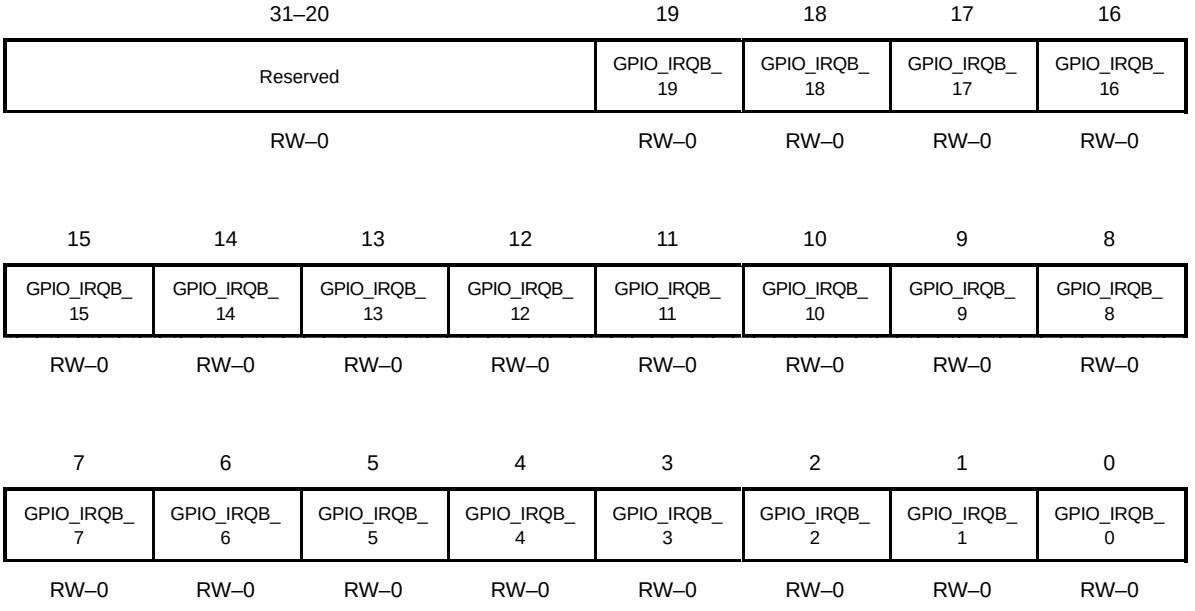
Bits 31–20 **Reserved**. These bits are reserved (read as zeros).

Bit 19 **GPIO_IRQA19**. In conjunction with gpio_irqB19, determines the behavior when GPIO19 configured as input IRQ.

Bits 18–0 **GPIO_IRQA18:0**. Same as gpio_irqA19.

Figure 7–4. GPIO_IRQB Register

Address (hex): Base = 0xFFFF:2800, Offset = 0x000C



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–20 **Reserved**. These bits are reserved (read as zeros).

Bit 19 **GPIO_IRQB19**. In conjunction with gpio_irqA19, determines the behavior when GPIO19 configured as input IRQ.

Bits 18–0 **GPIO_IRQB18:0**. Same as gpio_irqB19.

Interpretation of the IRQA and IRQB values is done according to Table 7–5.

Table 7–5. IRQA/IRQB Value Interpretations

gpio_irqB	gpio_irqA	Function
0	0	Disable IRQ
0	1	IRQ generated on rising edge
1	0	IRQ generated on falling edge
1	1	IRQ generated on state change

Figure 7–5. GPIO_DDIO – Delta Detect Register

Address (hex): Base = 0xFFFF:2800, Offset = 0x0010

31–20	19	18	17	16
Reserved	GPIO_DDIO_19	GPIO_DDIO_18	GPIO_DDIO_17	GPIO_DDIO_16
RW–0	RW–0	RW–0	RW–0	RW–0

15	14	13	12	11	10	9	8
GPIO_DDIO_15	GPIO_DDIO_14	GPIO_DDIO_13	GPIO_DDIO_12	GPIO_DDIO_11	GPIO_DDIO_10	GPIO_DDIO_9	GPIO_DDIO_8
RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1

7	6	5	4	3	2	1	0
GPIO_DDIO_7	GPIO_DDIO_6	GPIO_DDIO_5	GPIO_DDIO_4	GPIO_DDIO_3	GPIO_DDIO_2	GPIO_DDIO_1	GPIO_DDIO_0
RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–20 **Reserved**. These bits are reserved (read as zeros).

Bit 19 **GPIO_DDIO19**. Delta detect bit:
If gpio_cio19 = 0, always read as 0.
If gpio_cio19 = 1, reads 1 if gpio19 has changed since this bit was last cleared.
Write a 1 to clear gpio_ddio19.

Bits 18–0 **GPIO_DDIO18:0**. Same as gpio_ddio19.

GPIO_EN – GPIO Enable

This bit selects if a GPIO will be muxed with other I/Os. By default, the other functionality is selected. This bit controls the output multiplexers on the VC547x device. For GPIOs that are not shared, this bit has no meaning.

Figure 7–6. GPIO_EN Register

Address (hex): Base = 0xFFFF:2800, Offset = 0x0014

31–20	19	18	17	16
Reserved	GPIO_EN_19	GPIO_EN_18	GPIO_EN_17	GPIO_EN_16
RW–0	RW–1	RW–1	RW–1	RW–1

15	14	13	12	11	10	9	8
GPIO_EN_15	GPIO_EN_14	GPIO_EN_13	GPIO_EN_12	GPIO_EN_11	GPIO_EN_10	GPIO_EN_9	GPIO_EN_8
RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1

7	6	5	4	3	2	1	0
GPIO_EN_7	GPIO_EN_6	GPIO_EN_5	GPIO_EN_4	GPIO_EN_3	GPIO_EN_2	GPIO_EN_1	GPIO_EN_0
RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–20 **Reserved**. These bits are reserved (read as zeros).

Bit 19 **GPIO_EN19**. Selects signal for muxed GPIOs.

0 other I/O signal
1 GPIO

Bits 18–0 **GPIO_EN18:0**. Same as gpio_en19

Non-shared GPIOs are always available independent of the value of gpio_en.

7.2.2 KBGPIO Registers

Figure 7–7. KBGPIO_IO Register

Address (hex): Base = 0xFFFF:2900, Offset = 0x0000

31–16
Reserved
RW–0

15	14	13	12	11	10	9	8
KBGPIO_IO15	KBGPIO_IO14	KBGPIO_IO13	KBGPIO_IO12	KBGPIO_IO11	KBGPIO_IO10	KBGPIO_IO9	KBGPIO_IO8
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

7	6	5	4	3	2	1	0
KBGPIO_IO7	KBGPIO_IO6	KBGPIO_IO5	KBGPIO_IO4	KBGPIO_IO3	KBGPIO_IO2	KBGPIO_IO1	KBGPIO_IO0
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**. These bits are reserved (read as zeros).

Bit 15 **KBGPIO_IO15**. Writeable if I/O is configured as output (kbgpio_cio is 0). Reads value on KBGPIO15 pad in I/O is configured as input (kbgpio_cio is 1).

Bits 14–0 **KBGPIO_IO14:0**. Same as kbgpio_io15.

Figure 7–8. KBGPIO_CIO Register

Address (hex): Base = 0xFFFF:2900, Offset = 0x0004

31–16
Reserved
RW–0

15	14	13	12	11	10	9	8
KBGPIO_ CIO15	KBGPIO_ IO14	KBGPIO_ IO13	KBGPIO_ IO12	KBGPIO_ IO11	KBGPIO_ IO10	KBGPIO_ IO9	KBGPIO_ IO8
RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1

7	6	5	4	3	2	1	0
KBGPIO_ IO7	KBGPIO_ IO6	KBGPIO_ IO5	KBGPIO_ IO4	KBGPIO_ IO3	KBGPIO_ IO2	KBGPIO_ IO1	KBGPIO_ IO0
RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**. These bits are reserved (read as zeros).

Bit 15 **KBGPIO_CIO15**.

0 KBGPIO15 configured as output
1 KBGPIO15 configured as input (default)

Bits 14–0 **KBGPIO_CIO14:0**. Same as kbgpio_cio15.

Figure 7–9. KBGPIO_IRQA Register

Address (hex): Base = 0xFFFF:2900, Offset = 0x0008

31–16
Reserved
RW–0

15	14	13	12	11	10	9	8
KBGPIO_IRQA 15	KBGPIO_IRQA 14	KBGPIO_IRQA 13	KBGPIO_IRQA 12	KBGPIO_IRQA 11	KBGPIO_IRQA 10	KBGPIO_IRQA 9	KBGPIO_IRQA 8
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

7	6	5	4	3	2	1	0
KBGPIO_IRQA 7	KBGPIO_IRQA 6	KBGPIO_IRQA 5	KBGPIO_IRQA 4	KBGPIO_IRQA 3	KBGPIO_IRQA 2	KBGPIO_IRQA 1	KBGPIO_IRQA 0
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**. These bits are reserved (read as zeros).

Bit 15 **KBGPIO_IRQA15**. In conjunction with kbgpio_irqB15, determines the behavior when KBGPIO15 configured as input IRQ.

Bits 14–0 **KBGPIO_IRQA14:0**. Same as kbgpio_irqA15.

Figure 7–10. KBGPIO_IRQB Register

Address (hex): Base = 0xFFFF:2900, Offset = 0x000C

31–16
Reserved
RW–0

15	14	13	12	11	10	9	8
KBGPIO_IRQB 15	KBGPIO_IRQB 14	KBGPIO_IRQB 13	KBGPIO_IRQB 12	KBGPIO_IRQB 11	KBGPIO_IRQB 10	KBGPIO_IRQB 9	KBGPIO_IRQB 8
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

7	6	5	4	3	2	1	0
KBGPIO_IRQB 7	KBGPIO_IRQB 6	KBGPIO_IRQB 5	KBGPIO_IRQB 4	KBGPIO_IRQB 3	KBGPIO_IRQB 2	KBGPIO_IRQB 1	KBGPIO_IRQB 0
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**. These bits are reserved (read as zeros).

Bit 15 **KBGPIO_IRQB15**. In conjunction with kbgpio_irqA15, determines the behavior when KBGPIO15 configured as input IRQ.

Bits 14–0 **KBGPIO_IRQB14:0**. Same as kbgpio_irqB15.

Interpretation of the IRQA and IRQB values is made according to Table 7–6.

Table 7–6. KBGPIO_IRQA/IRQB Value Interpretations

kbgpio_irqB	kbgpio_irqA	Function
0	0	Disable IRQ
0	1	IRQ generated on rising edge
1	0	IRQ generated on falling edge
1	1	IRQ generated on state change

Figure 7–11. KBGPIO_DDIO – Delta Detect Register

Address (hex): Base = 0xFFFF:2900, Offset = 0x0010

31–16
Reserved
RW–0

15	14	13	12	11	10	9	8
KBGPIO_DDIO 15	KBGPIO_DDIO 14	KBGPIO_DDIO 13	KBGPIO_DDIO 12	KBGPIO_DDIO 11	KBGPIO_DDIO 10	KBGPIO_DDIO 9	KBGPIO_DDIO 8
RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1

7	6	5	4	3	2	1	0
KBGPIO_DDIO 7	KBGPIO_DDIO l6	KBGPIO_DDIO 5	KBGPIO_DDIO 4	KBGPIO_DDIO 3	KBGPIO_DDIO 2	KBGPIO_DDIO 1	KBGPIO_DDIO 0
RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**. These bits are reserved (read as zeros).

Bit 15 **KBGPIO_DDIO15**. Delta detect bit:
If kbgpio_cio15 = 1, always read as 0.
If kbgpio_cio15 = 0, reads 1 if io15 has changed since this bit was last cleared.
Write a 1 to clear gpio_ddio19.

Bits 14–0 **KBGPIO_DDIO14:0**. Same as kbgpio_ddio15.

KBGPIO_EN – KBGPIO Enable

This bit selects if a KBGPIO will be muxed with other I/Os. By default, the other I/O functionality is selected. This bit controls the output multiplexers on the VC547x device. For KBGPIOs that are not shared, this bit has no meaning.

Figure 7–12. KBGPIO_EN Register

Address (hex): Base = 0xFFFF:2900, Offset = 0x0014

31–16
Reserved
RW–0

15	14	13	12	11	10	9	8
KBGPIO_EN15	KBGPIO_EN14	KBGPIO_EN13	KBGPIO_EN12	KBGPIO_EN11	KBGPIO_EN10	KBGPIO_EN9	KBGPIO_EN8
RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1

7	6	5	4	3	2	1	0
KBGPIO_EN7	KBGPIO_ENI6	KBGPIO_EN5	KBGPIO_EN4	KBGPIO_EN3	KBGPIO_EN2	KBGPIO_EN1	KBGPIO_EN0
RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1	RW–1

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**. These bits are reserved (read as zeros).

Bit 15 **KBGPIO_EN15**. Selects register for muxed GPIOs.

0 other I/O signal
1 KBGPIO

Bits 14–0 **KBGPIO_EN14:0**. Same as kbgpio_en15.

Non-shared GPIOs are always available independent of the value of kbgpio_en.

7.2.3 Keyboard Connection

The keyboard can be connected to the chip using:

- ❑ KBGPIO(15:8) pins configured as input (*cio*=1) for row lines
- ❑ KBGPIO(7:0) pins configured as output (*cio*=0) pins for column lines with the *io* bit set to 0

If more keys are required, extra columns can be added using some of the spare GPIO pins.

If a key button of the keyboard matrix is pressed, the corresponding row and column lines are shorted together.

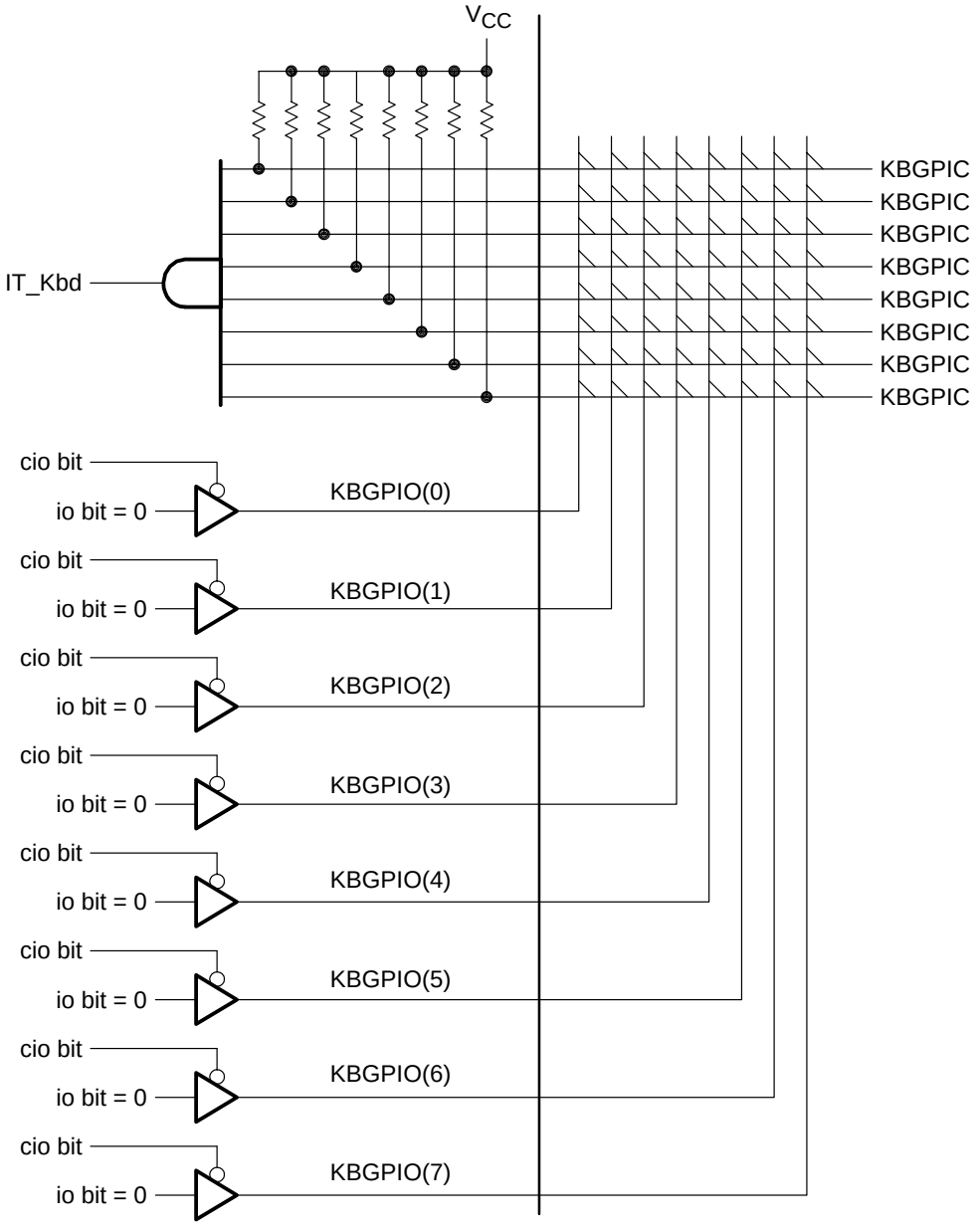
All input pins KBGPIO(15:8) are pulled up to VCC by the internal PU resistors and all output pins KBGPIO(7:0) are driving a low level (since *io* should have been set to 0). Any action on a button will generate an interrupt to the ARM. Upon interrupt reception the ARM SW shall write the *cio* bits of the column lines KBGPIO(7:0) with the sequence described below, while reading the *io* bits of the row lines KBGPIO(15:8).

This sequence allows the detection of simultaneously pressed keys.

Table 7–7. Keyboard Scanning Sequence

	IDLE cio bit	**Keyboard Scanning cio bit**									**IDLE cio bit**
KBGPIO(0)	0	1	0	1	1	1	1	1	1	1	0
KBGPIO(1)	0	1	1	0	1	1	1	1	1	1	0
KBGPIO(2)	0	1	1	1	0	1	1	1	1	1	0
KBGPIO(3)	0	1	1	1	1	0	1	1	1	1	0
KBGPIO(4)	0	1	1	1	1	1	0	1	1	1	0
KBGPIO(5)	0	1	1	1	1	1	1	0	1	1	0
KBGPIO(6)	0	1	1	1	1	1	1	1	0	1	0
KBGPIO(7)	0	1	1	1	1	1	1	1	1	0	0

Figure 7–13. Keyboard Connection



7.3 Input/Outputs of GPIO Module

Table 7–8. GPIO Module I/Os

Name	Function	Direction	Size
add_i	Address	IN	3
data_i	Data bus	IN	20
data_o	Data bus	OUT	32
nrw_i	Not Read Write	IN	1
cs_gpio_i	GPIO chip select	IN	1
cs_kb_i	KBGPIO chip select	IN	1
nreset_i	Reset timer module – Active at low level	IN	1
clk	GPIO and KBGPIO clock	IN	1
gpio_i	GPIO input	IN	20
gpio_o	GPIO output	OUT	20
gpio_en_o	GPIO enable output	OUT	20
gpio_io_enable_o	GPIO I/O output	OUT	20
nirq_gpio_0_o	GPIO bit 0 IRQ output	OUT	1
nirq_gpio_1_o	GPIO bit 1 IRQ output	OUT	1
nirq_gpio_2_o	GPIO bit 2 IRQ output	OUT	1
nirq_gpio_3_o	GPIO bit 3 IRQ output	OUT	1
nirq_gpio_19_4_o	OR of GPIO bit 19 to bit 4 IRQ output	OUT	1
gpiokb_i	KBGPIO input	IN	16
gpiokb_o	KBGPIO output	OUT	16
gpiokb_en_o	KBGPIO enable output	OUT	16
gpiokb_io_enable_o	KBGPIO i/o configuration output	OUT	16
nirq_gpiokb_15_8_o	OR of KBGPIO bit 15 to 8 IRQ output	OUT	1
nirq_gpiokb_7_0_o	OR of KBGPIO bit 7 to 0 IRQ output	OUT	1
test_mode	Test mode	IN	1
test_si	Test Scan Input	IN	1
test_so	Test Scan Output	OUT	1

Chapter 8

UART IRDA Module

This chapter explains the features of the UART IRDA module, shows the applicable registers, and discusses the serial infrared (SIR) mode and the universal asynchronous receiver/transmitter (UART) mode.

UART IRDA is associated with the ARM™ microcontroller unit (MCU) of the dual-core (MCU + DSP) VC547x device.

Topic **Page**

8.1 General Description

The UART IRDA module on the VC547x device is a universal asynchronous receiver/transmitter that can be used in two different operating modes: UART modem mode (UART mode) and IrDA serial infrared mode (SIR mode). This UART interface is compatible with 16C750-compliant devices. It includes the SIR protocol in order to be connected with an infrared transmitter to any external data peripherals with an IrDA-compliant data interface. The IR function can be disabled and the UART connected through a standard wired interface. This UART can be linked to an external PC for concurrent debugging.

In the UART modem mode, the UART IrDA module transmits characters sent to it by the ARM on the TX pin, and receives characters from the RX pin. In the SIR mode, transmissions are done by pulses generated by the UART and transformed into infrared pulses by a transceiver.

IrDA/SIR Background

IrDA is a standard defined by the IrDA consortium (**I**nfra**r**ed **D**ata **A**ssociation). It specifies a way to wirelessly transfer data via infrared radiation. The IrDA specifications include standards for both the physical devices and the protocols they use to communicate with each other. The IrDA standards have arisen from the need to connect various mobile devices together. (Primary use for IrDA is to link notebooks or various personal communicators; however, even video cameras are sometimes equipped with an IrDA interface.) IrDA devices communicate using infrared LEDs.

SIR/FIR Physical Layer Specifications

The serial IrDA (SIR) physical layer specification defines a short-range infrared asynchronous serial transmission mode with one start bit, eight data bits, and one stop bit. The maximum data rate is 115.2 Kbps (half-duplex). The primary benefit of this scheme is that existing serial hardware can be used very cheaply. This is one of the reasons for the widespread availability of IrDA. Compared to SIR, the Fast IrDA (FIR) physical layer specification defines short-range, low-power operation at 4 Mbps (half-duplex). All FIR devices are also required to support SIR operation.

The UART IrDA module on the VC547x device supports only the SIR protocol for IrDA communication.

8.2 Main Features

The UART IRDA module integrates two 64-word (9 and 11 bits) receive and transmit FIFOs and one 8-word (16 bits) status FIFO with programmable trigger levels. Hardware buffering allows higher transmission speed without data loss and without requiring frequent attention from the ARM. The baud rate is internally generated from a programmable divisor.

- UART mode:

  Under the UART mode, the module is configurable to send even, odd, or no parity, and 1, 1.5, or 2 stop bits. The word length for both TX and RX can be configured between five and eight bits. The receiver can detect break, idle or framing errors, FIFO overflow, and parity errors. The transmitter can detect FIFO underflow. All modem operations are controllable via a software interface.

- IrDA SIR mode:

  The protocol used is the serial infrared (SIR) protocol, defined as a standard (www.irda.org)

8.2.1 UART Mode Features

- Line break generation and detection
- Interrupt system control
- Baud rates up to 6.25M baud are supported
- Only software flow control

8.2.2 IrDA SIR Mode Features

- ❏ Frame format: addition of variable beginning-of-frame (xBOF) characters and end-of-frame (EOF) characters
- ❏ Uplink/downlink CRC generation/detection
- ❏ Asynchronous transparency (automatic insertion of break character)
- ❏ 8-character status FIFO available to monitor frame length and frame errors
- ❏ Variable frame length for RX and TX IrDA frame
- ❏ Sd_mode: output shutdown. When the UART/IrDA is in transmission, the pin is set to high; otherwise, the pin is maintained low. Software manages this functionality
- ❏ IrDA 1.0 SIR support, allows serial communication at baud rates from 2.4K baud to 115.2K baud.
- ❏ Pulse shaping and pulse recovering. Sending a single infrared pulse signals a zero. A one is signaled by not sending any pulse. The width of the pulse is programmable.
- ❏ The device operation, in IrDA 1.0 SIR mode, is similar to the operation in UART mode. The format of the serial data is similar to the UART data format. Each data word is sent with a zero value start bit followed by eight data bits, and ending with at least one stop bit with a binary value of one. The main differences are that the data transfer operations are normally performed in half-duplex, and the modem control and status signals are not used.

8.3 I/O Description

Table 8–1. UART_IRDA Signals

Signal	I/O	Function
CK16X_IRDA	I/O	16X serial transmission clock
TXIR_IRDA	I/O	Infrared transmit pulse
TX_IRDA	I/O	Transmit data
RXIR_IRDA	I/O	Infrared receive pulse
RX_IRDA	I/O	Receive data
SD_IRDA	I/O	IRDA transceiver shutdown mode

8.4 Register Mapping/Descriptions

8.4.1 UART IRDA Module Registers

Base address (hex): FFFF:0800

Register width: 32 bits

Table 8–2. UART IRDA Module Registers

Register	Description	Offset Address
UART_IRDA_RHR (UART Mode)	Receive Holding Register (UART Mode)	00h
UART_IRDA_RHR (SIR Mode)	Receive Holding Register (SIR Mode)	00h
UART_IRDA_THR	Transmit Holding Register	04h
UART_IRDA_FCR	FIFO Control Register	08h
UART_IRDA_SCR	Status Control Register	0Ch
UART_IRDA_LCR (UART Mode)	Line Control Register (UART Mode Only)	10h
UART_IRDA_LSR (UART Mode)	Line Status Register (UART Mode)	14h
UART_IRDA_LSR (SIR Mode)	Line Status Register (SIR Mode)	14h
UART_IRDA_SSR	Supplementary Status Register	18h
UART_IRDA_MCR	Modem Control Register	1Ch
UART_IRDA_MSR	Modem Status Register	20h
UART_IRDA_IER (UART Mode)	Interrupt Enable Register (UART Mode)	24h
UART_IRDA_IER (SIR Mode)	Interrupt Enable Register (SIR Mode)	24h
UART_IRDA_ISR (UART Mode)	Interrupt Status Register (UART Mode)	28h
UART_IRDA_ISR (SIR Mode)	Interrupt Status Register (SIR Mode)	28h

Table 8–2. UART IRDA Module Registers (Continued)

Register	Description	Offset Address
UART_IRDA_EFR	Enhanced Feature Register	2Ch
UART_IRDA_XON1	XON1 Character Register	30h
UART_IRDA_XON2	XON2 Character Register	34h
UART_IRDA_XOFF1	XOFF1 Character Register	38h
UART_IRDA_XOFF2	XOFF2 Character Register	3Ch
UART_IRDA_SPR	Scratch-Pad Register	40h
UART_IRDA_DIV_115K	Divisor for 115K-Baud Generation	44h
UART_IRDA_DIV_BIT_RATE	Divisor for Baud-Rate Generation	48h
UART_IRDA_TCR (UART Mode)	Transmission Control Register (UART Mode Only)	4Ch
UART_IRDA_TLR	Trigger Level Register	50h
UART_IRDA_MDR1	Mode Definition Register 1	54h
UART_IRDA_MDR2	Mode Definition Register 2	58h
UART_IRDA_TXFLL	Transmit Frame Length Register – LSB	5Ch
UART_IRDA_TXFLH	Transmit Frame Length Register – MSB	60h
UART_IRDA_RXFLL	Received Frame Length Register – LSB	64h
UART_IRDA_RXFLH	Received Frame Length Register – MSB	68h
UART_IRDA_SFLSR	Status FIFO Line Status Register	6Ch
UART_IRDA_SFREGL	Status FIFO Register – LSB	70h
UART_IRDA_SFREGH	Status FIFO Register – MSB	74h
UART_IRDA_BLR	Beginning-of-File-Length Register	78h
UART_IRDA_PULSE_WIDTH	Pulse Width Register	7Ch
UART_IRDA_ACREG	Auxiliary Control Register	80h
UART_IRDA_START_POINT	Start Point for IR Transmission	84h
UART_IRDA_WRPTR_URX	Write Pointer of RX FIFO	88h
UART_IRDA_RDPTR_URX	Read Pointer of RX FIFO	8Ch

Table 8–2. UART IRDA Module Registers (Continued)

Register	Description	Offset Address
UART_IRDA_WRPTR_UTX	Write Pointer of TX FIFO	90h
UART_IRDA_RDPTR_UTX	Read Pointer of TX FIFO	94h
UART_IRDA_WRPTR_STA	Write Pointer of Status FIFO	98h
UART_IRDA_RDPTR_STA	Read Pointer of Status FIFO	9Ch
UART_IRDA_RESUME	Resume Register	A0h

8.4.2 Special Access Registers

It is important to note here that some registers need special conditions to be accessed in write and/or read mode.

- ❏ UART_IRDA_MCR[7:5] and UART_IRDA_FCR[5:4] can only be written when UART_IRDA_EFR[4] = 1
- ❏ UART_IRDA_TCR and UART_IRDA_TLR can only be written when UART_IRDA_MCR[6] = 1
- ❏ UART_IRDA_WRPTR_URX, UART_IRDA_RDPTR_URX, UART_IRDA_WRPTR_UTX, UART_IRDA_RDPTR_UTX, UART_IRDA_WRPTR_STA, and UART_IRDA_RDPTR_STA can only be accessed when UART_IRDA_SCR[0] = 1

All the other registers can be accessed unconditionally.

8.4.3 Register Mapping

- ❏ Base address is 0xFFFF:0800
- ❏ Writing and reading in registers depends on the selected mode:
  - ■ UART mode (UART_IRDA_MDR1[2:0] = 000)
  - ■ IrDA mode (UART_IRDA_MDR1[2:0] = 001)

8.4.4 Receive Holding Register

This register holds the received word that is to be read by the MCU. Received data is stored in a 64-word FIFO. The first unread word is presented to the RHR and replaced by the second unread word after an RHR access. If the FIFO is disabled, the RHR register will contain the received data in the same way. If an overflow occurs, received data will not be written into the FIFO, and consequently, will not be read through RHR.

UART Mode

Bits 8, 9, and 10 of RHR indicate which kind of error has occurred on current read data.

Figure 8–1. Receive Holding Register (UART_IRDA_RHR) – UART Mode

Address (hex): Base = FFFF:0800, Offset = 0x0000

31–11	10	9	8
Reserved	RX_BI	RX_FE	RX_PE
R–0	R–0	R–0	R–0

7–0
RHR
R–U

Note: R = Read access; value following dash (–) = value after reset; U = Undefined

Bits 31–11 **Reserved**. Read as zeros.

Bit 10 **RX_BI.**

0 No break condition

1 A break was detected while receiving read data (i.e., RX_IRDA input signal was low for one character frame)

Bit 9 **RX_FE.**

0 No framing error in read data

1 A framing error occurred while receiving read data (i.e., received data did not have a valid stop bit)

Bit 8 **RX_PE.**

0 No parity error in read data

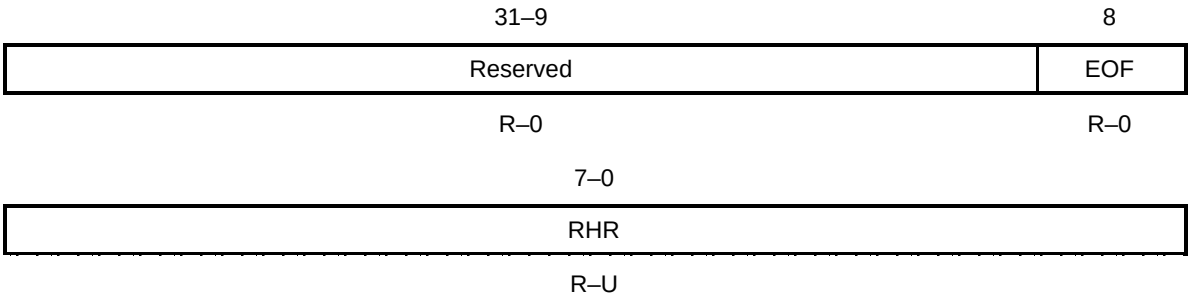
1 A parity error occurred while receiving read data

Bits 7–0 **RHR**. Receive holding register (contains the first unread byte of the 64-byte RX FIFO). If overflow occurs, data is not overwritten in FIFO.

SIR Mode

Figure 8–2. Receive Holding Register (UART_IRDA_RHR) – SIR Mode

Address (hex): Base = FFFF:0800, Offset = 0x0000



Note: R = Read access; value following dash (–) = value after reset; U = Undefined

Bits 31–9 **Reserved**. Read as zeros.

Bit 8 **EOF.** End-of-frame bit.

0 Not end

1 End of frame

Bits 7–0 **RHR**. Receive holding register (contains the first unread byte of the 64-byte RX FIFO). If overflow occurs, data is not overwritten in FIFO.

8.4.5 Transmit Holding Register

This register stores data to be sent. Once it is written by the CPU, data is transmitted to the transmitter FIFO and waits for its parallel-to-serial translation before being shifted onto the TX_IRDA output pin. If the FIFO is disabled, you should be sure that the TX FIFO is not full (SSR[0]) before writing to THR because you can overwrite data to send.

Figure 8–3. Transmit Holding Register (UART_IRDA_THR)

Address (hex): Base = FFFF:0800, Offset = 0x0004

31–8
Reserved
W–0

7–0
THR
W–U

Note: W = Write access; value following dash (–) = value after reset; U = Undefined

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **THR**. Transmit holding register (64-byte FIFO).

8.4.6 FIFO Control Register

Note that bits 4 and 5 can only be written when EFR[4] = 1. (UART_IRDA_EFR[4] enables/disables writing to UART_IRDA_FCR[5:4] and UART_IRDA_MCR[7:5].)

Figure 8–4. FIFO Control Register (UART_IRDA_FCR)

Address (hex): Base = FFFF:0800, Offset = 0x0008

31–8
Reserved
W–0

7–6	5–4	3	2	1	0
RX_FIFO_TRIG	TX_FIFO_TRIG	Reserved	RX_FIFO_CLEAR	TX_FIFO_CLEAR	FIFO_EN
W–0	W–0	W–0	W–0	W–0	W–0

Note: W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–6 **RX_FIFO_TRIG.** Sets the trigger level for the RX FIFO.

00	8 bytes
01	16 bytes
10	32 bytes
11	60 bytes

Bits 5–4 **TX_FIFO_TRIG.** Sets the trigger level for the TX FIFO if UART_IRDA_EFR(4) = 1.

00	8 bytes
01	16 bytes
10	32 bytes
11	56 bytes

Bit 3 **Reserved**. Read as zero.

Bit 2 **RX_FIFO_CLEAR.**

0	No change
1	Clears the RX FIFO and resets its counter logic to zero

Bit 1 **TX_FIFO_CLEAR.**

0	No change
1	Clears the TX FIFO and resets its counter logic to zero

Bit 0 **FIFO_EN.**

0	Disables the TX and RX FIFOs
1	Enables the TX and RX FIFOs

8.4.7 Status Control Register

Figure 8–5. Status Control Register (UART_IRDA_SCR)

Address (hex): Base = FFFF:0800, Offset = 0x000C

31–8	7	6
Reserved	FIFO_INIT_STATUS	FIFO_INIT
RW–0	R–0	W–0

5	4	3	2–1	0
Reserved	RX_CTS_WAKE_UP_ENABLE	TX_EMPTY_CTL_IT	Reserved	FIFO_PTR_ACCESS_EN
W–0	W–0	W–0	W–0	W–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bit 7 **FIFO_INIT_STATUS.**

0 Initialization of FIFOs is not finished if UART_IRDA_SCR(6) = 1

1 Initialization of FIFOs is finished. Clear on a read

Bit 6 **FIFO_INIT.**

0 FIFOs are not initialized

1 FIFOs are initialized to zero. This bit auto-clears

Bit 5 **Reserved**. Read as zero.

Bit 4 **RX_CTS_WAKE_UP_ENABLE.**

0 Disables the wake-up interrupt and clears UART_IRDA_SSR[1]

1 Waits for the falling edge of input pin RX_IRDA to generate an interrupt

Bit 3 **TX_EMPTY_CTL_IT.**

0 Normal mode for UART_IRDA_THR interrupt

1 The UART_IRDA_THR interrupt is generated when the TX FIFO and TX shift register are empty

Bits 2–1 **Reserved**. Read as zeros.

Bit 0 **FIFO_PTR_ACCESS_EN.**

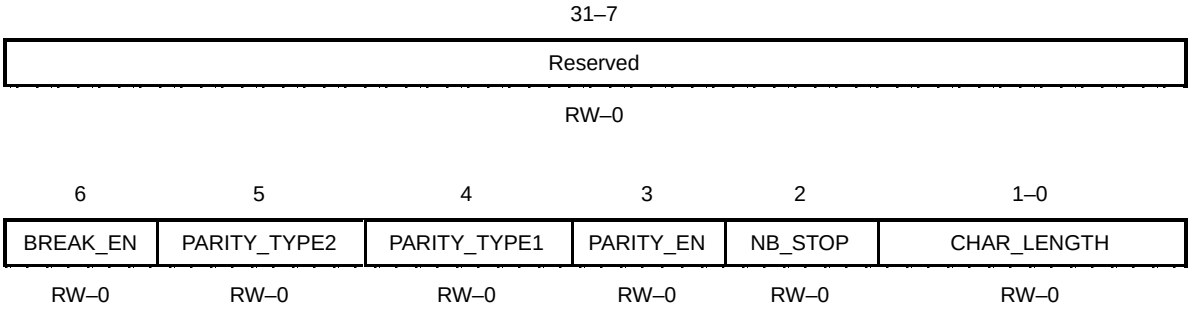
0 Disables FIFO's pointer access through registers

1 Enables FIFO's pointer access through UART_IRDA_WRPTR_URX, UART_IRDA_RDPTR_URX, UART_IRDA_WRPTR_UTX, UART_IRDA_RDPTR_UTX, UART_IRDA_WRPTR_STA, UART_IRDA_RDPTR_STA

8.4.8 Line Control Register (UART Mode Only)

Figure 8–6. Line Control Register (UART_IRDA_LCR) – UART Mode

Address (hex): Base = FFFF:0800, Offset = 0x0010



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–7 **Reserved**. Read as zeros.

Bit 6 **BREAK_EN.** Break Control Bit.

0 No break

1 Forces the transmitter output to go low to alert the communication terminal

Bit 5 **PARITY_TYPE2.** Selects the forced parity format if UART_IRDA_LCR(3) = 1

0 No forced parity

1 If UART_IRDA_LCR(4) = 0, the parity bit is forced to 1 in the TX and RX data. If UART_IRDA_LCR(4) = 1, the parity bit is forced to 0 in the TX and RX data

Bit 4 **PARITY_TYPE1.**

0 Odd parity is generated (if bit 3 = 1)

1 Even parity is generated (if bit 3 = 1)

Bit 3 **PARITY_EN.**

0 No parity

1 A parity bit is generated during transmission and the receiver checks for received parity

Bit 2 **NB_STOP.** Number of stop bits

0 1 stop bit (word length = 5, 6, 7, 8 bits)

1 1.5 stop bits (word length = 5)
2 stop bits (word length = 6, 7, 8)

Bits 1–0 **CHAR_LENGTH.** Word length for TX and RX

00	5 bits
01	6 bits
10	7 bits
11	8 bits

8.4.9 Line Status Register

UART Mode

Bit 7 indicates if there is an error in RX FIFO, which means that the UART received data with a framing error, parity error, or a break indication. This bit stays at one until no more errors remain in the FIFO, which means until all data with errors has been read.

Figure 8–7. Line Status Register (UART_IRDA_LSR) – UART Mode

Address (hex): Base = FFFF:0800, Offset = 0x0014

31–8
Reserved
R–0

7	6	5	4–2	1	0
RX_FIFO_STS	TX_SR_E	TX_FIFO_E	Reserved	RX_OE	RX_FIFO_E
R–0	R–1	R–1	R–0	R–0	R–0

Note: R = Read access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bit 7 **RX_FIFO_STS.**

0	Normal operation
1	At least one parity error, framing error, or breaking indication in the RX FIFO. Cleared when no more errors are present in the FIFO

Bit 6 **TX_SR_E.**

0	Transmitter hold and shift registers are not empty
1	Transmitter hold and shift registers are empty

Bit 5 **TX_FIFO_E.**

0 Transmitter hold register is not empty

1 Transmitter hold register is empty. The processor can load up to 64 bytes of data into THR if the TX FIFO is enabled.

Bits 4–2 **Reserved**. Read as zeros.

Bit 1 **RX_OE.**

0 No overrun error

1 Overrun error has occurred. Set when the character being held in the RX shift register is not transferred to the RX FIFO. This case can only occur when RX FIFO is full.

Bit 0 **RX_FIFO_E.**

0 No data in the RX FIFO

1 At least one data character in the RX FIFO

SIR Mode

Figure 8–8. Line Status Register (UART_IRDA_LSR) – SIR Mode

Address (hex): Base = FFFF:0800, Offset = 0x0014

31–8
Reserved
R–0

7	6	5	4	3	2	1	0
THR_EMPTY	STS_FIFO_ FULL	RX_LAST_ BYTE	FRAME_TOO_ LONG	ABORT	CRC	STS_FIFO_E	RX_FIFO_E
R–1	R–0	R–1	R–0	R–0	R–0	R–0	R–1

Note: R = Read access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bit 7 **THR_EMPTY.**

0 TX hold register is not empty

1 TX hold register is empty. The processor can load up to 64 bytes of data into THR if the TX FIFO is enabled.

Bit 6 **STS_FIFO_FULL.**

0 Status FIFO not full

1 Status FIFO full

Bit 5 **RX_LAST_BYTE.**

0 Did not receive the last byte of a frame from the FIFO

1 Received the last byte of a frame from the FIFO. This bit is set when the last byte of a frame is read. Cleared on a read of the UART_IRDA_LSR.

Bit 4 **FRAME_TOO_LONG.**

0 No frame_too_long error

1 Frame-too-long error at the top of the STATUS FIFO (next character to be read). This is set to 1 when a frame exceeding the maximum length set by UART_IRDA_RXFLL and UART_IRDA_RXFLH registers has been received. When the error is detected, current frame reception is terminated. Reception is stopped until the next START flag is detected.

Bit 3 **ABORT.**

0 No abort pattern error in frame

1 Abort pattern error is received

Bit 2 **CRC.**

0 No CRC error in frame

1 CRC error in the frame at the top of the STATUS FIFO (next character to be read).

Bit 1 **STS_FIFO_E.**

0 Status FIFO empty

1 Status FIFO not empty

Bit 0 **RX_FIFO_E.**

0 At least one data in the RX FIFO

1 No data in RX FIFO

8.4.10 Supplementary Status Register

Figure 8–9. Supplementary Status Register (UART_IRDA_SSR)

Address (hex): Base = FFFF:0800, Offset = 0x0018

31–8
Reserved
R–0

7–2	1	0
Reserved	RX_CTS_WAKE_UP_STS	TX_FIFO_FULL
R–0	R–0	R–0

Note: R = Read access; value following dash (–) = value after reset

Bits 31–2 **Reserved**. Read as zeros.

Bit 1 **RX_CTS_WAKE_UP_STS.** Enabled if UART_IRDA_SCR(4) = 1.

0 No falling edge event on RX_IRDA pin
1 A falling edge event occurred on RX_IRDA pin

Bit 0 **TX_FIFO_FULL.**

0 TX FIFO not full
1 TX FIFO full

8.4.11 Modem Control Register

Note that bits 5, 6, and 7 can be written only when UART_IRDA_EFR[4] = 1.

Figure 8–10. Modem Control Register (UART_IRDA_MCR)

Address (hex): Base = FFFF:0800, Offset = 0x001C

31–8
Reserved
RW–0

7	6	5	4	3–0
CLKSEL	TCR_TLR	XON_EN	MODE	Reserved
RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bit 7 **CLKSEL.**

0	No action
1	Divides clock input by four

Bit 6 **TCR_TLR.**

0	No action
1	Enables access to the UART_IRDA_TCR and UART_IRDA_TLR registers

Bit 5 **XON_EN.**

0	Disables *XON Any* function
1	Enables *XON Any* function

Bit 4 **MODE.**

0	Normal operating mode
1	Internal loopback mode. UART_IRDA_MCR[1:0] is looped into UART_IRDA_MSR[5:4]

Bits 3–0 **Reserved**. Read as zeros.

8.4.12 Modem Status Register

Figure 8–11. Modem Status Register (UART_IRDA_MSR)

Address (hex): Base = FFFF:0800, Offset = 0x0020

31–0
Reserved
R–0

Note: R = Read access; value following dash (–) = value after reset

Bits 31–0 **Reserved**. Read as zeros.

8.4.13 Interrupt Enable Register

UART Mode

The interrupt enable register is used to enable or disable any interrupt. There are five types of interrupts. You should be aware that the RHR interrupt enable is necessary in order to obtain a time-out interrupt (see the UART_IRDA_ISR register in section 8.4.14).

Figure 8–12. Interrupt Enable Register (UART_IRDA_IER) – UART Mode

Address (hex): Base = FFFF:0800, Offset = 0x0024

31–8
Reserved
RW–0

7–6	5	4	3	2	1	0
Reserved	XOFF_IT	Reserved	Reserved	LINE_STS_IT	THR_IT	RHR_IT
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–6 **Reserved**. Read as zeros.

Bit 5 **XOFF_IT.**

0 Disables the XOFF interrupt
1 Enables the XOFF interrupt

Bit 4 **Reserved**. Read as zero.

Bit 3 **Reserved**. Read as zero.

Bit 2 **LINE_STS_IT.**

0 Disables the receiver line status interrupt
1 Enables the receiver line status interrupt

Bit 1 **THR_IT.**

0 Disables the THR interrupt
1 Enables the THR interrupt

Bit 0 **RHR_IT.**

0 Disables the RHR interrupt
1 Enables the RHR interrupt

SIR Mode

Figure 8–13. Interrupt Enable Register (UART_IRDA_IER) – SIR Mode

Address (hex): Base = FFFF:0800, Offset = 0x0024

31–8							
Reserved							
RW–0							
7	6	5	4	3	2	1	0
EOF_IT	LINE_STS_IT	TX_UNDERRUN_IT	STS_FIFO_TRIG_IT	RX_OVERRUN_IT	LAST_RX_BYTE_IT	THR_IT	RHR_IT
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bit 7 **EOF_IT.**

0 Disables the received EOF interrupt
1 Enables the received EOF interrupt

Bit 6 **LINE_STS_IT.**

0 Disables the receiver line status interrupt
1 Enables the receiver line status interrupt

Bit 5 **TX_UNDERRUN_IT.**

0 Disables the TX underrun interrupt
1 Enables the TX underrun interrupt

Bit 4 **STS_FIFO_TRIG_IT.**

0 Disables the status FIFO trigger level interrupt
1 Enables the status FIFO trigger level interrupt

Bit 3 **RX_OVERRUN_IT.**

0 Disables the RX overrun interrupt

1 Enables the RX overrun interrupt

Bit 2 **LAST_RX_BYTE_IT.**

0 Disables the last-byte-in-RX-FIFO interrupt

1 Enables the last-byte-in-RX-FIFO interrupt

Bit 1 **THR_IT.**

0 Disables the THR interrupt

1 Enables the THR interrupt

Bit 0 **RHR_IT.**

0 Disables the RHR interrupt

1 Enables the RHR interrupt

8.4.14 Interrupt Status Register

UART Mode

All interrupts have a priority level. See Section 8.7.2, *Interrupts*, for more information on the management, priority, and clearing of interrupts.

Figure 8–14. Interrupt Status Register (UART_IRDA_ISR) – UART Mode

Address (hex): Base = FFFF:0800, Offset = 0x0028

31–8			
Reserved			
R–0			
7	6	5–1	0
FCR_MIRROR	FCR_MIRROR	IT_TYPE	IT_PENDING
R–0	R–0	R–0	R–1

Note: R = Read access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bit 7 **FCR_MIRROR.** Mirrors the content of FCR(0)

Bit 6 **FCR_MIRROR.** Mirrors the content of FCR(0)

Bits 5–1 **IT_TYPE.** Interrupts by priority

00011	Receiver line status error (priority level 1)
00110	RX time out (priority level 2)
00010	RHR interrupt (priority level 2)
00001	THR interrupt (priority level 3)
01000	XOFF/Special character interrupt (priority level 4)

Bit 0 **IT_PENDING.**

0	An interrupt (except the one defined by UART_IRDA_SCR(4)) is pending. $\overline{IRQ}$ is active.
1	No interrupt (except maybe the one defined by UART_IRDA_SCR(4)) is pending. $\overline{IRQ}$ is active.

SIR Mode

Figure 8–15. Interrupt Status Register (UART_IRDA_ISR) – SIR Mode

Address (hex): Base = FFFF:0800, Offset = 0x0028

31–8
Reserved
R–0

7	6	5	4	3	2	1	0
EOF_IT	LINE_STS_IT	TX_UE_IT	STS_FIFO_IT	RX_OE_IT	RX_FIFO_ LAST_BYTE_IT	THR_IT	RHR_IT
R–0	R–0	R–0	R–0	R–0	R–0	R–0	R–0

Note: R = Read access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bit 7 **EOF_IT.**

0	Receiver EOF interrupt inactive
1	Receiver EOF interrupt active

Bit 6 **LINE_STS_IT.**

0	Receiver line status interrupt inactive
1	Receiver line status interrupt active

Bit 5 **TX_UE_IT.**

0 TX underrun interrupt inactive
1 TX underrun interrupt active

Bit 4 **STS_FIFO_IT.**

0 Status FIFO trigger level interrupt inactive
1 Status FIFO trigger level interrupt active

Bit 3 **RX_OE_IT.**

0 RX overrun interrupt inactive
1 RX overrun interrupt active

Bit 2 **RX_FIFO_LAST_BYTE_IT.**

0 Last byte in RX FIFO interrupt inactive
1 Last byte in RX FIFO interrupt active

Bit 1 **THR_IT.**

0 THR interrupt inactive
1 THR interrupt active

Bit 0 **RHR_IT.**

0 RHR interrupt inactive
1 RHR interrupt active

8.4.15 Enhanced Feature Register

This register enables or disables enhanced features that concern flow control, except bit 4, which enables write operation onto MCR and FCR. You should note that XON1 and XON2 (and XOFF1 and XOFF2) must be different if the software flow control is used.

Figure 8–16. Enhanced Feature Register (UART_IRDA_EFR)

Address (hex): Base = FFFF:0800, Offset = 0x002C

31–8
Reserved
RW–0

7–6	5	4	3–0
Reserved	SPECIAL_CHAR_DETECT	ENHANCED_EN	SW_FLOW_CONTROL
RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W= Write access; value following dash (–) = value after reset

Bits 31–6 **Reserved**. Read as zeros.

Bit 5 **SPECIAL_CHAR_DETECT.**

0 Normal operating mode

1 Enables special character detection. Received character is compared to XOFF2. If a match occurs, the received character is transferred to RX FIFO and UART_IRDA_ISR[4] is set to 1.

Bit 4 **ENHANCED_EN.** Enhanced functions write enable.

0 Disables writing to UART_IRDA_FCR[5:4], UART_IRDA_MCR[7:5]

1 Enables writing to UART_IRDA_FCR[5:4], UART_IRDA_MCR[7:5]

Bits 3–0 **SW_FLOW_CONTROL.** Software flow control.

00XX No transmit flow control

01XX Transmit XON2, XOFF2

10XX Transmit XON1, XOFF1

11XX Transmit XON1, XON2, XOFF1, XOFF2

XX00 No receive flow control

XX01 Receiver compares XON2, XOFF2

XX10 Receiver compares XON1, XOFF1

XX11 Receiver compares XON1, XON2, XOFF1, XOFF2

Note: XON1/XON2 and XOFF1/XOFF2 must be set to different values if the software flow control is used

8.4.16 XON1 Character Register

Figure 8–17. XON1 Character Register (UART_IRDA_XON1)

Address (hex): Base = FFFF:0800, Offset = 0x0030

31–8
Reserved
RW–0
7–0
XON_WORD1
RW–0

Note: W= Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **XON_WORD1.** Used to store the 8-bit XON1 character.

8.4.17 XON2 Character Register

Figure 8–18. XON2 Character Register (UART_IRDA_XON2)

Address (hex): Base = FFFF:0800, Offset = 0x0034

31–8
Reserved
RW–0
7–0
XON_WORD2
RW–0

Note: W= Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **XON_WORD2.** Used to store the 8-bit XON2 character.

8.4.18 XOFF1 Character Register

Figure 8–19. XOFF1 Character Register (UART_IRDA_XOFF1)

Address (hex): Base = FFFF:0800, Offset = 0x0038

31–8
Reserved
RW–0
7–0
XOFF_WORD1
RW–0

Note: W= Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **XOFF_WORD1.** Used to store the 8-bit XOFF1 character.

8.4.19 XOFF2 Character Register

Figure 8–20. XOFF2 Character Register (UART_IRDA_XOFF2)

Address (hex): Base = FFFF:0800, Offset = 0x003C

31–8
Reserved
RW–0
7–0
XOFF_WORD2
RW–0

Note: W= Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

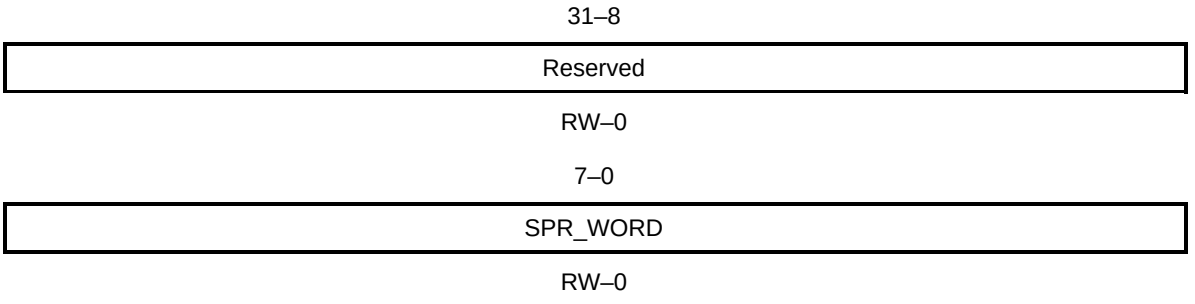
Bits 7–0 **XOFF_WORD2.** Used to store the 8-bit XOFF2 character.

8.4.20 Scratch Pad Register

This register has no control function. It is intended as a scratch pad to be used by the programmer for holding temporary data.

Figure 8–21. Scratch Pad Register (UART_IRDA_SPR)

Address (hex): Base = FFFF:0800, Offset = 0x0040



Note: R = Read access; W= Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **SPR_WORD.** Scratch pad register**.**

8.4.21 Divisor for 115K-Baud Generation Register

This 9-bit register is used to store the divisor needed to obtain a 115K-baud rate.

- UART mode

  The reset value (412) is the divisor needed at 47.5 MHz (47 500 000/115 200 = 412).

- SIR mode

  The same method can be followed, but most of the time it is necessary to calculate the register values of UART_IRDA_DIV_115K and UART_IRDA_BIT_RATE in order to get a precise pulse width (see section 8.6.4)

Figure 8–22. Divisor for 115K-Baud Generation Register (UART_IRDA_DIV_115K)

Address (hex): Base = FFFF:0800, Offset = 0x0044

31–9
Reserved
RW–0

8–0
DIV_115K
RW–110110010

Note: R = Read access; W= Write access; value following dash (–) = value after reset

Bits 31–9 **Reserved**. Read as zeros.

Bits 8–0 **DIV_115K.** div_115k = (fclk / 115200).
div_115k is the divisor needed to obtain the 115k-baud rate; it is the clock speed (in Hz) divided by 115200.

8.4.22 Divisor for Baud-Rate Generation Register

- UART mode

  This 7-bit register represents the needed divisor to obtain the right speed from 115K baud. For example, 0000010 will give 115K/2 = 57600 baud

- SIR mode

  This register must be adapted the same way as the UART_IRDA_DIV_115K register.

Figure 8–23. Divisor for Baud-Rate Generation Register (UART_IRDA_DIV_BIT_RATE)

Address (hex): Base = FFFF:0800, Offset = 0x0048

31–7
Reserved
RW–0

6–0
DIV_BITRATE
RW–0

Note: R = Read access; W= Write access; value following dash (–) = value after reset

Bits 31–7 **Reserved**. Read as zeros.

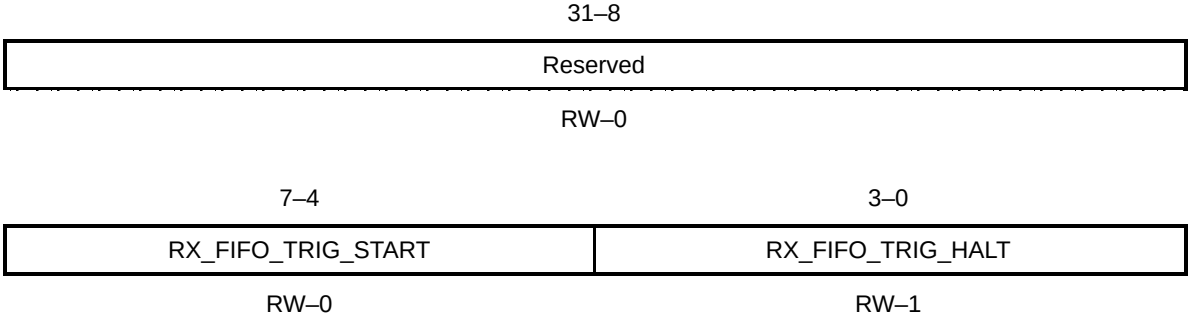
Bits 6–0 **DIV_BITRATE.** div_bit_rate = div_115k / bit rate.
div_bit_rate is the divisor needed to obtain the desired speed.

8.4.23 Transmission Control Register (UART Mode Only)

This register is used to store the receive FIFO threshold level to start and stop transmission during software flow control. This register can be written only if UART_IRDA_MCR[6] = 1.

Figure 8–24. Transmission Control Register (UART_IRDA_TCR) – UART Mode

Address (hex): Base = FFFF:0800, Offset = 0x004C



Note: R = Read accesss; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–4 **RX_FIFO_TRIG_START.** Receive FIFO trigger level to restore transmission.

0000	0 byte
0001	4 bytes
0010	8 bytes
...	
1111	60 bytes

Bits 3–0 **RX_FIFO_TRIG_HALT.** Receive FIFO trigger level to stop transmission.

0000	0 byte
0001	4 bytes
0010	8 bytes
...	
1111	60 bytes

8.4.24 Trigger Level Register

This register is used to store the programmable transmit and receive FIFO trigger levels used for IRQ generation. Trigger levels from 4 to 60 can be programmed with a granularity of 4.

Note that TLR can be written only if MCR[6] = 1.

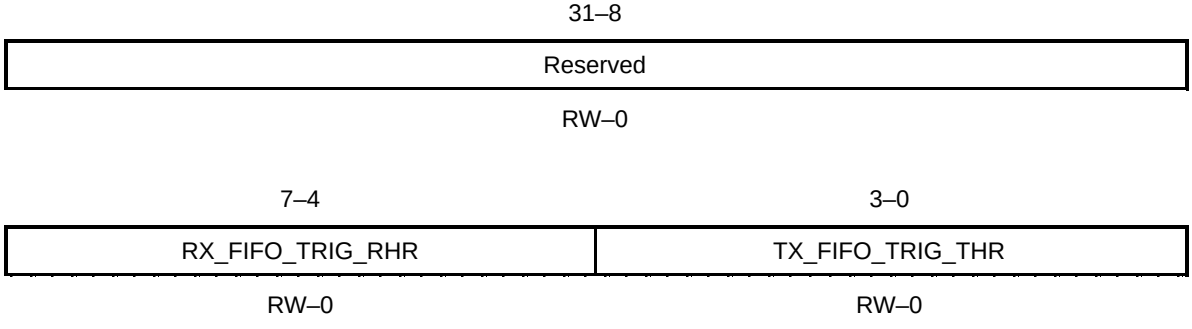
If TLR[7:4] = 0000, the programmable RX trigger levels are disabled and trigger RX levels in FCR[7:6] are enabled.

If TLR[3:0] = 0000, the programmable TX trigger levels are disabled and trigger TX levels in FCR[5:4] are enabled.

Note that for the TX FIFO, the TLR represents the number of empty spaces in the FIFO above which the THR interrupt will be activated. For example, if TLR[3:0] = 1111, and if there are four or less bytes in the transmit FIFO, the THR interrupt will be activated. If TLR[3:0] = 0001, and if there are 60 or less bytes in the transmit FIFO, the interrupt will be activated.

Figure 8–25. Trigger Level Register (UART_IRDA_TLR)

Address (hex): Base = FFFF:0800, Offset = 0x0050



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–4 **RX_FIFO_TRIG_RHR.** RX FIFO trigger level to generate RHR interrupt. It represents the number of empty spaces in the FIFO above which the RHR interrupt will be activated.

0000	Use UART_IRDA_FCR[7:6] trigger level
0001	4 bytes
0010	8 bytes
...	
1111	60 bytes

Bits 3–0 **TX_FIFO_TRIG_THR.** TX FIFO trigger level to generate THR interrupt. It presents the number of empty spaces in the FIFO above which the THR interrupt will be activated.

0000	Use UART_IRDA_FCR[5:4] trigger level
0001	4 bytes
0010	8 bytes
...	
1111	60 bytes

8.4.25 Mode Definition Register 1

Figure 8–26. Mode Definition Register 1 (UART_IRDA_MDR1)

Address (hex): Base = FFFF:0800, Offset = 0x0054

31–8
Reserved
RW–0

7–3	2–0
Reserved	MODE_SELECT
RW–0	RW–1

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–3 **Reserved**. Read as zeros.

Bits 2–0 **MODE_SELECT.** Mode select.

000	UART mode
001	SIR mode
111	Reset mode
	All other values are reserved

8.4.26 Mode Definition Register 2

Figure 8–27. Mode Definition Register 2 (UART_IRDA_MDR2)

Address (hex): Base = FFFF:0800, Offset = 0x0058

31–8
Reserved
RW–0

7–3	2–1	0
Reserved	STS_FIFO_TRIG	Reserved
RW–0	RW–0	RW–U

Note: R = Read access; W = Write access; value following dash (–) = value after reset; U = Undefined

Bits 31–3 **Reserved**. Read as zeros.

Bits 2–1 **STS_FIFO_TRIG.** Status FIFO threshold select.

00	1 character
01	4 characters
10	7 characters
11	8 characters

Bit 0 **Reserved.**

8.4.27 Transmit Frame Length Register (LSB)

Figure 8–28. Transmit Frame Length Register – LSB (UART_IRDA_TXFLL)

Address (hex): Base = FFFF:0800, Offset = 0x005C

31–8
Reserved
W–0
7–0
TXFLL
W–0

Note: W= Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **TXFLL.** LSB register used to specify the frame length for transmission.

8.4.28 Transmit Frame Length Register (MSB)

Figure 8–29. Transmit Frame Length Register – MSB (UART_IRDA_TXFLH)

Address (hex): Base = FFFF:0800, Offset = 0x0060

31–8
Reserved
W–0
7–0
TXFLH
W–0

Note: W= Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **TXFLH.** MSB register used to specify the frame length for transmission.

8.4.29 Receive Frame Length Register (LSB)

Figure 8–30. Receive Frame Length Register – LSB (UART_IRDA_RXFLL)

Address (hex): Base = FFFF:0800, Offset = 0x0064

31–8
Reserved
W–0
7–0
RXFLL
W–0

Note: W= Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **RXFLL.** LSB register used to specify the frame length in reception.

8.4.30 Receive Frame Length Register (MSB)

Figure 8–31. Receive Frame Length Register – MSB (UART_IRDA_RXFLH)

Address (hex): Base = FFFF:0800, Offset = 0x0068

31–8
Reserved
W–0
7–0
RXFLH
W–0

Note: W= Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **RXFLH.** MSB register used to specify the frame length in reception.

8.4.31 Status FIFO Line Status Register

Figure 8–32. Status FIFO Line Status Register (UART_IRDA_SFLSR)

Address (hex): Base = FFFF:0800, Offset = 0x006C

31–8
Reserved
R–0

7–5	4	3	2	1	0
Reserved	OE_ERROR	FRAME_LENGTH_ERROR	ABORT_DETECT	CRC	Reserved
R–0	R–0	R–0	R–0	R–0	R–U

Note: R = Read access; value following dash (–) = value after reset; U = Undefined

Bits 31–5 **Reserved**. Read as zeros.

Bit 4 **OE_ERROR.**

0 No error
1 Overrun error occurred in RX FIFO when frame at top of FIFO was received

Bit 3 **FRAME_LENGTH_ERROR.**

0 No error
1 Frame length error in frame at top of FIFO

Bit 2 **ABORT_DETECT.**

0 No error
1 Abort detected in frame at top of FIFO

Bit 1 **CRC.**

0 No error
1 CRC error in frame at top of FIFO

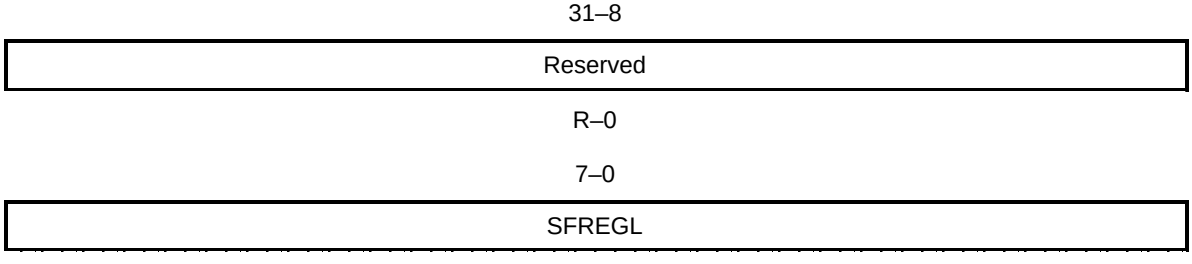
Bit 0 **Reserved.**

8.4.32 Status FIFO Register

The frame lengths of received frames are written into the status FIFO. This information can be read by reading the SFREGL and SFREGH registers (these registers do not physically exist). The least significant bits (LSBs) are read from the SFREGL register, and the most significant bits (MSBs) are read from SFREGH. Reading these registers does not alter the status FIFO read pointer. These registers should be read before the pointer is incremented by reading the SFLSR.

Figure 8–33. Status FIFO Register – LSB (UART_IRDA_SFREGL)

Address (hex): Base = FFFF:0800, Offset = 0x0070



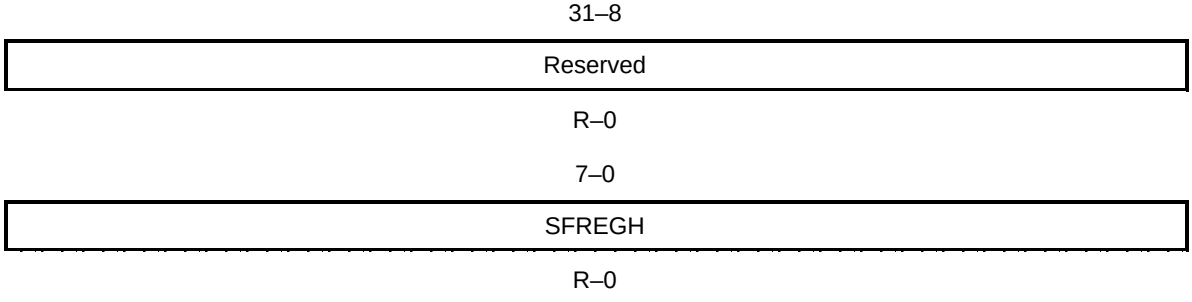
Note: R = Read access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **SFREGL**. LSB part of the frame length for the received frame.

Figure 8–34. Status FIFO Register – MSB (UART_IRDA_SFREGH)

Address (hex): Base = FFFF:0800, Offset = 0x0074



Note: R = Read access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **SFREGH.** MSB part of the frame length for the receive frame.

8.4.33 Beginning-of-File Length Register

Note that BLR(6) is used to select whether C0h or FFh start patterns are to be used when multiple start flags are required in SIR mode. If only one start flag is required, it will always be C0h. If n start flags are required, then either (n – 1) C0h or (n –1) FFh flags will be sent (if BLR(6) is 1 or 0, respectively), followed by a single C0h flag (immediately preceding the first data byte).

Figure 8–35. Beginning-of-File Length Register (UART_IRDA_BLR)

Address (hex): Base = FFFF:0800, Offset = 0x0078

31–8
Reserved
RW–0

7	6	5–0
STS_FIFO_RESET	BOF_TYPE	NB_XBOF
RW–0	RW–1	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bit 7 **STS_FIFO_RESET.** Status FIFO reset. This bit is self-clearing**.**

Bit 6 **BOF_TYPE.** BOF type. SIR flag select.

0	0xFF
1	0xC0

Bits 5–0 **NB_XBOF**. Number of xBOF to be transmitted at the beginning of an IRDA frame (0–63). The main purpose of the parameter is to provide a delay at the beginning of each frame for devices with long interrupt latency.

8.4.34 Pulse Width Register

Figure 8–36. Pulse Width Register (UART_IRDA_PULSE_WIDTH)

Address (hex): Base = FFFF:0800, Offset = 0x007C

31–8
Reserved
RW–0

7–0
PULSE_WIDTH
RW–0

Note: R = Read access; W= Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **PULSE_WIDTH.** Width of the pulse in number of ARM/div_115k clock cycles. Example: If ARM clock frequency/div_115k = 5 MHz, pulse_width = 8 generates a pulse width of 1.6 μs.

8.4.35 Auxiliary Control Register

Figure 8–37. Auxiliary Control Register (UART_IRDA_ACREG)

Address (hex): Base = FFFF:0800, Offset = 0x0080

31–7
Reserved
RW–0

6	5–3	2	1	0
SD_MODE	Reserved	SCTX_EN	ABORT_EN	EOT_EN
RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–7 **Reserved**. Read as zeros.

Bit 6 **SD_MODE.** Output pin to control transmission.

0 SD_IRDA pin is set to high

1 SD_IRDA pin is set to low

Bits 5–3 **Reserved**. Read as zeros.

Bit 2 **SCTX_EN.** When the CPU writes 1 to this bit, the TX state machine starts frame transmission. This bit is self_clearing.

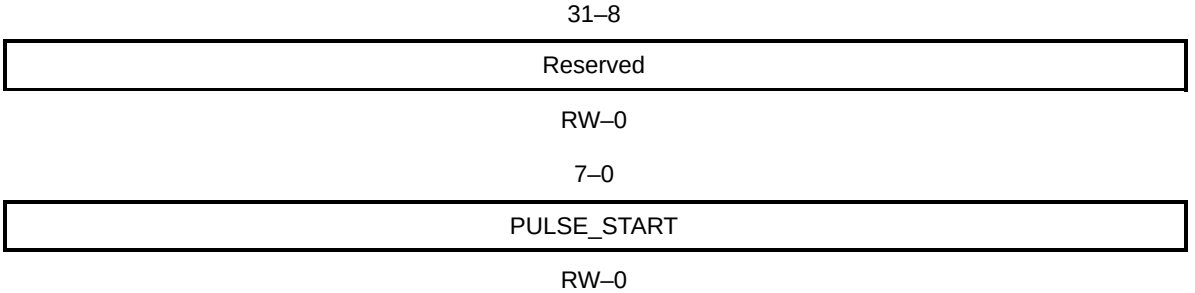
Bit 1 **ABORT_EN.** Frame abort.
The CPU can abort transmission of a frame by writing 1 to this bit. Neither the end flag nor the CRC bits are appended to the frame. The CPU should reset the TX FIFO before transmitting the next frame.

Bit 0 **EOT_EN.** End of transmission.
The CPU should write 1 to this bit just before writing the last byte to the FIFO in set_EOT bit frame closing method. This bit automatically gets cleared when the CPU writes to the THR.

8.4.36 Start Point for IR Transmission

Figure 8–38. Start Point for IR Transmission (UART_IRDA_START_POINT)

Address (hex): Base = FFFF:0800, Offset = 0x0084



Note: R = Read access; W= Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **PULSE_START.** Start point for IR transmission (in function of bit rate)**.**

8.4.37 Access to Read and Write Pointers

Access to read and write pointers is enabled by setting UART_IRDA_SCR[0] to 1.

Figure 8–39. Write Pointer of RX FIFO (UART_IRDA_WRPTR_URX)

Address (hex): Base = FFFF:0800, Offset = 0x0088

31–7
Reserved
RW–0

6–0
RX_WRITE_PTR
RW–0

Note: R = Read access; W= Write access; value following dash (–) = value after reset

Bits 31–7 **Reserved**. Read as zeros.

Bits 6–0 **RX_WRITE_PTR.** Write pointer of RX FIFO. Can be accessed only if UART_IRDA_SCR[0] = 1.

Figure 8–40. Read Pointer of RX FIFO (UART_IRDA_RDPTR_URX)

Address (hex): Base = FFFF:0800, Offset = 0x008C

31–7
Reserved
RW–0

6–0
RX_READ_PTR
RW–0

Note: R = Read access; W= Write access; value following dash (–) = value after reset

Bits 31–7 **Reserved**. Read as zeros.

Bits 6–0 **RX_READ_PTR.** Read pointer of RX FIFO. Can be accessed only if UART_IRDA_SCR[0] = 1.

Figure 8–41. Write Pointer of TX FIFO (UART_IRDA_WRPTR_UTX)

Address (hex): Base = FFFF:0800, Offset = 0x0090

31–7
Reserved
RW–0

6–0
TX_WRITE_PTR
RW–0

Note: R = Read access; W= Write access; value following dash (–) = value after reset

Bits 31–7 **Reserved**. Read as zeros.

Bits 6–0 **TX_WRITE_PTR.** Write pointer of TX FIFO. Can be accessed only if UART_IRDA_SCR[0] = 1.

Figure 8–42. Read Pointer of TX FIFO (UART_IRDA_RDPTR_UTX)

Address (hex): Base = FFFF:0800, Offset = 0x0094

31–7
Reserved
RW–0

6–0
TX_READ_PTR
RW–0

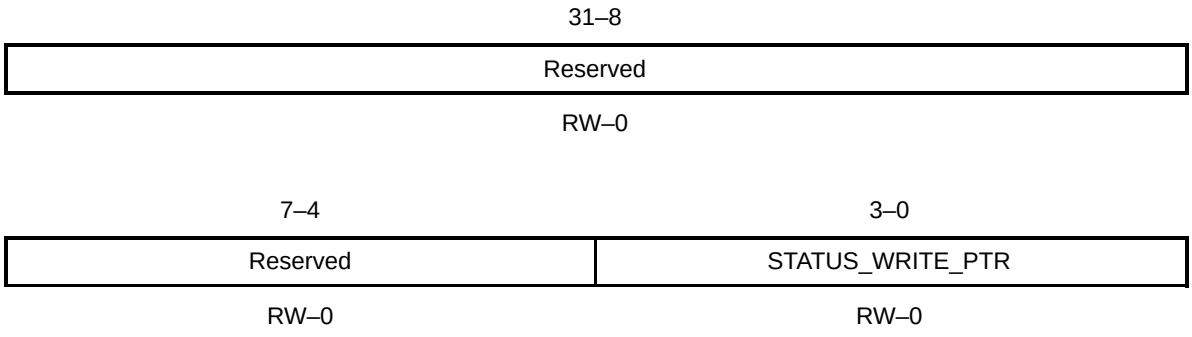
Note: R = Read access; W= Write access; value following dash (–) = value after reset

Bits 31–7 **Reserved**. Read as zeros.

Bits 6–0 **TX_READ_PTR.** Read pointer of TX FIFO. Can be access only if UART_IRDA_SCR[0] = 1.

Figure 8–43. Write Pointer of Status FIFO (UART_IRDA_WRPTR_STA)

Address (hex): Base = FFFF:0800, Offset = 0x0098



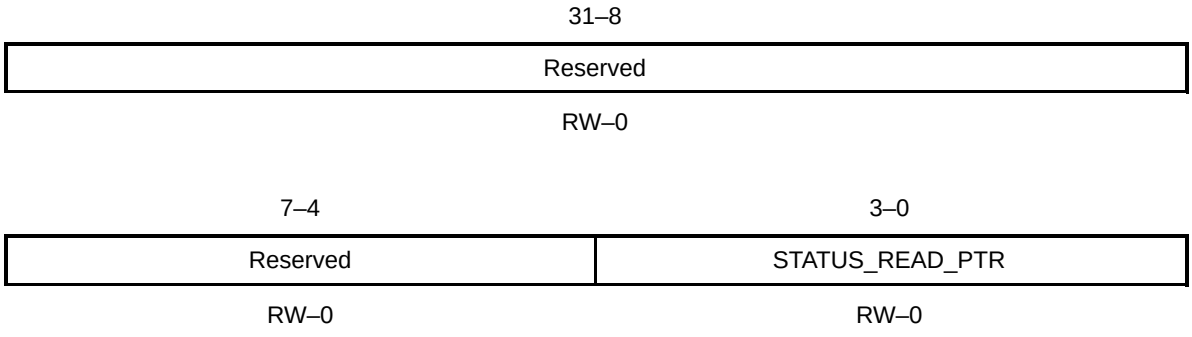
Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–4 **Reserved**. Read as zeros.

Bits 3–0 **STATUS_WRITE_PTR.** Write pointer of status FIFO. Can be accessed only if UART_IRDA_SCR[0] = 1.

Figure 8–44. Read Pointer of Status FIFO (UART_IRDA_RDPTR_STA)

Address (hex): Base = FFFF:0800, Offset = 0x009C



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–4 **Reserved**. Read as zeros.

Bits 3–0 **STATUS_READ_PTR.** Read pointer of status FIFO. Can be accessed only if UART_IRDA_SCR[0] = 1.

8.4.38 Resume Register

Figure 8–45. Resume Register (UART_IRDA_RESUME)

Address (hex): Base = FFFF:0800, Offset = 0x00A0

31–8
Reserved
R–0

7–0
DI
R–0

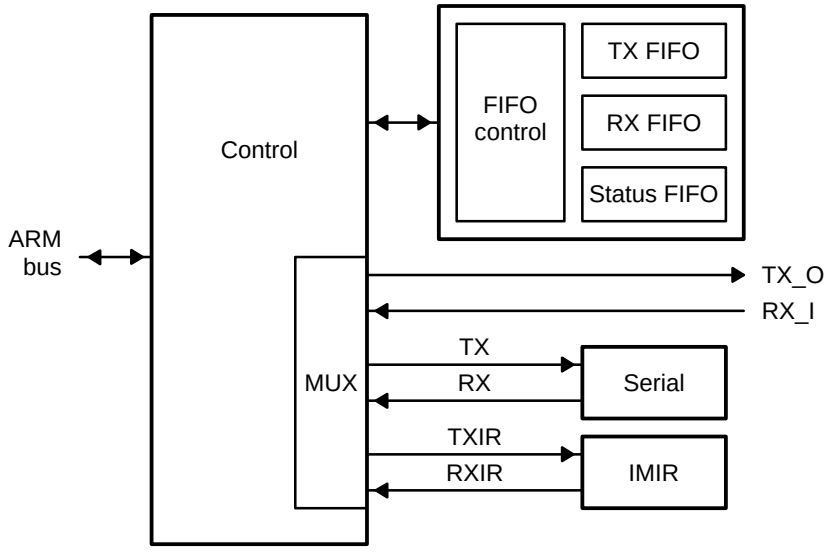
Note: R = Read access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **DI.** Dummy read to restart the transmission or reception.

8.5 UART IRDA Functional Block Diagram

Figure 8–46. Function Block Diagram



8.6 Serial Infrared Mode and Protocol

In serial infrared mode (SIR), data transfer takes place between the CPU and peripheral devices allowing serial communication at baud rates of up to 115.2 Kbit/s. A SIR transmit frame starts with start flags (either a single C0h, multiple C0h or a single C0h preceded by a number of FFh flags) followed by data frame, CRC–16 (16-bit cyclic redundancy check field), and ends with a stop flag (C1h).

Note that UART_IRDA_BLR[6] is used to select whether C0h or FFh start patterns are to be used when multiple start flags are required.

❑ Transmission:

The transmit state machine attaches start flags, CRC-16 and stop-flags. It checks the outgoing data to establish if data transparency is required.

SIR transparency is carried out if the outgoing data (between the start and stop flags) contains C0h, C1h or 7Dh. If one of these is about to be transmitted, the state machine sends an escape character (7Dh) first, then inverts the fifth bit of the real data to be sent, and sends this data immediately after the 7Dh character.

❑ Reception:

On receive, the receive state machine recovers the receive clock, removes the start flags, removes any transparency from the incoming data, and determines frame boundary with reception of the stop flag. It also checks for errors such as: frame abort (7Dh character followed immediately by a C1h stop flag, without transparency), CRC error and frame-length error. At the end of a frame reception, the CPU reads the LSR (line status register) to find out the errors, if any, of the received frame.

Data can be transferred both ways simultaneously by the module, but transmit and receive should not take place at the same time according to the standard.

The infrared pulse width output is determined by the standard in function of the baud rate frequency (see Section 8.6.4).

Note that if back-to-back frames are to be received by this device in SIR mode, the transmitting device must send at least 2 start flags at the start of every frame.

8.6.1 CRC Generation

Figure 8–47. IrDA Frame Format

xBOF	Begin of File	Address	Control	Information	End of File
N × 8 bits	8 bits	8 bits	8 bits	M × 8 bits	8 bits

The CRC polynomial to apply is $1 + X^5 + X^{12} + X^{16}$.

The CRC needs to be applied on the address (A), control (C), and information (I) bytes. The data is sent to CRC block LSB first.

The CRC is initialized to all ones.

The one's complement of the CRC is transmitted rather than the CRC itself.

The two words of CRC are written in the FIFO in reception.

8.6.2 Asynchronous Transparency

Prior to transmitting a byte, the UART_IRDA controller examines each byte in the payload and the CRC field (between BOF and EOF). For each byte equals to 0xC0 (BOF), 0xC1 (EOF), 0x7D (control escape) it does the following:

- In transmission:
  - Inserts a control escape (CE) byte preceding the byte
  - Complements bit 5 of the byte (i.e., exclusive ORs the byte with 0x20)

  The byte sent for the CRC computation is the initial byte written in the TX FIFO (before the XOR with 0x20).

- In reception:

  For the address, control, information, and FCS fields:

  - Compares the byte with the CE byte. If it is not equal, sends it to the CRC detector and stores it in the RX FIFO (store not required for the last two bytes)
  - If equal to CE, discards the CE byte
  - Complements bit 5 of the byte following the CE
  - Sends the complemented byte to the CRC detector and stores it in the RX FIFO (store not required in the last two bytes)

8.6.3 Abort Sequence

The transmitter may decide to prematurely close a frame. The transmitting station aborts by sending a CE byte immediately followed by a flag sequence 0x7DC1. The abort pattern closes the frame without an FCS field or an ending flag.

It is possible to abort a transmission frame by programming UART_IRDA_ACREG[1].

When this bit is set to 1, 7Dh and C1h are transmitted and the frame is not terminated with CRC or stop flags.

The receiver treats a frame as an aborted frame when a 7Dh character followed immediately by a C1h character has been received without transparency.

8.6.4 Pulse Shaping

The following describes programmable pulse shaping:

- The pulse width is entirely programmable, depending on the two divisors UART_IRDA_DIV_115K and UART_IRDA_DIV_BIT_RATE
- The principle is to calculate the value of these divisors to reach the desired frequency and the right pulse width at the same time
- The objective is to reach 115 200 bauds with a 50-MHz frequency and to send 1.6-µs pulse width

  1.6 µs corresponds to 625 000 Hz.
- The goal is to get 625 kHz as precision step

Table 8–3 gives the values for the divisors and the value for the UART_IRDA_PULSE_WIDTH register to obtain the expected pulse as a function of frequency, baud rates and pulse width.

Table 8–3. Pulse Shaping at a Frequency of 50 MHz

Baud Rate (kb/s)	Pulse Width	1st Counter X 2nd Counter	UART_irda_pulse_width
115.2	1.63 µs	16 x 27	5
57.6	3.26 µs	27 x 32	6
38.4	4.88 µs	27 x 48	9
19.2	9.77 µs	81 x 32	6
9.6	19.53 µs	65 x 80	15
2.4	78.13 µs	651 x 32	6

Notes: 1) (1^{st} counter = uart_irda_bit_rate and 2^{nd} counter = uart_irda_div_115k)
2) Baud rate and pulse width values are directly taken from the IrDA standard.
3) 1< start_point < value(UART_IRDA_DIV115K – 1)
1< pulse width < value(UART_IRDA_115K – 1) – value(START_POINT)

Encoder

Serial data from the transmit state machine is encoded to transmit data to the optoelectronics. While the serial data input to the TXD is high, the output (TXIR) is always low, and the counter used to form a pulse on TXIR is continuously cleared.

After TXD resets to 0, TXIR rises when the div_bit_rate counter reaches the start point value of the pulse (start value is set in UART_IRDA_START_POINT). TXIR falls when div_bit_rate counter (counts from 0 to DIV_115K–1) reaches pulse width value (set in UART_IRDA_PULSE_WIDTH register).

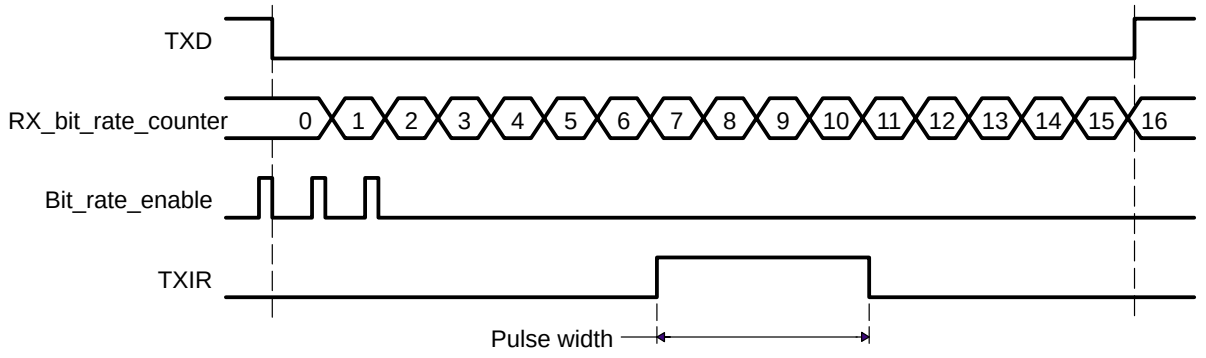
In the following example, the register values are set to:

UART_IRDA_START_POINT = 7

UART_IRDA_PULSE_WIDTH = 4

UART_IRDA_DIV_115K = 15

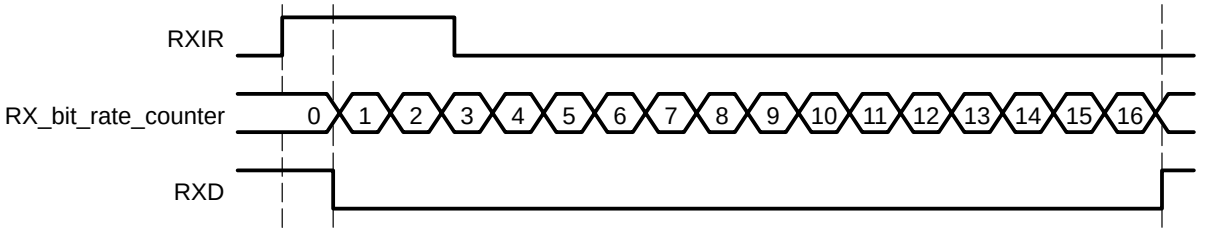
Figure 8–48. Encoder Timing Diagram



Decoder

After reset, RXD is high. When a rising edge is detected on RXIR, RXD falls on the next rising edge on the next rising edge of clk. RXD stays low until RX_bit_rate_counter reaches its maximum value and then returns to high as required by the IrDA specification. As long as no pulses (rising edges) are detected on the RXIR, RXD remains high.

Figure 8–49. Decoder Timing Diagram



It is possible for jitter or slight frequency differences to cause the next falling on RXIR to be missed for one counter cycle. In that case, a 1-clock-wide pulse appears on RXD between consecutive zeros. Thus, it is important for the receiver block to strobe the RX in the middle of the bit time to avoid latching this 1-clock-wide pulse.

8.6.5 Address Checking

In SIR mode, only frames intended for the device are written into the RX FIFO, if address checking has been enabled. This is to avoid receiving frames not meant for this device in a multipoint infrared environment. It is possible to program two frames addresses that the UART/IrDA module will receive with XON1/ADDR1 and XON2/ADDR2 registers.

Selecting address1 checking is done by setting UART_IRDA_EFR[1] to 1, and address2 by setting UART_IRDA_EFR[0] to 1. Setting EFR(1:0) to 0 disables all address checking operations. If both bits are set, the incoming frame will be checked for both address1 and address2.

If address checking is disabled, all received frames will be written into the reception FIFO.

8.7 Functional Descriptions

8.7.1 Trigger Levels

The UART provides independent programmable trigger levels for both receiver and transmitter interrupt generation. In the UART mode, trigger levels can be accessed through the UART_IRDA_FCR and UART_IRDA_TLR registers. These trigger levels are activated upon the enabling of the RHR and THR interrupts in the UART_IRDA_IER register. In the SIR mode, the Status FIFO trigger level interrupt can be accessed through UART_IRDA_MDR2[2:1] bits, and other triggers can be accessed in the same method as in the UART mode.

Interrupts in SIR and UART mode are independent. They are both activated with the IER register, but different in SIR or UART mode, even it they have the same address.

8.7.2 Interrupts

The UART IrDA module generates interrupts to the ARM through the interrupt line, $\overline{IRQ}$. All interrupts can be enabled or disabled through the interrupt enable register (UART_IRDA_IER). The interrupt status can be checked at any time by reading the interrupt status register (UART_IRDA_ISR).

8.7.2.1 Problem Definition

There are two interrupt conditions, which can cause undesirable UART IrDA logic behavior. These two interrupt conditions can be found in IT_TYPE (bits 5:1) of UART_IRDA_ISR (FFFF:0828) as defined below:

- 00110 => RX timeout (priority level 2)
- 00010 => RHR interrupt (priority level 2)

Without the workaround defined below, it would appear that the only way to clear the previously defined interrupt conditions would be to reset the UART IrDA logic module.

8.7.2.2 Problem Workaround

- When any UART IrDA interrupt is received, check the IT_TYPE for either an RX timeout or RHR interrupt as defined above.
- If the interrupt has been set due to either the RX timout or RHR interrupt, then perform the following maneuvers to reset the interrupt condition:
  - Disable all the interrupts by writing zero ‘0’ into UART_IRDA_IER (FFFF:0824).

- Read from UART_IRDA_RHR (FFFF:0800) until the FIFO is empty. The FIFO will be empty when UART_IRDA_LSR (FFFF:0814) is read with '0' in RX_FIFO_E (bit 0).
- Enable desired interrupts by writing into the UART_IRDA_IER (FFFF:0824).
- Return to normal operation.

❑ If the interrupt has been set due to any other interrupt condition, then the interrupt can be reset normally.

UART mode

There are five different possible interrupts, prioritized to four levels.

When an interrupt is generated, the UART_IRDA_ISR register indicates that an interrupt is pending by bringing UART_IRDA_ISR[0] to zero, and provides the type of interrupt through UART_IRDA_ISR[5:1]. Table 8–4 gives you the priority, ISR coding, type, source, and clear method for all interrupts in UART mode.

Table 8–4. Interrupts in UART Mode

UART_ISR[5–0]	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Method
000001	None	None	None	None
000110	1	Receiver line status	Overrun, framing error, parity error, or break detection occurred in characters in the RX FIFO	Framing, parity, or break: read all erroneous characters from the RX FIFO. Overrun: read LSR.
001100	2	RX time-out	Stale data in RX FIFO	Read RHR
000100	2	RHR interrupt	RX FIFO not empty if FIFO disabled RX FIFO above trigger level in other cases	Read RHR until interrupt condition disappears
000010	3	THR interrupt	TX FIFO empty if FIFO disabled TX FIFO below trigger level in other cases	Write to THR until interrupt condition disappears
010000	4	XOFF interrupt	Receive XOFF character or special character	Receive XON character or read ISR

SIR mode

In SIR mode, there are eight possible interrupts with no priority.

Table 8–5 gives you the ISR coding, type, source, and clear method for all interrupts in SIR mode.

Table 8–5. Interrupts in SIR Mode

UART_IRDA_ISR Bit	Interrupt Type	Interrupt Source	Interrupt Reset Method
0	RHR interrupt	DRDY (data ready) (FIFO disable)	Read RHR until condition disappears
		RX FIFO above trigger level (FIFO enable)	Read UART_IRDA_ISR
1	THR interrupt	TFE (TX FIFO empty) (FIFO disable)	Write to THR until interrupt condition disappears
		TX FIFO below trigger level (FIFO enable)	Read UART_IRDA_ISR
2	Last byte in RX FIFO	Last byte of frame in RX FIFO	Read UART_IRDA_ISR
3	RX overrun	Write to RHR when RX FIFO full	Read UART_IRDA_RESUME
4	Status FIFO interrupt	Status FIFO trigger level reached	Read Status FIFO register (UART_IRDA_SFLSR)
5	TX underrun	THR empty before EOF sent	Read UART_IRDA_RESUME
6	Receiver Line Status interrupt	CRC, abort or frame length error written into status FIFO	Read Status FIFO (UART_IRDA_SFLSR)
7	Received EOF	Received End of Frame	Read UART_IRDA_ISR

8.7.3 Features Available in UART Mode

8.7.3.1 Time-Out and Break Conditions

Time-Out

A time-out occurred if the receiver line RX_IRDA has been high for a time equivalent to four times the programmed word length plus 12 bits, and if the RHR register is not read during this time.

Break Condition

A break is detected if the RX line has been high for a time equivalent to word length plus 1 bit if there is no parity, or plus 2 bits in other cases.

8.7.3.2 Software Flow Control

The principle of software flow control is the same as that for hardware flow control: Transmissions are accepted or rejected by the receiving UART as a function of its RX FIFO pointer position from the trigger levels set. The UART IRDA module does not support hardware flow control; only the UART Modem module does. For more information, see section 9.6.4, *Hardware Flow Control*.

RX

When software flow control is activated, the UART compares incoming data with the programmed XOFF1 or XOFF2 character (in some cases, it can be XOFF1 followed by XOFF2). When the correct XOFF characters are received, transmission is halted after completing transmission of the current character. XOFF detection also sets ISR[4], if enabled.

To resume transmission, an XON character must be received, detected, and compared to the programmed XON character. (In some cases it can be XON2 following XON1). When the correct XON is detected, the transmission is resumed and the XOFF interrupt disappears.

The software flow control characters are not stored in RX FIFO once they are recognized. However, if there is one error (framing, parity, break) in them, those characters will be treated as normal data and will not be recognized as XOFF/XON characters.

If EFR[1:0] = “11”, which means that XOFF1/XOFF2 (XON1/XON2) must be received sequentially, the UART only recognizes the completely received couples as the control characters. If the first received character is recognized as XOFF1 but the subsequent character is not recognized as XOFF2, then both the XOFF1 and the subsequent character are written in RX FIFO as normal characters. It is the same case for XON1 and XON2.

TX

An XOFF1 or XOFF2 character is transmitted when the RX FIFO has passed the stop-trigger level programmed in TCR[3:0] (XOFF1 followed by XOFF2 in some cases).

An XON1 or XON2 character is transmitted when the RX FIFO reaches the start-trigger level programmed in TCR[7:4] (XON1 followed by XON2 in some cases).

General Features

It is important to note that if software flow control is disabled after sending XOFF, the XON will be sent automatically to enable normal transmission to proceed. The same occurs if the software flow combination changes after having sent XOFF: the original XON will be send. For example, if the couple XOFF1/XON1 was selected for software control, and if the user change to XOFF2/XON2 after having send XOFF1, XON1 will be sent when RX FIFO reaches the start trigger in spite of XOFF2/XON2 selection.

Transmission of the software flow control characters follows the same protocol as data, which means that the word's size, parity type, and number of stop bits defined for data will be used for control transmission. If the control characters are defined for 8 bits and the transmission is, for example, defined for 6 bits, there is no problem because the six LSBs of the control character will be sent by the transmitter UART and will be compared to the six LSBs of the control character stored in the receiving UART.

8.7.4 Features Available in SIR Mode

8.7.4.1 Frame Closing

There are two ways by which a transmission frame can be properly terminated.

1) **Frame length method:** This method is selected when UART_IRDA_MDR1(7) = 0. The CPU writes the frame-length value to TXFLL and TXFLH registers. The device automatically attaches end flags to the frame once the number of bytes transmitted becomes equal to the frame-length value.

2) **Set EOT bit method:** Set-EOT bit method is selected when UART_IRDA_MDR!(7) = 1. The CPU writes 1 to UART_IRDA_ACREG(0) (EOT bit) just before it writes the last byte to the TX FIFO. When the CPU writes the last byte to the TX FIFO, the device internally sets the tag bit for that particular character in the TX FIFO. As the TX state machine reads data from the TX FIFO, it uses this tag-bit information to attach end flags and properly terminate the frame.

8.7.4.2 Store and Control Transmission (SCT)

In SCT, the CPU first starts writing data into the TX FIFO. Then after it writes a part of a frame, it writes a 1 to UART_IRDA_ACREG(2) (deferred TX start) to start transmission. SCT is enabled when MDR1(5) = 1. This method of transmission is different in comparison to the normal mode where transmission of data starts immediately after data is written to the TX FIFO. SCT is useful to send short frames without TX underrun.

8.7.4.3 Underrun During Transmission

Underrun in transmission occurs when the TX FIFO becomes empty before the end of the frame is transmitted. When underrun occurs, the device closes the frame with end–flags but attaches an incorrect CRC value. The receiving device will detect a CRC error and discard the frame, and it can then ask for a re-transmission. Underrun also causes an internal flag to be set which disables further transmission. Before the next frame can be transmitted, the system CPU must:

- ❏ Reset the TX FIFO
- ❏ Read the UART_IRDA_RESUME register. This clears the internal flag

8.7.4.4 Overrun During Receive

Overrun occurs during receive if the RX state machine tries to write data into the RX FIFO when it is already full. When overrun occurs, the device interrupts the CPU with UART_IRDA_ISR(3) and discards the remaining portion of the frame. Overrun also causes an internal flag to be set which disables further reception. Before the next frame can be received, the system CPU must:

- ❏ Reset the RX FIFO
- ❏ Read the UART_IRDA_RESUME register. This clears the internal flag

8.7.4.5 Status FIFO

In SIR mode, a status FIFO is used to record the received frame status. When a complete frame is received, the length of the frame and the error bits associated with the frame are written into the STATUS FIFO.

The frame length and error status can be read by reading SFREGL/H and SFLSR. Reading the SFLSR causes the read pointer to be incremented. The status FIFO is eight entries deep and therefore can hold the status of eight frames.

The CPU uses the frame length information to locate the frame boundary in the received frame data. The CPU can screen bad frames using the error-status information, and later request the sender to resend only the bad frames.

Chapter 9

UART Modem Interface

This chapter explains the features of the UART Modem module, shows the applicable registers, and provides a functional description that includes trigger levels, interrupts, break and time-out conditions, hardware and software flow control, and the Autobauding mode.

The UART Modem Interface is associated with the ARM™ microcontroller unit (MCU) of the dual-core (MCU + DSP) VC547x device.

Topic **Page**

9.1 General Description

The UART modem module is a universal asynchronous receiver/transmitter that transmits characters sent to it by the ARM™ microcontroller unit (MCU) on the TX pin, and receives characters from the RX pin. It enables serial communication between the ARM and other outside devices.

The UART contains a transmitter (parallel-to-serial converter) and a receiver (serial-to-parallel converter) each clocked separately. The parallel side of the UART is connected to the ARM bus. It converts the parallel data (so called because they are stored, retrieved, and transferred eight or more bits at a time) from the ARM to serial asynchronous data communictions (in which data is sent and received one bit at a time) for transmission over a data link such as an EIA-232 cable or modem connection. The UART interface in the VC547x is compatible with the NS16C750 device. This UART is devoted to the connection of a PC-based software debugger tool through a standard wired interface.

9.2 Main Features

This module contains a 64-word (9 and 11 bits)-deep FIFO for received characters and another 64-word FIFO for transmitted characters with programmable trigger levels for the FIFOs. Hardware buffering allows higher transmission speed without data loss and without requiring frequent attention from the ARM.

The module is configurable to send even, odd, or no parity, and 1, 1.5, or 2 stop bits. The number of data bits can be configured between five and eight bits. Break characters can be generated and detected.

The baud rate is internally generated by a programmable divisor.

All transmitting parameters can be detected in reception mode by an autobauding mechanism that recognizes speed, word size, parity, and the number of stop bits.

Data flow control can be managed either in hardware (auto-RTS, auto-CTS) or in software with XON/XOFF character detection. Note: Only the UART Modem module, not the UART IRDA module, has the hardware flow control mechanism. Hardware flow control relieves the ARM of excessive software overhead.

All the UART features (even autobauding) are completely independent of clock-speed, which allows an all-system frequency up to 50 MHz.

9.2.1 UART Mode Features

- The UART supports five modem signals:
  - TX: Output – transmit data
  - RX: Input – receive data
  - RTS: Output – request to send
  - CTS: Input – clear to send
  - DCD: Output – data carrier detected
- Line break generation and detection
- Interrupt system control
- Loopback capabilities for internal test
- Baud rates up to 6.25 Mbaud are supported (at 50 MHz)
- Autobauding supports speeds between 1200 baud and 115.2K baud, and a 7- or 8-bit word size
- Hardware flow control (DCD, RTS/CTS)

9.3 I/O Description

Table 9–1. Modem I/O Signals

Signal	I/O	Function
RX_MODEM	I	Modem serial data input
CTS_MODEM	I	Modem clear to send input signal. Active-low
TX_MODEM	O	Modem serial data output
RTS_MODEM	O	Modem request to send output signal. Active-low
DCD_MODEM	O	Modem carrier detect output. Active-low

9.4 Register Mapping/Descriptions

9.4.1 UART Modem Module Registers

Base address (hex): FFFF:1000

Register width: 32 bits

Table 9–2. UART Modem Module Registers

Register	Description	Offset Address
UART_RHR	Receive Holding Register	00h
UART_THR	Transmit Holding Register	04h
UART_FCR	FIFO Control Register	08h
UART_SCR	Status Control Register	0Ch
UART_LCR	Line Control Register	10h
UART_LSR	Line Status Register	14h
UART_SSR	Supplementary Status Register	18h
UART_MCR	Modem Control Register	1Ch
UART_MSR	Modem Status Register	20h
UART_IER	Interrupt Enable Register	24h
UART_ISR	Interrupt Status Register	28h
UART_EFR	Enhanced Feature Register	2Ch
UART_XON1	XON1 Character Register	30h
UART_XON2	XON2 Character Register	34h
UART_XOFF1	XOFF1 Character Register	38h
UART_XOFF2	XOFF2 Character Register	3Ch
UART_SPR	Scratch-Pad Register	40h
UART_DIV_115K	Divisor for 115K-Baud Generation	44h
UART_DIV_BIT_RATE	Divisor for Baud-Rate Generation	48h
UART_TCR	Transmission Control Register	4Ch

Table 9–2. UART Modem Module Registers (Continued)

Register	Description	Offset Address
UART_TLR	Trigger-Level Register	50h
UART_MDR	Mode Definition Register	54h
UART_UASR	UART Autobauding Status Register	58h
UART_RDPTR_URX	RX FIFO Read Pointer Register	5Ch
UART_WRPTR_URX	RX FIFO Write Pointer Register	60h
UART_RDPTR_UTX	TX FIFO Read Pointer Register	64h
UART_WRPTR_UTX	TX FIFO Write Pointer Register	68h

9.4.2 Special Access Registers

It is important to note here that some registers need special conditions to be accessed in write mode.

- ❏ UART_MCR[7:5] and UART_FCR[5:4] can only be written when UART_EFR[4] = 1.
- ❏ UART_TCR and UART_TLR can only be written when UART_MCR[6] = 1.
- ❏ UART_WRPTR_URX, UART_RDPTR_URX, UART_WRPTR_UTX, and UART_RDPTR_UTX can only be written when UART_SCR[0] = 1.

All the other registers can be accessed unconditionally.

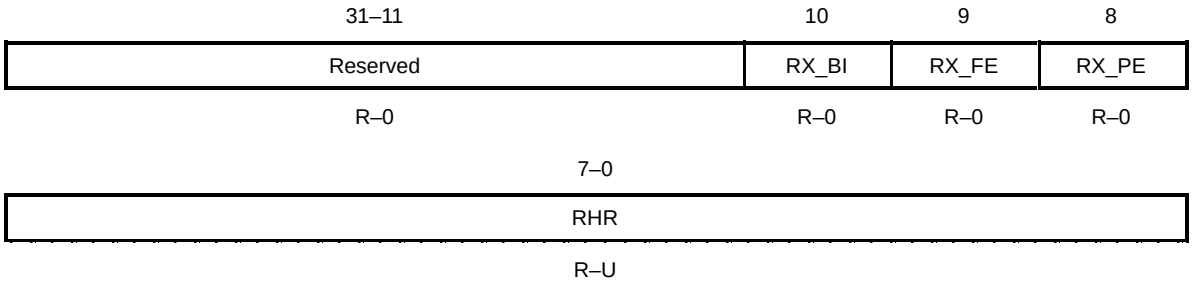
9.4.3 Receive Holding Register

This register holds the received word that is to be read by the MCU. Received data is stored in a 64-word FIFO and the first unread word is presented to the RHR, replaced by the second unread word after an RHR access. If the FIFO is disabled, the RHR register will contain the received data in the same way. If an overflow occurs, received data will not be written into the FIFO, and consequently, will not be able to be read through RHR.

Bits 8, 9, and 10 of RHR indicate which kind of error has occurred on current read data.

Figure 9–1. Receive Holding Register (UART_RHR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0000



Note: R = Read access; value following dash (–) = value after reset; U = Undefined

Bits 31–11 **Reserved**. Read as zeros.

Bit 10 **RX_BI.**

0 No break condition

1 A break was detected while receiving read data (i.e., RX_MODEM input signal was low for one character frame)

Bit 9 **RX_FE.**

0 No framing error in read data

1 A framing error occurred while receiving read data (i.e., received data did not have a valid stop bit)

Bit 8 **RX_PE.**

0 No parity error in read data

1 A parity error occurred while receiving read data

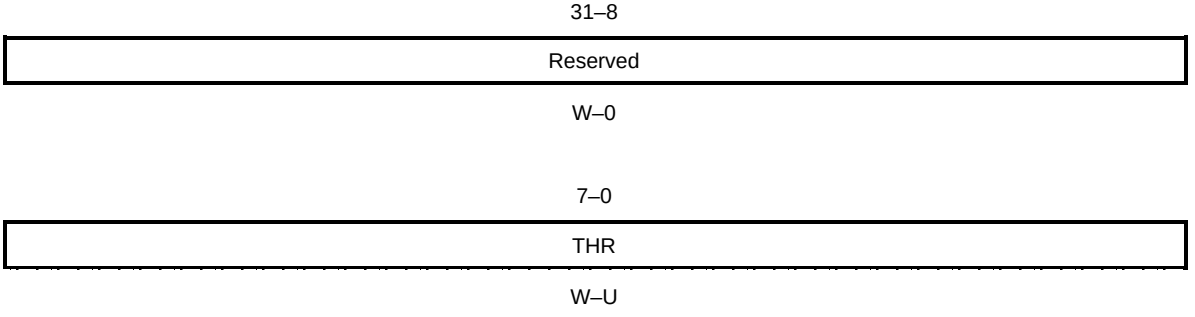
Bits 7–0 **RHR.** Receive holding register (contains the first unread byte of the 64-byte RX FIFO). If overflow occurs, data is not overwritten in FIFO.

9.4.4 Transmit Holding Register

This register stores data to send. Once it is written by the CPU, data is transmitted into the transmitter FIFO, and waits for its parallel-to-serial translation before being shifted onto the TX_MODEM output pin. If the FIFO is disabled, you should be sure that the TX FIFO is not full (SSR[0]) before writing to THR because you can overwrite data to send.

Figure 9–2. Transmit Holding Register (UART_THR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0004



Note: W = Write access; value following dash (–) = value after reset; U = Undefined

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **THR.** Transmit holding register (64-byte FIFO)

9.4.5 FIFO Control Register

Note that bits 4 and 5 can only be written when EFR[4] = 1. (UART_EFR[4] enables/disables writing to UART_FCR[5:4] and UART_MCR[7:5].)

Figure 9–3. FIFO Control Register (UART_FCR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0008

31–8
Reserved
W–0

7–6	5–4	3	2	1	0
RX_FIFO_TRIG	TX_FIFO_TRIG	Reserved	RX_FIFO_CLEAR	TX_FIFO_CLEAR	FIFO_EN
W–0	W–0	W–0	W–0	W–0	W–0

Note: W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–6 **RX_FIFO_TRIG.** Sets the trigger level for the RX FIFO.

00 8 bytes
01 16 bytes
10 32 bytes
11 60 bytes

Bits 5–4 **TX_FIFO_TRIG.** Sets the trigger level for the TX FIFO if EFR(4) = 1.

00 8 bytes
01 16 bytes
10 32 bytes
11 56 bytes

Bit 3 **Reserved**. Read as zero.

Bit 2 **RX_FIFO_CLEAR.**

0 No change
1 Clears the RX FIFO and resets its counter logic to zero

Bit 1 **TX_FIFO_CLEAR.**

0 No change
1 Clears the TX FIFO and resets its counter logic to zero

Bit 0 **FIFO_EN.**

0 Disables the TX and RX FIFOs

1 Enables the TX and RX FIFOs

9.4.6 Status Control Register

The FIFO initialization writes zeros into the TX and RX FIFO in order not to have unknown values in some registers, like RHR, before a word's reception.

To initialize FIFO, bit 6 of the SCR register must be set to one and must stay as one until initialization has ended. Once initialization has ended, SCR[6] goes to zero and SCR[7] goes to one, indicating FIFO initialization is finished. SCR[7] is cleared on a read.

Figure 9–4. Status Control Register (UART_SCR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x000C

31–8	7	6
Reserved	FIFO_INIT_STATUS	FIFO_INIT
RW–0	R–0	W–0

5	4	3	2–1	0
Reserved	RX_CTS_WAKE_UP_ENABLE	TX_EMPTY_CTL_IT	Reserved	FIFO_PTR_ACCESS_EN
W–0	W–0	W–0	W–0	W–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bit 7 **FIFO_INIT_STATUS.**

0 Initialization of FIFOs is not finished if SCR(6) = 1

1 Initialization of FIFOs is finished. Clear on a read

Bit 6 **FIFO_INIT.** Initialize FIFO bit.

0 FIFOs are not initialized

1 FIFOs are initialized to zero. This bit auto-clears

Bit 5 **Reserved**. Read as zero.

Bit 4 **RX_CTS_WAKE_UP_ENABLE.**

0 Disables the wake-up interrupt and clears UART_SSR(1)

1 Waits for the falling edge of input pins RX_MODEM or CTS_MODEM to generate an interrupt

Bit 3 **TX_EMPTY_CTL_IT.**

0 Normal mode for UART_THR interrupt

1 The UART_THR interrupt is generated when TX FIFO and TX shift register are empty

Bits 2–1 **Reserved**. Read as zeros.

Bit 0 **FIFO_PTR_ACCESS_EN.**

0 Disables FIFO's pointer access through registers

1 Enables FIFO's pointer access through UART_WRPTR_URX, UART_RDPTR_URX, UART_WRPTR_UTX, and UART_RDPTR_UTX

9.4.7 Line Control Register

Figure 9–5. Line Control Register (UART_LCR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0010

31–7
Reserved
RW–0

6	5	4	3	2	1–0
BREAK_EN	PARITY_TYPE2	PARITY_TYPE1	PARITY_EN	NB_STOP	CHAR_LENGTH
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–7 **Reserved**. Read as zeros.

Bit 6 **BREAK_EN.** Break control bit.

0 No break

1 Forces the transmitter output to go low to alert the communication terminal

Bit 5 **PARITY_TYPE2.** Selects the forced parity format if UART_LCR(3) = 1.

0	No forced parity
1	If UART_LCR(4) = 0, the parity bit is forced to 1 in the TX and RX data. If UART_LCR(4) = 1, the parity bit is forced to 0 in the TX and RX data.

Bit 4 **PARITY_TYPE1.**

0	Odd parity is generated (if bit 3 = 1)
1	Even parity is generated (if bit 3 = 1)

Bit 3 **PARITY_EN.**

0	No parity
1	A parity bit is generated during transmission and the receiver checks for received parity

Bit 2 **NB_STOP.** Number of stop bits.

0	1 stop bit (word length = 5, 6, 7, 8 bits)
1	1.5 stop bits (word length = 5) 2 stop bits (word length = 6, 7, 8)

Bits 1–0 **CHAR_LENGTH.** Word length for TX and RX.

00	5 bits
01	6 bits
10	7 bits
11	8 bits

9.4.8 Line Status Register

Bit 7 indicates if there is an error in RX FIFO, which means that the UART received data with a framing error, parity error, or a break indication. This bit stays at one until no more errors remain in the FIFO, (i.e., until all data with errors is read).

Figure 9–6. Line Status Register (UART_LSR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0014

31–8
Reserved
R–0

7	6	5	4–2	1	0
RX_FIFO_STS	TX_SR_E	TX_FIFO_E	Reserved	RX_OE	RX_FIFO_E
R–0	R–1	R–1	R–0	R–0	R–0

Note: R = Read access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bit 7 **RX_FIFO_STS.**

0 Normal operation

1 At least one parity error, framing error, or break indication in the RX FIFO. Cleared when no more errors are present in the FIFO

Bit 6 **TX_SR_E.**

0 Transmitter hold and shift registers are not empty

1 Transmitter hold and shift registers are empty

Bit 5 **TX_FIFO_E.**

0 Transmitter hold register is not empty

1 Transmitter hold register is empty. The processor can load up to 64 bytes of data into THR if the TX FIFO is enabled

Bits 4–2 **Reserved**. Read as zeros.

Bit 1 **RX_OE.**

0 No overrun error

1 Overrun error has occurred. Set when the character being held in RX shift register is not transferred to the RX FIFO. This case can only occur when RX FIFO is full

Bit 0 **RX_FIFO_E.**

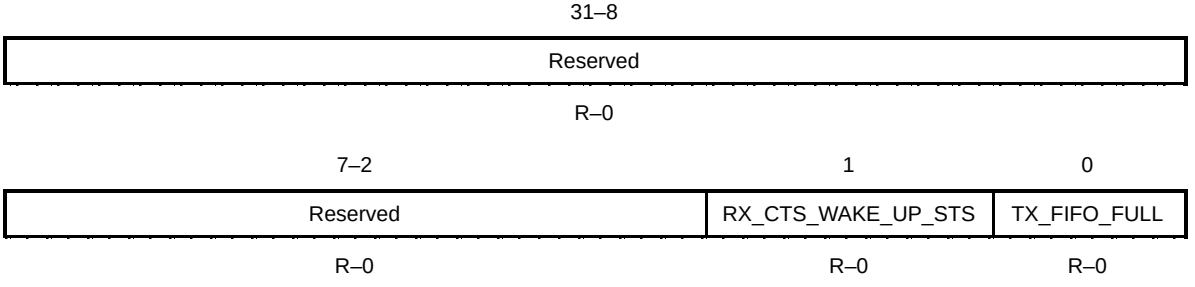
0 No data in the RX FIFO.

1 At least one data character in the RX FIFO.

9.4.9 Supplementary Status Register

Figure 9–7. Supplementary Status Register (UART_SSR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0018



Note: R = Read access; value following dash (–) = value after reset

Bits 31–2 **Reserved**. Read as zeros.

Bit 1 **RX_CTS_WAKE_UP_STS.** Enabled if UART_SCR[4] = 1.

0 No falling edge event on either RX_MODEM or CTS_MODEM

1 A falling edge occurred on RX_MODEM or CTS_MODEM

Bit 0 **TX_FIFO_FULL.**

0 TX FIFO not full

1 TX FIFO full

9.4.10 Modem Control Register

Note that bits 5, 6 and 7 can be written only when UART_EFR[4] = 1.

Figure 9–8. Modem Control Register (UART_MCR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x001C

31–8
Reserved
RW–0

7	6	5	4	3–2	1	0
CLKSEL	TCR_TLR	XON_EN	LOOPBACK_EN	Reserved	RTS	DCD
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bit 7 **CLKSEL.**

0 No action

1 Divide clock input by four

Bit 6 **TCR_TLR.**

0 No action

1 Enables access to the UART_TCR and UART_TLR registers

Bit 5 **XON_EN.**

0 Disables *XON any* function

1 Enables *XON any* function

Bit 4 **LOOPBACK_EN.**

0 Normal operating mode

1 Internal loopback mode. UART_MCR[1:0] is looped into UART_MSR[5:4]

Bits 3–2 **Reserved**. Read as zeros.

Bit 1 **RTS.** RTS control if auto-RTS is disabled. If auto-RTS is enabled, the RTS output is controlled by hardware flow control.

0 Forces RTS_MODEM to be inactive (high)

1 Forces RTS_MODEM to be active (low)

Bit 0 **DCD.**

0 Forces DCD_MODEM signal to be inactive (high)
1 Forces DCD_MODEM signal to be active (low)

9.4.11 Modem Status Register

Figure 9–9. Modem Status Register (UART_MSR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0020

31–8
Reserved
R–0

7–6	5	4	3–2	1	0
Reserved	NDSR_STS	NCTS_STS	Reserved	DSR_STS	CTS_STS
R–0	R–0	R–Input Signal	R–0	R–0	R–0

Note: R = Read access; value following dash (–) = value after reset

Bits 31–6 **Reserved**. Read as zeros.

Bit 5 **NDSR_STS.** In loop-back mode, it is equivalent to UART_MCR(0).

Bit 4 **NCTS_STS.** This bit is the complement of the CTS_MODEM input. In loopback mode, it is equivalent to UART_MCR(1).

Bits 3–2 **Reserved**. Read as zeros.

Bit 1 **DSR_STS.**

0 UART_MCR(0) has NOT changed state in loopback mode
1 UART_MCR(0) has changed state in loopback mode. Cleared on a read

Bit 0 **CTS_STS.**

0 CTS_MODEM input (or UART_MCR(1) in loopback mode) has NOT changed state
1 CTS_MODEM input (or UART_ MCR(1) in loopback mode) has changed state. Cleared on a read

9.4.12 Interrupt Enable Register

The Interrupt Enable register is used to enable or disable any interrupt. There are seven types of interrupts. You should be aware that the RHR interrupt enable is necessary in order to obtain a time-out interrupt (see Section 9.4.13, *Interrupt Status Register*).

Figure 9–10. Interrupt Enable Register (UART_IER)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0024

31–8
Reserved
RW–0

7	6	5	4	3	2	1	0
CTS_IT	RTS_IT	XOFF_IT	Reserved	MODEM_STS_IT	LINE_STS_IT	THR_IT	RHR_IT
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bit 7 **CTS_IT.**

0 Disables the CTS interrupt
1 Enables the CTS interrupt

Bit 6 **RTS_IT.**

0 Disables the RTS interrupt
1 Enables the RTS interrupt

Bit 5 **XOFF_IT.**

0 Disables the XOFF interrupt
1 Enables the XOFF interrupt

Bit 4 **Reserved**. Read as zero.

Bit 3 **MODEM_STS_IT.**

0 Disables the UART_MSR interrupt
1 Enables the UART_MSR interrupt

Bit 2 **LINE_STS_IT.**

0 Disables the UART_LSR interrupt

1 Enables the UART_LSR interrupt

Bit 1 **THR_IT.**

0 Disables the THR interrupt

1 Enables the THR interrupt

Bit 0 **RHR_IT.**

0 Disables the RHR interrupt

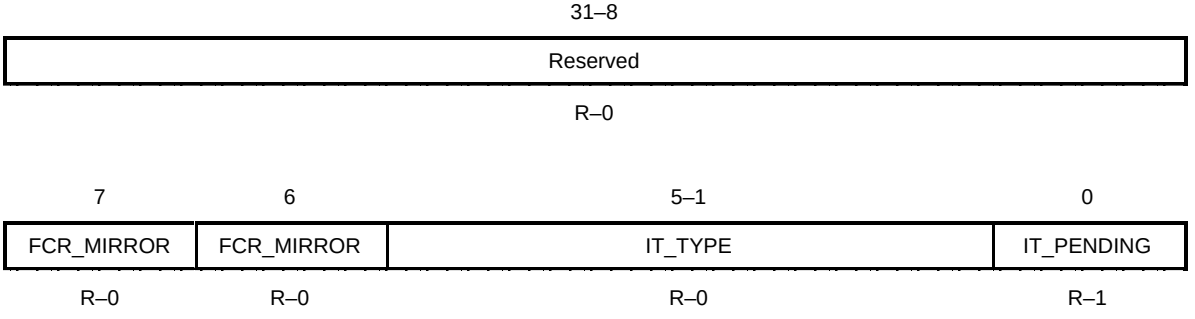
1 Enables the RHR interrupt

9.4.13 Interrupt Status Register

All interrupts have a priority level. See Section 9.6.2, *Interrupts*, for more information on the management, priority, and clearing of interrupts.

Figure 9–11. Interrupt Status Register (UART_ISR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0028



Note: R = Read access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bit 7 **FCR_MIRROR.** Mirrors the content of FCR(0)

Bit 6 **FCR_MIRROR.** Mirrors the content of FCR(0)

Bits 5–1 **IT_TYPE.** Interrupts by priority.

00011	Receiver line status error (Priority level 1)
00110	RX time out (Priority level 2)
00010	RHR interrupt (Priority level 2)
00001	THR interrupt (Priority level 3)
00000	Modem status interrupt (Priority level 4)
01000	XOFF/special character interrupt (Priority level 5)
10000	CTS, RTS change of state (Priority level 6)

Bit 0 **IT_PENDING.**

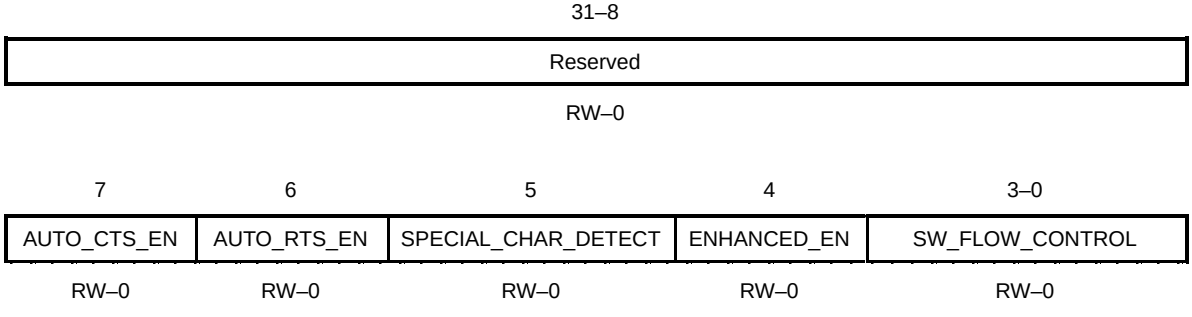
0	An interrupt (except the one defined by UART_SCR(4)) is pending. $\overline{IRQ}$ is active
1	No interrupt (except the one defined by UART_SCR(4)) is pending. $\overline{IRQ}$ is active

9.4.14 Enhanced Feature Register

This register enables or disables enhanced features related to flow control, except bit 4, which enables write operation onto MCR and FCR. Note that XON1 and XON2 (and XOFF1 and XOFF2) must be different if software flow control is used.

Figure 9–12. Enhanced Feature Register (UART_EFR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x002C



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bit 7 **AUTO_CTS_EN.** Auto-CTS enable bit.

0	Normal operating mode
1	Auto-CTS flow-control enable (i.e., transmission is halted when CTS is inactive.

Bit 6 **AUTO_RTS_EN.** Auto-RTS enable bit.

0	Normal operating mode
1	Auto-RTS flow control enable (i.e., RTS goes inactive when RX FIFO halt-trigger level is reached, and goes active when restore trigger is reached.

Bit 5 **SPECIAL_CHAR_DETECT.**

0	Normal operating mode
1	Enables special character detection. Received character is compared to XOFF2. If a match occurs, the received character is transferred to RX FIFO and UART_ISR(4) is set to 1

Bit 4 **ENHANCED_EN.** Enhances the write enable function.

0	Disables writing to UART_FCR[5:4], UART_MCR[7:5]
1	Enables writing to UART_FCR[5:4], UART_MCR[7:5]

Bits 3–0 **SW_FLOW_CONTROL.** Selection of software flow control.

00XX	No transmit flow control
01XX	Transmit XON2, XOFF2
10XX	Transmit XON1, XOFF1
11XX	Transmit XON1, XON2, XOFF1, XOFF2
XX00	No receive flow control
XX01	Receiver compares XON2, XOFF2
XX10	Receiver compares XON1, XOFF1
XX11	Receiver compares XON1, XON2, XOFF1, XOFF2

Note: XON1/XON2 and XOFF1/XOFF2 must be set to different values if the software flow control is used.

9.4.15 XON1 Character Register

Figure 9–13. XON1 Character Register (UART_XON1)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0030

31–8
Reserved
RW–0

7–0
XON_WORD1
RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **XON_WORD1.** Used to store the 8-bit XON1 character.

9.4.16 XON2 Character Register

Figure 9–14. XON2 Character Register (UART_XON2)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0034

31–8
Reserved
RW–0

7–0
XON_WORD2
RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

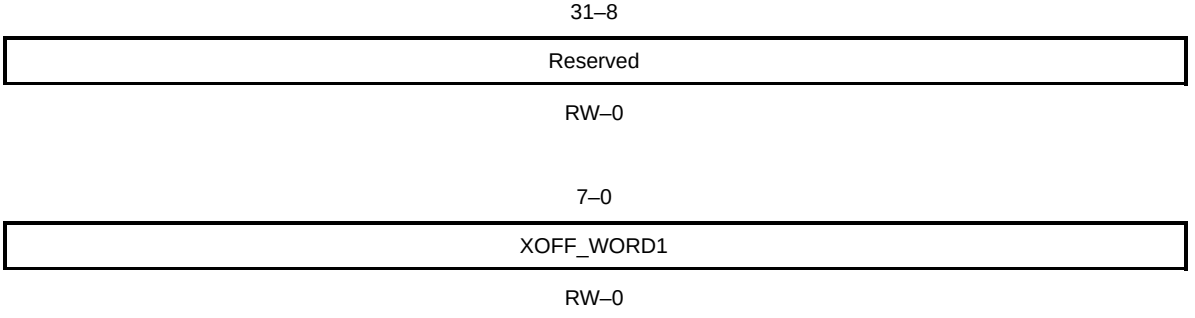
Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **XON_WORD2.** Used to store the 8-bit XON2 character.

9.4.17 XOFF1 Character Register

Figure 9–15. XOFF1 Character Register (UART_XOFF1)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0038



Note: R = Read access; W = Write access; value following dash (–) = value after reset

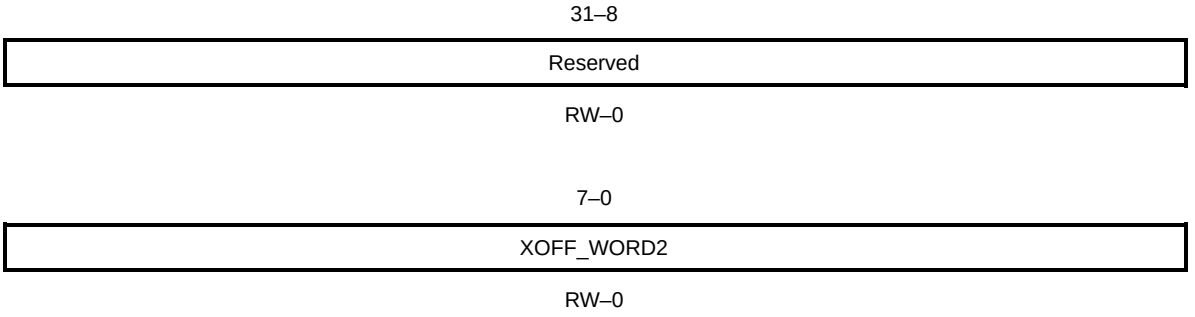
Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **XOFF_WORD1.** Used to store the 8-bit XOFF1 character.

9.4.18 XOFF2 Character Register

Figure 9–16. XOFF2 Character Register (UART_XOFF2)

Address (hex): Base = 0xFFFF:1000, Offset = 0x003C



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

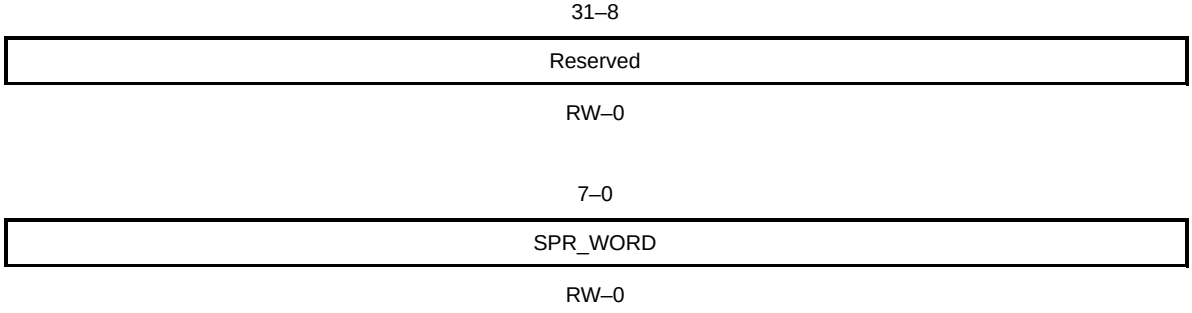
Bits 7–0 **XOFF_WORD2.** Used to store the 8-bit XOFF2 character.

9.4.19 Scratch-Pad Register

This register does not have a control function. It is intended as a scratch-pad to be used by the programmer for holding temporary data.

Figure 9–17. Scratch-Pad Register (UART_SPR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0040



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

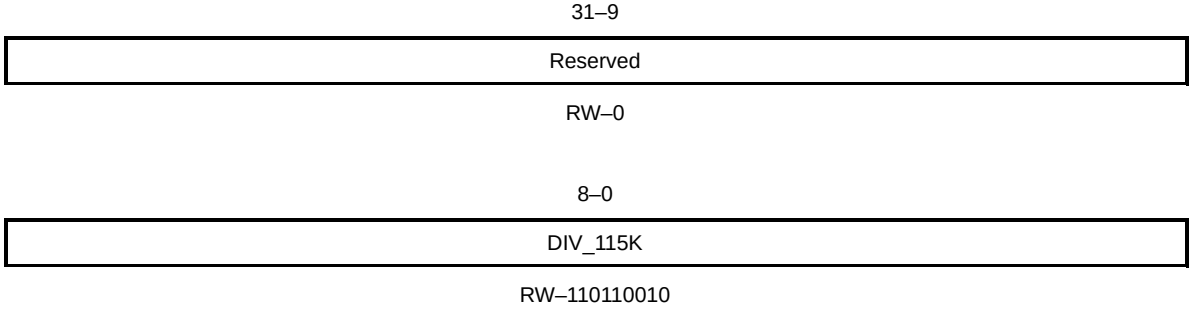
Bits 7–0 **SPR_WORD.** Scratch pad register.

9.4.20 Divisor for 115k-Baud Generation

This 9-bit register is used to store the divisor needed to obtain a 115K-baud rate. The reset value (434) is the divisor needed at 50 MHz (50,000,000/115,200 = 434).

Figure 9–18. Divisor for 115K-Baud Generation (UART_DIV_115K)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0044



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–9 **Reserved**. Read as zeros.

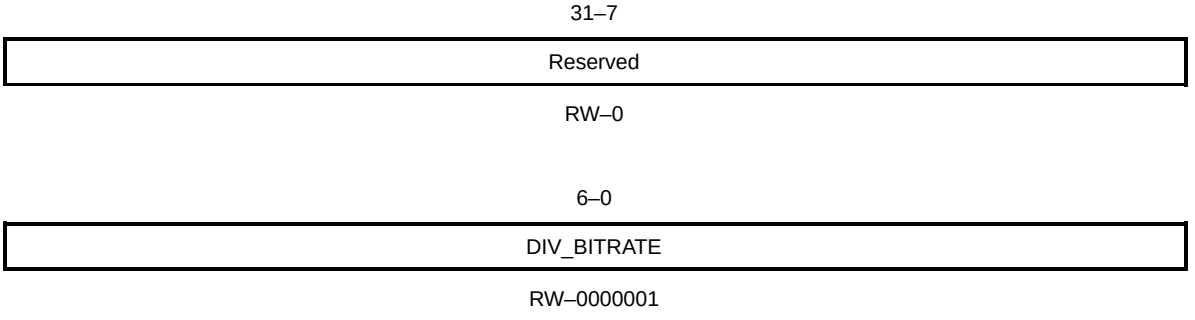
Bits 8–0 **DIV_115K.**
div_115k = (fclk / 115200)
Div_115k is the divisor needed to obtain the 115K baud rate; it is the clock speed (in Hz) divided by 115200.

9.4.21 Divisor for Baud-Rate Generation

This 7-bit register represents the divisor needed to obtain the desired speed from 115K baud; for example, 0000010 gives 115K baud/2 = 57500 baud.

Figure 9–19. Divisor for Baud-Rate Generation (UART_DIV_BIT_RATE)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0048



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–7 **Reserved**. Read as zeros.

Bits 6–0 **DIV_BITRATE.**
div_bit_rate = div_115K / bit rate.
Div_bit_rate is the divisor needed to obtain the desired speed.

9.4.22 Transmission Control Register

This register is used to store the receive FIFO threshold levels to start and stop transmission during hardware or software flow control.

This register can be written only if MCR[6] = 1

Figure 9–20. Transmission Control Register (UART_TCR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x004C

31–8
Reserved
RW–0

7–4	3–0
RX_FIFO_TRIG_START	RX_FIFO_TRIG_HALT
RW–0	RW–1111

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–4 **RX_FIFO_TRIG_START.** Receive FIFO trigger level to restore transmission.

0000	0 bytes
0001	4 bytes
0010	8 bytes
⇓	⇓
1111	60 bytes

Bits 3–0 **RX_FIFO_TRIG_HALT.** Receive FIFO trigger level to stop transmission.

0000	0 bytes
0001	4 bytes
0010	8 bytes
⇓	⇓
1111	60 bytes

9.4.23 Trigger-Level Register

This register is used to store the programmable transmit and receive FIFO trigger levels used for IRQ generation. Trigger levels from 4 to 60 can be programmed with a granularity of 4.

Note that TLR can be written only if MCR[6] = 1.

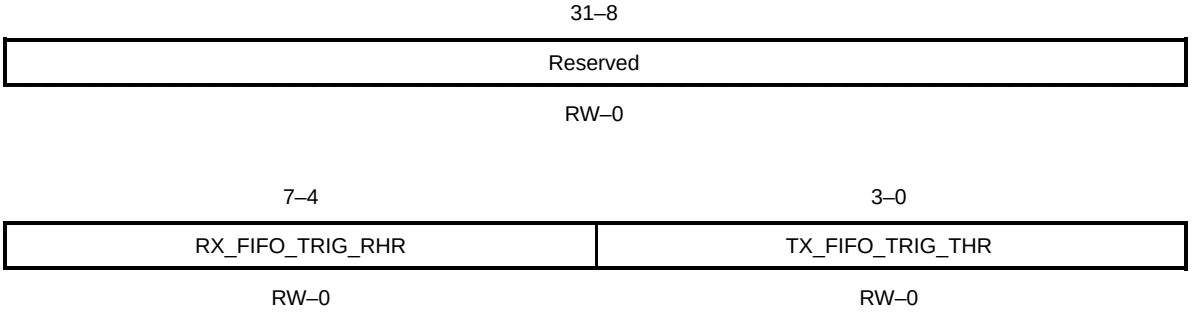
If TLR[7:4] = 0000, the programmable RX trigger levels are disabled and trigger RX levels in FCR[7:6] are enabled.

If TLR[3:0] = 0000, the programmable TX trigger levels are disabled and trigger TX levels in FCR[5:4] are enabled.

Note that for the TX FIFO, the TLR represents the number of empty spaces in the FIFO above which the THR interrupt will be activated. For example, if TLR[3:0] = 1111, and if there are four or less bytes in the transmit FIFO, the THR interrupt will be activated. If TLR[3:0] = 0001, and if there are 60 or less bytes in the transmit FIFO, the interrupt will be activated.

Figure 9–21. Trigger-Level Register (UART_TLR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0050



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–4 **RX_FIFO_TRIG_RHR.** RX FIFO trigger level to generate RHR interrupt.

0000	Use UART_FCR[7:6] trigger level
0001	4 bytes
0010	8 bytes
⇓	⇓
1111	60 bytes

Bits 3–0 **TX_FIFO_TRIG_THR.** TX FIFO trigger level to generate THR interrupt. It presents the number of empty spaces in the FIFO above which the THR interrupt will be activated.

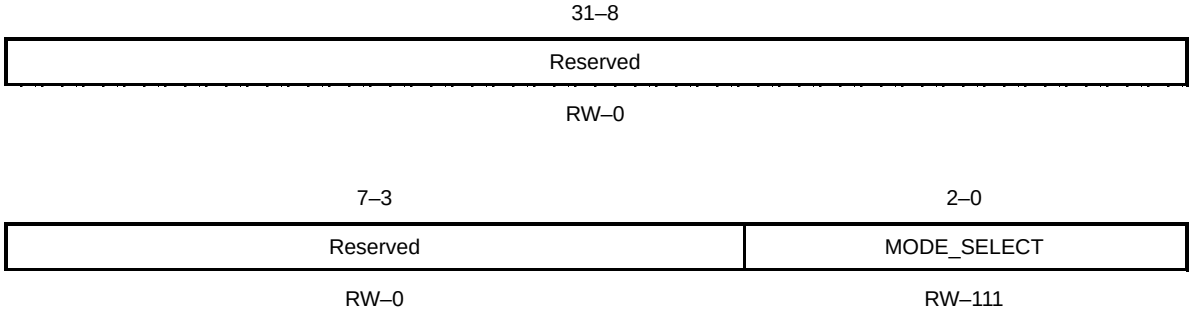
0000	Use UART_FCR[5:4] trigger level
0001	4 bytes
0010	8 bytes
⇓	⇓
1111	60 bytes

9.4.24 Mode Definition Register

To change the UART mode, you must set MDR[2:0] to the reset state and then to the new mode in order to avoid undefined behavior. The reset mode resets transmission, reception, and autobaud parameter detection, but not TX and RX FIFO pointers.

Figure 9–22. Mode Definition Register (UART_MDR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0054



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–3 **Reserved**. Read as zeros.

Bits 2–0 **MODE_SELECT.**

000	UART mode
010	Autobauding mode
111	Reset mode
	All other values are reserved

9.4.25 UART Autobauding Status Register

This register determines the speed, word size, parity, and stop-bit number in Autobauding mode. Parameters are detected and the UART is ready to receive or transmit data only when UASR is not null. Parameters are detected after receiving the ***AT*** or ***at*** characters, which are not stored in the RX FIFO.

Figure 9–23. UART Autobauding Status Register (UART_UASR)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0058

31–8
Reserved
RW–0

7–6	5	4	3–0
PARITY_TYPE	BIT_BY_CHAR	STOP_BIT	SPEED
RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–6 **PARITY_TYPE.**

00	No parity identified
01	space parity (0 forced)
10	even parity
11	odd parity

Bit 5 **BIT_BY_CHAR.** Word's size.

0	7-bit character identified
1	8-bit character identified

Bit 4 **STOP BIT.**

0	1 stop bit identified
1	2 stop bits identified

Bits 3–0 **SPEED.** Speed identifier.

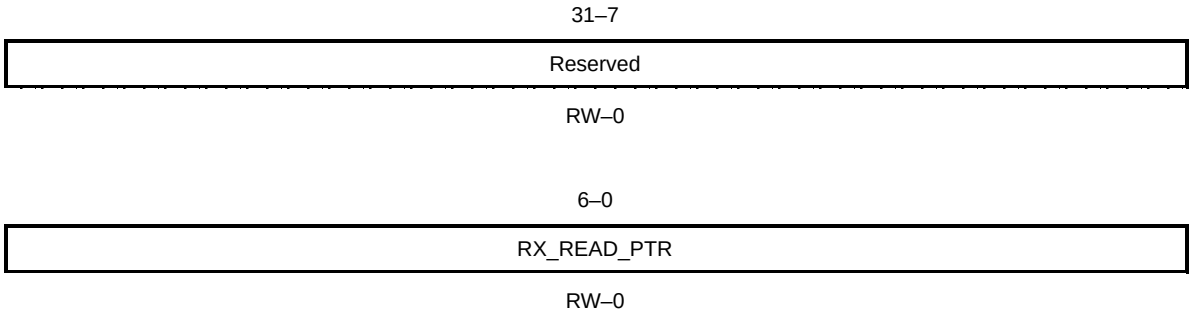
0000	no speed identified
0001	115200 baud
0010	57600 baud
0011	38400 baud
0100	28800 baud
0101	19200 baud
0110	14400 baud
0111	9600 baud
1000	4800 baud
1001	2400 baud
1010	1200 baud

9.4.26 RX FIFO Read Pointer Register

Note that this register can be written only if SCR[0] = 1.

Figure 9–24. RX FIFO Read Pointer Register (UART_RDPTR_URX)

Address (hex): Base = 0xFFFF:1000, Offset = 0x005C



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–7 **Reserved**. Read as zeros.

Bits 6–0 **RX_READ_PTR.** Read pointer of RX FIFO**.**

9.4.27 RX FIFO Write Pointer Register

Note that this register can be written only if SCR[0] = 1.

Figure 9–25. RX FIFO Write Pointer Register (UART_WRPTR_URX)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0060

31–7
Reserved
RW–0

6–0
RX_WRITE_PTR
RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–7 **Reserved**. Read as zeros.

Bits 6–0 **RX_WRITE_PTR.** Write pointer of RX FIFO.

9.4.28 TX FIFO Read Pointer Register

Note that this register can be written only if SCR[0] = 1.

Figure 9–26. TX FIFO Read Pointer Register (UART_RDPTR_UTX)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0064

31–7
Reserved
RW–0

6–0
TX_READ_PTR
RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–7 **Reserved**. Read as zeros.

Bits 6–0 **TX_READ_PTR.** Read pointer of TX FIFO.

9.4.29 TX FIFO Write Pointer Register

Note that this register can be written only if SCR[0] = 1.

Figure 9–27. TX FIFO Write Pointer Register (UART_WRPTR_UTX)

Address (hex): Base = 0xFFFF:1000, Offset = 0x0068

31–7
Reserved
RW–0

6–0
TX_WRITE_PTR
RW–0

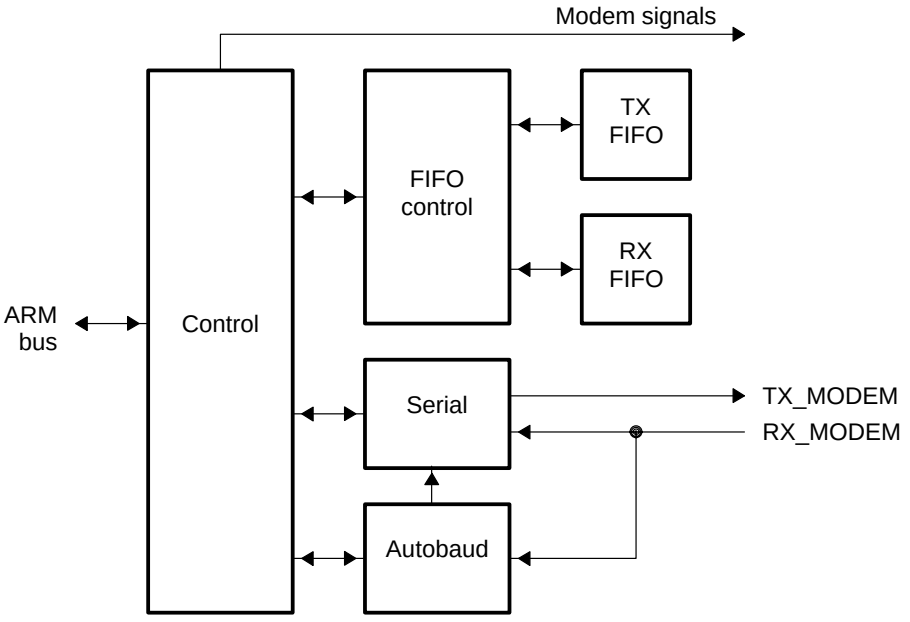
Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–7 **Reserved**. Read as zeros.

Bits 6–0 **TX_WRITE_PTR.** Write pointer of TX FIFO.

9.5 Functional Block Diagram

Figure 9–28. UART Modem Interface Block Diagram



9.6 Functional Descriptions

9.6.1 Trigger Levels

The UART provides independent programmable trigger levels for both receiver and transmitter interrupt generation. Trigger levels can be set through the UART_FCR and UART_TLR registers. These trigger levels are activated upon the enabling of the RHR and THR interrupts in the UART_IER register.

9.6.2 Interrupts

The UART generates interrupts to the ARM through the interrupt line, $\overline{\text{IRQ}}$. All interrupts can be enabled or disabled through the interrupt enable register (UART_IER). The interrupt status can be checked at any time by reading the interrupt status register (UART_ISR).

There are seven different types of interrupts, prioritized to six levels.

When an interrupt is generated, the UART_ISR register indicates that an interrupt is pending by bringing UART_ISR[0] to zero. The interrupt status register also provides the type of interrupt through UART_ISR[5:1]. Table 9–3 lists the priority, ISR coding, type, source, and clear method for all interrupts.

Note that the XOFF interrupt is reset only by receiving an XON character and by reading the UART_ISR register for the special character detection.

Table 9–3. UART Modem Interrupts

ISR[5–0]	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Method
000001	None	None	None	None
000110	1	Receiver line status	Overrun, framing error, parity error, or break detection occurred in characters in the RX FIFO	Framing, parity, or break: read all erroneous characters from the RX FIFO. Overrun: read LSR
001100	2	RX timeout	Stale data in RX FIFO	Read RHR
000100	2	RHR interrupt	RX FIFO not empty if FIFO disabled RX FIFO above trigger level in other cases	Read RHR until interrupt condition disappears
000010	3	THR interrupt	TX FIFO empty if FIFO disabled TX FIFO below trigger level in other cases	Write to THR until interrupt condition disappears
000000	4	Modem status	MSR[3:0] not null	Read MSR
010000	5	XOFF interrupt	Receive XOFF character / special character	Receive XON character / read ISR
100000	6	CTS, RTS	RTS or CTS pin change state from active to inactive	Read ISR

9.6.3 Break and Time-Out Conditions

9.6.3.1 Time-Out

A time-out occurs if the receiver line RX_Modem has been high for a time equivalent to four times the programmed word length plus 12 bits, and if the RHR register is not read during this time.

9.6.3.2 Break Condition

A break is detected if the RX line has been low for a time equivalent to the word length plus one bit if there is no parity, or plus two bits in other cases.

9.6.4 Hardware Flow Control

Hardware flow control is composed of auto-CTS and auto-RTS, which are enabled or disabled independently by programming EFR[7:6].

In auto-CTS mode, the CTS pin must be active (low level) before the module can transmit data. If CTS becomes inactive during a transmission, the UART will finish sending the current word and will wait for CTS to become active again to empty its TX FIFO.

In auto-RTS mode, the RTS pin is active (low level) until the stop trigger is reached by the RX FIFO, and then is inactive until the start trigger is reached. This means that RTS is activated when there is enough room in the FIFO to receive data and is deactivated when the RX FIFO is sufficiently full.

It is important to note that the stop trigger must be set at a higher level than the start trigger if the hardware flow control is used.

The stop trigger is represented by TCR[3:0] and the start trigger by TCR[7:4].

9.6.5 Software Flow Control

The principle of software flow control is the same as that for hardware flow control: Transmissions are authorized by the receiving UART as a function of its RX FIFO pointer position from the trigger levels set.

9.6.5.1 RX

When software flow control is activated, the UART compares incoming data with the programmed XOFF1 or XOFF2 character (in some cases, it can be XOFF1 followed by XOFF2). When the correct XOFF characters are received, transmission is halted after completing transmission of the current character. XOFF detection also sets ISR[4], if enabled.

To resume transmission, an XON character must be received, detected, and compared to the programmed XON character (in some cases, it can be XON2 following XON1). When the correct XON is detected, the transmission is resumed and XOFF interrupt disappears.

The software flow control characters are not stored in RX FIFO once they are recognized. However, if there is one error (framing, parity, break) in them, those characters will be treated as normal data and will not be recognized as XOFF/XON characters.

If EFR[1:0] = 11, which means that XOFF1/XOFF2 (XON1/XON2) must be received sequentially, the UART only recognizes the completely received couples as the control characters. If the first received character is recognized as XOFF1 but the subsequent character is not XOFF2, then both the XOFF1 and the subsequent characters are written into RX FIFO as normal characters. The similar case applies for XON1 and XON2 characters.

9.6.5.2 TX

An XOFF1 or XOFF2 character is transmitted when the RX FIFO has passed the stop-trigger level programmed in TCR [3:0] (XOFF1 followed by XOFF2 in some cases).

An XON1 or XON2 character is transmitted when the RX FIFO reaches the start-trigger level programmed in TCR[7:4] (XON1 followed by XON2 in some cases).

9.6.5.3 General Features

It is important to note that if software flow control is disabled after sending XOFF, the XON will be automatically sent to enable normal transmission to proceed. The same occurs if the software flow combination changes after having sent XOFF: the original XON will be send. For example, if the couple XOFF1/XON1 is selected for software control, and if the user changes to XOFF2/XON2 after having sent XOFF1, XON1 will be sent when the RX FIFO reaches the start trigger, in spite of the XOFF2/XON2 selection.

The software flow control character transmission follows the same protocol as data, which means that the word size, parity type, and number of stop bits defined for data will be used for control transmission. If the control characters are defined for eight bits and the transmission is, for example, defined for six bits, there is no difficulty because the six less significant bits of the control character will be sent by the transmitter UART and will be compared to the six less significant bits of the control character stored in the receiving UART.

Note that software flow control and hardware flow control should not be enabled simultaneously.

9.6.6 Autobauding Mode

The UART_MDR [2:0] register needs to be set to 010 in order to activate the autobauding mode.

In this mode, the UART has to recognize the transmission speed, the parity type, the number of stop bits, and the word size on two receiving characters, ***AT*** or ***at***.

During this sequence's reception, break condition, framing, and parity errors are not detected. If the character or the parameters are not identified, nothing is stored in the RX FIFO and the UART will not receive anything.

The good detection in autobauding is based on the value of the UART_DIV_115K register.

Until parameters are detected, UASR register bits 3 to 0 are set to zero, which means that no speed is identified. Receiving parameters must not be taken into consideration before these bits change.

Whatever appends the ***AT*** or ***at*** chain is not stored in RX FIFO. Non-null values in the UASR register indicate that the UART is aware of the parameters that have been detected and is ready to receive and transmit data.

In Autobauding mode, the transmitting parameters are determined by autobauding. The LCR register has no specific function except in the case of break transmission.

You should be aware that 5-bit and 6-bit character transmissions cannot be recognized when in autobauding mode.

The ***At*** and ***at*** sequences are not recognized.

Each error forces the system to a new ***A*** or ***a*** acquisition.

After parameters are recognized in the reception state, the entire received sequence is pushed into the RX FIFO, including the final <CR>.

<CR> (13 in ASCII) is the final character for the Autobauding mode. Once it is stored in RX FIFO, the transmission parameters are cleared, UASR goes to null value, and the UART resumes speed/number-of-bits/parity/number-of-stop bits acquisition.

Chapter 10

Serial Port Interface (SPI)

This chapter describes the operation of the serial port interface (SPI) and includes register definitions and timing diagrams.

The SPI is associated with the ARM™ microcontroller unit (MCU) of the dual-core (MCU + DSP) VC547x device.

Topic **Page**

10.1 SPI Main Features

The serial interface is a bidirectional 3-line interface dedicated to the transfer of data to and from external devices offering a 3-line serial interface.

This interface is specified to be compatible with the UMA1018M Philips, the FUJITSU MB15F02 and the SIEMENS PMB2306T synthesizers, and the TI VEGA and AD7015 GSM analog-to-digital and digital-to-analog (A/D–D/A) devices.

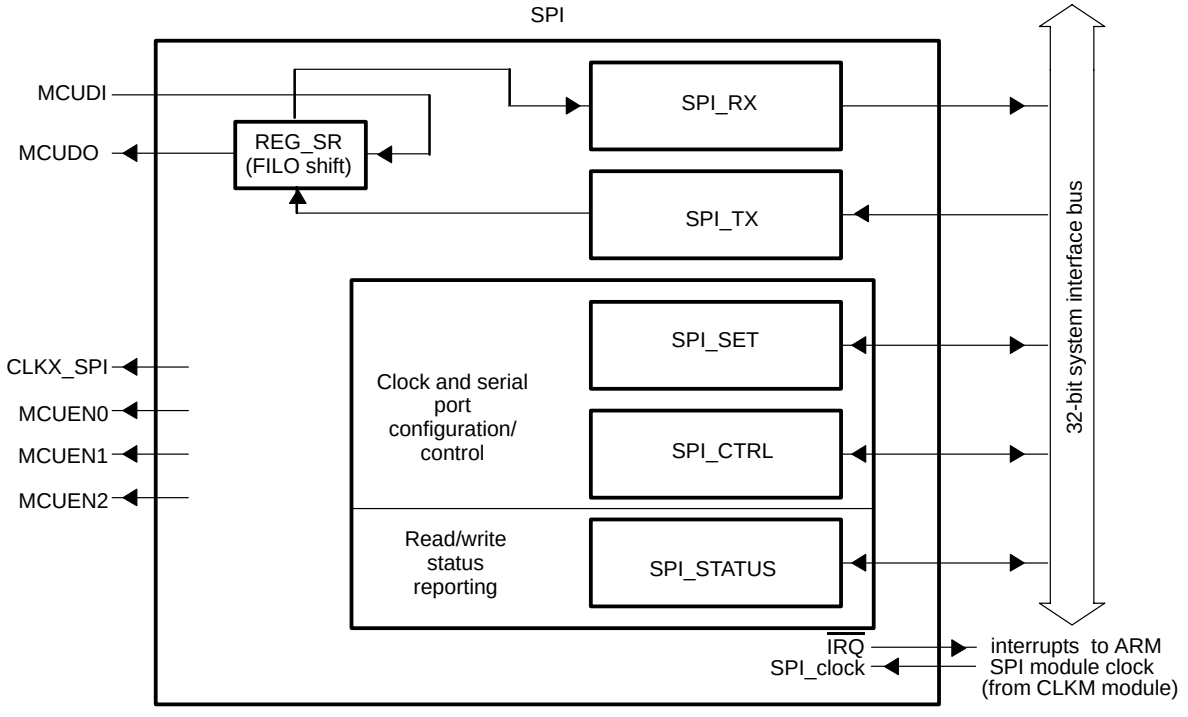
This serial port is based on a looped shift-register, thus allowing both transmit (PISO) and receive (SIPO) modes.

The serial port is fully controlled by the VC547x System Interface bus (data write, data read, and activation of serialization operations).

10.2 SPI General Description

The SPI consists of a data path and a control path connected to external devices by six pins as shown in Figure 10–1.

Figure 10–1. SPI Block Diagram



Data/Clock Communications

Data is communicated to devices interfacing the SPI via the data output (MCUDO) pin for transmit and the data input (MCUDI) pin for receive. Clocking information is communicated via the CLKX_SPI. Up to three external devices can be connected to the SPI interface. MCUENi (i = 0, 1, or 2) can be used to individually enable the connecting devices. The ARM core communicates with the SPI through 32-bit-wide control registers accessible via the system interface bus.

ARM CPU Read/Write Operations

The ARM CPU reads the received data from the SPI Receive register (SPI_RX) and writes the data to be transmitted to the SPI Transmit register (SPI_TX). Data written to SPI_TX is shifted out to the MCUDO pin via the internal shift register (REG_SR).

On the reception side, data received on the MCUDI pin is also shifted into the internal REG_SR register and then copied to SPI_RX. This internal shift register is based on a loop (FILO principle). Therefore, even though only one shift register is available for both transmit and receive, a read process can be done simultaneously with a write process. Also, the concurrent write process can be dummied if no data has to be transmitted. See Section 10.5.2 for more information on the receive protocol.

Control Configuration

The remaining registers that are accessible to the ARM CPU configure the control mechanism of the SPI and also do transmission/reception status reporting. Notification of important events/interrupts to the ARM is done through the interrupt line, $\overline{\text{IRQ}}$, which is handled by the Interrupt Handler module.

10.3 SPI I/O Description

Table 10–1. ARM Serial Port Interface Signals

Signal	Direction	Description
MCUDI	IN	SPI serial data input
MCUDO	OUT	SPI serial data output
CLKX_SPI	OUT	SPI serial clock
MCUEN0	OUT	Enables device 0
MCUEN1	OUT	Enables device 1
MCUEN2	OUT	Enables device 2

10.4 SPI Registers

The serial port offers input and output registers for, respectively, the loading of data to serialize (TRANSMIT) and the reading of data parallelized (RECEIVE).

Base address (hex): FFFF:2000

Register width: 32 bits

Table 10–2. SPI Registers

Register	Description	Offset Address
REG_SR†	Internal Shift Register	– – –
SPI_SET	SPI Setup Register	00h
SPI_CTRL	SPI Control Register	04h
SPI_STATUS	SPI Status Register	08h
SPI_TX	SPI Transmit Register	0Ch
SPI_RX	SPI Receive Register	10h

† REG_SR is an internal register that is not directly accessible by the ARM core.

10.4.1 SPI Setup Register

SPI_SET is dedicated to the configuration of the serial port.

Figure 10–2. SPI Setup Register (SPI_SET)

Address (hex): Base = FFFF:2000, Offset = 0x0000

31–14	13	12	11	10	9	8
Reserved	L2	L1	L0	P2	P1	P0
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

7	6	5	4	3	2–0
C2	C1	C0	MSK1	MSK0	PTV
RW–0	RW–0	RW–0	RW–1	RW–1	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–14 **Reserved**. Read as zeros.

Bit 13 **L2.** Format of enable signal MCUEN2 for device 2.

0 Level trigger
1 Edge trigger

Bit 12 **L1.** Format of enable signal MCUEN1 for device 1.

0 Level trigger
1 Edge trigger

Bit 11 **L0.** Format of enable signal MCUEN0 for device 0.

0 Level trigger
1 Edge trigger

Bit 10 **P2.** Format of enable signal MCUEN2 for device 2.

0 Negative level
1 Positive level

Bit 9 **P1.** Format of enable signal MCUEN1 for device 1.

0 Negative level
1 Positive level

Bit 8 **P0.** Format of enable signal MCUEN0 for device 0.

0 Negative level
1 Positive level

Bit 7 **C2.** Active edge of the clock for device 2.

0 Falling edge
1 Rising edge

Bit 6 **C1.** Active edge of the clock for device 1.

0 Falling edge
1 Rising edge

Bit 5 **C0.** Active edge of the clock for device 0.

0	Falling edge
1	Rising edge

Bit 4 **MSK1.** Disable interrupt for Read/Write cycle.

0	Interrupt active
1	Interrupt disabled

Bit 3 **MSK0.** Disable interrupt for write cycle.

0	Interrupt active
1	Interrupt disabled

Bits 2–0 **PTV.** Prescale clock divisor.

000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	Reserved

10.4.2 SPI Control Register

SPI_CTRL is dedicated to the activation of the serial port and starts the operation of the interface as soon as one of its two bits is set. It defines:

- ❏ WRITE activation of the serial port (transmit only)
- ❏ READ activation of the serial port (simultaneously receive and transmit)
- ❏ Number of bits to transfer (in the range of 1 to 32)
- ❏ External device address (between 0 and 2)

Figure 10–3. SPI Control Register (SPI_CTRL)

Address (hex): Base = FFFF:2000, Offset = 0x0004

31–9	8–7	6–2	1	0
Reserved	AD	NB	WR	RD
RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W= Write access; value following dash (–) = value after reset

Bits 31–9 **Reserved**. Read as zeros.

Bits 8–7 **AD.** Index of the addressed device (3-device maximum)

00	Device number 0
01	Device number 1
10	Device number 2
11	Reserved

Bits 6–2 **NB.** Word size (from 1 to 32 bits) transmission of NB+1 bits.

00000	1-bit transmit
00001	2-bit transmit
⇓	⇓
11111	32-bit transmit

Bit 1 **WR.** Write process activation

0	Not active
1	Active

Bit 0 **RD.** Read and write process activation (toggle at 1)

0	Not active
1	Active

10.4.3 SPI Status Register

This register shows the status of data transmission and reception.

Figure 10–4. SPI Status Register (SPI_STATUS)

Address (hex): Base = FFFF:2000, Offset = 0x0008

31–2	1	0
Reserved	WE	RE
R–0	R–0	R–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–2 **Reserved**. Read as zeros.

Bit 1 **WE.** Write end.

0	The serialization is not completed
1	The serialization is completed

Bit 0 **RE.** Read end.

0	The loading of the SPI Receive register is not completed
1	The loading of the SPI Receive register is completed

10.4.4 SPI Transmit Register

Data to transmit is loaded in the register SPI_TX, which is accessible by the interface bus in read or write.

Figure 10–5. SPI Transmit Register (SPI_TX)

Address (hex): Base = FFFF:2000, Offset = 0x000C

31–0
DATA_TX
RW–0

Note: R = Read access; W= Write access; value following dash (–) = value after reset

Bits 31–0 **DATA_TX.** Data to transmit.

10.4.5 SPI Receive Register

Received data is accessible from the interface bus.

Figure 10–6. SPI Receive Register (SPI_RX)

Address (hex): Base = FFFF:2000, Offset = 0x0010

31–0
DATA_RX
R–0

Note: R = Read access; W= Write access; value following dash (–) = value after reset

Bits 31–0 **DATA_RX.** Received data.

10.5 SPI Protocol Description

The serial port is configured using the SPI_SET register.

After the loading of the transmit register, the serialization and parallelization processes are started by setting to one either the RD bit or the WR bit of the control register.

A read process is always simultaneous with a write process because the internal shift register is based on a loop (filo principle); however, the concurrent write process can be dummied if no data has to be transmitted.

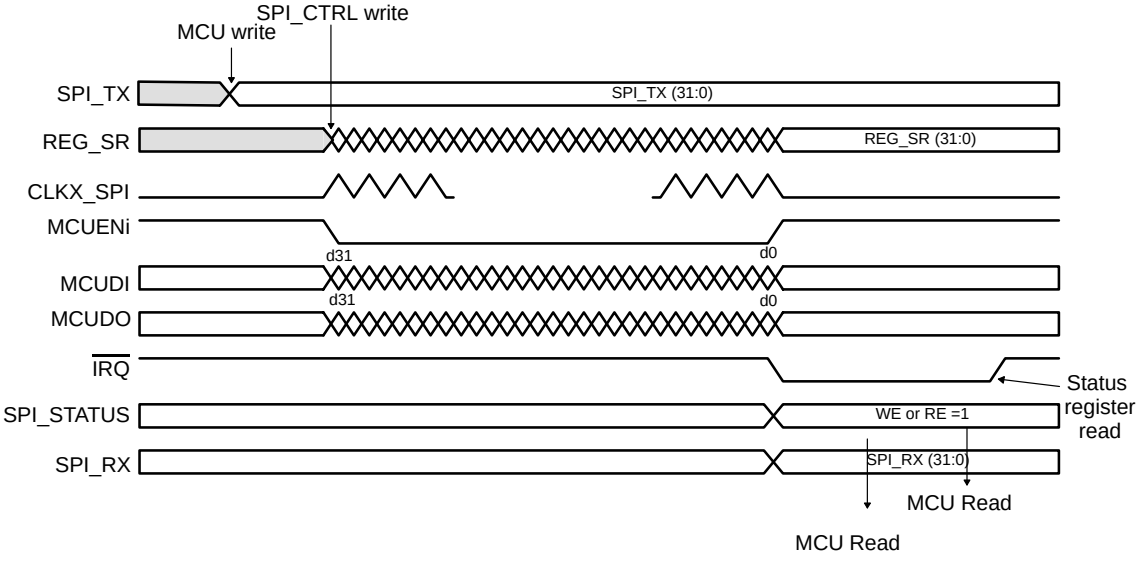
The external transfer of a word starts as soon as the transmit clock is generated. The transmitted data word is shifted out on the rising or falling edge of the transmission clock (CLKX_SPI); whereas, the received data word is shifted in on the falling or rising edge of CLKX_SPI (complementary edge). The loading of the word in the external device is then validated on either:

- The deactivation of the enable signal (MCUENi) (rising or falling edge) or
- The high level/low level of the enable signal (MCUENi) (load enable of the receiver latch), depending on the value of the P (negative/positive level) and L (level/edge trigger) parameters in the SPI_SET register

An interrupt can be generated (depending on the setup register) at the end of the write or read/write cycle.

The IRQ request is cleared when the MCU reads the status register.

Figure 10–7. Protocol Waveforms



10.5.1 Transmit Protocol

The serialization protocol of a word is as follows:

1) Initialization: MCU writes in Setup register => write in **SPI_SET**

2) Loading of the Data Word: MCU writes word => write in **SPI_TX**

3) Start of the Serialization: MCU set WR bit in Control register
   => write in **SPI_CTRL**

4) Upon WR Setting: Then **REG_SR <– SPI_TX**
   activate serialization on the rising edge of SPI_clock (internal clock)

5) Select external device on the rising (C = 1) or falling edge (C = 0) of SPI_clock.
   For example, select external device serial PORTi => **MCUENi** = 0 if SPI_CTRL [8:7] = 00.

6) Data Serialization:
   activate **CLKX_SPI**
   Loop i from 0 to n
   MCUDO=REG_SR(n – i)
   End Loop
   deactivate **CLKX_SPI**

7) Deselect external device serial PORTi => **MCUENi** =1 (if P = 0 in SPI_SET)

8) Interrupt Generation

10.5.2 Receive Protocol

The parallelization protocol of a word is as follows:

1) Initialization: MCU writes in Setup register => write in **SPI_SET**

2) Loading of the dummy data word
   MCU writes word => write in **SPI_TX**

3) Start of the concurrent parallelization/serialization: MCU set RD bit in control register => write in **SPI_CTRL**

4) Upon RD setting then **REG_SR <– REG_TX** activate parallelization/serialization on the rising edge of SPI_clock (internal clock)

5) Select external device on the rising (C=1) or falling edge (C=0) of SPI_clock.
   For example, select external device serial PORTi => **MCUENi** = 0 if SPI_CTRL [8:7] = 00.

6) Data Serialization:
   activate **CLKX_SPI**
   Loop i from 0 to n
   MCUDO = REG_SR(n – i)
   REG_SR(i) = MCUDI
   End Loop
   deactivate **CLKX_SPI**

7) Deselect external device serial PORTi => **MCUENi** = 1 (if P = 0 in SPI_SET)

8) Interrupt Generation

9) Reading of received data word by the MCU
   MCU reads total part => read **SPI_RX**

Notes: 1) It is important to note that the MSB bits of a word are first transmitted. If an 8-bit word must be transmitted, write them into SPI_TX[31:24].

Notes: 2) *By design, the read process requires two data transfers because the external device needs to receive first the address of the data to be read before being able to transmit this data to the MCU.*
However, in case of consecutive read processes or read and write processes, the processes can be piped, thus saving the dummy transfer which can be replaced by an effective transfer.

10.5.3 Transmission Mode Waveforms

The serial interface is active as soon as the transmit clock is activated.

The transmitted data word is shifted out on the rising or falling edge of the transmission clock (CLKX_SPI); whereas, the received data word is shifted in on the falling or rising edge of CLKX_SPI (complementary edge).

On the deactivation of the enable signal:

- ❏ The transmitted word is stored in the external device
- ❏ The received word is stored in the receive register of the serial port (SPI_RX) if the read mode has been selected (bit 0 of register SPI_CTRL)

Figure 10–8. Case C=0, DO on Rising Edge, DI on Falling Edge, P=0, L=0

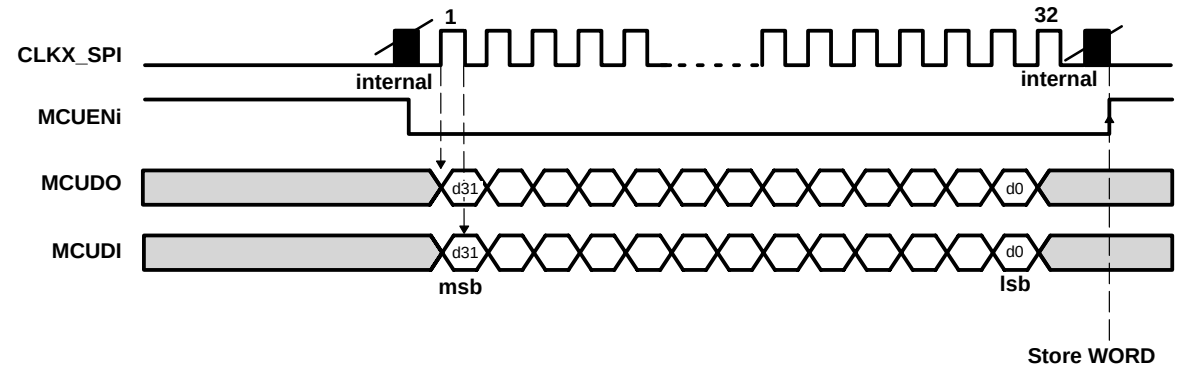


Figure 10–9. Case C=1, DO on Falling Edge, DI on Rising Edge, P=1, L=0

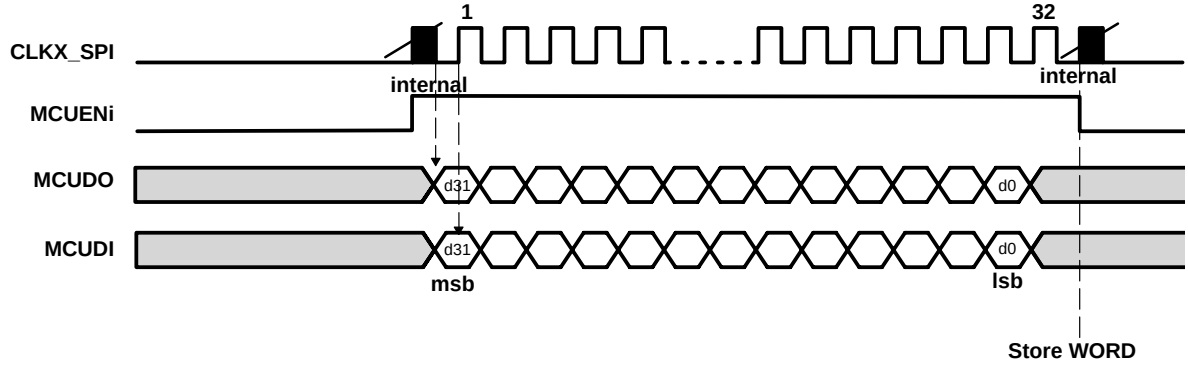
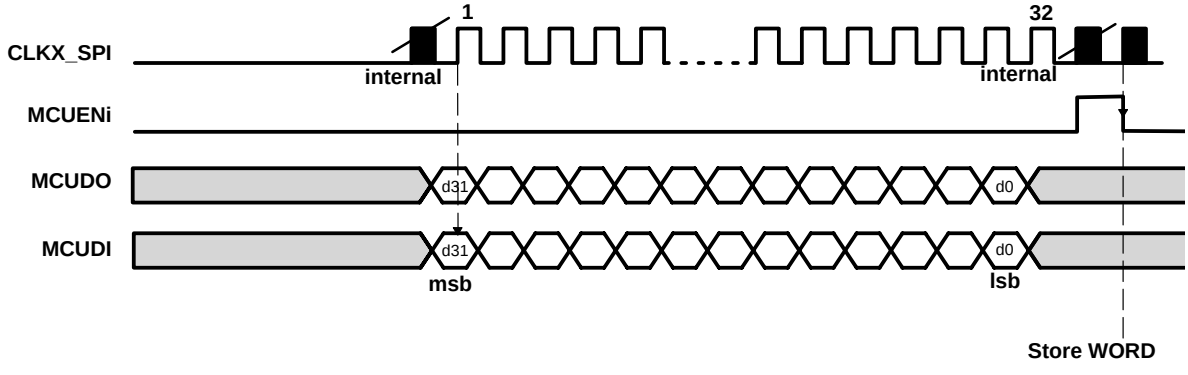


Figure 10–10. Case C=0, DO= on Falling Edge, DI on Rising Edge, P=1, L=1



Chapter 11

Master I2C Interface

This chapter provides a general description of the I2C Interface, shows the applicable registers, describes the I2C bus protocol and Master I2C interface resets, and discusses interrupt, FIFO, and clock management.

The Master I2C Interface is associated with the ARM™ microcontroller unit (MCU) of the dual-core (MCU + DSP) TMS320VC547x device.

Topic **Page**

11.1 Master I2C Interface Module General Description

The Master Inter-Integrated Circuit (I2C) Interface Module provides an interface between the VC547x system-interface bus and the I2C bus. The VC547x system bus, through the Master I2C Interface Module, can control the external peripheral devices on the I2C bus.

11.1.1 Overview

Fundamentally, the Master I2C Interface Module is a parallel-to-serial and serial-to-parallel converter. The parallel data received from the VC547x bus has to be converted to a suitable serial form for external peripheral devices on the I2C bus. Also, the serial data received from the I2C bus has to be converted to a suitable parallel form for the VC547x bus.

The Master I2C Interface supports write cycles and simple and combined read cycles.

11.1.2 Main Features

The Master I2C Interface Module supports the I2C Master-Only mode with:

- ❏ 7-bit address DEVICE
- ❏ 8-bit subaddress
- ❏ Master write to slave receiver in single- or multiple-mode (data loop) 16-byte-deep transmit FIFO
- ❏ Master simple read to slave receiver
- ❏ Read combined cycle
- ❏ 3-bit programmable prescale internal clock divider and 7-bit programmable SCL clock divider to supports a wide clock frequency range of module input clock signals. The I2C SCL clock frequencies are:
  - ■ I2C Standard mode: 100 kHz
  - ■ I2C Fast mode: 400 kHz
- ❏ 3-bit programmable spike filter to provide I2C bus input signal-noise filtering ability
- ❏ Error handling capability during I2C bus access

The Master I2C Interface Module does not support:

- ❏ I2C bus 10-bit addressing
- ❏ I2C bus CBUS compatibility
- ❏ Multimaster I2C
- ❏ I2C bus high-speed mode: 3.4 MHz
- ❏ I2C bus master reads slave immediately after first byte (sequential read mode)

11.1.3 Special Considerations

There are two problems which have been identified in the I2C logic. These problems are listed below.

- ❏ Required FIFO reset prior to any I2C operation
- ❏ Inability to generate “Not Acknowledge”

11.1.3.1 FIFO Reset

For any I2C serial activity (this includes reads as well as writes), a reset must be performed to the FIFO prior to the start of each operation. There appears to be a logic bug, which leaves the FIFO in an unhealthy state following any I2C usage. The following workaround appears to have good results.

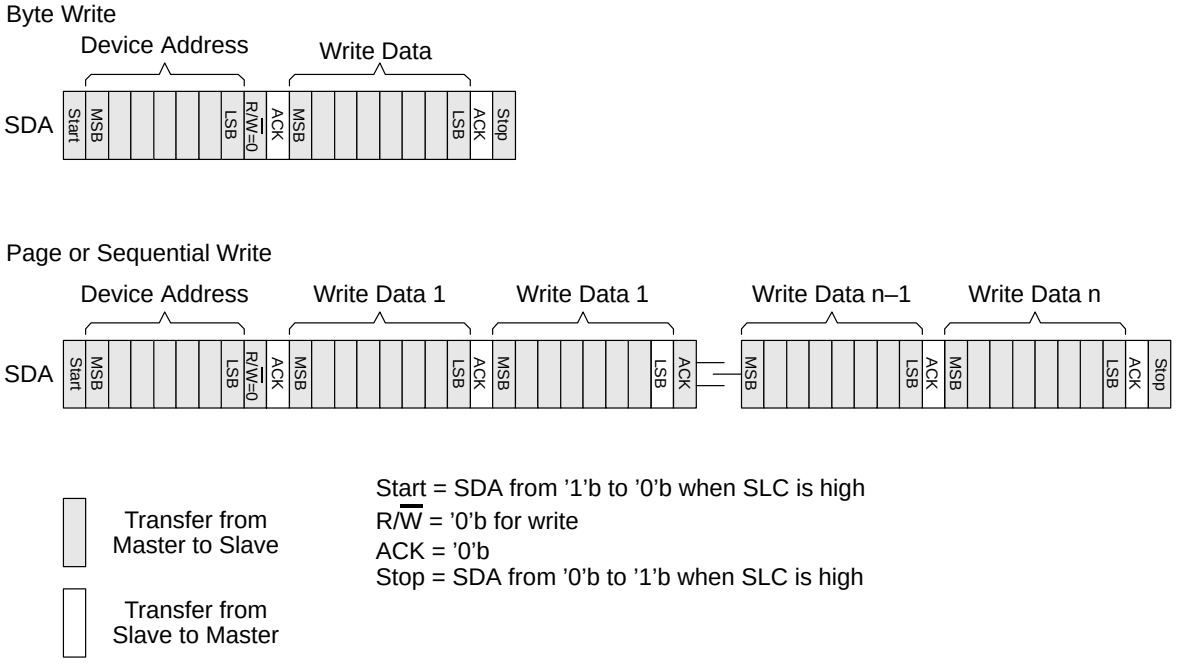
1) Reset the FIFO (CMD_REG [FFFF:1810] bit 0, SOFT_RESET = 1).
2) Delay while the reset works through the FIFO.
3) Remove the FIFO reset (CMD_REG [FFFF:1810] bit 0, SOFT_RESET = 0).
4) Delay once again.
5) Reload the FIFO size (CONF_FIFO_REG [FFFF:1814], FIFO_SIZE).
6) Load the FIFO (DATA_WRITE_REG [FFFF:1808], DATA_WRITE).
7) Assuming the following registers are prepared:
    - ■ Type of FIFO operation (CMD_REG [FFFF:1810] bit 2, RW or bit 3, COMB_READ).
    - ■ The Device Address (DEVICE_REG [FFFF:1800], DEVICE).
    - ■ The Register Address (ADDRESS_REG [FFFF:1804], ADDRESS).

8) Then the I2C FIFO operation can be started (CMD_REG [FFFF:1810] bit 1, START).

11.1.3.2 Not Acknowledge

The Not Acknowledge bit transferred from the Master to the Slave is a critical element of all Read operations, but is not used in any Write operation. This is illustrated in Figure 11–1.

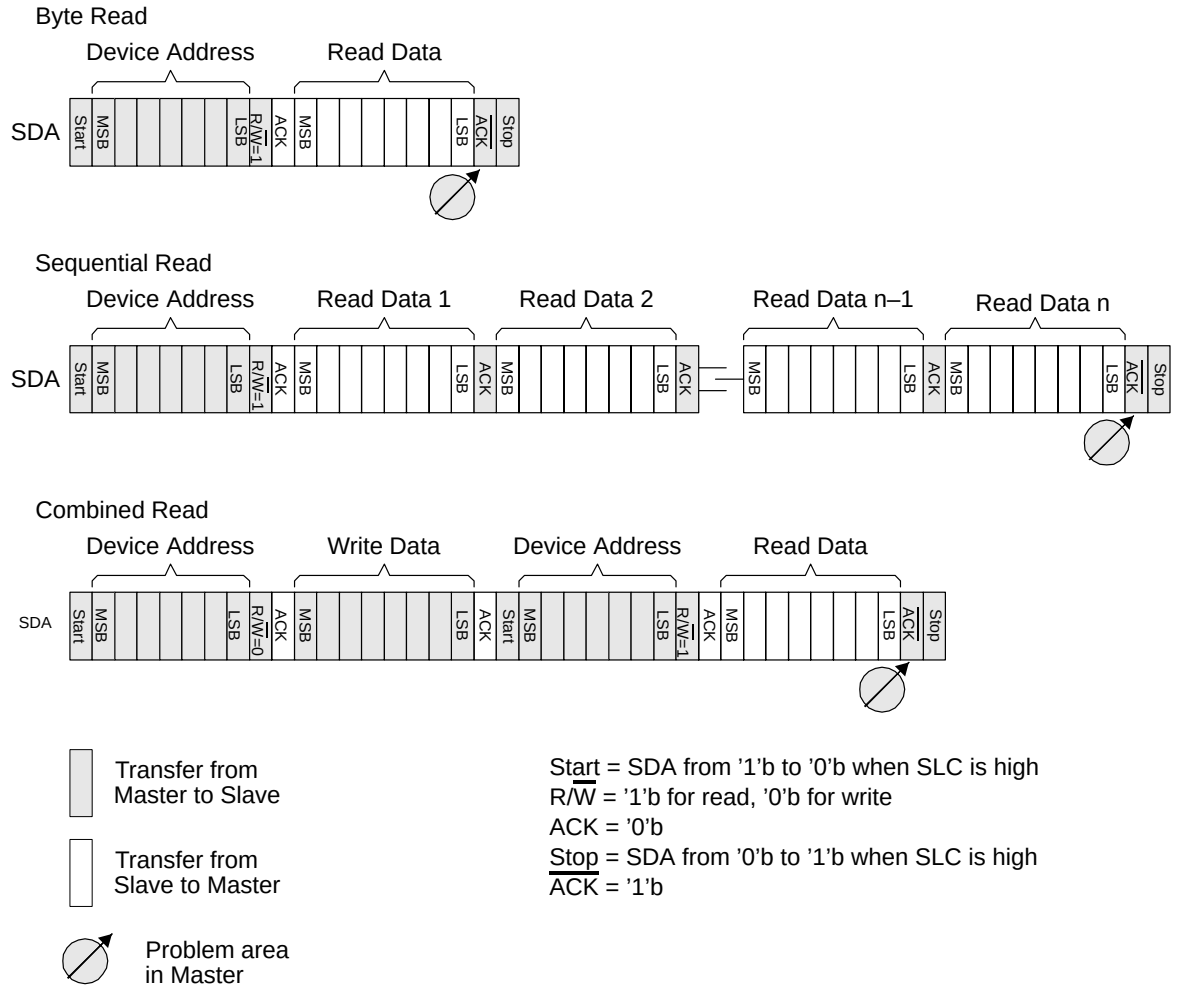
Figure 11–1. I2C Write Operation



For all Read operations, the Not Acknowledge bit transferred from the Master to the Slave clearly defines the successful capture of the last byte transferred by the Slave to the Master. It also informs the Slave that the Master cannot accept anymore read data.

Unfortunately, the VC547x sends an Acknowledge in place of the Not Acknowledge bit. This is highly inappropriate behavior, since the Slave device believes that a Sequential Read operation is now in progress. As the VC547x device does not have a read FIFO, it is incapable of accepting anymore than one byte during any given Read operation. Worse yet, many Slave devices require the Not Acknowledge signal to appropriately complete a Read operation and without it, they may simply hang.

Figure 11–2. I2C Read Operation



11.1.4 Standard I2C Bus Protocol

Note: Support for Master I2C Interface Module

This section describes the standard I2C bus protocol and lists the common terminology used (Table 11–1). Not all features described or listed in this section are supported on the Master I2C Interface Module of the VC547x.

I2C Bus

The I2C bus is a multimaster synchronous serial bus that includes an arbitration procedure to prevent data corruption if more than one master simultaneously initiates a data transfer. The bit-transfer rate can be up to 100 kbit/s in the standard mode. (An extension to the I2C bus specification defines the fast mode that allows a bit-transfer rate of up to 400 kbit/s.)

Each device connected to the bus is recognized by a unique 7-bit address (a 10-bit address is defined in the extended I2C bus specification) and can operate as either a transmitter or receiver. In addition, devices can also be considered as masters or slaves when performing data transfers.

Bus Lines

Only two bidirectional bus lines are required: a serial data line (SDA) and a serial clock line (SCL). The output stage of the devices that are connected to the I2C bus must have an open drain to perform the wired-AND function. The arbitration procedure developed relies on the wired-AND connection of all I2C interfaces to the I2C bus. The clock signals during arbitration are a synchronized combination of the clocks generated by the masters using the wired-AND connection to the SCL line. The master devices always generate the clock signals on the I2C bus, and each master provides its own clock signal. The clock signal from the master onto the I2C bus can only be altered when it is stretched by a slow-slave device holding down the clock line, or by another master when arbitration occurs.

The data on the SDA line must be stable during the high period of the clock. The high or low state of the data line can only change when the clock signal on the SCL line is low. Data is transferred with the most significant bit (MSB) first. Every data character transmitted must be eight bits long, but the number of bytes-per-transfer is unrestricted. Each byte has to be followed by an acknowledge bit. The first byte of the transfer includes the 7-bit address of the slave, and the LSB is the data direction bit (R/W). An LSB R/W to 0 indicates a transmit from the master to the slave and, if set to 1, indicates a transmit to the master from the slave.

All initial transmissions include seven address bits and one R/W bit, with the exception of the general call address and the start byte procedure.

Table 11–1 provides definitions of I2C terminology and explanations of events occurring on the I2C bus.

I2C Bus Terminology

Table 11–1. I2C Bus Terminology

Term	Description
Transmitter	The device that sends the data to the bus.
Receiver	The device that receives the data from the bus.
Master	The device that initiates a transfer, generates clock signals, and terminates a transfer.
Slave	The device addressed by a master.
Multimaster†	More than one master can attempt to control the bus at the same time without corrupting the message.
Arbitration†	Procedure to ensure that if more than one master simultaneously tries to control the bus, only one is allowed to do so and the message is not corrupted (the first master to produce a one when the other produces a zero will lose the arbitration).
Synchronization	Procedure to synchronize the clock signals of two or more devices; The low period of the SCL clock line is determined by the device with the longest clock-low period. The high period of the SCL line is determined by the device with the shortest clock-high period.
Start condition	A high-to-low transition on the SDA line while SCL is high defines a start condition (S). The start condition is always generated by the master. The bus is considered to be busy after the START condition.
Stop condition	A low-to-high transition on the SDA line while SCL is high defines a stop condition (P). The stop condition is always generated by the master. The bus is considered to be free after the STOP condition.
Acknowledge bit	An addressed receiver is obliged to generate an acknowledge or a non-acknowledge bit after a byte has been received (during the 9th SCL clock cycle). – The receiver pulls down the SDA line during the acknowledge clock pulse so that it remains stable low during the high period of this clock pulse. This indicates an acknowledge, and the transfer can then continue. – The receiver leaves the SDA line high during the acknowledge clock pulse to indicate a non-acknowledge. The master can then generate a stop condition to abort the transfer.

Table 11–1. I^2C Bus Terminology (Continued)

Term	Description
General call address†	The general call address is for addressing every device connected to the I^2C bus. It consists of a "00000000" address byte sent after the start condition.
Start Byte	The Start-Byte Procedure consists of : – a start condition (S) – a start byte (00000001) – an acknowledge clock pulse – a restart condition (Sr)
10-Bit Addressing†	*Defined in the Extended I^2C bus specification.*

† Features not supported by the Master I^2C Interface Module.

The documentation, *I^2C Bus and How To Use It* (including specifications), is issued by Philips Semiconductors and should be consulted for a complete description of the I^2C bus.

11.2 I/O Description

Table 11–2. I^2C Signals

Signal	I/O	Function
SDA	I/O	I^2C serial data
SCL	I/O	I^2C serial clock

11.3 Register Descriptions

The Master I2C Interface Module has 10 registers for communication between the VC547x bus and the I2C bus.

Base address (hex): FFFF:1800

Bit width: 32 bits

Table 11–3. Master I2C Register Descriptions

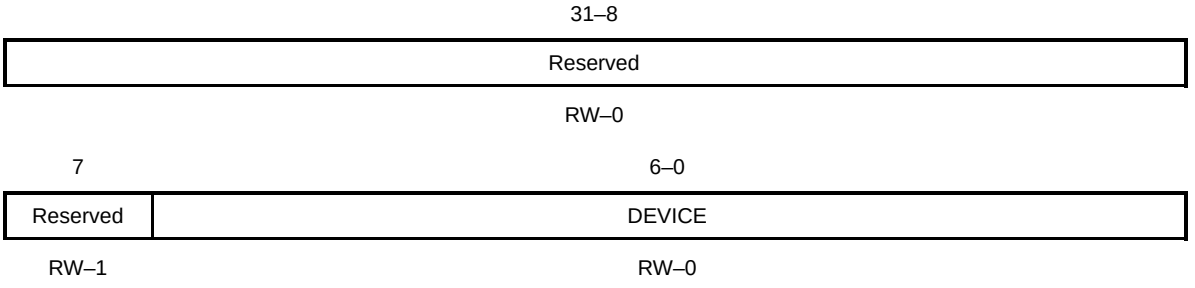
Register	Description	Offset Address
DEVICE_REG	Device Register	00h
ADDRESS_REG	Address Register	04h
DATA_WRITE_REG	Data Write Register	08h
DATA_READ_REG	Data Read Register	0Ch
CMD_REG	Command Register	10h
CONF_FIFO_REG	Configuration FIFO Register	14h
CONF_CLK_REG	Configuration Clock Register	18h
CONF_CLK_REF_REG	Configuration Clock Functional Reference Register	1Ch
STATUS_FIFO_REG	Status FIFO Register	20h
STATUS_ACTIVITY_REG	Status Activity Register	24h

11.3.1 Device Register

At the beginning of an I2C bus read/write access, the Device register is loaded with the 7-bit slave device identification information.

Figure 11–3. Device Register (DEVICE_REG)

Address (hex): Base = FFFF:1800, Offset = 0x0000



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

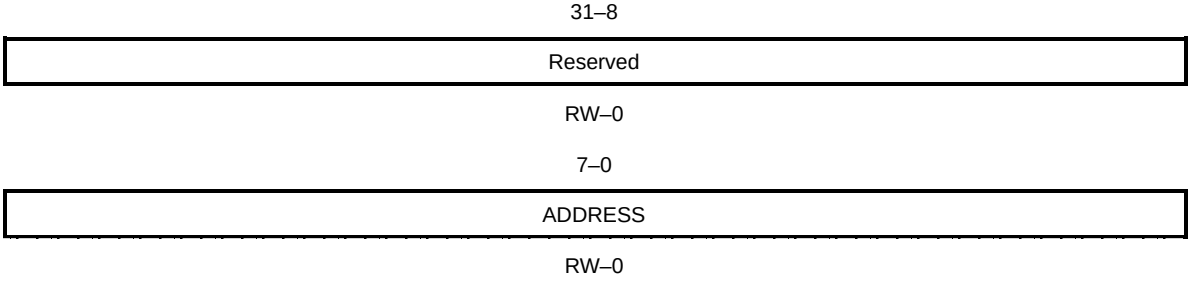
Bit 7 **Reserved**. Read as one.

Bits 6–0 **DEVICE**. Device identification code for I2C bus slave device.

11.3.2 Address Register

Figure 11–4. Address Register (ADDRESS_REG)

Address (hex): Base = FFFF:1800, Offset = 0x0004



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **ADDRESS**. I2C slave device internal register address.

11.3.3 Data Write Register

The Data Write register is the input register of the 16-byte transmit FIFO.

Figure 11–5. Data Write Register (DATA_WRITE_REG)

Address (hex): Base = FFFF:1800, Offset = 0x0008

31–8
Reserved
RW–0
7–0
DATA_WRITE
RW–0[0]

Note: R = Read access; W = Write access; value following dash (–) = value after hard reset; value in brackets [ ] = value after soft reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **DATA_WRITE**. Data to write on I^2C bus.

11.3.4 Data Read Register

Figure 11–6. Data Read Register (DATA_READ_REG)

Address (hex): Base = FFFF:1800, Offset = 0x000C

31–8
Reserved
R–0
7–0
DATA_READ
R–0[0]

Note: R = Read access; value following dash (–) = value after hard reset; value in brackets [ ] = value after soft reset

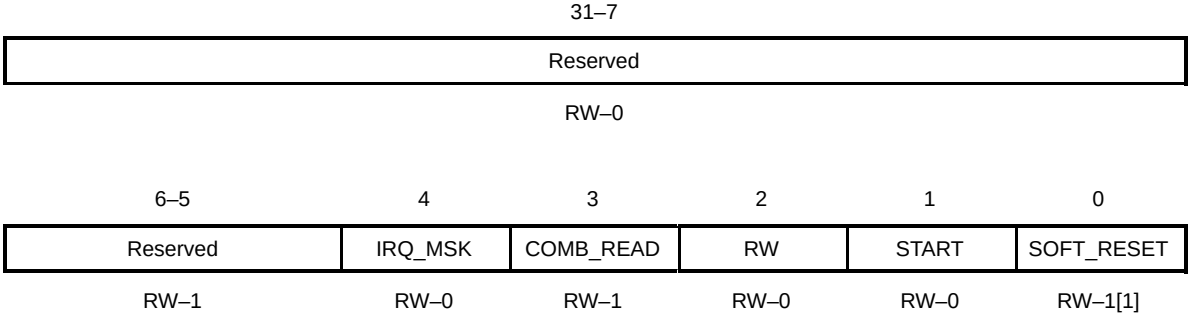
Bits 31–8 **Reserved**. Read as zeros.

Bits 7–0 **DATA_READ**. Data to read on I^2C bus.

11.3.5 Command Register

Figure 11–7. Command Register (CMD_REG)

Address (hex): Base = FFFF:1800, Offset = 0x0010



Note: R = Read access; W = Write access; value following dash (–) = value after hard reset; value in brackets [ ] = value after soft reset

Bits 31–7 **Reserved**. Read as zeros.

Bits 6–5 **Reserved**. Read as ones.

Bit 4 **IRQ_MSK.** Interrupt masking bit.

0 Interrupt request is disabled
1 Interrupt request is enabled

Bit 3 **COMB_READ.** Simple or combined read access.

0 A master read immediately, if RW = 1
1 A combined read access, if RW = 1

Bit 2 **RW.** Read Not Write Bit.

0 I^2C bus write access
1 I^2C bus read access

Bit 1 **START.** Start the I^2C transmission (toggle bit).

Note: the START toggle bit is activated when writing a 1. This bit does not need to be released to 0. Writing a 0 means no action.

Bit 0 **SOFT_RESET.** Reset the FIFO.

0 No reset soft

1 Reset soft

11.3.6 Configuration FIFO Register

Figure 11–8. Configuration FIFO Register (CONF_FIFO_REG)

Address (hex): Base = FFFF:1800, Offset = 0x0014

31–8
Reserved
RW–0

7–4	3–0
Reserved	FIFO_SIZE
R–1	RW–1

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Reads return zeros.

Bits 7–4 **Reserved**. Reads return ones.

Bits 3–0 **FIFO_SIZE**. Size of the FIFO (16 max) to generate the FIFO_FULL.

When FIFO_SIZE = 000, read 1 value in the FIFO.
When FIFO_SIZE = n, read n+1 value in the FIFO.

11.3.7 Configuration Clock Register

Figure 11–9. Configuration Clock Register (CONF_CLK_REG)

Address (hex): Base = FFFF:1800, Offset = 0x0018

31–8
Reserved
RW–0

7–6	5–3	2–0
Reserved	SPK_F	PTV
RW–1	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–6 **Reserved**. Read as ones.

Bits 5–3 **SPK_F.** Spike filter factor.

000	No filtering
001	Checks signal stability for 2 master clock
010	Checks signal stability for 3 master clock
011	Checks signal stability for 4 master clock
100	Checks signal stability for 5 master clock
101	Checks signal stability for 6 master clock
110	Checks signal stability for 7 master clock
111	Checks signal stability for 8 master clock

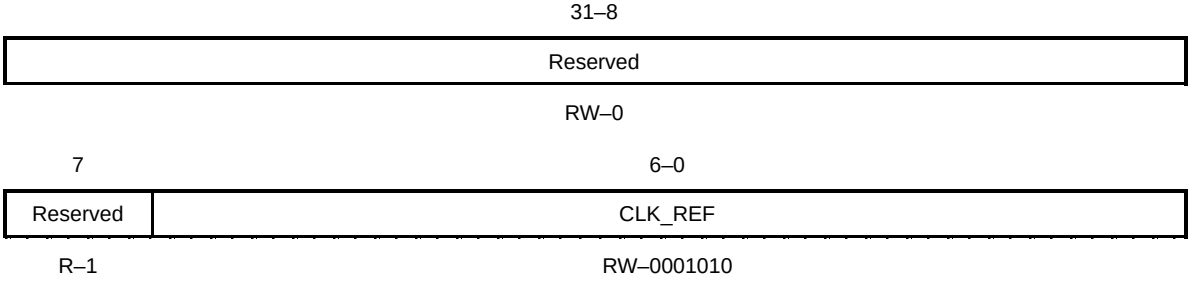
Bits 2–0 **PTV.** Prescale clock divider factor.

000	Divisor_1 = 2
001	Divisor_1 = 4
010	Divisor_1 = 8
011	Divisor_1 = 16
100	Divisor_1 = 32
101	Divisor_1 = 64
110	Divisor_1 = 128
111	Divisor_1 = 256

11.3.8 Configuration Clock Functional Reference Register

Figure 11–10. Configuration Clock Functional Reference Register (CONF_CLK_REF_REG)

Address (hex): Base = FFFF:1800, Offset = 0x001C



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**. Reads return zeros.

Bit 7 **Reserved**. Reads return one.

Bits 6–0 **CLK_REF**. Functional clock reference.

0000001 Divisor_2 = 1
0000010 Divisor_2 = 2
0000011 Divisor_2 = 3
...
1111110 Divisor_2 = 126
1111111 Divisor_2 = 127

CLK_FUNC_REF = I2C_clk/(divisor_1*[divisor_2+1])
SCL_OUT = CLK_FUNC_REF / 2

11.3.9 Status FIFO Register

Figure 11–11. Status FIFO Register (STATUS_FIFO_REG)

Address (hex): Base = FFFF:1800, Offset = 0x0020

31–8
Reserved
R–0

7–6	5–2	1	0
Reserved	READ_CPT	FIFO_EMPTY	FIFO_FULL
R–1	R–0[0]	R–1[1]	R–0[0]

Note: R = Read access; value following dash (–) = value after hard reset; value in brackets [ ] = value after soft reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–6 **Reserved**. Read as ones.

Bits 5–2 **READ_CPT**. Indicates the FIFO count value.

Bit 1 **FIFO_EMPTY**. Indicates if the FIFO is empty.

0 FIFO not empty
1 FIFO empty

Bit 0 **FIFO_FULL**. Indicates if the FIFO is full.

0 FIFO not full
1 FIFO full

11.3.10 Status Activity Register

Figure 11–12. Status Activity Register (STATUS_ACTIVITY_REG)

Address (hex): Base = FFFF:1800, Offset = 0x0024

31–8
Reserved
R–0

7–4	3	2	1	0
Reserved	INTERRUPT	IDLE	ERROR_DEVICE	ERROR_DATA
R–1	R–0[0]	R–0[0]	R–0[0]	R–0[0]

Note: R = Read access; value following dash (–) = value after hard reset; value in brackets [ ] = value after soft reset

Bits 31–8 **Reserved**. Read as zeros.

Bits 7–4 **Reserved**. Read as ones.

Bit 3 **INTERRUPT**. Interrupt bit.

0 I^2C bus transfer is not completed or *I^2C Module* is in Idle mode
1 I^2C bus transfer is completed or aborted on non-acknowledge

Bit 2 **IDLE**. Indicates whether the I^2C bus transfer is completed or not.

Bit 1 **ERROR_DEVICE**. Indicates an error on device transmission.

0 No error is detected during the device address transmit
1 Error is detected during the device transmit

Bit 0 **ERROR_DATA**. Indicates an error on the sub-address or data transmission.

0 No error is detected during I^2C bus access
1 Error is detected during I^2C bus access

11.4 FIFO Management

A FIFO is used in the Master I2C Interface to buffer transmit data. This buffer has a maximum depth of 16 bytes.

The register CONF_FIFO_REG defines the number (FIFO_SIZE) of data to transmit on the I2C bus, and consequently, the number of data to be loaded in the FIFO.

When FIFO_SIZE = N,

- If configured in write-access mode, the interface reads n+1 data in the FIFO
- N+1 data can be loaded in the FIFO before FIFO is reported as full (FIFO_FULL = 1)

The FIFO includes two counters, one for write data and one for read data. These two counters are independent, and therefore, allow a transmission start without reloading the FIFO content.

Figure 11–13. FIFO Management State

11.5 Master I2C Interface Resets

The Master I2C Interface Module provides for both hardware and software resets.

Hardware Reset

The Master I2C Interface Module *hardware* reset is derived from the VC547x reset.

Software Reset

The Master *software* reset is activated by the SOFT_RESET bit in the command register (CMD_REG). This reset is limited to the FIFO counter (wr_cpt and rd_cpt), FIFO state machine, and the contents in the FIFO.

In the VC547x device, the I2C module can also be put in reset mode by controlling the module input nreset_i signal, which is an output of the clock and reset management module (see Chapter 5, *Clock Management Module*).

It is strongly recommended that you do not use the soft reset bit of the I2C Module. Instead, you should use the corresponding reset bit inside the VC547x clock module.

11.6 Clock Management

There is no clock management provided inside the I2C Interface Module; however, it is possible in the VC547x device to shut down the I2C clock (see Chapter 5, *Clock Management Module).*

11.7 Interrupt Management

An interrupt is generated after every read or write I2C bus transfer and when an error occurs during a transmission. An interrupt line can be masked with the dedicated programmable control bit (IRQ_MSK) in the command register (CMD_REG).

Chapter 12

Ethernet Interface Module (EIM)

This chapter describes the Ethernet interface module (EIM), its registers, interfaces, memory, and operation.

The EIM is associated with the ARM™ microcontroller unit (MCU) of the dual-core (MCU + DSP) TMS320VC5471 device.

Topic **Page**

12.1 EIM Overview

The Ethernet Interface Module (EIM) provides a straightforward and effective method of integrating an IEEE802.3/Ethernet MAC functionality onto a processor I/O subsystem. EIM filters packets for the ARM host system.The main features of the EIM are:

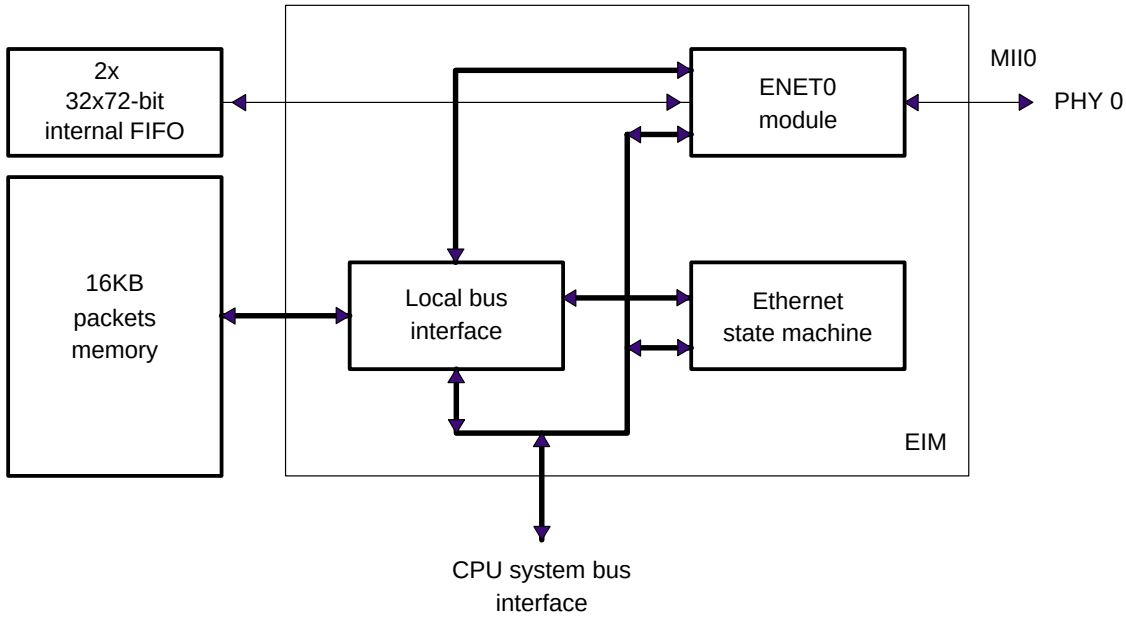
- ENET module implementing 10/100 Mbit/s Ethernet media access controller (MAC) operation
  - Fully compliant with IEEE Std 802.3 for 10/100 Mbit/s
  - Full- and half-duplex mode
  - Media-independent interface (MII)
  - Flow control support in full-duplex mode
- Hardwired State Machine for Ethernet packet transfers without CPU processor intervention
- Packet memory for temporary storage of TX and RX packets plus control information stored in a *descriptor ring* software structure
- 16K bytes of internal packet-memory-mapped in ARM memory. The packet memory is accessible in 8/16/32 bits in one cycle (zero wait state)
- local packet memory bus allowing internal packet transfer without ARM CPU bus occupation

12.1.1 General Description

The Ethernet interface module (EIM) consists of four primary blocks: ENET module, the Ethernet state machine (ESM), the Packet Memory, and the local bus interface.

Figure 12–1 shows a block diagram of the EIM.

Figure 12–1. EIM Block Diagram



Packet Channels

The EIM Module handles packet routing between two packet channels that are mapped as follows:

- ❏ Ethernet MAC port connected to PHY interface (called ENET0 port)
- ❏ One virtual port connected to the LCC layer of the ARM software (called CPU port)

Configuring the CPU Port

The CPU port is configurable through a matching destination register, a matching multicast address mask register, and two logical address registers.

Each port can be independently enabled/disabled. Routing rules are defined for each port.

Routing Rules for Packets Coming From the ENET

The routing rule for packets coming from the ENET is:

- ❏ Packets matching one of the enabled CPU matching rules (unicast, broadcast, logical) are sent to the CPU port

Routing Rules for Packets Coming From the CPU

The routing rule for packets coming from the CPU is:

- ❏ All packets are sent to the ENET0 port (ENET0 is designated for connection to the LAN)

12.2 Ethernet Interface Signals

Table 12–1. Ethernet Interface Signals (ENET0 MII Interface Signals)

Signal	Direction	Function
COL0	IN	MII collision: COLL from PHY. Indicates a collision has been detected on media.
CRS0	IN	MII carrier sense: CRS from PHY. Indicates that a carrier is present on the media.
TCLK0	IN	MII transmit clock: TCLK from PHY. TX reference clock generated by the PHY.
RCLK0	IN	MII receive clock: RCLK from PHY. RX reference clock generated by the PHY
RXD0[3:0]	IN	MII RX data: RXD from PHY. Nibble-wide data interface presenting received data from PHY to the MAC synchronous to RCLK. In SNI mode, bit 0 carries the serial data stream.
RXDV0	IN	MII RX data valid: RXDV from PHY. Indicates that the data on RXD is valid.
RXER0	IN	MII RX error: RXER from PHY. Indicates an error detected in the RX data stream by the PHY.
TXEN0	OUT	MII TX enable: TXEN to PHY. Indicates to PHY that the MAC is transmitting data.
TXER0	OUT	MII TX error: TXER to PHY. Indicates to PHY that a coding error is sent. This is asserted synchronously with TCLK
TXD0[3:0]	OUT	MII TX data: TXD to PHY. When MII is in nibble mode, data is sent one nibble at a time. When in SNI mode, data is sent serially on TXD[0].

12.3 ENET Functional Description

The Ethernet core (ENET) provides a straightforward and effective method of integrating an IEEE 802.3/Ethernet port onto an ARM subsystem. An integrated DMA provides advanced buffer management allowing multiple packets to be moved without ARM processor intervention.

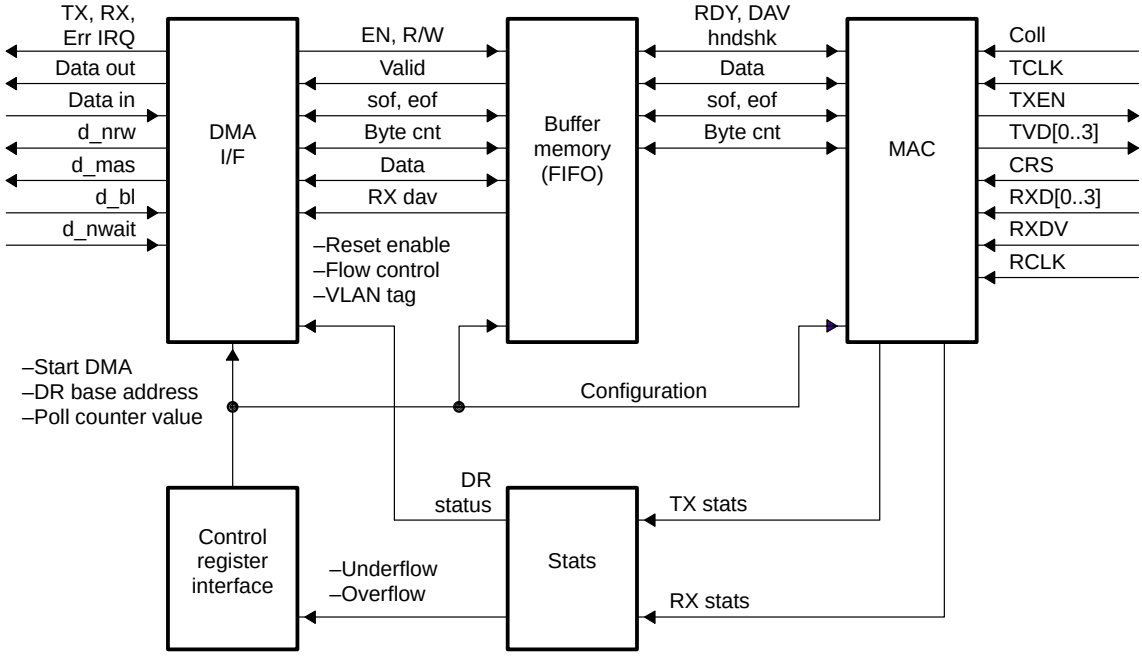
12.3.1 ENET Overview

- 32-bit ARM bus interface (big-endian)
- 32-bit intelligent bus master DMA interface for ring buffer management
- Ring buffer chaining
- RX and TX status reporting
- 10/100 MBps bit rate
- Programmable full-/half-duplex MII/SNI
- Internal loopback
- Full-duplex flow control
- Address filtering
- 256-byte TX FIFO divided into four 64-byte partitions
- 256-byte RX FIFO divided into four 64-byte partitions

Important Note: the integrated DMA allows the transfer of packets based on a software descriptors ring structure between a host memory and the internal ENET FIFO. In a typical standalone implementation, the host memory is the generic CPU memory. This is not the case with EIM. The host memory is a dedicated local memory and the DMA has no visibility over other ARM CPU memory areas. This dedicated local memory is a 16K-byte RAM called the Packet Memory. See Section 12.8, *EIM Packet RAM Structure*, on page 12-59 for more information.

The ENET core consists of five primary blocks: the Buffer Memory Unit (FIFO function and flow control), the Media Access Controller (MAC), Control Register Interface, DMA controller, and Statistics block.

Figure 12–2. ENET Module Functional Block Diagram



12.3.2 Buffer Memory Unit (FIFO)

The Buffer Memory Unit is provided to decouple system memory access timing from packet timing. Without the buffer, interrupts and data transfers would have to be treated at relatively high priority to prevent underrun or overrun conditions. In addition to buffering, the Buffer Memory Unit takes care of network retries, runs, and flow control within the ENET module. The Buffer Memory Unit is a collection of state machines which implement the FIFO pointer maintenance, full and empty flag maintenance, and DMA control signals. The memory is divided into RX and TX buffer space implemented as a single port SRAM for area efficiency. Access is provided through a simple arbitration scheme with ARM host given highest priority while guaranteeing RX/TX MAC access at least once every 64 bit times. Each buffer is organized as 32x72 bit words. Each access provides 8 bytes of data and 1 byte of flag information.

Figure 12–3. Buffer Organization

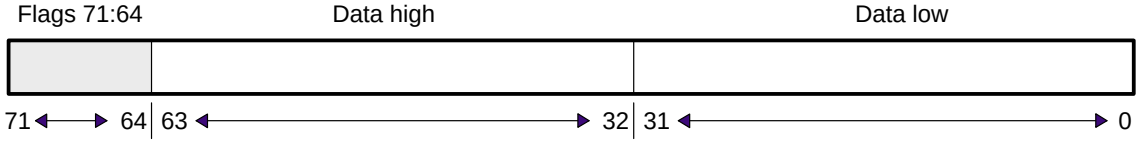
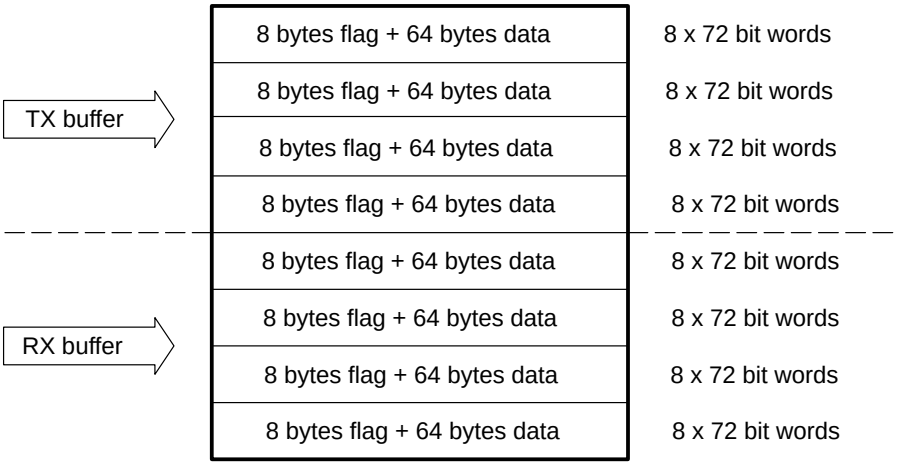


Figure 12–4. Single-Port RAM



The buffers act as a circular queue with all accesses occurring on word boundaries. The DMA controller makes the Buffer Memory access and allocation scheme transparent to the user of the ENET module.

The flag information is generated by the MAC interfaces and passed with the data through the buffer memory, providing useful status and control information. The flag field is used internally by the Buffer Memory Unit to communicate with the MAC. Start-of-Frame, End-of-Frame, and byte count are primary examples of the type of information contained in the flag field.

12.3.3 DMA Controller

The DMA controller is responsible for moving packet data between the system memory (16K-byte packet memory) and the Buffer Memory Unit (FIFO). The DMA memory interface protocol and timing are the same as the ARM MCU memory interface timing.

The movement of packet data is controlled by two very specific data structures in packet memory called Descriptor Rings. In general, the DR data structures provide flexible packet management allowing variable memory buffer sizes and locations. The DR also provides a means to synchronize shared ownership of data buffers in memory between DMA and ARM host. Each ring consists of a variable number of descriptors that can be chained to handle long packets in multiple data buffer areas. The location of the descriptor rings in

packet memory is programmable using ENET configuration registers. One descriptor ring is for transmit operations and the other is for receive operations. The bit format of the RX and TX descriptors are very similar. The formats of each are described in more detail in Section 12.4, *EIM Descriptors Structure*, on page 12-22.

12.3.4 Control Registers Interface

The Control Registers Interface is a generic 32-bit CPU compatible interface that is used primarily to configure the ENET registers prior to processing packets.

The list of ENET registers is available in Section 12.5.

12.3.5 Media Access Controller (MAC)

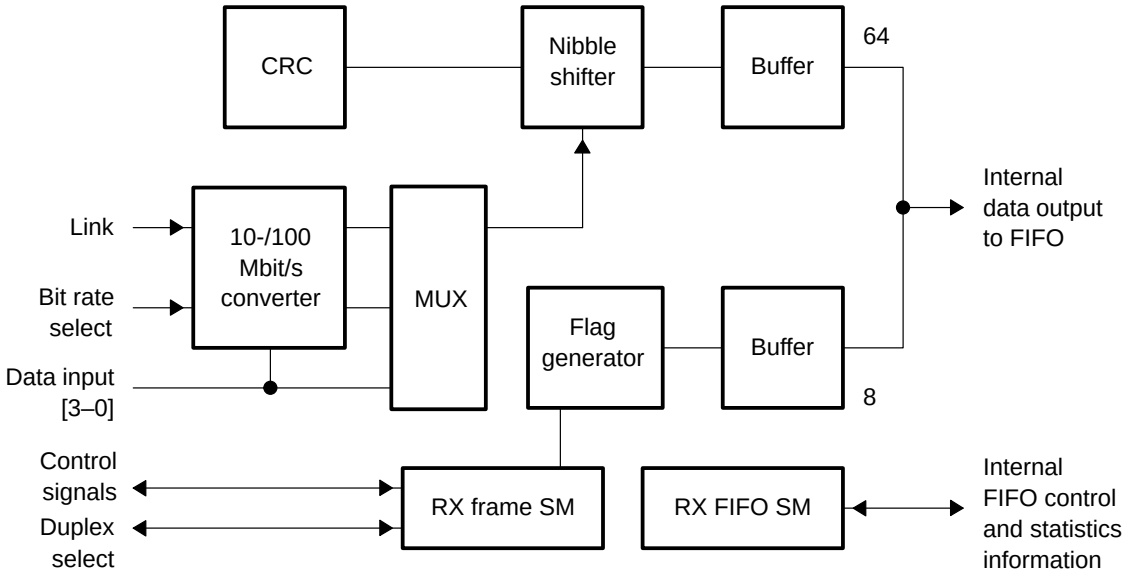
The MAC is a full-featured, configurable, 802.3-compliant controller. Half-/Full-duplex configuration is controlled by setting the *Duplex* bit in the Mode register. See Section 12.7 for definitions of the ENET peripheral registers. When the *Duplex* bit is set to one, the MAC operates in full-duplex mode. While operating in the full-duplex mode, transmit is independent of carrier sense (MCRS) status. When the *Duplex* bit is set to zero, the MAC operates in the half-duplex mode. While operating in half-duplex mode, the MII receiver is disabled when the MTXEN signal is active.

12.3.5.1 Data Reception

The MAC Receive Block implements all the required IEEE 802.3 standards for either 10-Mbit/s operation or 100-Mbit/s operation. When operating at 10 Mbit/s, the 10/100-Mbit/s port can operate either in nibble-serial or bit-serial interface mode (as set by the MWIDTH bit of the Mode register). The MAC block implements reception of data from the MII, handles receive path control and identification of errors. The MAC is also capable of operating in 1-Mbit/s and below range in support of applications such as home PNAs, provided that protocol compliance is maintained.

The Internal Memory Interface Block takes the data from the MAC control block and places the frame data, with the associated 8-bit status flag field, in the data buffer for transfer to the Buffer Memory Unit (FIFO).

Figure 12–5. Media Access Controller (MAC) Receive Block Functional Diagram



Preamble and SFD

Preamble & Start of Frame Delimiter (SFD) handling is accomplished after the MAC becomes active upon detection of valid data on the MII by monitoring the receive data valid (**RDV)** and receive data (**RD[3:0])** signals. A valid preamble and a valid 8-bit Start of Frame Delimiter (SFD) are used to establish the start of an incoming packet. Once alignment is established, the preamble and SFD bytes are stripped and not forwarded with the remaining frame. At least one nibble of *1010* of the preamble must be present prior to the SFD of *1011* with the valid **RDV** signal for the MAC to recognize a start of frame. If no preamble is present prior to the SFD, the MAC will not recognize the beginning of an incoming frame. Once the MAC recognized the *1010* of the preamble, it will look for the SFD of *1011*, regardless of the length of the preamble, or go to idle if **RDV** becomes invalid.

CRC Check

Upon packet reception, cyclic redundancy check (CRC) is checked. If the CRC is valid, then the EOF flag value is set to indicate good CRC and the CRC is passed along with the frame to the Buffer Memory FIFO. If the CRC is bad, then the EOF flag value is set to indicate CRC Error and the CRC is passed along with the frame to the Buffer Memory. The Buffer Memory FIFO uses EOF to construct completion status before it writes to Descriptor Word 0.

64-Byte Frame Requirement

The MAC enforces the minimum 64-byte frame requirement upon a receive. Even if the frame was a well formed frame, any frame less than 64 bytes long is discarded from the Buffer Memory one word at a time, and not forwarded to the ARM host. In this event, an error status is reported via the *Status* field (short frame) of Receive Descriptor Word 0, and the received byte count is set to zero. No data from the short frame will be passed to the system's Packet Memory.

Nibble Dribble

Nibble Dribble (non-octet alignment error) is allowed on receive and the data is passed intact. The CRC is aligned on the last valid byte boundary. A maximum of one nibble may be appended to the end of a frame without causing CRC failures.

The MAC is configured to receive the extended frame type that is currently being defined within the IEEE 802.1 working group. Any incoming frame will be truncated to 1536 bytes no matter what the actual length.

DA, SA, and Type Fields

Independent of packet length, the destination address (DA), source address (SA), and Type fields are not modified by the MAC and are always forwarded with the packet. The destination address is examined according to the addressing mode set in Address Mode Enable register. If the DA is accepted, the frame data will be forwarded to the Packet Memory location, pointed to by Descriptor Words 2 and 3. If the DA address field contains an address not enabled by the Address Mode Enable register, the frame will be flushed from Buffer Memory FIFO and no other action will be taken.

IFG

On a receive for both 10-Mbit/s operation and 100-Mbit/s operation, there is no enforcement of Inter-Frame Gap **(IFG).** A frame received that is not spaced with the appropriate IFG interval (96 bit times) is processed and passed intact through the MAC and is reported as normal received packets**.** The minimum IFG is the time to complete the receive cycle and return to idle. The time is dependent on the receive data as follows:

- ❏ If the frame is 64 bytes or less and a collision occurred, then IFG is 1–8 ARM CLK cycles plus 2 RCLK cycles
- ❏ If the frame is greater than 64 bytes and a collision occurred, then IFG is 3 ARM CLK cycles plus 2 RCLK cycles

- For any good frame, IFG is 2 RCLK cycles

  Since ENET can not do full-frame buffering, all frames are passed to Packet Memory following address filtering regardless of error conditions.

Receive Status Reporting

Receive status reporting includes:

- **Non-octet Alignment Error:** This is also called Nibble Dribble. The MAC detects an alignment error when it receives a non-integral number of octets. This condition should result in a CRC error as well. Alignment error is reported in Descriptor Word 0 frame status field. The DMA controller will set the LIF bit and then set ownership of the descriptor entry back to ARM.

- **Overrun (overflow):** An overrun condition can occur when the DMA controller finds that the ownership of the next receive descriptor entry is set to ARM preventing the controller from emptying the Buffer Memory FIFO. Upon the Buffer Memory going full, the MAC will cease to receive data from the MII port and will not attempt to transfer any additional data to the Buffer Memory Module. The MAC does not exert any back pressure to the inbound MII in an attempt to slow down the incoming Frames. There is no flexible allocation of FIFO depth within the Buffer Memory Module.

  In some cases, the overrun status will not be able to be logged into the descriptor ring status since the ARM host has the ownership. The SYS_ERR_IRQ interrupt will be activated when overrun occurs. The ARM host can read the SE_STATUS register to determine the cause of the interrupt.

  During overrun conditions, the DMA controller empties the Buffer Memory when it is able to (when the host relinquishes descriptor ownership, for example).

- **CRC Error:** A CRC error occurs when the CRC field of the Ethernet frame does not match the MAC's computed CRC. The two values are compared at the end of the frame.

  CRC error is reported in the Descriptor Word 0 frame status field. The DMA controller sets the LIF bit and then sets ownership of the descriptor entry back to the ARM.

  Collisions can cause CRC errors. If other bits are set, it is possible to infer when the collision occurred. For example, if the CRC error bit is set in the RX descriptor status with the short frame bit set and zero bytes in the byte count field, then a collision is likely to have occurred in the first 64 bits of the RX packet.

If a normal collision occurs, it will be within the first 64 bytes of the RX packet. The MAC automatically purges any data sent to the Buffer Memory FIFO.

- **Short Frame:** Short frames consist of less than 64 octets (excluding framing bits, but including CRC). The MAC automatically purges short frames from the Buffer Memory FIFO. A Short Frame error is reported in the Descriptor Word 0 frame status field. The DMA controller sets the FIF and LIF bits and then sets ownership of the descriptor entry back to the ARM host. There is no receive data associated with short frames.

  If a receive error occurs within the first 64 bits (8 bytes), then the short frame bit will not be set. The MAC ignores the frame.

- **Late Collision:** Late Collisions are any collisions occurring after the initial 64 bytes have been received. The MAC does not detect the error as a late collision but rather as a CRC, alignment, or long frame error. As such, there is no status bit for late collision. If late collision occurs, it will be reported as one or more of these other errors. Also, the MAC will not back out of the Buffer Memory Module. The data is saved in the Buffer Memory FIFO and then forwarded to system packet memory via the DMA controller with some error status bits set. Purging of the errant frame is the responsibility of the higher level control. The DMA controller sets the LIF bit and then sets ownership of the descriptor entry back to ARM.

- **Long Frame:** A Long Frame is a frame with more than 1518 octets (excluding framing bits but including CRC). A Long Frame error is reported in Descriptor Word 0 frame status field. The DMA controller sets the LIF bit and then sets ownership of the descriptor entry back to the ARM host. Long Frame errors coinciding with a CRC error are called RX Jabber errors. RX Jabber errors are reported by setting both the Long Frame error bit and the CRC error bit.

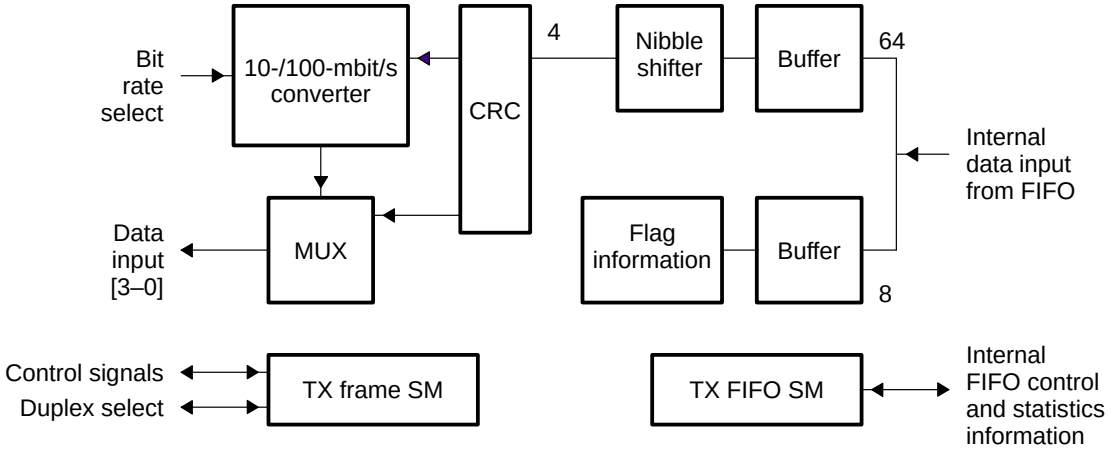
- **Miss:** Miss conditions are based on frame destination address comparisons. It is not an error but rather a status condition indicating that the frame is being accepted only because the snoop addressing mode was enabled and no other addressing modes produced an address match. This bit can only be set if snoop mode is enabled. See Address Mode Enable register bit 3. After the DMA controller finishes sending BYTES of data to the RX descriptors buffer space, the DMA controller sets the FIF bit and then sets ownership of the descriptor entry back to the host. The LIF bit is also set if a full frame was received.

- **Virtual LAN (VLAN) Address Detect:** If the value of the VTYPE field in the received data matches the contents of the VLAN tag register, the VLAN status bit is set in the RX descriptor word.

12.3.5.2 Data Transmission

The MAC transmit block takes the data from the Buffer Memory and serializes it, then implements all the IEEE 802.3 standards requirements for either 10-Mbit/s operation or 100-Mbit/s operation. It implements identification of transmit errors, and transmits the data to the MII.

Figure 12–6. Media Access Controller (MAC) Transmit Block Functional Diagram



If the packet data stored in the descriptor word transmit buffer is less than 64 bytes long, then it is not a valid Ethernet frame. If the packet is less than 64 bytes, the PAD_CRC_EN bit of TX descriptor word 1 must be set to command ENET to create a valid frame. If PAD_CRC_EN is set, packets greater than 59 bytes have only CRC appended. The 4 bytes of CRC make the packet length at least 64 bytes long. Packets less than 60 bytes are padded with data and have a new CRC. For frames that have pad data inserted, the CRC is appended after the pad using the padded data in the calculation of the CRC value. The pad is placed in the data field. The transmit frame is padded by bytes with the value 00(h). In frames longer than 60 bytes, a CRC is appended after the last data in the frame.

CRC can be appended to the end of any transmit frame if PAD_CRC_EN is set. This allows the minimum frame size of 64 bytes (512 bits) for 802.3 Ethernet standard to be guaranteed. Pad insertion is controlled on a frame by frame basis by setting the **PAD_CRC_EN** bit in Descriptor Word 1.

Invalid frames (frames less than 14 bytes or less than 64 bytes with pad_crc_en off) are ignored. They are never read from system packet memory. The descriptor word ownership is set back to the ARM host and the system error TX_FE is set in the system error interrupt register and a system error interrupt occurs.

Transmit Error Handling

Transmit error handling involves:

- **Loss of Carrier Error:** This error indicates that the carrier sense condition was lost during transmission or was never asserted when attempting to transmit a frame. The ENET skips over the rest of the frame (chained descriptors), resetting the ownership bit back to the host, and writes loss of carrier status in the descriptor that has LIF set. ENET then looks for the next new frame to transmit (the next FIF with ownership set to ENET).

  Since loss of carrier only happens due to a jabber indication from the PHY, the MAC does not start checking for LOC error until the end of a slot-time. If the frame is a 64-byte frame, it would appear that checking is happening at CRC time.

  Carrier sense error-checking is disabled in full-duplex mode.

- **Underrun Error:** If the host does not provide data to the MAC after a transmission has started, the transmission is stopped and the MAC generates a transmit Underrun status. The ENET skips over the rest of the frame (chained descriptors), resetting the ownership bit back to host, and writes loss of carrier status in the descriptor that has LIF set. An inverse CRC is currently appended to the outgoing fragment of the abruptly cut frame, and the MAC resumes operation when a new frame is available for transmission. An underrun situation on the MAC is a serious failure; and therefore, the status collection must report this to the control engine. The MAC looks for the next Start Of Frame flag and resynchronizes its transmit process to the next frame.

  The ENET does not attempt to retransmit a frame after an underrun occurs. An Underrun error is reported in the Descriptor Word 0 frame status field as well as the System Error Interrupt Status Register. The DMA controller sets the ownership bit back to the ARM host and activates SYS_ERR_IRQ.

- **CRC Error:** On transmission, CRC is checked. If the generated CRC is found to correspond to the value found in the frame, no errors are reported to the status field of Descriptor Word 0, and the frame finishes transmission. If the generated CRC is different than the CRC in the frame, then an error is reported to the DMA controller and the frame finishes transmission with a CRC field of the inverse of the calculated CRC.

  The ENET does not attempt retransmission following a CRC error since the cause of the CRC error is most likely to be in the frame data itself. The retry limit for TX frames with CRC errors is considered to be 1 (one). Upon detection of a CRC error, the DMA controller skips over any remaining

descriptors in the frame and sets the CRC status bit when LIF is found. Unless there is an error in the packet BYTE count, CRC Error bits are only set in descriptors with LIF=1, due to the nature of CRC.

- **Collision Errors:** The ENET MAC detects several types of transmit collisions. Each are handled in a slightly different manner and are outlined separately below. All error checking is done when in half-duplex mode. Collision detection error checking is disabled in full-duplex mode.

- **Normal Collisions:** In case of a collision occurring before 64 bytes (512 bit times), the MAC generates a 32-bit JAM pattern of all ones. It reads the internal backoff random number generator register value and counts down to zero. (The MAC implements the IEEE Std 802.3 binary exponential backoff algorithm.) Once the counter is decremented to zero, then the MAC tries to retransmit the frame. The frame is discarded after 15 failed transmit attempts (initial attempt plus 14 retries). The frame is purged from the Buffer Memory FIFO and the Exceed Retry Limit bit is set in the descriptor word 0 status field. The DMA controller then sets descriptor ownership back to the ARM host and continues to the next descriptor entry. If the next descriptor does not have the FIF bit set (the discarded frame was part of a long frame consisting of multiple descriptor entries), it sets the ownership bit of the descriptor back to the ARM host and continues to the next descriptor until it finds either FIF set to 1 or the ownership bit set to ARM.

  If 1 to 14 collisions are encountered, the frame will be transmitted successfully (assuming Descriptor Word 1 BYTES field ≥ 64). Upon completion of the frame, the Collision bit of the descriptor word 0 status field is set to 1, but Exceed Retry remains set to 0.

  Automatic retransmission due to collisions is handled completely between the Buffer Memory and the MAC. If the MAC encounters $1 < n < 15$ collisions, it rereads the last 64 bytes of data from the Buffer Memory FIFO. Collision statistics will be saved for later reporting when the transmission of the BYTES of data for the current descriptor completes.

- **Late Collisions:** A late collision is reported if a collision condition occurs more than one slot time (64 bytes or 512 bit times) after the transmit process was initiated. In this event, the frame is dropped. As with all transmit errors except normal collisions, the ENET does not attempt retransmission. As the DMA controller skips over the remainder of the frame's descriptor entries, it sets the ownership bits back to the ARM host until it finds FIF = 1, indicating a new frame. A late collision is reported when the DMA controller closes the descriptor. If the LIF bit is set, the collision bit may be set as well.

- **Excessive Collisions:** If the frame transmission encounters a normal collision on the 15th retry, then the frame transmission is stopped and the frame is purged from the Buffer Memory. The DMA controller skips ahead to the descriptor with the LIF bit set, and then writes status (exceed retry and collision). The ENET returns ownership back to the ARM host and continues to the next descriptor entry.

- **Heartbeat (SQE):** Heartbeat is a self-test of collision detection logic in the PHY. This test is also called Signal Quality Error. If the SQE mode of the PHY is enabled, the PHY creates a simulated collision during the Interframe Gap (IFG) following every transmission. The PHY activates the MCOL signal for a short period. If the MAC does not see MCOL go active within a specific time after the end of the transmission, the MAC flags an SQE error that is reported by the DMA controller to the descriptor word 0 status register before setting the ownership back to the ARM host.

  Heartbeat is only valid in half-duplex mode. The MAC automatically disables the SQE check when in full-duplex mode.

- **Exceed Retry:** Exceed Retry when set to one indicates that the transmission failed. This bit is always set in combination with another status bit explaining the nature of the failure.

- **No Last-in-Frame (LIF) Indicator:** If the frame has no Last-In-Frame indicator and the next frame in the next descriptor entry has no First-In-Frame indicator, the MAC continues to transmit on the wire until a carrier sense error is indicated by the MII. Upon detection of this error, the MAC will read out the data from the transmit Buffer Memory until the next Last-In-Frame or First-In-Frame is found. It should be noted that the DMA controller does not keep track of a running frame byte count. This is done by the MAC while the transmit is proceeding. The DMA controller only keeps track of byte counts on descriptor by descriptor basis.

- **No First-in-Frame (FIF) Indicator:** The DMA controller behaves in a similar manner if the LIF bit was set in the previous descriptor entry and the next descriptor entry does not have the FIF bit set (when ownership bit is set to ENET). The DMA control immediately activates SYS_ERR_IRQ, sets the descriptor ownership back to the ARM host, and proceeds to the next descriptor entry. In this case, however, there is no data for the MAC to transmit.

12.3.6 Statistics Block

The statistics block ensures that the correct status is reported as TX and RX frames are closed. It provides different functions for TX and RX status. But in both cases it monitors the TX and RX status buses coming from the MAC. The statistics block maintains RX queues to match the data queues in the Buffer Memory. Since the Buffer Memory can store up to 4 independent RX packets in its queue, the RX statistics block stores up to 4 sets of status. The DMA controller pops the RX status queue as it reads the data from the receive data buffers. TX status is different in that no queue is necessary, since only one packet will be transmitted at a time. Due to the DMA controller functionality, multiple TX packets are never queued up in Buffer Memory, even if they are short packets.

The statistics block also contains some logic for determining when a packet has completed transmission. The statistics block sends a packet completion signal to the DMA controller, which then reads the completion status and closes the TX descriptor.

12.3.7 Loopback

Loopback can be done non-intrusively while in normal operation. This mode can only be done in half-duplex/serial mode (MWIDTH = 0). It allows transmitted data to be looped back onto the receive channel. This mode can be controlled on a frame-by-frame basis by setting the loopback bit of Descriptor Word 1. The DMA controller automatically marks looped backed packets from packets received normally by setting the loopback bit in the RX descriptor word 1.

12.3.8 Flow Control

Flow Control in compliance with IEEE Std. 802.3 is implemented in the ENET Buffer Memory subsystem. It is used for full-duplex mode only. Loading the TX Flow Pause count register is all it takes to command the ENET to generate flow control frames for transmission. A write to the pause count register triggers a flow control frame to be generated in between outgoing data frames. Responding (TX pause) to the receipt (RX) of a flow control frame requires the FLW_EN bit of the Flow Control register to be set.

12.3.8.1 TX Pause Operation

Upon receipt of a valid Flow Control Frame, the count value of the payload is loaded into the specified channel pause counter. Each channel is handled independently, with separate pause counters and separate pause control circuitry. Each channel may have Flow Control either enabled or disabled independent of the state of other channels. (Note: ENET currently has only one channel).

Any channel that has Flow Control disabled, upon receiving a flow control frame, purges that frame. If Flow Control is enabled, then the frame is acted upon and will be filtered, not passing into memory.

Initiation of the pause period begins immediately following the completion of any current TX frames, or immediately if the TX channel is idle. Once the pause period is complete, it does not repeat until the receipt of another flow control frame.

12.3.8.2 Generation of TX Flow Control Command Frame

Flow Control Command frames are automatically generated in the Buffer Memory module. A Flow Control Command Frame is initiated by writing a count value to the Flow Control Count register.

If the channel is currently transmitting a frame, the Flow Control Frame is held off until the current frame is completed. Any other frames pending in the buffer for the channel are held off until the Flow Control Frame is transmitted completely. Flow Control Commands may be issued regardless of the state of RX_FLOW_EN.

Generation of a proper Ethernet frame is done by applying the destination address (i.e., DA = 01–80–C2–00–00–01), a source address (i.e., SA = 00–00–00–00–00–00), and a type field (i.e., Pause Opcode = 8808). Appended to this stream is the counter value loaded into the Flow Control Count Register, and padded to 60 bytes with Pad = 00(h). A correct CRC is then added to make the frame a total of 64 bytes.

A form of half-duplex flow control can be done by setting the back pressure bit of the mode register. This forces a JAM condition on TXD.

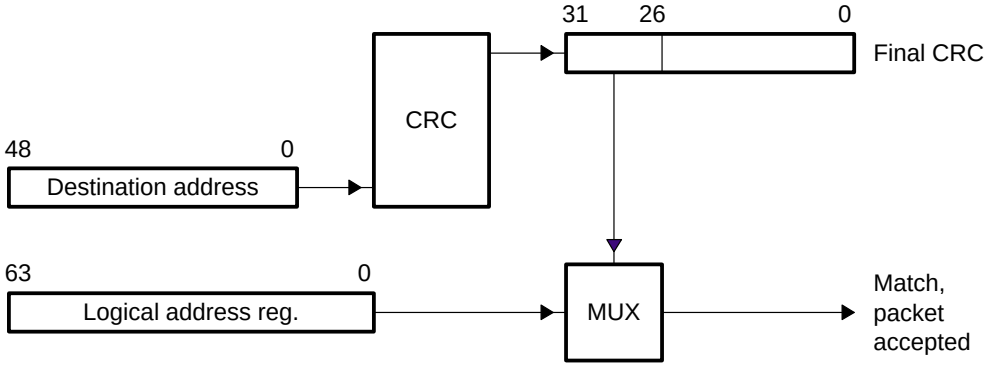
12.3.9 Addressing Modes

This section is relevant to receive operations only. ENET provides four types of addressing to be used as criteria for accepting an incoming receive packet. The first 6 bytes of a new packet after the preamble and start of frame delimiter are defined as destination address. The type of comparison done on the DA depends on the address mode selected with the Address Mode Enable register. Multiple address modes can be enabled at a time. Of course, Promiscuous with anything else is redundant. If all address modes are disabled, the ENET flushes all received packets.

- **Physical Addressing:** The 48-bit DA is compared with the contents of the Destination Physical Address match register. If the two values compare, the packet is accepted and written to the receive descriptor buffer. Physical addressing is also called unicast addressing.

- **Logical Addressing:** The 48-bit DA is sent through the MAC CRC circuit and the high-order 6 bits are used to select one of the 64 bits in the logical address filter. If the selected bit is set to *1*, the packet is accepted and written to the receive descriptor buffer. However, this hardware approach is not a perfect filter. It filters out most, but not all, unwanted addresses, so it is up to the software to do the final checking. Logical addressing is also called multicast addressing. It is a subset of broadcast addressing. A diagram of logical address filter implementation is shown below.

Figure 12–7. Logical Address Filter Implementation



- **Broadcast Addressing:** The 48-bit DA address is checked for all ones. This is the indication of a broadcast address. If all ones, the packet is accepted and written to the receive descriptor buffer.

- **Promiscuous:** The 48-bit DA address does not need to be compared. All addresses are accepted. This mode is also called snoop mode.

12.3.10 ENET Interrupts

Three interrupt request pins are provided on the ENET module. TX_IRQ and RX_IRQ may be masked on a descriptor-by-descriptor basis using the INTRE bit of Descriptor Word 0.

- **TX_IRQ:** Transmit Interrupt is simply a descriptor completion interrupt. It is meant to help the ARM processor gauge the rate that packets are being processed rather than report any type of status. The interrupt is enabled by INTRE of descriptor word 0, and if enabled, goes active after ENET sets the ownership bit of the packet back to ARM.
- **RX_IRQ:** Receive Interrupt behaves the same as TX_IRQ except that it is used for receive operations.
- **SYS_ERR_IRQ:** Occurs when a memory access times out or a packet is missed because the next receive or transmit descriptor is in use. This interrupt also occurs if a descriptor error is detected such as an invalid first-in-frame bit setting. There is no capability to mask this interrupt within the ENET module.

  For use of these interrupts lines in EIM, please refer to the related chapter.

12.3.11 Configuration

Following reset and before any frame can be transmitted or received, the ENET module needs to be configured. Mode registers can be programmed to specify operating modes such as MII width, duplex, random collision backoff seeds, and several others. All registers are readable and writable to allow the processor to check current configuration status and to set/reset individual bits. The Mode register must be used to define duplex, MII width, and bit rate if the default reset values are not the desired configuration. Configuration registers for changing the random back off seed value, VLAN-tagged frame compare value, and flow control pause count must be programmed if the features are to be used.

TX and RX operations will be disabled upon reset. No receiving occurs while the Enable bit in the MODE configuration register is set to zero. Also, the DMA controller does not load the transmit buffer with data to transmit until descriptor ring polling configuration register is enabled with a non-zero value or the transmit descriptor buffer ready command is issued by the ARM host.

12.4 EIM Descriptors Structure

Data transfer between the ENET and Host is controlled by two very specific data structures in a shared memory (accessible by both parts) called Descriptor Rings (DRs). The DR data structures provide flexible packet management allowing variable memory buffer sizes and locations. The DR also provides a means to synchronize shared ownership of data buffers in memory between DMA and host.

Each ring consists of a variable number of descriptors which can be chained to handle long packets in multiple data buffer areas. The location of the descriptors rings in memory is programmable using ENET configuration registers. One descriptor ring is for transmit operations and the other is for receive operations. The bit format of the RX and TX descriptors are very similar. The formats of each are described in more details below. The DMA controller operates on each descriptor ring in strict sequential fashion. If the controller wants to process a descriptor entry to transmit a packet, it checks the ownership bit. If the host owns the entry, the controller periodically polls the entry waiting for ownership to be passed to the ENET module. If ownership is not passed before the buffer empties, an underflow occurs. Likewise, for receive operations, the controller will poll and wait for the next descriptor entry to become available. If ownership is not asserted before the buffer fills, an overflow occurs. The user may set the poll time by loading the 16-bit poll interval register, which decrements once per clock.

The DMA controller does not check for any limit on the chaining length.

12.4.1 TX Descriptor Ring

Each descriptor has four 16-bit words. Depending on the Host system, they can be accessed as two 32-bit words.

Table 12–2. TX Descriptor Word #0

Bit	Name	Host R/W	Function
15	OWN	R/W	Ownership bit 0 = ENET, 1 = HOST After initialization, descriptor has to be owned by ENET.
14	WRAP	W	Descriptor chain wrap 0 = go to the next sequential descriptor entry 1 = go to first descriptor as defined in configuration register
13	FIF	W	First-In-Frame: When set to 1 indicates that this descriptor contains the first bytes of a new frame

Table 12–2. TX Descriptor Word #0 (Continued)

Bit	Name	Host R/W	Function
12	LIF	W	Last-In-Frame: When set to 1 indicates that this descriptor contains the last bytes of a new frame
11–8	RETRY	R	Retry Count status: 0 = initial values 1–15 = valid values
7	INTRE	W	Interrupt enable: 1 = Generate an TX_IRQ interrupt after BYTES of data have been transmitted (TX) or end of a packet is received. 0 = disable interrupt Should always set to 1.
6–0	STATUS	R	Frame status 6: Exceed Retry Error 5: Heartbeat (SQE) 4: Late Collision Error 3: Collision 2: CRC Error 1: Underrun Error 0: Loss of Carrier Error

Word0 contains status and control bits. The DMA controller reads DW0 when its poll counter expires or the TX descriptor buffer ready command is given.

After reading DW0, the DMA controller checks bit 15 (OWN) to determine if the host has relinquished ownership of the 4-word descriptor and data buffer space. The OWN bit is used as a simple semaphore between the host and the DMA controller that defines ownership of the descriptor. Only the owner of a descriptor can write to the descriptor words. A buffer memory overflow (or underflow) condition can occur if the ownership bit is still set to HOST when the buffer memory fills up (empties).

The next bit (WRAP) controls descriptor chaining and wrapping (ring).

Following WRAP are two bits for marking the start (First-In-Frame: FIF) and end of frames (Last-In-Frame: LIF). If the FIF bit is set, the data buffer pointed to by DW2 and DW3 contains the start of a new packet. If the LIF bit is set, then the data buffer pointed to by DW2 and DW3 contains the last bytes of the packet. If multiple descriptors are chained together to build a single packet, then it is possible that only one or neither of FIF/LIF bits are set. The last control bit DW0 is bit 7, INTRE. It commands the DMA controller whether or not to generate a transmit interrupt after the controller has finished processing the descriptor. The interrupt bit allows interrupts to be generated on descriptor basis.

The remaining 11 bits of DW0 are status bits. Whenever the DMA controller finishes processing the descriptor, it writes to the status bits. One field unique to TX operations is the retry count status field in word 0. This field is used to indicate how many collisions occurred while attempting to transmit the packet. The controller writes a value to the RETRY field representing the retry initialization value plus the number of collisions that occurred (the retry initialization count value can be set with the backoff seed mode register).

Table 12–3. TX Descriptor Word #1

Bit	Name	Host R/W	Function
15	—	—	*reserved*
14	PAD_CRC	W	Enable padding for frames < 64 bytes and regenerate CRC. For frames > 64 bytes enable generation of CRC for inclusion in transmitted frame. Valid only if FIF = 1. 1 = pad/CRC 0 = no pad/CRC
13–11	—	—	*reserved*
10–0	BYTES	W	Descriptor byte count. Includes DA, SA, data length, and CRC fields.

TX DW1 contains an 11-bit byte-count field and two control bits.

BYTES is an 11-bit field for descriptor data-byte count (1536 max). If a descriptor entry happens to be the start of a new frame (FIF = 1), the byte count value must be less than 1519 bytes to be a valid Ethernet frame. However, extended frames are supported up to 1536 bytes. The TX byte count in Descriptor Word 1 must be a multiple of 8 bytes unless LIF field of DW0 is set to 1. This restriction is due to the 8-byte granularity of the buffer Memory FIFO. If the descriptor has both FIF and LIF bits set, then the byte count represents the number of bytes in the packet (not including CRC if PAD_CRC is set).

The two control bits in DW1 are for loopback test control and for short frame padding and CRC generation. See Section 12.3.5.2, *Data Transmission*, on page 12-14 for further details regarding pad/crc insertion. PAD_CRC is valid only if the FIF bit is also set.

Table 12–4. TX Descriptor Word #2

Bit	Name	Host R/W	Function
15–0	MS_ADDR	W	Most significant 16 bits of 32-bit data packet address

Table 12–5. TX Descriptor Word #3

Bit	Name	Host R/W	Function
15–0	LS_ADDR	W	Least significant 16 bits of 32-bit data packet address

Words 2 and 3 contain a 32-bit address that points to where the packet data is located. Only the host can write to words 2 and 3.

DMA Reads

The buffer memory is capable of storing up to four independent packets (if they are each less than 65 bytes). To reduce risks of underruns and to achieve the minimum IPG between packets while transmitting, the DMA controller reads a minimum of two blocks of 64 bytes before starting to transmit. Moreover, it starts to read the following descriptors before closing the current one, if it has the LIF bit set.

For long transmits consisting of more than one descriptor packet, ENET passes ownership of each descriptor back to the host after it has finished transferring BYTES of data from system memory to the buffer memory. Note that the descriptor is passed back to the host even before the data is transmitted by the MAC. ENET writes whatever status it has at the time it closes each intermediate descriptor.

Final status is set in the descriptor that has the LIF bit set. This behavior is the same even if a transmit error occurs.

Reporting Errors

It is possible for the DMA controller to find itself in a situation where it closes one intermediate descriptor and finds that the host still owns the next. If the host does not relinquish ownership of the next descriptor before the MAC empties the data in buffer memory, the MAC signals an underflow error. This underflow error will be reported with a system error interrupt. However, other errors, such as late collisions, cannot be reported until after the host relinquishes ownership of the next descriptor.

If an error other than a normal collision occurs while transmitting a packet with multiple descriptors, the ENET ceases transmission and closes out all subsequent descriptors for the packet until it finds the descriptor with LIF set. When LIF is detected, ENET closes this descriptor and writes the appropriate error status.

Re-Transmission

The ENET does not attempt a re-transmission of the packet unless the error was a normal collision, even if the packet was contained in a single descriptor.

If multiple attempts are required to transmit a packet consisting of multiple descriptor entries, only the descriptors not yet closed and the descriptor with the LIF bit set will have the valid retry count status reported.

When the DMA controller finishes processing a descriptor entry that had the wrap bit set, the controller returns to the first descriptor entry by reloading the descriptor base address from the configuration registers. Only the host can change the value of the wrap bit.

12.4.2 RX Descriptor Ring

The main difference between the TX and RX descriptors is found in the RX descriptor status field. The other difference is in the usage of the LIF, FIF, and loopback bits. For RX operations, the ENET writes to these bits to frame status; whereas for TX operations, ENET reads these bits and takes action on them.

Table 12–6. RX Descriptor Word #0

Bit	Name	Host R/W	Function
15	OWN	R/W	Ownership bit 0 = ENET, 1 = HOST (ESM) After initialization, descriptor has to be owned by ENET.
14	WRAP	W	Descriptor chain wrap 0 = go to the next sequential descriptor entry 1 = go to first descriptor as defined in configuration register
13	FIF	R	First-In-Frame: When set to 1 indicates that this descriptor contains the first bytes of a new frame
12	LIF	R	Last-In-Frame: When set to 1 indicates that this descriptor contains the last bytes of a new frame
11–8	—	—	*reserved*
7	INTRE	W	Interrupt enable: 1 = Generate an RX_IRQ interrupt after BYTES of data have been received (RX) or end of a packet is received. 0 = disable interrupt Should always set to 1.
6–0	STATUS	R	Frame status 6: Miss 5: VLAN 4: Long Frame Error 3: Short Frame Error 2: CRC Error 1: Overrun Error 0: Non Octet Alignment Error

Table 12–7. RX Descriptor Word #1

Bit	Name	Host R/W	Function
15–11	—	—	*reserved*
10–0	BYTES	R/W	Descriptor buffer size set by the host to indicate the size of the data buffer. When ENET finishes receiving a packet or fills the buffer, it will overwrite this field with the actual received byte count. Value set by the host must be a multiple of 64 bytes.

Table 12–8. RX Descriptor Word #2

Bit	Name	Host R/W	Function
15–0	MS_ADDR	W	Most significant 16 bits of 32-bit data packet address

Table 12–9. RX Descriptor Word #3

Bit	Name	Host R/W	Function
15–0	LS_ADDR	W	Least significant 16 bits of 32-bit data packet address

The buffer memory block signals the DMA controller when it has at least one 64-byte block of data or an end of packet condition occurs. The flag field in the buffer memory contains Start-Of-Frame (SOF) and End-Of-Frame (EOF) and byte count information. The DMA controller uses the flag field status to set the FIF, LIF and BYTE fields of receive descriptor words 0 and 1.

The RX DMA makes memory access requests to look for an open descriptor only after it has data that needs to be stored to memory. If the DMA controller reads a descriptor and finds that it does not own it, the controller repeats the memory access every other clock until the host sets the ownership back to DMA or an overrun error occurs.

Reject Short Frame Error

Normally, a valid RX packet will always be at least 64 bytes long. The MAC block automatically flushes any packet shorter than 64 bytes. However, ENET has a mechanism for recording the occurrence of short frame errors using the Reject Short Frame Error (RJCT_SFE) bit in MODE register. The controller then reports the error status if the RJCT_SFE of the ENET mode register is off(0). The DMA controller opens the next descriptor, writes the appropriate status, and then closes the descriptor, setting the ownership bit back to the host. When set to zero, every RX error occurring within the first 64 bytes will be reported by opening an RX descriptor, writing status, and then closing it. So normal collisions (which look like runts or short frames to the MAC) can cause consumption of the RX descriptors, even for packets with a non-matching destination address.

See section 12.3.5.1, *Data Reception*, on page 12-9 for details on the definition of the receive status bits.

For long receives consisting of more than one descriptor packet, ENET closes each descriptor when it finishes writing BYTES of data to system memory. The DMA controller passes descriptor ownership back to the host as it closes each descriptor. If the controller finds that the next descriptor is owned by the host, it continues to read RX DW0 from memory until it finds that the host has relinquished ownership of the descriptor or until a receive error occurs such as overflow. The DMA controller writes whatever RX status it has at the time a descriptor is closed.

The DMA controller can generate an interrupt when it closes each descriptor, depending on the INTRE bit.

12.5 EIM Peripheral Register Tables

The EIM peripheral registers are divided into Ethernet state machine (ESM) peripheral registers and ENET peripheral registers.

ESM Peripheral Registers

Base Address (hex): FFFF:0000

Register width: 32 bits

All registers are big-endian aligned. Unmapped bits are always read as zero. Access to unmapped registers leads to an undefined result.

Table 12–10. ESM Peripheral Registers

Register	Description	Offset Address
EIM_CTRL	ESM Control Register	00h
EIM_STATUS	ESM Status Register	04h
EIM_CPUTXBA	CPU TX Descriptors Base Address Register	08h
EIM_CPURXBA	CPU RX Descriptors Base Address Register	0Ch
EIM_BUFSIZE	Packet Buffer Size Register	10h
EIM_FILTER	CPU Filtering Control Register	14h
EIM_CPUDA_1	CPU Destination Address Register, High Word	18h
EIM_CPUDA_0	CPU Destination Address Register, Low Word	1Ch
EIM_MFV_1	Multicast Filter Valid Register, High Word	20h
EIM_MFV_0	Multicast Filter Valid Register, Low Word	24h
EIM_MFM_1	Multicast Filter Mask Register, High Word	28h
EIM_MFM_0	Multicast Filter Mask Register, Low Word	2Ch
EIM_RXTH	RX Threshold Register	30h
EIM_RX_CPU_RDY	CPU RX Ready Register	34h
EIM_INT_EN	ESM Interrupt Enable Register	38h
Reserved	—	3Ch
EIM_ENET0_TX_DESC	ENET0 TX Queue Current Pointer Register	40h

Table 12–10. ESM Peripheral Registers (Continued)

Register	Description	Offset Address
EIM_ENET0_RX_DESC	ENET0 RX Queue Current Pointer Register	44h
Reserved		48h
Reserved		4Ch
EIM_CPU_TX_DESC	CPU TX Queue Current Pointer Register	50h
EIM_CPU_RX_DESC	CPU RX Queue Current Pointer Register	54h

ENET Peripheral Registers

Base Address (hex): FFFF:0000

Register width: 32 bits

All registers are big-endian aligned. This means that all accesses are performed on all register bits. Therefore, ARM software can perform 8-/16-/32-bit reads; it can also perform 8-/16-bit writes on an 8-/16-bit register, but an 8-/16-bit write to a 32-bit register will result in an unexpected write on bits 16 to 31.

Table 12–11. ENET0 Registers

Register	Description	Offset Address
EIM_MODE_E0	Mode Register	100h
EIM_NEW_RBOF_E0	Backoff Seed Register	104h
EIM_RBOF_CNT_E0	Backoff Count Register	108h
EIM_FLW_CNT_E0	TX Flow Pause Count Register	10Ch
EIM_FLW_CNTRL_E0	Flow Control Register	110h
EIM_VTYPE_E0	VTYPE Tag Register	114h
EIM_SE_SR_E0	System Error Interrupt Status Register	118h
EIM_TX_BUF_RDY_E0	Transmit Descriptor Buffer Ready Register	11Ch
EIM_TDBA_E0	Transmit Descriptor Base Address Register	120h
EIM_RDBA_E0	Receive Descriptor Base Address Register	124h
EIM_PAR1_E0	Destination Physical Address Match Register, High Word	128h
EIM_PAR0_E0	Destination Physical Address Match Register, Low Word	12Ch
EIM_LAR1_E0	Logical Address Hash Filter Register, High Word	130h
EIM_LAR0_E0	Logical Address Hash Filter Register, Low Word	134h
EIM_ADR_MODE_E0	Address Mode Enable Register	138h
EIM_DRP_E0	Descriptor Ring Poll Interval Count Register	13Ch

12.6 ESM Peripheral Registers

12.6.1 EIM ESM Control Register

Base Address (hex): FFFF:0000

Register width: 32 bits

When set, ESMEN will reset the ESM and all of the ESM registers.

When a queue is not enabled, it cannot be accessed by the ESM, and therefore, its size is null and its space can be used for other purposes.

Normal behavior is to transfer packets received on the CPU port to the ENET0 port.

Figure 12–8. EIM ESM Control Register (EIM_CTRL)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0000

31–16	15	14–9	8
Reserved	ESMEN	Reserved	CPU_ENET0_EN
	RW–0		RW–0

7	6	5	4	3–2	1	0
Reserved	ENET0_FLWCNTEN	ENET0_RXEN	ENET0_TXEN	Reserved	CPU_RXEN	CPU_TXEN
	RW–0	RW–0	RW–0		RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**.

Bit 15 **ESMEN**. Ethernet State Machine Enable.

0 Disable
1 Enable

Bits 14–9 **Reserved**.

Bit 8 **CPU_ENET0_EN**. When set, this bit enables routing of packets received on CPU port to ENET0.

0 Disable
1 Enable—must be set if ESMEN = 1

Bit 7 **Reserved**.

Bit 6 **ENET0_FLWCNTEN**. When set, enables flow control threshold trigger on ENET0 RX queue

Bit 5 **ENET0_RXEN**. When set, enables processing of ENET0 RX queue

Bit 4 **ENET0_TXEN**. When set, enables processing of ENET0 TX queue

Bits 3–2 **Reserved**.

Bit 1 **CPU_RXEN**. When set, enables processing of CPU RX queue

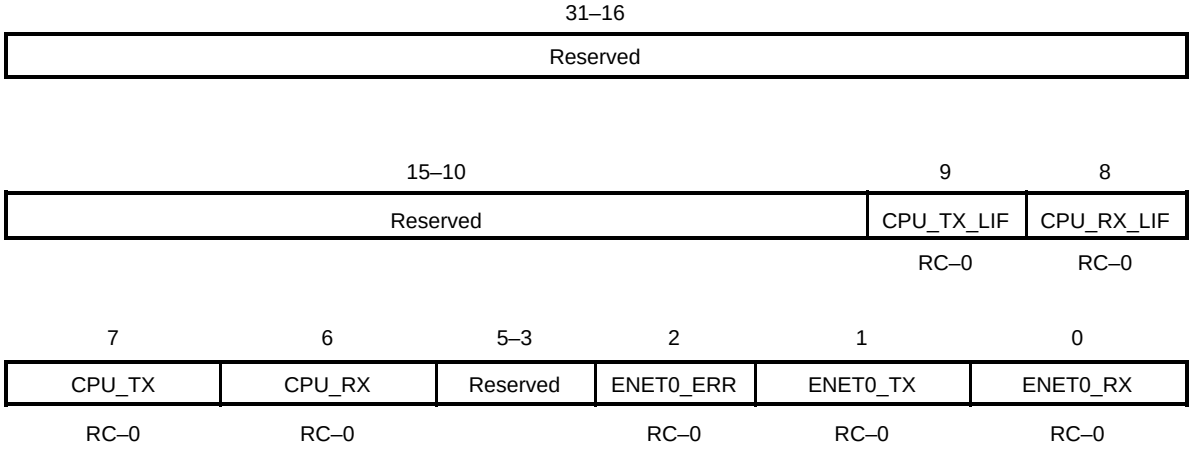
Bit 0 **CPU_TXEN**. When set, enables processing of CPU TX queue

12.6.2 EIM ESM Status Register

ESM status is set independent of the related interrupt enable bit. Status bits are read-and-clear registers: bits are automatically cleared when the CPU performs a read, thus avoiding a manual clear with the risk of status loss. Writing to Status bits has no effect.

Figure 12–9. EIM ESM Status Register (EIM_STATUS)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0004



Note: R = Read access; C = cleared on a read; value following dash (–) = value after reset

Bits 31–10 **Reserved**.

Bit 9 **CPU_TX_LIF**. Set after the last descriptor of a packet has been filled in the CPU TX queue

Bit 8 **CPU_RX_LIF**. Set after the last descriptor of a packet has been processed (and given back to the host) in the CPU RX queue

Bit 7 **CPU_TX**. Set after a descriptor has been filled in the CPU TX queue

Bit 6 **CPU_RX**. Set after a descriptor has been filled in the CPU RX queue

Bits 5–3 **Reserved**.

Bit 2 **ENET0_ERR**. Set after an ENET0 ERR interrupt has been triggered

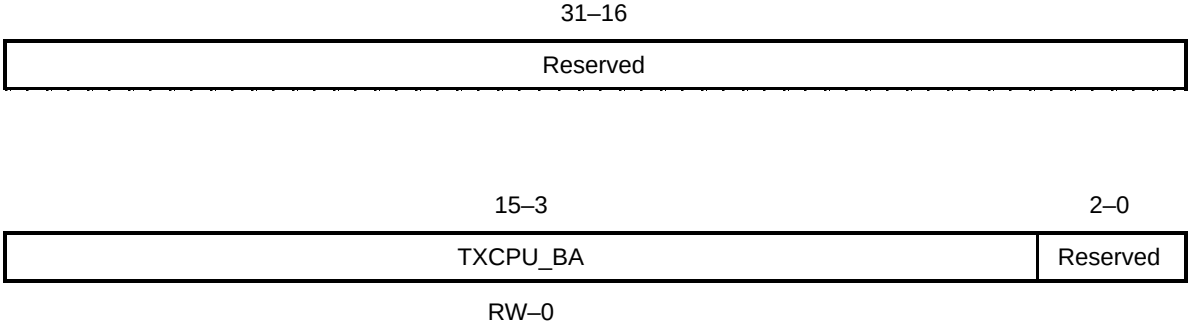
Bit 1 **ENET0_TX**. Set after an ENET0 TX interrupt has been triggered

Bit 0 **ENET0_RX**. Set after an ENET0 RX interrupt has been triggered

12.6.3 EIM CPU TX Descriptors Base Address Register

Figure 12–10. EIM CPU TX Descriptors Base Address Register (EIM_CPUTXBA)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0008



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**.

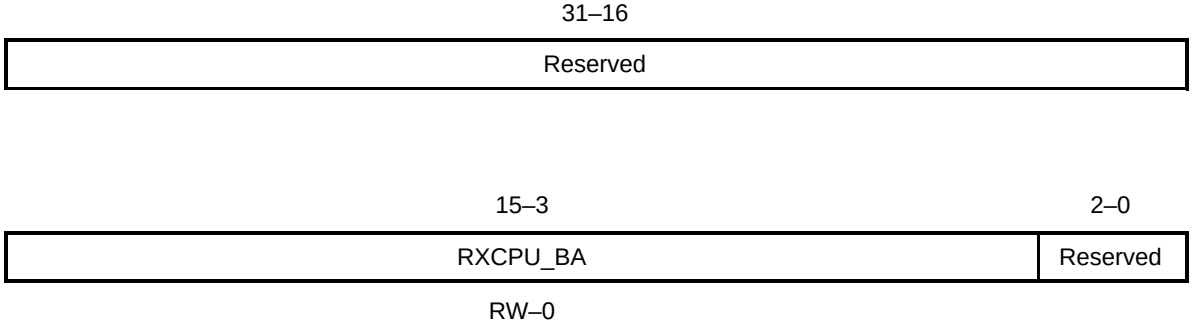
Bits 15–3 **TXCPU_BA**. LSW offset address of first descriptor of CPU TX ring. Lower 16 bits of the 32-bit address range, with bits 2–0 fixed at zero.

Bits 2–0 **Reserved** (always read as zeros)

12.6.4 EIM CPU RX Descriptors Base Address Register

Figure 12–11. EIM CPU RX Descriptors Base Address Register (EIM_CPURXBA)

Address (hex): Base = 0xFFFF:0000, Offset = 0x000C



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**.

Bits 15–3 **RXCPU_BA**. LSW offset address of first descriptor of CPU RX ring. Lower 16 bits of the 32-bit address range, with bits 2–0 fixed at zero.

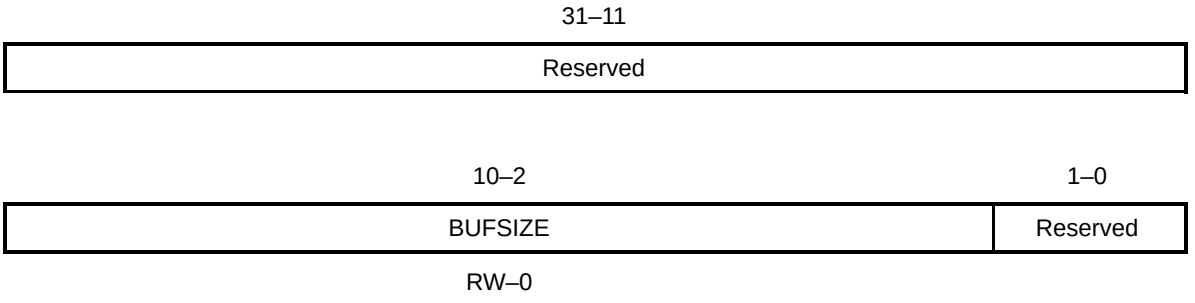
Bits 2–0 **Reserved** (always read as zeros)

12.6.5 EIM Packet Buffer Size Register

Buffer size has to be set with a multiple of 4 and a minimum of 64 (and ≤ 1536).
Buffer size is directly expressed in bytes when accessing BUFSIZE (10 – 0).

Figure 12–12. EIM Packet Buffer Size Register (EIM_BUFSIZE)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0010



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–11 **Reserved**.

Bits 10–2 **BUFSIZE**. Size of packet buffers

Bits 1–0 **Reserved** (always read as zeros)

12.6.6 EIM CPU Filtering Control Register

Figure 12–13. EIM CPU Filtering Control Register (EIM_FILTER)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0014

31–8
Reserved

7–5	4	3	2	1	0
Reserved	MCLAEN	LOGICALEN	MULTICASTEN	BROADCASTEN	DAEN
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–5 **Reserved**.

Bit 4 **MCLAEN**. When set, enables matching on Logical Address and Multicast filtering. LOGICALEN and MULTICASTEN must be cleared when MCLAEN is set.

Bit 3 **LOGICALEN**. When set, enables use of the Logical Filtering mechanism of the ENET

Bit 2 **MULTICASTEN**. When set, enables the Multicast Filter matching

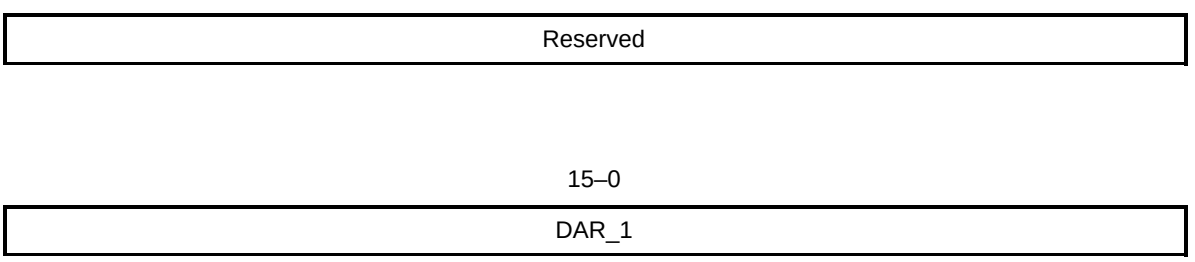
Bit 1 **BROADCASTEN**. When set, enables Broadcast matching

Bit 0 **DAEN**. When set, enables CPU Destination Address matching

12.6.7 EIM CPU Destination Address Register, High Word

Figure 12–14. EIM CPU Destination Address Register, High Word (EIM_CPUDA_1)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0018



Note: R = Read access; W = Write access; value following dash (–) = value after reset

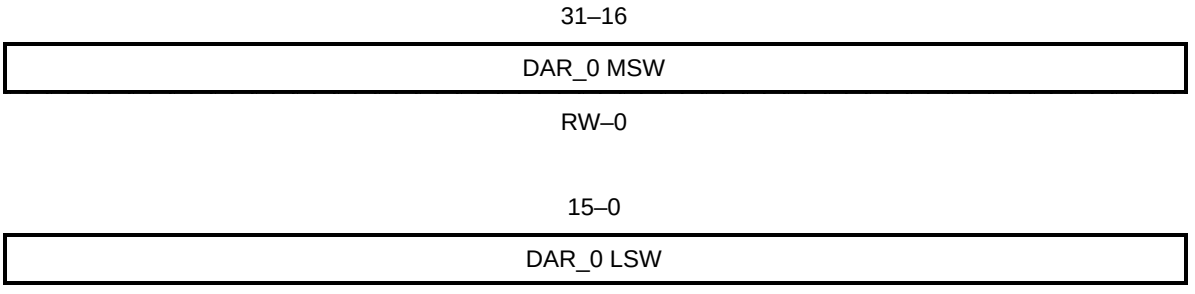
Bits 31–16 **Reserved**. Always returns “0”.

Bits 15–0 **DAR_1**. CPU Destination Address 1, LSW. Combined with EIM_CPUDA_0 to produce a 48-bit-wide value capable of address matching MAC address “01:02” hex.

12.6.8 EIM CPU Destination Address Register, Low Word

Figure 12–15. EIM CPU Destination Address Register, Low Word (EIM_CPUDA_0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x001C



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **DAR_0 MSW**. CPU Destination Address 0, MSW. Combined with EIM_CPUDA_1 to produce a 48-bit-wide value capable of address matching MAC address “03:04” hex.

Bits 15–0 **DAR_0 LSW**. CPU Destination Address 0, LSW. Combined with EIM_CPUDA_1 to produce a 48-bit-wide value capable of address matching MAC address “05:06” hex.

12.6.9 EIM Multicast Filter Valid Register, High Word

Figure 12–16. EIM Multicast Filter Valid Register, High Word (EIM_MFV_1)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0020

31–16
Reserved

15–0
MFV_1
RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**.

Bits 15–0 **MFV_1**. Bit field used to indicate bits that have to match in Multicast Filter Mask Register, bits 47 to 32 (EIM_MFM1)

12.6.10 EIM Multicast Filter Valid Register, Low Word

Figure 12–17. EIM Multicast Filter Valid Register, Low Word (EIM_MFV_0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0024

31–0
MFV_0
RW–0

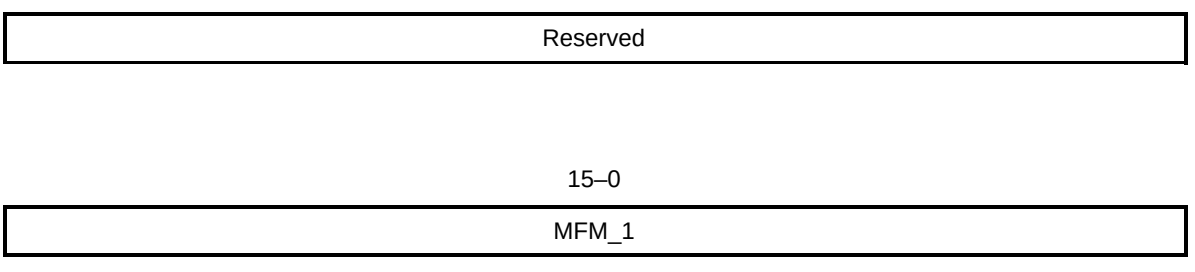
Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–0 **MFV_0**. Bit field used to indicate bits that have to match in Multicast Filter Mask Register, bits 31 to 0 (EIM_MFM0)

12.6.11 EIM Multicast Filter Mask Register, High Word

Figure 12–18. EIM Multicast Filter Mask Register, High Word (EIM_MFM_1)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0028



Note: R = Read access; W = Write access; value following dash (–) = value after reset

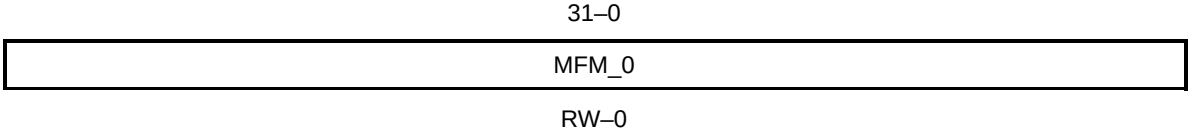
Bits 31–16 **Reserved**.

Bits 15–0 **MFM_1**. Address mask corresponding to multicast packets, bits 47 to 32.

12.6.12 EIM Multicast Filter Mask Register, Low Word

Figure 12–19. EIM Multicast Filter Mask Register, Low Word (EIM_MFM_0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x002C



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–0 **MFM_0**. Address mask corresponding to multicast packets, bits 31 to 0.

12.6.13 EIM RX Threshold Register

Figure 12–20. EIM RX Threshold Register (EIM_RXTH)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0030

31–8
Reserved

7–0
RXTH
RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–8 **Reserved**.

Bits 7–0 **RXTH**. Number of pending RX descriptors to reach in an ENET ring to trigger the TX flow control frame on this ENET.

12.6.14 EIM CPU RX Ready Register

Figure 12–21. EIM RX CPU Ready Register (EIM_RX_CPU_RDY)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0034

31–8
Reserved

7–1	0
Reserved	CPURX_RDY
	W–0

Note: W = Write access; value following dash (–) = value after reset

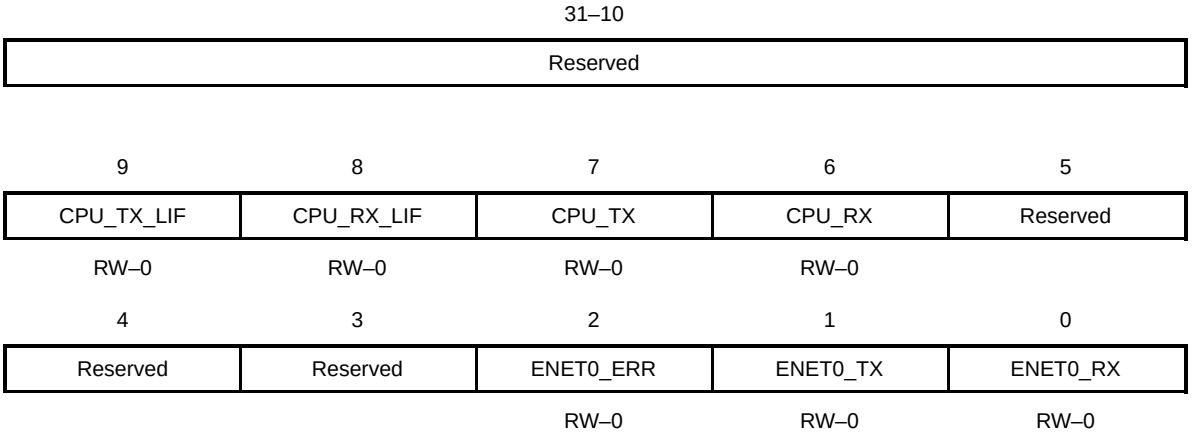
Bits 31–1 **Reserved**.

Bit 0 **CPURX_RDY**. A 1 has to be written to this bit to indicate to the ESM module that a descriptor is ready to be switched in the CPU RX queue.

12.6.15 EIM ESM Interrupt Enable Register

Figure 12–22. EIM ESM Interrupt Enable Register (EIM_INT_EN)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0038



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–10 **Reserved**.

Bit 9 **CPU_TX_LIF**.

0 CPU TX LIF interrupt disabled
1 CPU TX LIF interrupt enabled

Bit 8 **CPU_RX_LIF**.

0 CPU RX LIF interrupt disabled
1 CPU RX LIF interrupt enabled

Bit 7 **CPU_TX**.

0 CPU TX interrupt disabled
1 CPU TX interrupt enabled

Bit 6 **CPU_RX**.

0 CPU RX interrupt disabled
1 CPU RX interrupt enabled

Bits 5–3 **Reserved**.

Bit 2 **ENET0_ERR**.

0 ENET0 ERR interrupt disabled
1 ENET0 ERR interrupt enabled

Bit 1 **ENET0_TX**.

0 ENET0 TX interrupt disabled

1 ENET0 TX interrupt enabled

Bit 0 **ENET0_RX**.

0 ENET0 RX interrupt disabled

1 ENET0 RX interrupt enabled

12.6.16 EIM ENET0 TX Queue Current Pointer Register

The Packet Memory, where the descriptors reside, is byte-addressable. Each descriptor has 8 bytes of information. Therefore, the pointer must increment at 8-byte intervals, leaving the three least significant bits of the pointer to be zero.

Figure 12–23. EIM ENET0 TX Queue Current Pointer Register (EIM_ENET0_TX_DESC)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0040

31–16
Reserved

15–3	2–0
ENET0_TX_PTR	Reserved
R-U	

Note: R = Read access; value following dash (–) = value after reset; U = undefined

Bits 31–16 **Reserved**.

Bits 15–3 **ENET0_TX_PTR**. Pointer to next free descriptor in ENET0 TX queue. LSW of full 32-bit address range, with bits 2–0 fixed at zero.

Bits 2–0 **Reserved** (always read as zeros)

12.6.17 EIM ENET0 RX Queue Current Pointer Register

Figure 12–24. EIM ENET0 RX Queue Current Pointer Register (EIM_ENET0_RX_DESC)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0044

31–16
Reserved

15–3	2–0
ENET0_RX_PTR	Reserved
R-U	

Note: R = Read access; value following dash (–) = value after reset; U = undefined

Bits 31–16 **Reserved**.

Bits 15–3 **ENET0_RX_PTR**. Pointer to next free descriptor in ENET0 RX queue. LSW of full 32-bit address range, with bits 2–0 fixed at zero.

Bits 2–0 **Reserved** (always read as zeros)

12.6.18 EIM CPU TX Queue Current Pointer Register

Figure 12–25. EIM CPU TX Queue Current Pointer Register (EIM_CPU_TX_DESC)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0050

31–16
Reserved

15–3	2–0
CPU_TX_PTR	Reserved
R-U	

Note: R = Read access; value following dash (–) = value after reset; U = undefined

Bits 31–16 **Reserved**.

Bits 15–3 **CPU_TX_PTR**. Pointer to next free descriptor in CPU TX queue. LSW of full 32-bit address range, with bits 2–0 fixed at zero.

Bits 2–0 **Reserved** (always read as zeros)

12.6.19 EIM CPU RX Queue Current Pointer Register

Figure 12–26. EIM CPU RX Queue Current Pointer Register (EIM_CPU_RX_DESC)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0054

31–16
Reserved

15–3	2–0
CPU_RX_PTR	Reserved
R-U	

Note: R = Read access; value following dash (–) = value after reset; U = undefined

Bits 31–16 **Reserved**.

Bits 15–3 **CPU_RX_PTR**. Pointer to next free descriptor in CPU RX queue. LSW of full 32-bit address range, with bits 2–0 fixed at zero.

Bits 2–0 **Reserved** (always read as zeros)

12.7 ENET0 Registers

12.7.1 EIM ENET0 Mode Register

Base address (hex): FFFF:0000

Register width: 32 bits

Figure 12–27. EIM ENET0 Mode Register (EIM_MODE_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0100

31–16	15	14–8
Reserved	FIFO_EN	Reserved
	RW–0	

7	6	5	4	3	2	1	0
RJCT_SFE	DPNET	MWIDTH	WRAP	FDWRAP	DUPLEX	Reserved	ENABLE
RW–0	RW–0	RW–1	RW–0	RW–0	RW–0		RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**.

Bit 15 **FIFO_EN**. Enables CPU access to ENET FIFO.

0 FIFO access disabled (normal mode)
1 FIFO access enabled (debug mode)

Bits 14–8 **Reserved**.

Bit 7 **RJCT_SFE**. Reject short frame errors during receive. When set, short frame errors are ignored.

0 Report SFE
1 Reject SFE

Bit 6 **DPNET**. Demand priority network rather than CSMA/CD.

0 Normal operation
1 Demand priority

Bit 5 **MWIDTH**. Selects the width of the MII port. In serial mode, data is received on RXD0 and transmitted on TXD0.

0 Serial mode
1 Nibble mode

Bit 4 **WRAP**. Internal MAC loopback. When set, the MAC will loop transmitted data back to received data at furthest point possible while still in MAC module.

0 Normal operation

1 WRAP

Bit 3 **FDWRAP**. Full-duplex wrap. When asserted and in wrap mode, this register stops packet reception from network.

0 Normal operation

1 Full-duplex wrap mode

Bit 2 **DUPLEX**.

0 Half-duplex

1 Full-duplex

Bit 1 **Reserved**.

Bit 0 **ENABLE**. Port Enable. When disabled, MAC attempts to finish active frames. This is a MAC enable only. It does not directly affect DMA.

0 Port disable

1 Port enable

12.7.2 EIM ENET0 Backoff Seed Register

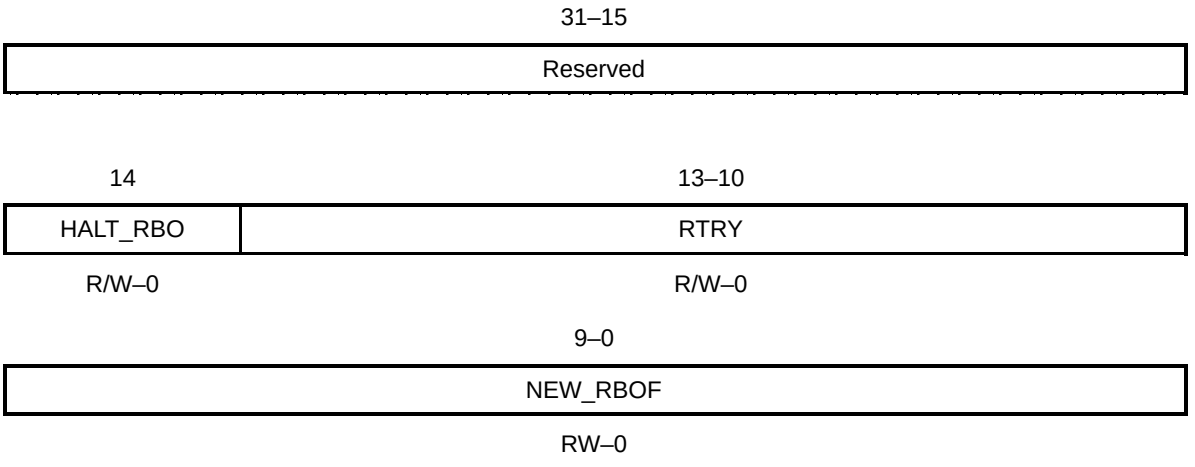
The backoff count is a random number computed using the TCLK and host clock relationship. As soon as these clocks are active, the backoff value is continually being updated even before there is a collision. The HALT_RBO bit can be used to halt the random backoff action before a collision. This is simulation debug aid.

The 4-bit RTRY field can be used to reduce the number of retries before a packet is discarded. The MAC will allow 15 retry attempts before the packet is discarded.

This register is only valid in half-duplex mode as defined by EIM_MODE_E0 (FFFF:0100) DUPLEX (bit 2).

Figure 12–28. EIM ENET0 Backoff Seed Register (EIM_NEW_RBOF_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0104



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–15 **Reserved**.

Bit 14 **HALT_RBO**. Halt random backoff

Bits 13–10 **RTRY**. Preset backoff and retry counter, where 0x0 represents a maximum value and 0xF represents a minimum value.

Bits 9–0 **NEW_RBOF**. Load new random backoff seed

12.7.3 EIM ENET0 Backoff Count Register

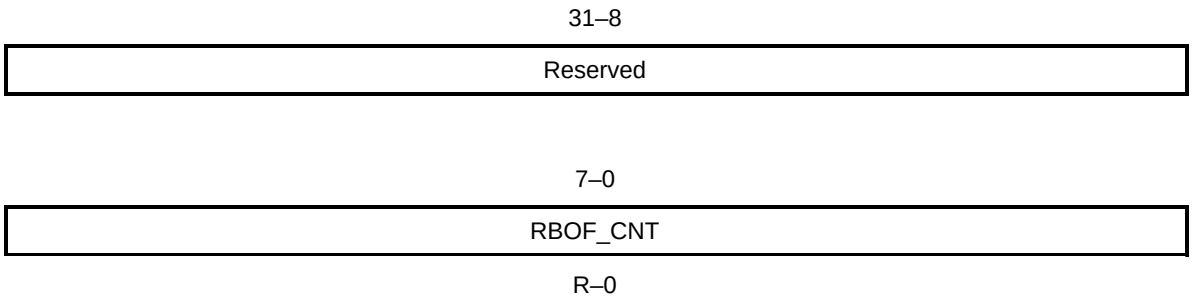
This register is used to read the eight least significant bits of the 10-bit collision backoff counter. During reset, the RBOF_CNT register is loaded with a random value.

The backoff count will decrement once for every 128 TCLKs (512 bit times when in nibble mode).

This register is only valid in half-duplex mode as defined by EIM_MODE_E0 (FFFF:0100) DUPLEX (bit 2).

Figure 12–29. EIM ENET0 Backoff Count Register (EIM_RBOF_CNT_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0108



Note: R = Read access; value following dash (–) = value after reset

Bits 31–8 **Reserved**.

Bits 7–0 **RBOF_CNT**. Current random backoff count

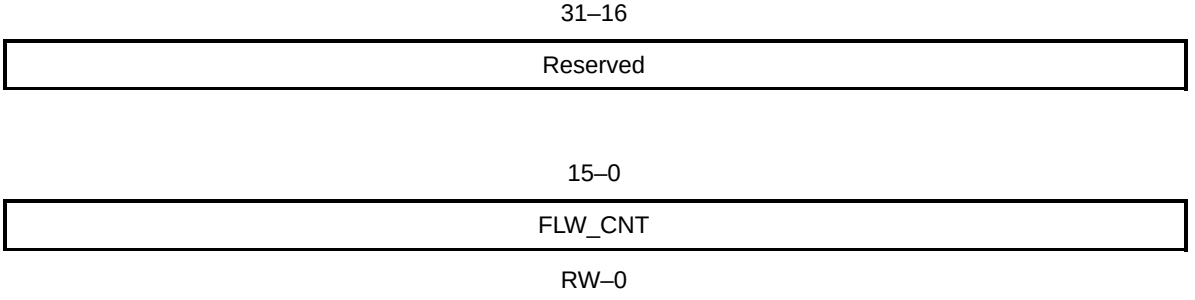
12.7.4 EIM ENET0 TX Flow Pause Count Register

Loading this register with any value will automatically cause the MAC to transmit a flow control frame with the FLW_CNT value placed in the type/length field. The Count value represents the number of *slot times* where one slot is 64 bytes.

The flow control frame will not be sent if both RX_FLW_EN and BACK_PSR (see below) are set to zero.

Figure 12–30. EIM ENET0 TX Flow Pause Count Register (EIM_FLW_CNT_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x010C



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**.

Bits 15–0 **FLW_CNT**. Flow Control Pause Count: Pause count to be transmitted as a flow control frame

12.7.5 EIM ENET0 Flow Control Register

RX_FLW_EN is for full-duplex operation while BACK_PSR can be used as a form of flow control for half-duplex operation.

Figure 12–31. EIM ENET0 Flow Control Register (EIM_FLW_CNTRL_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0110

31–8
Reserved

7–2	1	0
Reserved	BACK_PSR	RX_FLW_EN
	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–2 **Reserved**.

Bit 1 **BACK_PSR**. Enable Back Pressure – force collisions

Bit 0 **RX_FLW_EN**. Enable Pause based flow control for received frames

0 Disabled
1 Enabled

12.7.6 EIM ENET0 VTYPE Tag Register

This register is used for setting the compare value for the Virtual LAN (VLAN) tag field in the received data. The VLAN tag field is the first 16 bits in the data field of the received data. No filtering is performed on frames based on VLAN comparison; only status is reported in RX descriptor.

Figure 12–32. EIM ENET0 VTYPE Tag Register (EIM_VTYPE_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0114

31–16
Reserved

15–0
VTYPE
RW–0xFFFF

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**.

Bits 15–0 **VTYPE**. VLAN tagged frame compare value.

12.7.7 EIM ENET0 System Error Interrupt Status Register

Setting any bit to 1 will activate the $\overline{\text{SYS\_ERR\_IRQ}}$ signal of ENET module. Note that this is not sufficient to generate an interrupt to the CPU. The actual interrupt is generated by the EIM ESM module if validated (see EIM ESM Interrupt Enable Register).

Figure 12–33. EIM ENET0 System Error Interrupt Status Register (EIM_SE_SR_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0118

31–8
Reserved

7–5	4	3	2	1	0
Reserved	TX_FE	RX_BCE	FIFE	OFLW	UFLW
	RW–0	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–5 **Reserved**.

Bit 4 **TX_FE**. TX Only.

1 TX byte count or PAD_ CRC improperly set

Bit 3 **RX_BCE**. RX Only

1 RX byte count error.
This bit is set if new RX descriptor entry has byte count field set to less than 8 bytes.

Bit 2 **FIFE**. TX Only.

1 First-in-frame error

Bit 1 **OFLW**. RX Only.

1 Buffer memory overflow

Bit 0 **UFLW**. TX Only.

1 Buffer memory underflow

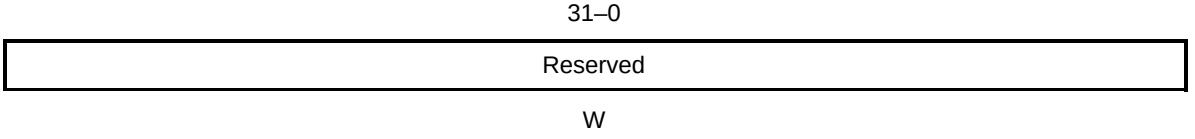
12.7.8 EIM ENET0 Transmit Descriptor Buffer Ready Register

There is no physical peripheral register associated with the Transmit Descriptor Buffer Ready Register. When a write occurs, it prompts the DMA controller to start processing the next transmit descriptor entry independent of the poll counter value.

This is a write-only register, where any value written into this register will prompt the DMA controller.

Figure 12–34. EIM ENET0 Transmit Descriptor Buffer Ready Register (EIM_TX_BUF_RDY_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x011C

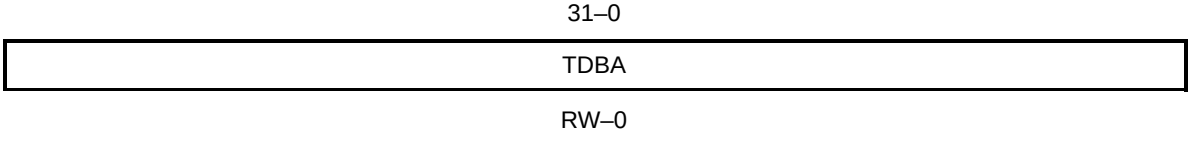


Bits 31–0 **Reserved**.

12.7.9 EIM ENET0 Transmit Descriptor Base Address Register

Figure 12–35. EIM ENET0 Transmit Descriptor Base Address Register (EIM_TDBA_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0120



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–0 **TDBA**. Transmit Descriptor Base Address. This register is written by the ARM host and is not changed by the ENET.

12.7.10 EIM ENET0 Receive Descriptor Base Address Register

Figure 12–36. EIM ENET0 Receive Descriptor Base Address Register (EIM_RDBA_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0124

31–0
RDBA
RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–0 **RDBA**. Receive Descriptor Base Address. This register is written by the ARM host and is not changed by the ENET.

12.7.11 EIM ENET0 Destination Physical Address Match Register, High Word

Figure 12–37. EIM ENET0 Destination Physical Address Match Register, High Word (EIM_PAR1_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0128

31–16
Reserved

15–0
PAR_1
RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**. Always returns “0”.

Bits 15–0 **PAR_1**. ENET0 Destination Address Match 1, LSW. Combined with EIM_PAR0_E0 to produce a 48-bit-wide value capable of address matching MAC address “06:05” hex.

12.7.12 EIM ENET0 Destination Physical Address Match Register, Low Word

Figure 12–38. EIM ENET0 Destination Physical Address Match Register, Low Word (EIM_PAR0_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x012C

31–16
PAR_0 MSW
RW–0

15–0
PAR_0 LSW
RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **PAR_0 MSW**. ENET0 Destination Address Match 0, MSW. Combined with EIM_PAR1_E0 to produce a 48-bit-wide value capable of address matching MAC address “04:03” hex.

Bits 15–0 **PAR_0 LSW**. ENET0 Destination Address Match 1, LSW. Combined with EIM_PAR1_E0 to produce a 48-bit-wide value capable of address matching MAC address “02:01” hex.

12.7.13 EIM ENET0 Logical Address Hash Filter Register, High Word

Figure 12–39. EIM ENET0 Logical Address Hash Filter Register, High Word (EIM_LAR1_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0130

31–0
LAR_1
RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–0 **LAR_1**. Logical Hash Filter Address Register bits 63:32. These bits are checked based on a 6-bit CRC compressed Destination Address. If set, packet is accepted.

12.7.14 EIM ENET0 Logical Address Hash Filter Register, Low Word

Figure 12–40. EIM ENET0 Logical Address Hash Filter Register, Low Word (EIM_LAR0_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0134

31–0
LAR_0
RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–0 **LAR_0**. Logical Hash Filter Address Register bits 31:0. These bits are checked based on a 6-bit CRC compressed Destination Address. If set, packet is accepted.

12.7.15 EIM ENET0 Address Mode Enable Register

The type of comparison done on the Destination Address depends on the address mode selected. Multiple address modes can be enabled at a time. Of course, enabling Snoop Address mode with anything else is redundant. If all address modes are disabled, the ENET flushes all received packets.

Figure 12–41. EIM ENET0 Address Mode Enable Register (EIM_ADR_MODE_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x0138

31–8
Reserved

7–4	3	2	1	0
Reserved	ESAC	EBAC	ELAC	EPAC
	RW–0	RW–0	RW–0	RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–4 **Reserved**.

Bit 3 **ESAC**. Enables Snoop Address Compare (Promiscuous) when set.

Bit 2 **EBAC**. Enables Broadcast Address Compare when set.

Bit 1 **ELAC**. Enables Logical Address Compare when set.

Bit 0 **EPAC**. Enables Physical Address Compare when set.

12.7.16 EIM ENET0 Descriptor Ring Poll Interval Count Register

Figure 12–42. EIM ENET0 Descriptor Ring Poll Interval Count Register (EIM_DRP_E0)

Address (hex): Base = 0xFFFF:0000, Offset = 0x013C

31–16
Reserved

15–0
DRPC
RW–0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 31–16 **Reserved**.

Bits 15–0 **DRPC**. Descriptor Ring Poll Interval Count. Value used to load a counter that is decremented once per ENET clock. When a terminal count is reached, the DMA controller polls the current descriptor entry to check the ownership bit. Polling is disabled when count is initialized to zero.

12.8 EIM Packet RAM Structure

12.8.1 Logical Organization

The Packets Memory provides up to four queues for packet storing: 1 queue for received packets and 1 queue for transmit packets of each port (CPU and ENET0). Each queue element consists of a packet (or a part of a packet) plus some status and control flags.

A queue is a ring of entries called descriptors. Each descriptor contains a pointer to an allocated buffer containing a data packet.

When a packet has to be transferred between two queues, only the descriptors (pointing to the packet) are copied; i.e., the data packet is passed by reference.

Each descriptor has to be linked to a packet buffer. To keep track of free buffers, a Free Buffer Entry is added after each packet buffer. This allows a buffer to be referenced by two descriptors of two different queues.

From the ESM point of view, the CPU is seen as a port like ENET0. Therefore, the following conventions are defined:

- A packet sent to the ARM is considered as a transmitted packet and goes to the CPU TX queue
- A packet sent from the ARM is considered as a received packet and goes to the CPU RX queue
- A packet sent to the ENET0 port is considered as a transmitted packet and goes to the ENET0 TX queue.
- A packet sent from the ENET0 port is considered as a received packet and goes to the ENET0 RX queue.

12.8.2 Packets Memory Physical Organization

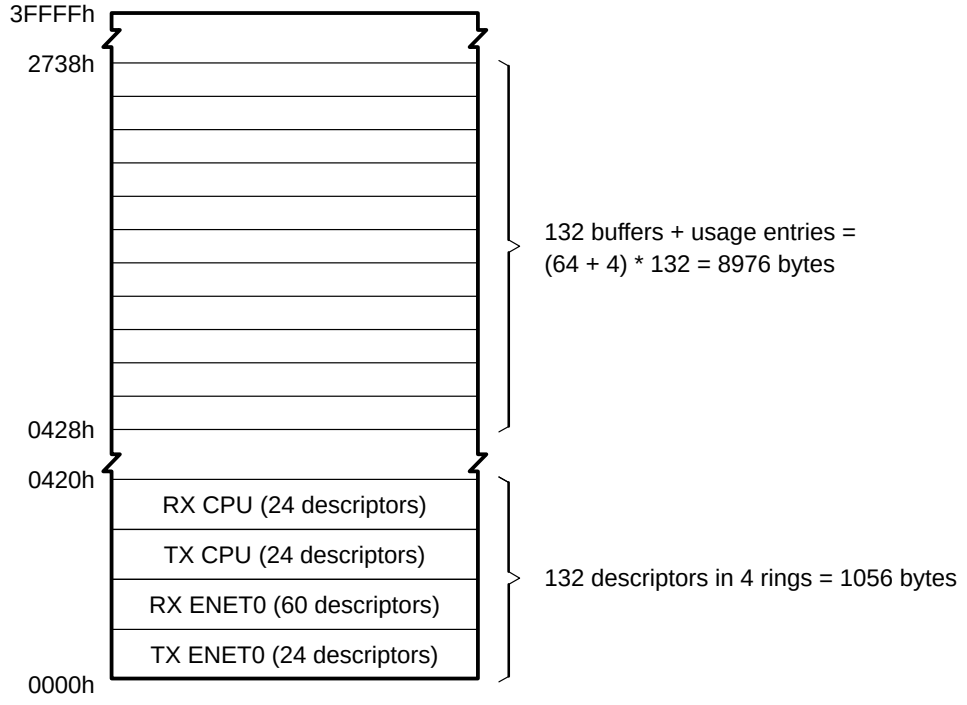
It is the responsibility of the ARM software to initialize the Packets Memory structure and to insure coherence with the configuration values programmed in the ESM and ENET registers.

Figure 12–43 describes a possible structure of the Packets Memory.

In this example, the ENET0 RX queue has been set to a length of more than two Ethernet maximum frame lengths. All other queues are set to a length corresponding to a maximum size Ethernet packet.

Other queuing strategies may be implemented. The only constraints are that descriptors have to be aligned on an 8-byte boundary, that buffer length has to be a multiple of 4 with a minimum of 64 bytes, and that the total number of descriptors of each ring has to be less than 255.

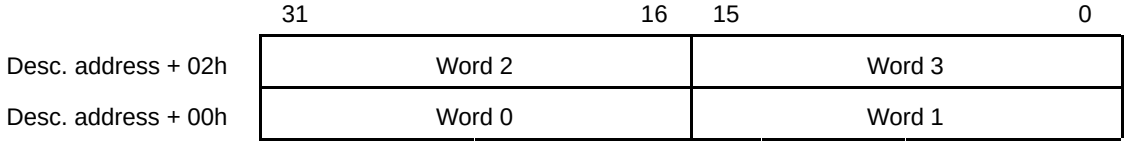
Figure 12–43. Packets Memory Physical Organization



12.8.3 Descriptor Words

A Descriptor is an 8-byte-wide structure containing packet status and pointer to data. Descriptor word structure is based on ENET software configuration, with some restriction in the flags usage. The descriptor word structure is mapped as in Figure 12–44.

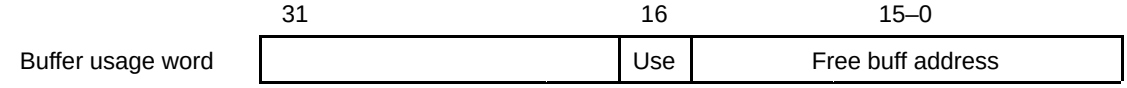
Figure 12–44. Descriptor Word Structure



Inside a queue, each descriptor is managed sequentially. The end of a queue is signaled by a descriptor flag (WRAP).

In the EIM Module, interrupts must be validated (INTRE=1 for each descriptor). For normal operation, you do not have to manipulate the descriptor rings. This is done in HW (Ethernet State Machine). Only the initialization is the responsibility of the software.

Figure 12–45. Buffer Usage Word Structure



12.8.4 CPU TX Descriptor

Table 12–12. CPU TX Descriptor Words #0 and 1

Bit	Name	Host R/W	Function
31	OWN	R/W	Ownership bit. 0 = CPU, 1 = HOST
30	WRAP	W	Descriptor chain wrap 0 = go to the next sequential descriptor entry 1 = go to first descriptor as defined in configuration register.
29	FIF	R	First-In-Frame: When set to 1, indicates that this descriptor contains the first byte of a new frame.

Table 12–12. CPU TX Descriptor Words #0 and 1 (Continued)

Bit	Name	Host R/W	Function
28	LIF	R	Last-In-Frame: When set to 1, indicates that this descriptor contains the last bytes of a new frame.
27 – 24	—	—	Reserved. Always set to 0.
23	INTRE	R	Reserved. Always set to 1.
22 – 11	—	—	Reserved
10–0	BYTES	—	Descriptor byte count

Table 12–13. CPU TX Descriptor Words #2 and 3

Bit	Name	Host R/W	Function
31 – 0	ADDR	R/W	Packet Buffer Pointer Address. Only bits 15–2 will be evaluated and modified by ESM. Bits 31–16 won't be touched at all. Bits 1–0 are not evaluated and are always set to zero when written by ESM.

12.8.5 CPU RX Descriptor

Table 12–14. CPU RX Descriptor Words #0 and 1

Bit	Name	Host R/W	Function
31	OWN	R/W	Ownership bit 0 = CPU, 1 = HOST
30	WRAP	W	Descriptor chain wrap 0 = go to the next sequential descriptor entry 1 = go to first descriptor as defined in configuration register
29	FIF	W	First-In-Frame: When set to 1, indicates that this descriptor contains the first bytes of a new frame
28	LIF	W	Last-In-Frame: When set to 1, indicates that this descriptor contains the last bytes of a new frame
27 – 24	—	—	*reserved*

Table 12–14. CPU RX Descriptor Words #0 and 1 (Continued)

Bit	Name	Host R/W	Function
23	—	—	*reserved*
22 – 16	STATUS	—	Must be written with zeros. Coming from CPU, Status don't have sense. Anyway if any of bit 20–16 is set to one, the packet will be considered in error and so will be discarded.
15	—	—	Reserved
14	PAD_CRC	W	Enable padding for frames < 64 bytes and regenerate CRC. For frames > 64 bytes, enable generation of CRC for inclusion in transmitted frame. 1 = pad/CRC 0 = no pad/CRC This bit should be set only with FIF bit.
13 – 11	—	—	Reserved
10 – 0	BYTES	W	Descriptor byte count. Includes DA, SA, data length, and CRC (if present) fields.

Table 12–15. CPU RX Descriptor Words #2 and 3

Bit	Name	Host R/W	Function
31 – 0	ADDR	W	Data Packet Buffer Address. Only bits 15–3 are evaluated by EIM, thought it's easier for applicative software to deal with all 32 bits of pointer.

12.8.6 ENET0 RX Descriptors

Table 12–16. ENET0 RX Descriptor Word # 1

Bit	Name	Host R/W	Function
31	OWN	R/W	Ownership bit 0 = ENET, 1 = HOST (ESM) After initialization, descriptor has to be owned by ENET
30	WRAP	W	Descriptor chain wrap. 0 = Go to the next sequential descriptor entry 1 = Go to first descriptor as defined in configuration register
29	FIF	R	First In Frame. When set to 1, indicates that this descriptor contains the first bytes of a new frame

Table 12–16. ENET0 RX Descriptor Word # 1 (Continued)

Bit	Name	Host R/W	Function
28	LIF	R	Last in Frame. When set to 1, indicates that this descriptor contains the last bytes of a new frame
27 – 24	—	—	Reserved
23	INTRE	W	Interrupt Enable 1 = Generate an RX_IRQ interrupt after BYTES of data have been received (RX) or end of a packet is received. 0 = disable interrupt Should always set to 1
22 – 16	STATUS	R	Frame Status 6: Miss 5: VLAN 4: Long Frame error 3: Short Frame error 2: CRC error 1: Overrun error 0: Non-Octet Alignment error
15 – 11	–	–	Reserved
10 – 0	BYTES	R/W	Descriptor buffer size set by the host to indicate the size of the data buffer. When ENET finishes receiving a packet or fills the buffer, it will overwrite this field with the actual received byte count. Value set by the host must be a multiple of 64 bytes.

Table 12–17. ENET0 RX Descriptor Word #2

Bit	Name	Host R/W	Function
31 – 0	ADDR	R/W	Data Packet Buffer Address Only bits 15–3 are evaluated by EIM, although it is easier for applicable software to deal with all 32 bits all pointer

12.8.7 ENET0 TX Descriptors

Table 12–18. ENET0 TX Descriptor Word #1

Bit	Name	Host R/W	Function
31	OWN	R/W	Ownership bit 0 = ENET, 1 = HOST (ESM) After initialization, descriptor has to be owned by ENET
30	WRAP	W	Descriptor Chain Wrap 0 = Go to the next sequential descriptor entry 1 = Go to first descriptor as defined in configuration register
29	FIF	W	First In Frame. When set to 1, indicates that this descriptor contains the first bytes of a new frame
28	LIF	W	Last in Frame When set to 1, indicates that this descriptor contains the last bytes of a new frame
27 – 24	RETRY	R	Retry Count Status 0 = Initial values 1–15 = Valid values
23	INTRE	W	Interrupt Enable 1 = Generate an TX_IRQ interrupt after BYTES of data have been transmitted (TX) or end of a packet is received. 0 = disable interrupt. Should always set to 1
22 – 16	STATUS	R	Frame Status 6: Exceed Retry error 5: Heartbeat (SQE) 4: Late collision error 3: Collision 2: CRC error 1: Underrun error 0: Loss of carrier error
15	–	–	Reserved
14	PAD_CRC	W	Enable padding for frames < 64 bytes and regenerate CRC. For frames > 64 bytes, enable generation of CRC for inclusion in transmitted frame 1 = pad/CRC 0 = no pad/CRC

Table 12–18. ENET0 TX Descriptor Word #1 (Continued)

Bit	Name	Host R/W	Function
13 – 11	—	—	Reserved
10 – 0	BYTES	W	Descriptor Byte Count Includes DA, SA, data length, and CRC fields

Table 12–19. ENET0 TX Descriptor Word #2

Bit	Name	Host R/W	Function
31 – 0	ADDR	R/W	Data Packet Buffer Address Only bits 15–3 are evaluated by EIM, although it is easier for applicable software to deal with all 32 bits all pointer

12.8.8 Buffer Usage Word

This word has to be present right after each Packet Buffer (at address Buff Address + BUFSIZE). It is a 32-bit word with the following function:

Figure 12–46. Buffer Usage Table Structure

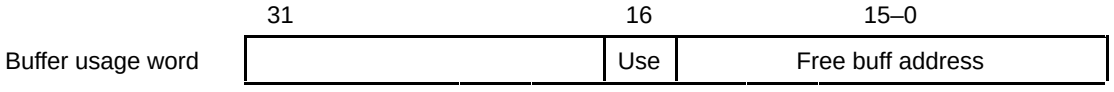


Table 12–20. Buffer Word

Bit	Name	Function
16	USE	0: buffer is referenced by 1 descriptor 1: buffer is referenced by 2 descriptors
15 – 0	FREEBUF	Pointer to a free buffer if USE=1

12.9 EIM ESM Functional Description

12.9.1 Main State Machine Description

The Ethernet state machine (ESM) module is responsible for packet routing. Its main task is to wait for an Ethernet packet available in an RX queue, look at its Destination Address, and pass it to a TX queue corresponding to the Destination Address value.

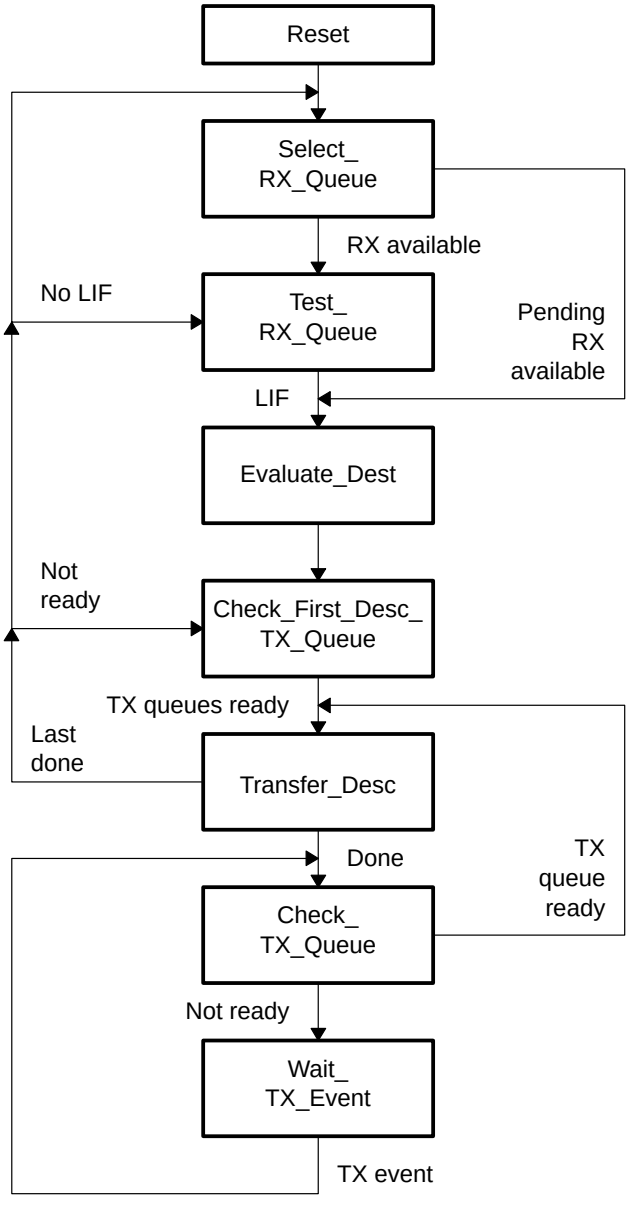
The main loop of the ESM acts under control of interrupt lines of the ENET and CPU modules signaling events in RX queues. Requests are served according to a priority mechanism: CPU operations have priority over ENET requests.

A request is valid when all packets are in the buffer. When a request cannot be served immediately (a destination queue is full), the ESM will pass to the following request.

Once processing of a request has started (*i.e.,* transfer has started), it cannot be aborted and other requests are delayed. If an error has been flagged on a packet, it will not be transferred; instead, it is discarded from the RX queue.

No sanity check is performed by the ESM module. If the packets and descriptors structure are corrupted (no wrap bit, doubled FIF, LIF without FIF, bad pointers, bad Buffer Usage content, etc.), unexpected behavior may occur. It is always possible to reset the EIM to restart with clean conditions.

Figure 12–47. ESM Flow Diagram



12.9.2 Reset State

The state machine enters this state when the EIM is reset or when the ESMEN bit of the EIM_CTRL register is cleared. When leaving this state, all state machine internal registers (queues pointers, status, etc.) are cleared.

12.9.3 Select_RX_Queue State

All RX events (new descriptors available) are monitored if enabled by the corresponding bit in ESM control register.

Events are registered if they can be processed immediately.

Once an event has been selected, the state machine goes into Test_RX_Queue state. If the packet cannot be routed in following states, (TX queue not ready), the state machine goes back to the Select_RX_Queue and marks the queue as “waiting”.

When several events are available at the same time, the following arbitration scheme is applied:

- CPU event has the highest priority if queue is not marked as *waiting*
- CPU *waiting* event has the highest priority one time over two and the lowest in the other case

Once an RX queue has been selected, the state machine goes into Test_RX_Queue state or in Evaluate_Dest state, if the queue is a waiting one.

12.9.4 Test_RX_Queue State

The current descriptor of the current RX queue is read.

If any error status is set (bits 16 – 20 of DW01), an error flag is set for the current queue (for future use).

If the FIF bit is set, the descriptor address is stored as the Start address of the packet.

If the LIF bit is set, a full packet is available in the RX queue and the state machine goes into Evaluate_Dest state.

In the other case, the state machine goes back to the Select_RX_Queue state.

12.9.5 Evaluate_Dest State

The first descriptor of the packet is read and stored in an internal register.

Six first-bytes of the packet buffer (packet Destination Address) are read and matched with CPU rules when coming from RX-ENET0 queue:

- CPU Destination: Destination Address is matched against the value of DA_CPU register.
- Broadcast: Destination Address is matched against the value FF:FF:FF:FF:FF:FF.
- Logical Address: Destination Address matching is done by ENET module. Result is stored in MISS bit of RX descriptor.
- Multicast Address: Destination Address is matched against the Multicast Filter Mask Register. The Destination Address must verify DA and MVF = MFM.
- Multicast and Logical: This rule is true when Logical Address and Multicast Address rules are both true.

Destination queues are evaluated:

- Packet is sent to TX-CPU if any of the previous rules are activated and if this rule is enabled in the CPU Filtering Control Register.
- Packet is sent to TX-ENET0 if coming from RX-CPU queue.

If these operations have already been performed on the current packet, the machine goes directly in Check_First_Desc_TX_Queue state.

If any descriptor has been flagged in error, this state is skipped and the machine goes directly in Check_First_Desc_TX_Queue state.

12.9.6 Check_First_Desc_TX_Queue State

Current RX queue will be routed to 1, 2, or 0 (Packet is in error) TX queues.

Current descriptor of selected TX queues are read to check ownership.

If ESM has ownership of current descriptor over all involved TX queues (or if there is no TX queue), the machine goes to Transfer_Desc state.

In the other case, RX queue is flagged as “waiting” and the machine goes back to Select_RX_Queue state.

12.9.7 Transfer_Desc State

This state is the core of the ESM operation. Its responsibility is to transfer the RX descriptor pointer to TX descriptors and manage the free buffer entries.

The state manages only one descriptor. The loop over all packet descriptors is done through the Check_TX_Queue state (see Section 12.9.8 on page 12-76).

When entering this state, TX queues are known to be owned by the ESM module.

ESM packet routing is performed in the following three steps:

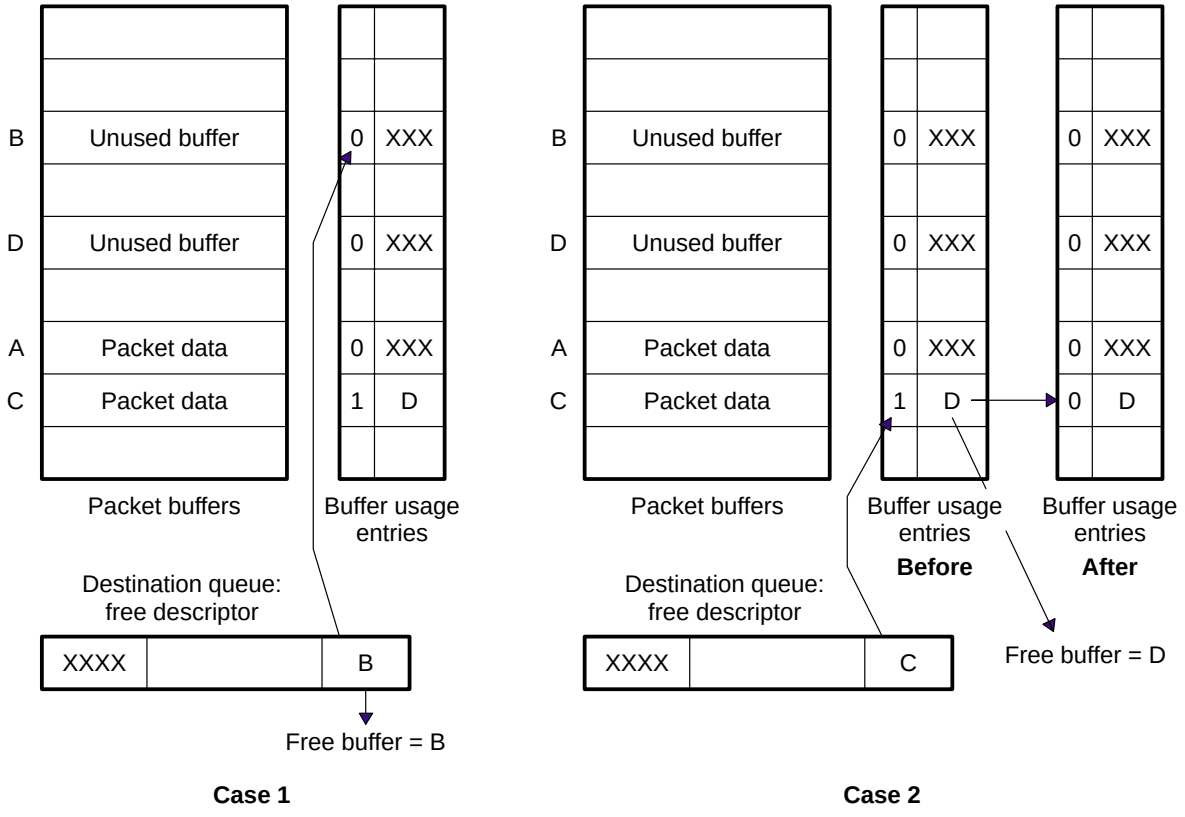
Step 1: Retrieval of Free Buffer in Destination Descriptors

This operation is done on each concerned destination descriptor (0, 1, or 2).

The destination register buffer address is read and its corresponding Buffer Usage Table entry is read. The following conditions apply, depending on the USE value:

- When USE = 0, this buffer is only referenced once.
- When USE = 1, this buffer is referenced in another descriptor. FREEBUF is used as a free buffer and USE is cleared.
- When completed, one free buffer per destination is available.

Figure 12–48. Free Buffer Retrieval



Step 2: Copy of Source Descriptor

The source descriptor is read.

The destination descriptor is built as follows:

- ❏ OWN bit is set to 0 (owner is ENET)
- ❏ WRAP bit is not modified
- ❏ FIF and LIF are copied from RX descriptor
- ❏ RETRY is set to 0000
- ❏ INTRE is set to 1
- ❏ STATUS is set to 0000000
- ❏ PAD_CRC is copied from RX descriptor if RX queue is RX-CPU, and set to zero in other cases
- ❏ BYTES is copies from RX descriptor

If two destinations are selected, the USE flag is set and the Free buffer entry is set to the value of the second stored free buffer retrieved during the previous step.

Figure 12–49. Copy of Source Descriptor to Two Destinations

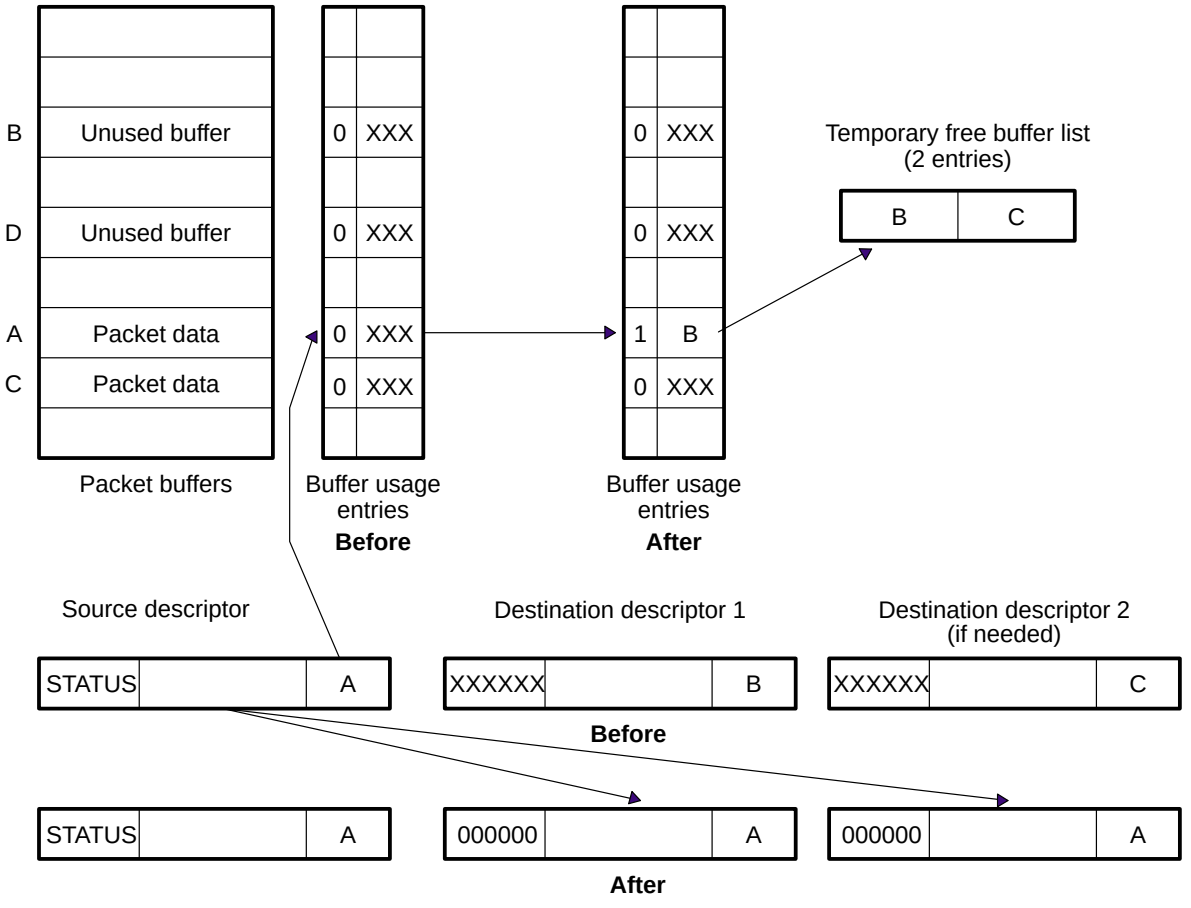
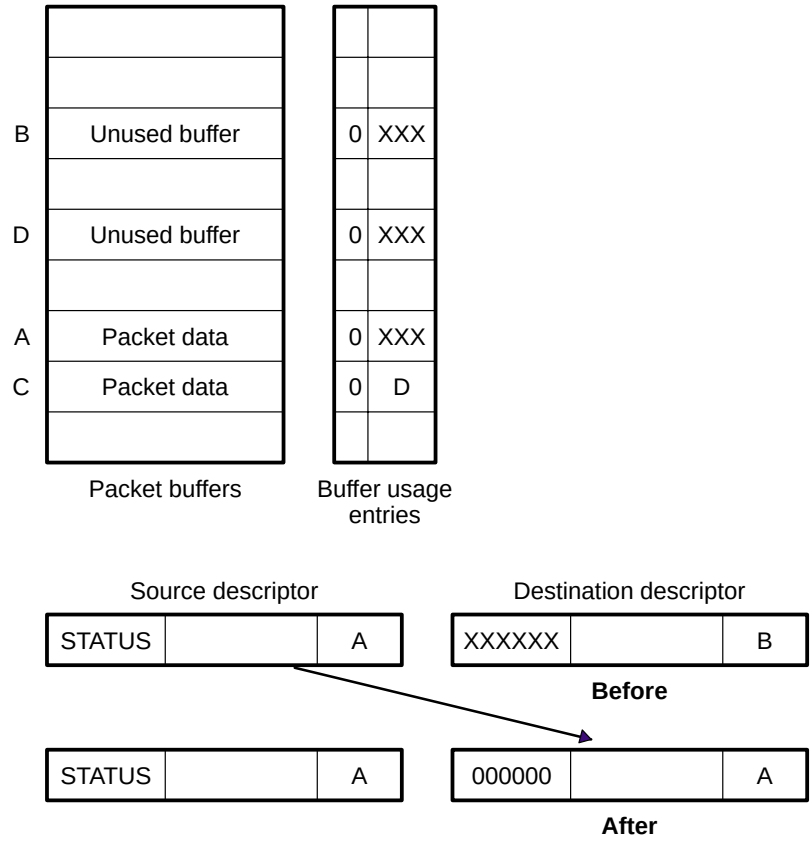


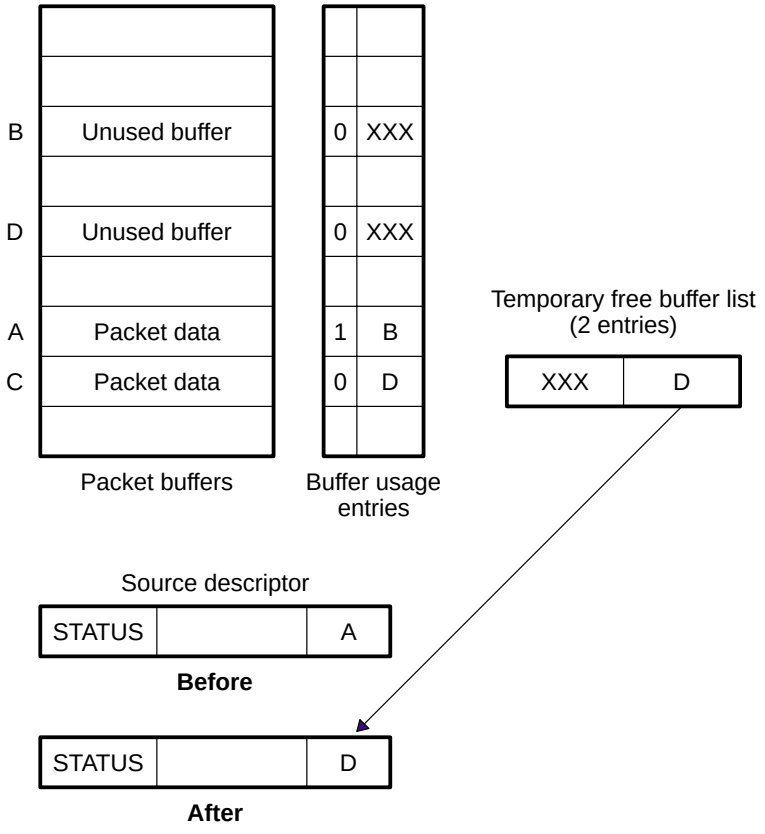
Figure 12–50. Copy of Source Descriptor to a Single Destination



Step 3: Free Buffer Allocation to Source Descriptor

The first retrieved free buffer pointer is stored in the source descriptor. Ownership of the source descriptor is given back to the ENET/CPU.

Figure 12–51. Free Buffer Allocation to Source Descriptor



12.9.8 Check_TX_Queue State

Current descriptor of selected TX queues are read to check ownership.

If the ESM has ownership of current descriptor over all involved TX queues (or if there is no TX queue), the machine goes to Transfer_Desc state.

In the other case, the machine goes to the Wait_TX_Event state.

12.9.9 Wait_TX_Event State

This state generates only a transient before looping back in Transfer_Desc state.

12.10 EIM Operation

12.10.1 Setting Up

EIM initialization by the CPU must be done in the following three steps:

Step 1: Packet Buffer Memory Initialization

The packet buffer memory initialization is as follows:

- ❏ Software has to initialize the descriptor ring structures according to the proposed organization described in Section 12.8.2, *Packets Memory Physical Organization*, or customize it (with more or less buffers on some queues, with a different packet size, etc.).
- ❏ Descriptor flags have to be initialized according to values described in Section 12.4.1, *TX Descriptor Ring*, on page 12-22 and in Section 12.4.2, *RX Descriptor Ring*, on page 12-26.
- ❏ Buffer Usage Entries have to be initialized to zero.

Step 2: ESM Initialization

The ESM initialization is as follows:

- ❏ Packet Memory Registers have to be initialized according to the configuration performed above.
- ❏ All used ports have to be enabled in the ESM control register.
- ❏ ESM itself has to be enabled, if needed.
- ❏ Routing rules for CPU ports have to be initialized.

Step 3: ENET0 Initialization

The ENET0 initialization is as follows:

- ❏ The Mode/Backoff, Seed/Backoff, Count/TX, Flow Go/Flow, and Control/VTYPE/Ring Poll Interval registers have to be set according to the operating configuration. IEEE802-3, *Auto-negotiation*, is beyond the scope of this document and has to be managed by another software layer, talking directly to the MII.
- ❏ Base Address registers have to be set according to Packet RAM mapping.

- ❏ The Logical Address Hash Filter register has to be initialized according to applicable needs.
- ❏ The ENET has to be set in Promiscuous and Logical Addressing modes in the Address Mode Enable Register.

EIM Start-Up

The EIM start-up is as follows:

- ❏ ESM has to be enabled by setting both the ESMEN and CPU_ENET0_EN bits, if needed.
- ❏ The ENET has to be started by setting the ENABLE bit in its MODE register.

12.10.2 Packets Operation

Software has to maintain two pointers to the current RX-CPU and TX-CPU descriptors. At initialization time, they have to be set to the first descriptors of each queue, and they have to be incremented each time a descriptor ownership is given to the ESM.

Packet Transmission

The Packet has to be filled into the RX-CPU queue.

The CPU initially must verify that it has ownership of a descriptor. In any case, a descriptor owned by ESM must be manipulated by software.

FIF, LIF, and CRC_PAD have to be set correctly on each involved descriptor regarding the current packet.

TX interrupt can be programmed to be generated after each descriptor release or only after the release of the last descriptor.

Ownership of descriptors should be set to ESM at once after all packets have been filled in the queue, or descriptor-by-descriptor. Contrary to ENET behavior, if the software is late in giving the ownership to the ESM, this will not lead to underruns, since the packet is switched only when it is fully available in RX queue.

Each time a descriptor is closed, the software has to increment its current TX descriptor pointer. Moreover, software has to advert it to the ESM by writing into the EIM CPURX_RDY register.

There is no feedback about the success or failure in transmitting the packet by the ENET. This responsibility is delegated to upper layer software (like TCP).

Packet Reception

Received packets can be held in wait loops under interrupt control or by polling the ownership bit of the current TX-CPU descriptor.

When a descriptor ownership is given to software, a packet (or a fragment of a packet) is available for read. All descriptors have to be read sequentially up to the detection of the LIF bit (with wait loops on each descriptor for ownership).

When data has been read, the descriptor should be given back to the ENET. This can be done descriptor by descriptor or when all packets have been read; however, it is better to give ownership back for each descriptor as soon as possible to avoid wait loops in the ESM, if receiving bursts for CPU.

Multicast subscribing/unsubscribing can be done at any time by computing the logical address hash filter value and by reprogramming the ENET.

Interrupts

In a standard application, only CPU_Interrupts are useful to process.

ENET0_ERR flags can be monitored to check an eventful problem on Ethernet ports, but no real action can be decided on.

In the same way, ENET0_RX/TX flags give a heartbeat of Ethernet traffic.

12.11 ENET Operation

ENET DMA operations are limited to the 16 KB Packet RAM. The ENET cannot see the whole ARM memory space. Bits 31 – 16 of addresses are not significant.

ENET interrupts are grouped into one EIM interrupt. Interrupt Status is accessible through the EIM Status register.

12.11.1 Setting Up

ENET initialization by CPU must be done in the following three steps:

- ❏ RX and TX queues initialization
- ❏ Configuration registers initialization
- ❏ ENET start by setting the enable bit of the Mode register

The actual values to be filled into descriptors and registers is greatly dependent upon the application. See the previous ENET sections for more information.

IEEE802-3, *Auto Negotiation*, which is beyond the scope of this document, has to be managed by another software layer talking directly to the MII.

12.11.2 Packet Operations

CPU software has to maintain two pointers to the current RX and TX descriptor. At initialization time, they have to be set to the first descriptors of each queue, and they have to be incremented each time a descriptor ownership is given to the ENET. Wraps have to be manipulated by software just as it is done by ENET.

Packet Transmission

The Packet has to be filled into TX queue.

The CPU initially must verify that it has ownership of a descriptor. In any case, a descriptor owned by ENET must not be manipulated by software.

FIF, LIF, and CRC_PAD have to be set correctly on each involved descriptor regarding the current packet.

TX interrupt can be programmed to be generated after each descriptor release or only after the release of the last descriptor.

Ownership of descriptors should be set to ENET at once after all packets have been filled in the queue, or descriptor-by-descriptor. In the last case, it is the responsibility of the software to be able to give back ownership in rate with the descriptor requests made by ENET during transmission.

Failing to insure this point leads to an underrun condition.

Each time a descriptor is closed, the software has to increment its current TX descriptor pointer.

When a descriptor is given back to software by ENET (this can be detected with TX interrupts or by polling the ENET Status bits), statistics can be collected by looking at the status bits in the descriptor and at the System error status flag.

Packet Reception

Received packets can be waited under interrupt control if the RX interrupt has been enabled for each RX descriptor or by polling the ownership bit of the current RX descriptor.

When a descriptor ownership is given to software, a packet (or a fragment of a packet) is available for read. All descriptors have to be read sequentially up to the detection of the LIF bit (with waiting loops on each descriptor for ownership).

When data has been read, the descriptor should be given back to the ENET. This can be done descriptor-by-descriptor or when all packets have been read; however, it is better to give ownership back to each descriptor as soon as possible to be used by ENET for further receptions and minimize the risk of overruns.

When reading a descriptor, statistics can be collected by looking at the status bit in the descriptor and at the System error status flag.

Chapter 13

Initialization Protocol

This chapter explains hardware logic reset, ARM code downloading, and DSP Boot mode.

Topic **Page**

13.1 Initialization Protocol

13.1.1 Hardware Logic Reset

The chip receives one external reset signal (active low) $\overline{\text{RESET}}$ that initializes all digital logic modules. The $\overline{\text{RESET\_OUT}}$ signal can be used to reset external peripherals under control of the ARM.

Table 13–1. Reset Management

Activated Reset Signal	ARM Reset	LEAD Reset	ARM Modules Reset	External Reset ($\overline{\text{RESET\_OUT}}$)
$\overline{\text{RESET}}$	Yes	Yes	Yes	Yes
RST_CMD watchdog	Yes	No	Yes	No
CNTL_RST (lead_reset)	No	Yes	No	No
CNTL_RST (ext_reset)	No	No	No	Yes

CNTL_RST is a control register mapped in the MCU memory space.

After an ARM reset, the ARM program counter points to 0000:0000 address.

After a DSP reset, the DSP program counter begins execution from 0xFF80.

Each ARM module implements in its control register a software reset bit that can be activated by the ARM processor. By default, these bits are set as soon as the ARM reset is activated.

13.1.2 ARM Code Downloading

In debug mode, when external RAM is used to replace Flash or ROM, the program code is downloaded via the JTAG facilities. ROM or EPROM devices must be programmed before assembly.

13.1.3 DSP Boot Mode

When the DSP comes out of reset, it begins execution from 0xFF80. Before releasing the DSP from reset, the ARM can program the DSP memory that is mapped to the 0xFF80 location. This is done by setting the *DSP_mpnmc* and *DSP_apibn* bits in the DSP_REG ARM memory-mapped register. Table 13–2 shows the possible options for the DSP boot memory.

Table 13–2. DSP Boot Memory

DSP_mpnmc	DSP_REG[9] (DSP_APIBN), BSCR[4] (ABMDIS)†	DSP Boot Memory
0	0 (both)	API memory
0	1 (both or any)	On-chip RAM
1	0 (both)	API memory
1	1 (both or any)	External DSP memory

† "Both" implies that DSP_APIBN, as well as ABMDIS, are set to the same value. "Any" implies that one of the fields, DSP_APIBN or ABMDIS, is set.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W

X