Core10100 v4.0

Handbook

Actel
POWER MATTERS

Actel Corporation, Mountain View, CA 94043

Printed in the United States of America

Part Number: 50200077-6

Release: February 2009

Trademarks

Table of Contents

Introduction

Core10100 is a high-speed media access control (MAC) Ethernet controller (Figure 1). It implements Carrier Sense Multiple Access with Collision Detection (CSMA/CD) algorithms defined by IEEE 802.3 for MAC over an Ethernet connection. Communication with an external host is implemented via a set of Control and Status registers and the DMA controller for external shared RAM. For data transfers, Core10100 operates as a DMA master. It automatically fetches from transmit data buffers and stores receive data buffers into external RAM with minimum CPU intervention. Linked list management enables the use of various memory allocation schemes. Internal RAMs are used as configurable FIFO memory blocks, and there are separate memory blocks for transmit and receive processes. The core has a generic host-side interface that connects with external CPUs. This host interface can be configured to work with 8-, 16-, or 32-bit data bus widths with big- or little-endian byte ordering.

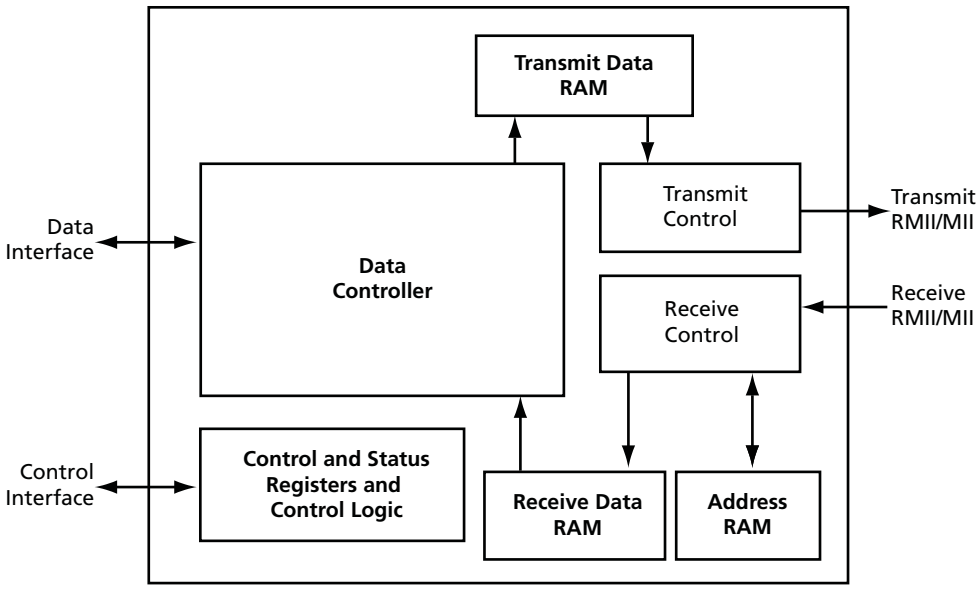

Figure 1 · Core10100 Block Diagram

Figure 2 shows a typical application using Core10100. Typical applications include LAN controllers, AFDX controllers, and embedded systems. Figure 1-1 on page 11 shows the primary blocks of Core10100.

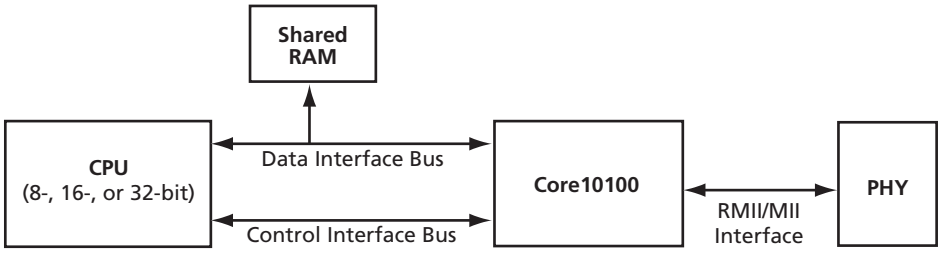

Figure 2 · Typical Core10100 Application

Figure 3 shows an ARM®-based system using Core10100_AHBAPB. This system can be automatically created in SmartDesign.

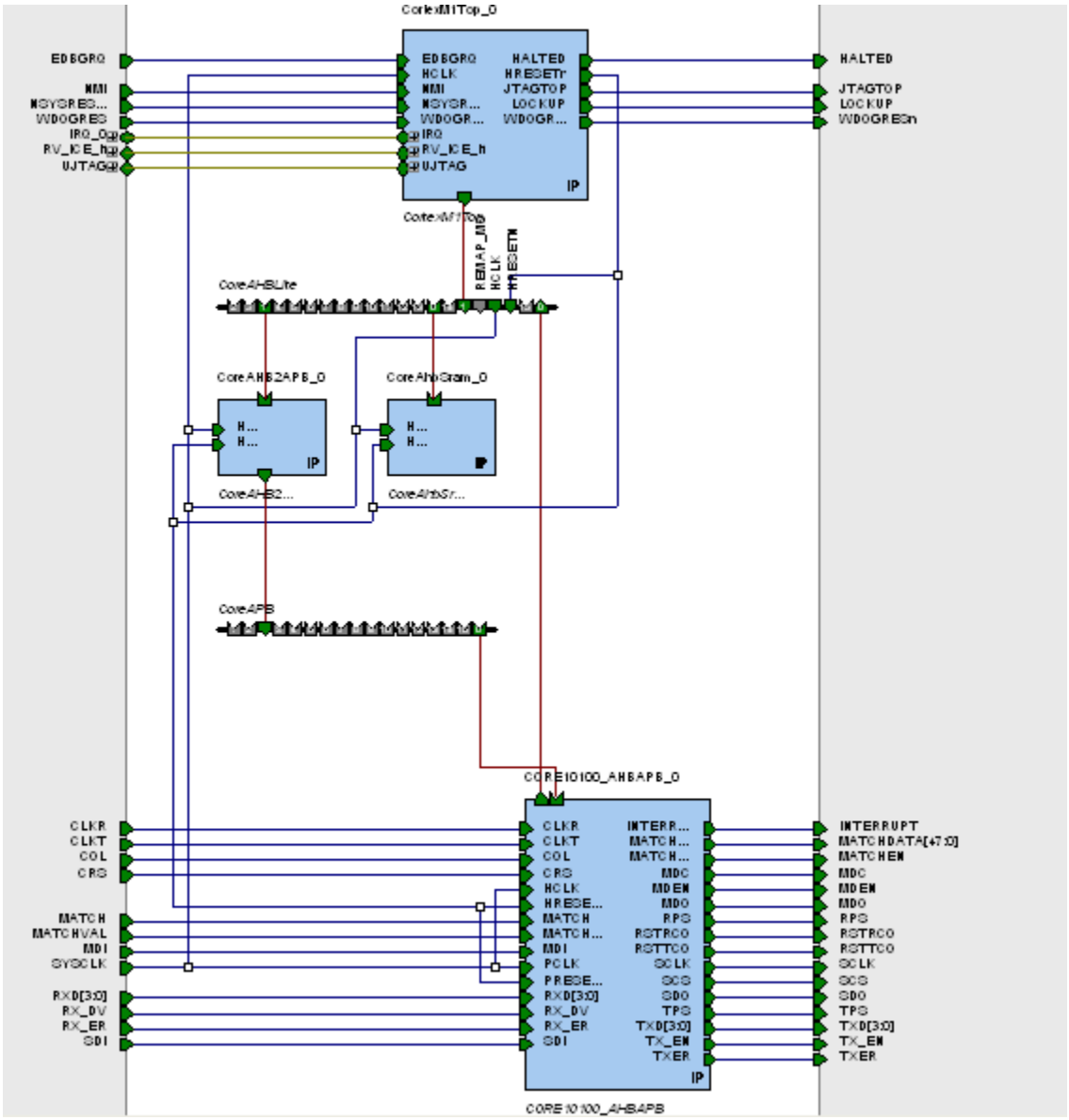

Figure 3 · ARM-Based System Using Core10100_AHBAPB

Supported Device Families

IGLOO®
IGLOOe
ProASIC3
ProASIC3E
ProASIC®3L
Fusion
ProASIC^{PLUS}®
Axcelerator®
RTAX-S

Core Versions

This handbook applies to Core10100 and Core10100_AHB v4.0. The release notes provided with the core list known discrepancies between this handbook and the core release associated with the release notes.

Supported Interfaces

Core10100 is available with the following interfaces:

- Core10100—synchronous CPU and memory interfaces (legacy interface)
- Core10100_AHBAPB—APB slave CPU interface and AHB master memory interface

Actel recommends that new designs using the SmartDesign environment use the Core10100_AHBAPB version of the core. Core10100 is provided for backwards compliance to previous versions of Core10100.

The above interfaces are described in "Interface Descriptions" on page 19.

Device Utilization and Performance

Core10100 can be implemented in the following Actel FPGA devices. Table 1 through Table 6 on page 9 provide the typical utilization and performance data for the core implemented in these devices.

Table 1 · Core10100 Device Utilization and Performance for an 8-Bit Datapath

Family	Cells or Tiles			RAM	Utilization		Performance (MHz)
	Combinatorial	Sequential	Total		Device	Total	
IGLOO®/e	4,330	1,918	6,248	14	AGLE600	45%	30
ProASIC®3 ProASIC3E ProASIC3L	4,173	1,923	6,096	14	A3P6000	44%	49
Fusion	4,215	1,918	6,133	14	AFS600	44%	56
ProASIC^{PLUS}®	5,547	1,958	7,505	29	APA600	35%	27
Axcelerator®	3,087	2,207	5,114	13	AX1000	28%	73
RTAX-S	3,055	2,014	5,069	13	RTAX1000S	28%	57

Table 2 · Core10100 Device Utilization and Performance for a 16-Bit Datapath

Family	Cells or Tiles			RAM	Utilization		Performance (MHz)
	Combinatorial	Sequential	Total		Device	Total	
IGLOO/e	4,715	2,045	6,760	14	AGLE600	49%	30
ProASIC3 ProASIC3E ProASIC3L	4,529	2,050	6,579	14	A3P600	49%	37
Fusion	4,693	2,043	6,736	14	AFS600	49%	36
ProASIC^{PLUS}	6,163	2,087	8,250	29	APA600	38%	26
Axcelerator	3,328	2,170	5,498	13	AX1000	30%	67
RTAX-S	3,316	2,153	5,469	13	RTAX1000S	30%	49

Table 3 · Core10100 Device Utilization and Performance for a 32-Bit Datapath

Family	Cells or Tiles			RAM	Utilization		Performance (MHz)
	Combinatorial	Sequential	Total		Device	Total	
IGLOO/e	4,715	1,963	6,678	14	AGLE600	48%	30
ProASIC3 ProASIC3E ProASIC3L	4,435	1,967	6,402	14	A3P600	46%	36
Fusion	4,597	1,961	6,558	14	AFS600	47%	36
ProASIC^{PLUS}	5,938	1,997	7,935	29	APA600	65%	26
Axcelerator	3,216	2,090	5,306	13	AX1000	29%	55
RTAX-S	3,225	2,089	5,314	13	RTAX1000S	29%	44

Table 4 · Core10100_AHBAPB Device Utilization and Performance for an 8-Bit Datapath

Family	Cells or Tiles			RAM	Utilization		Performance (MHz)
	Combinatorial	Sequential	Total		Device	Total	
IGLOO/e	4,408	1,936	6,344	14	AGLE600	46%	30
ProASIC3 ProASIC3E ProASIC3L	4,234	1,941	6,175	14	A3P600	45%	54
Fusion	4,306	1,939	6,245	14	AFS600	45%	54
ProASIC^{PLUS}	5,656	1,975	7,660	29	APA600	35%	32
Axcelerator	2,974	2,049	5,023	13	AX1000	28%	65
RTAX-S	2,946	2,041	4,987	13	RTAX1000S	27%	46

Table 5 · Core10100_AHBAPB Device Utilization and Performance for a 16-Bit Datapath

Family	Cells or Tiles			RAM	Utilization		Performance (MHz)
	Combinatorial	Sequential	Total		Device	Total	
IGLOO/e	4,749	2,067	6,816	14	AGLE600	49%	30
ProASIC3 ProASIC3E ProASIC3L	4,579	2,065	6,644	14	A3P600	48%	36
Fusion	4,620	2.065	6,685	14	AFS600	48%	46
ProASIC^{PLUS}	6,219	2,106	8,354	29	APA600	39%	25
Axcelerator	3,054	2,166	5,220	13	AX1000	29%	65
RTAX-S	3,036	2,161	5,197	13	RTAX1000S	28%	43

Table 6 · Core10100_AHBAPB Device Utilization and Performance for a 32-Bit Datapath

Family	Cells or Tiles			RAM	Utilization		Performance (MHz)
	Combinatorial	Sequential	Total		Device	Total	
IGLOO/e	5,231	2,199	7,430	14	AGLE600	54%	30
ProASIC3 ProASIC3E ProASIC3L	5,011	2,197	7,208	14	A3P600	53%	35
Fusion	5,169	2,195	7,364	14	AFS600	53%	35
ProASIC^{PLUS}	6,625	2,243	8,897	29	APA600	41%	25
Axcelerator	3,340	2,348	5,688	13	AX1000	31%	56
RTAX-S	3,380	2,359	5,739	13	RTAX1000S	32%	45

Note: Data in the above tables was achieved using Actel Libero® Integrated Design Environment (IDE), using the parameter settings given in Table 7 on page 10. Performance is for Std. speed grade parts, was achieved using the Core10100 macro alone, and represents the system clock (CLKDMA/HCLK) frequency. The CLKR and CLKT clock domains are capable of operating at 25 MHz or 2.5 MHz, depending on the link speed. The CLKCSR/PCLK clock domain is capable of operating in excess of CLKDMA/HCLK.

Table 7 · Parameter Settings

Parameter	Core10100			Core10100_AHBAPB		
	8-Bit	16-Bit	32-Bit	8-Bit	16-Bit	32-Bit
ENDIANESS	0	0	0	1	1	1
ADDRFILTER	1	1	1	0	0	0
FULLDUPLEX	0	0	0	0	0	0
CSRWIDTH APB_DWIDTH	8	16	32	8	16	32
DATAWIDTH AHB_DWIDTH	8	16	32	8	16	32
DATADEPTH AHB_AWIDTH	20	24	32	20	24	32
TFIFODEPTH	11	10	9	11	10	9
RFIFODEPTH	12	11	10	12	11	10
TCDEPTH	1	1	1	1	1	1
RCDEPTH	2	2	2	2	2	2
RMII	1	1	1	1	1	1

Memory Requirements

Core10100 uses FPGA memory blocks. The actual number of memory blocks varies based on the parameter settings. The approximate number of RAM blocks is given by EQ 1 and EQ 2.

IGLOO/e, ProASIC3/E, ProASIC3L, Fusion, Axcelerator, and RTAX-S

$$NRAMS = (DW / 8 \times (2^{TFIFODEPTH} / 512 + 2^{RFIFODEPTH} / 512) + ADDRFILTER$$

EQ 1

where DW is DATAWIDTH or AHB_DWIDTH.

ProASIC[PLUS]

$$NRAMS = (DW / 8 \times (2^{TFIFODEPTH} / 256 + 2^{RFIFODEPTH} / 256) + 2 \times ADDRFILTER$$

EQ 2

where DW is DATAWIDTH or AHB_DWIDTH.

The number of RAM blocks may vary slightly from the above equations due to the Synthesis tool selecting different aspect ratios and inferring memories for internal logic.

Functional Block Descriptions

Core10100 architecture, shown in Figure 1-1, consists of the functional blocks described in this section.

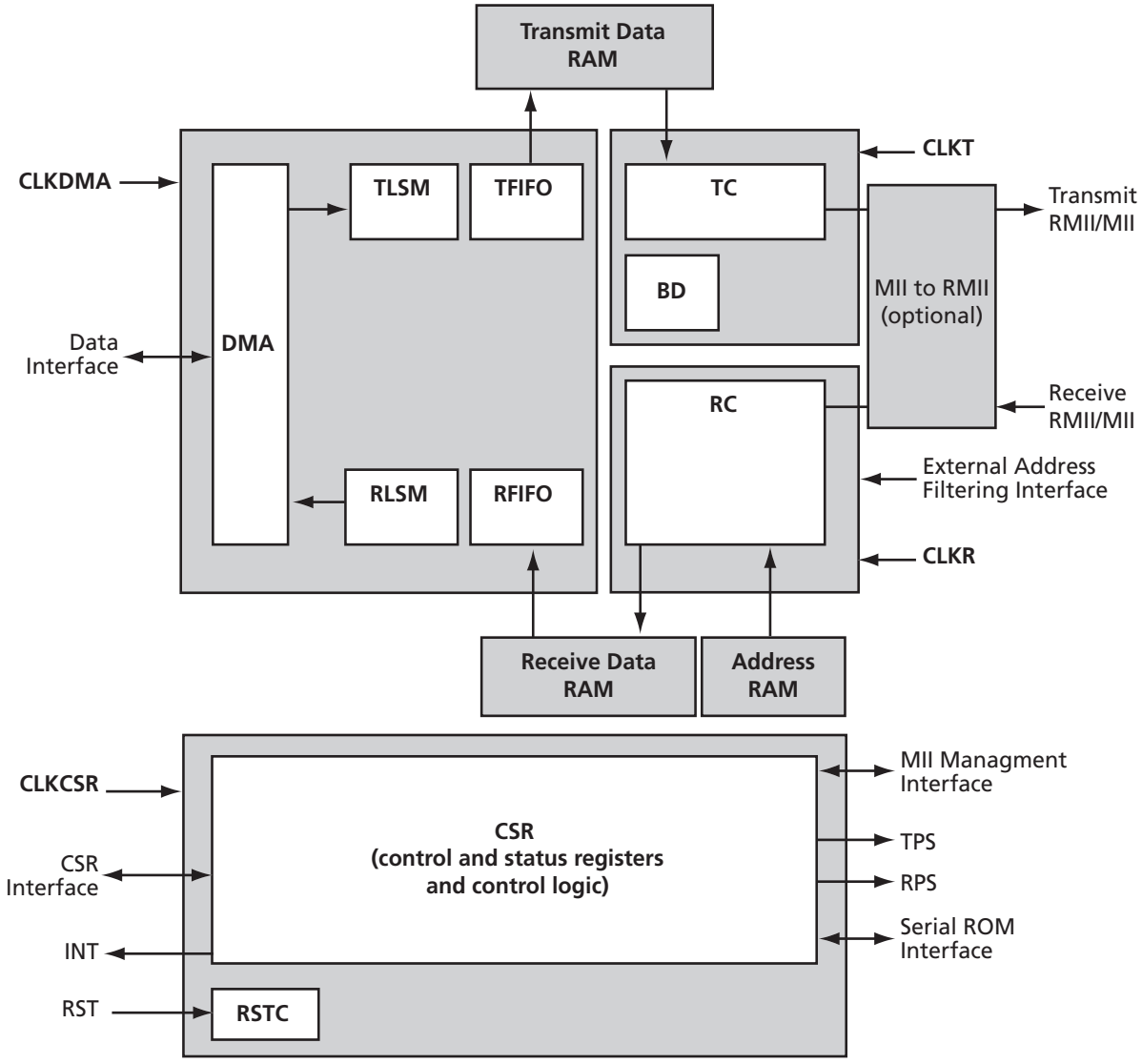

Figure 1-1 · Core 10100 Architecture

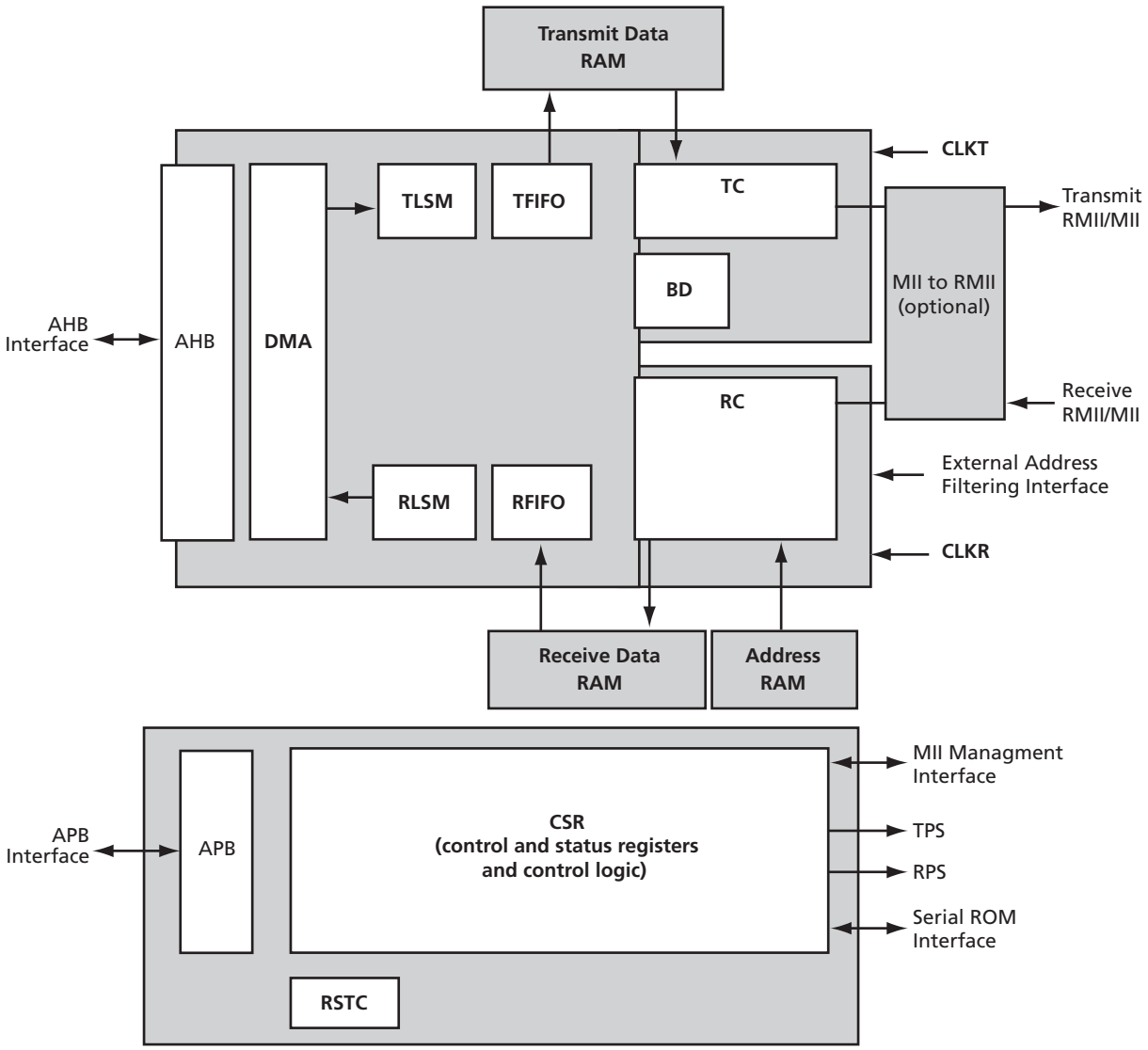

Figure 1-2 · Core10100_AHBAPB Architecture

AHB – AHB Interface

The AHB block implements an AHB master function, allowing the DMA controller to access memory on the AHB bus.

APB – APB Interface

This APB block implements an APB slave interface, allowing the CPU to access the CSR registers set.

CSR – Control/Status Register Logic

The CSR component is used to control Core10100 operation by the host. It implements the CSR register set and the interrupt controller. It also provides a generic host interface supporting 8-, 16-, and 32-bit transfer. The CSR component operates synchronously with the clkcsr clock from the host CSR interface. The CSR also provides a Serial ROM interface and MII Management interface. The host can access these two interfaces via read/write CSR registers.

DMA – Direct Memory Access Controller

The direct memory access controller implements the host data interface. It services both the receive and transmit channels. The TLSM and TFIFO have access to one DMA channel. The RLSM and RFIFO have access to the other DMA channel. The direct memory access controller operates synchronously with the CLKDMA clock from the host data interface.

TLSM – Transmit Linked List State Machine

The transmit linked list state machine implements the descriptor/buffer architecture of Core10100. It manages the transmit descriptor list and fetches the data prepared for transmission from the data buffers into the transmit FIFO. The transmit linked list state machine controller operates synchronously with the CLKDMA clock from the host data interface.

TFIFO – Transmit FIFO

The transmit FIFO is used for buffering data prepared for transmission by Core10100. It provides an interface for the external transmit data RAM working as FIFO memory. It fetches the transmit data from the host via the DMA interface. The FIFO size can be configured via the core parameters. The transmit FIFO controller operates synchronously with the CLKDMA clock from the host data interface.

TC – Transmit Controller

The transmit controller implements the 802.3 transmit operation. From the network side, it uses the standard 802.3 MII interface for an external PHY device. The TC unit reads transmit data from the external transmit data RAM, formats the frame, and transmits the framed data via the MII. The transmit controller operates synchronously with the CLKT clock from the MII interface.

BD – Backoff/Deferring

The backoff/deferring controller implements the 802.3 half-duplex operation. It monitors the status of the Ethernet bus and decides whether to perform a transmit or backoff/deferring of the data via the MII. It operates synchronously with the CLKT clock from the MII interface.

RLSM – Receive Linked List State Machine

The receive linked list state machine implements the descriptor/buffer architecture of Core10100. It manages the receive descriptor list and moves the data from the receive FIFO into the data buffers. The receive linked list state machine controller operates synchronously with the clkdma clock from the host data interface.

RFIFO – Receive FIFO

The receive FIFO is used for buffering data received by Core10100. It provides an interface for the external RAM working as FIFO memory. The FIFO size can be configured by the generic parameters of the core. The receive FIFO controller operates synchronously with the CLKDMA clock from the host data interface.

RC – Receive Controller

The receive controller implements the 802.3 receive operation. From the network side it uses the standard 802.3 MII interface for an external PHY device. The RC block transfers data received from the MII to the receive data RAM. It supports internal address filtering. It also supports an external address filtering interface. The receive controller operates synchronously with the CLKR clock from the MII interface.

RSTC – Reset Controller

The reset controller is used to reset all components of Core10100. It generates a reset signal asynchronous to all clock domains in the design from the external reset line and software reset.

Memory Blocks

There are three internal memory blocks required for the proper operation of Core10100:

- Receive data RAM – Synchronous RAM working as receive FIFO
- Transmit data RAM – Synchronous RAM working as transmit FIFO
- Address RAM – Synchronous RAM working as MAC address memory

RMII – RMII to MII Interface

The Reduced Media Independent Interface (RMII) reduces the number of pins required for connecting to the PHY from 16 to 8.

2 Tool Flows

Licensing

Core10100 is licensed in two ways: Obfuscated and RTL. Depending on your license, tool flow functionality may be limited.

Obfuscated

Complete RTL code is provided for the core, enabling the core to be instantiated with SmartDesign. Simulation, Synthesis, and Layout can be performed with Actel Libero® Integrated Design Environment (IDE). The RTL code for the core is obfuscated,[1] and the some of the testbench source files are not provided. They are precompiled into the compiled simulation library instead.

RTL

Complete RTL source code is provided for the core and testbenches.

1. *Obfuscated means the RTL source files have had formatting and comments removed, and all instance and net names have been replaced with random character sequences.*

The core can be configured using the configuration GUI within SmartDesign, as shown in Figure 2-1 and Figure 2-2 on page 17.

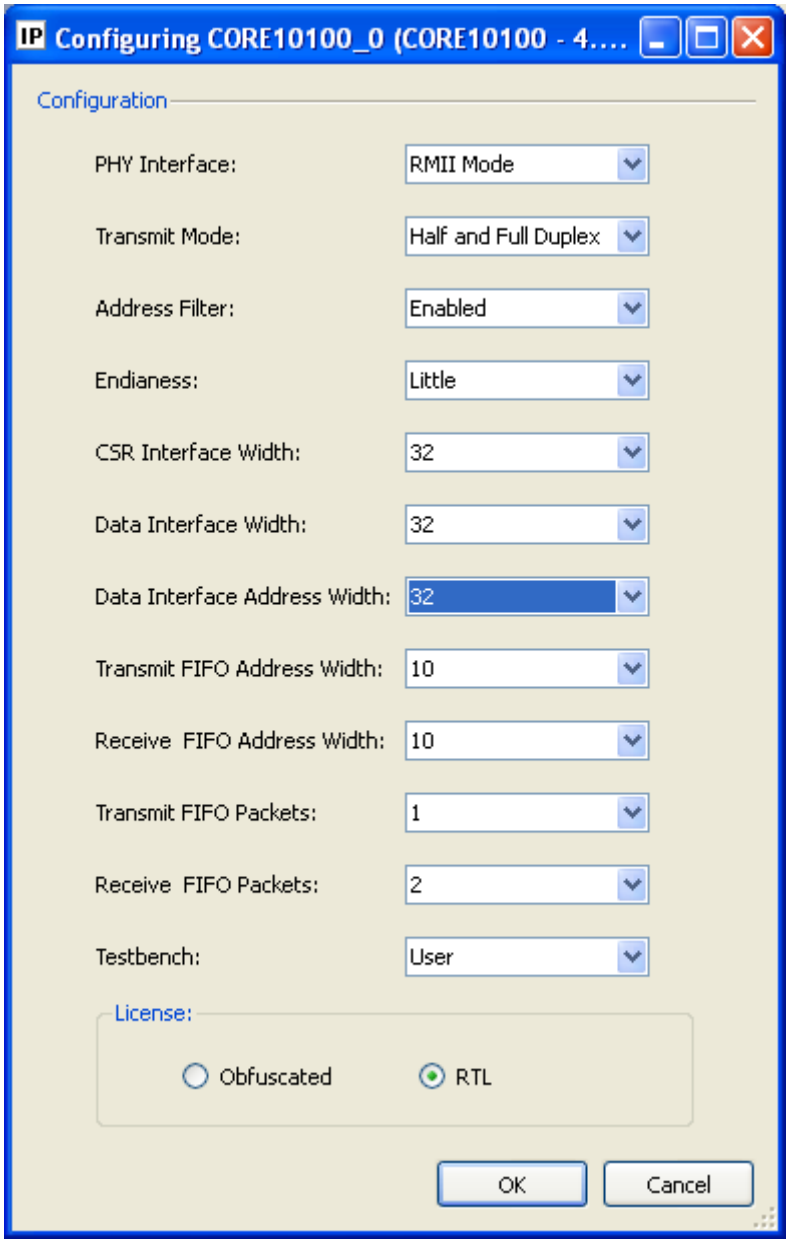

Figure 2-1 · Core10100 Configuration within SmartDesign

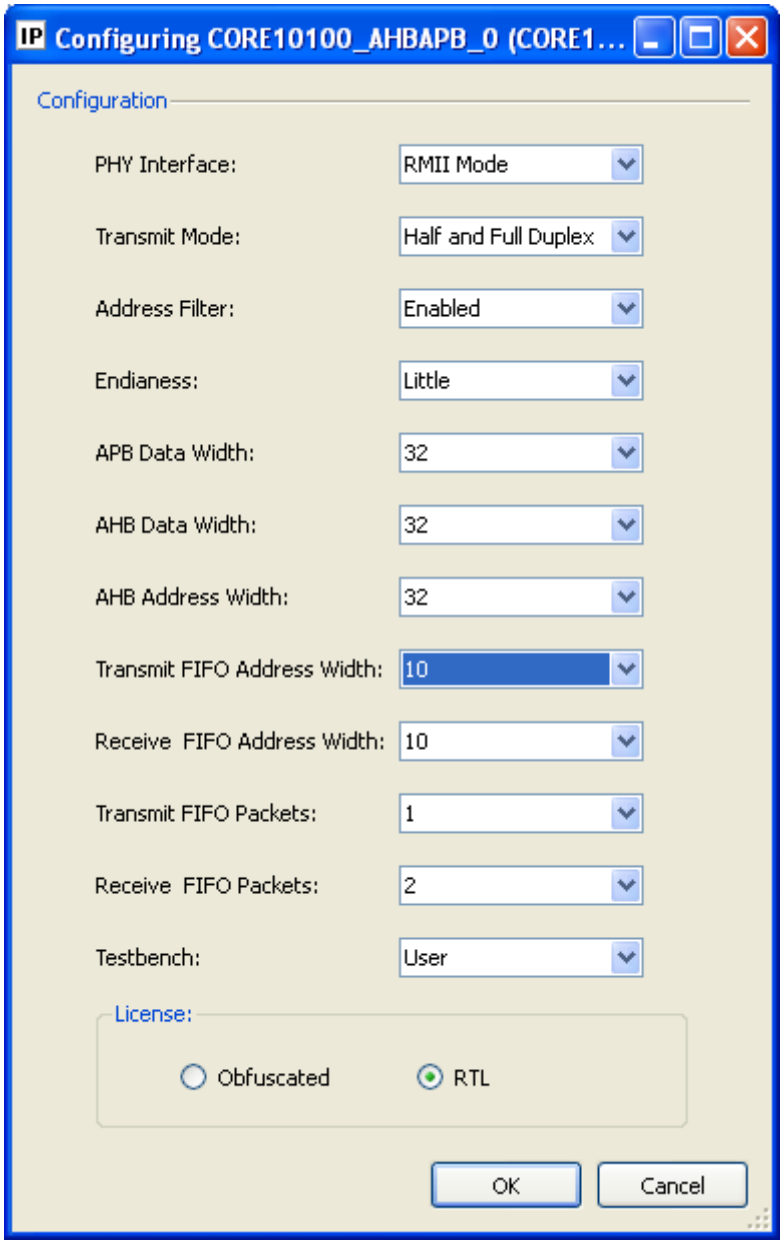

Figure 2-2 · Core10100_AHBAPB Configuration within SmartDesign

Importing into Libero IDE

Core10100 is available for download to the SmartDesign IP Catalog, via the Libero IDE web repository. For information on using SmartDesign to instantiate, configure, connect, and generate cores, refer to the Libero IDE online help.

Simulation Flows

To run simulations, select the user testbench within the SmartDesign Core10100 configuration GUI, right-click, and select **Generate Design** (see Figure 2-1 on page 16).

When SmartDesign generates the design files, it will install the appropriate testbench files. To run the simulation, simply **set the design root to the Core10100 instantiation in the Libero IDE design hierarchy pane** and click the **Simulation** icon in the Libero IDE Design Flow window. This will invoke Model*Sim* and automatically run the simulation.

Synthesis in Libero IDE

Set the design root appropriately and click the **Synthesis** icon in the Libero IDE. The synthesis window appears, displaying the Synplicity® project. Set Synplicity to use the Verilog 2001 standard if Verilog is being used. To perform synthesis, click the **Run** icon.

"Timing Constraints" on page 70 details the recommended timing constraints that should be used during Synthesis.

Place-and-Route in Libero IDE

Having set the design route appropriately and run Synthesis, click the **Layout** icon in Libero IDE to invoke Designer. Core10100 requires no special place-and-route settings.

"Timing Constraints" on page 70 details the recommended timing constraints that should be used during Layout.

3 Interface Descriptions

Core10100 is available with the following interfaces:

- CSR Interface
- AMBA

Both Core10100 and Core10100_AHBAPB share a common set of set signals to the backend physical layer (PHY) and address filtering interface.

Parameters on Core10100

Table 3-1 details the parameters on Core10100.

Table 3-1 · Core10100 Parameters

Parameter	Values	Default Value	Description
FAMILY	0 to 99	17	Must be set to match the supported FPGA family: 11 – Axcelerator 12 – RTAX-S 14 – ProASIC^{PLUS} 15 – ProASIC3 16 – ProASIC3E 17 – Fusion 20 – IGLOO 21 – IGLOOe 22 – ProASIC3L
FULLDUPLEX	0 to 1	0	This controls the core's support of half-duplex operation. 0 – Half- and full-duplex operation supported 1 – Full-duplex only When set to '1', the collision and backoff logic required to support half-duplex operation is omitted, reducing the size of the core.
ENDIANESS	0 to 2	1	Sets the endianess of the core: 0 – Programmable by software 1 – Little 2 – Big When set to a nonzero value, the size of the core is reduced.
ADDRFILTER	0 to 1	1	Enables the internal address filter RAM. 0 – Internal address filter RAM disabled 1 – Internal address filter RAM enabled
DATADEPTH	20 to 32	32	Sets the width of the address bus used to interface to the system memory.
DATAWIDTH	8, 16, 32	32	Sets the width of the data bus used to interface to the system memory.

Table 3-1 · Core10100 Parameters (continued)

Parameter	Values	Default Value	Description
CSRWIDTH	8, 16, 32	32	Sets the width of the data bus used to access the registers within the core.
TCDEPTH	1 to 4	1	Defines the maximum number of frames that can reside in the transmit FIFO at one time. The maximum number of frames that reside in the TX FIFO at one time is $2^{TCDEPTH}$.
RCDEPTH	1 to 4	2	Defines the maximum number of frames that can reside in the receive FIFO at one time. The maximum number of frames that reside in the RX FIFO at one time is $2^{RCDEPTH}$ -1.
TFIFODEPTH	7 to 12	9	Sets the size of the internal FIFO used to buffer transmit data. The size is $2^{TFIFODEPTH}$ × AHB_DWIDTH / 8 bytes. The transmit FIFO size must be greater than TCDEPTH times the maximum permitted frame size.
RFIFODEPTH	7 to 12	10	Sets the size of the internal FIFO used to buffer receive data. The size is $2^{RFIFODEPTH}$ × AHB_DWIDTH / 8 bytes. The receive FIFO size must be greater than RCDEPTH times the maximum permitted frame size.
RMII	0, 1	0	When set to 1, the core supports RMII interface. When set to 0, the core supports MII interface.

Parameters on Core10100_AHBAPB

Table 3-2 details the parameters on Core10100_AHBAPB.

Table 3-2 · Core10100_AHBAPB Parameters

Parameter	Values	Default Value	Description
FAMILY	0 to 99	17	Must be set to match the supported FPGA family. 11 – Axcelerator 12 – RTAX-S 14 – ProASIC^{PLUS} 15 – ProASIC3 16 – ProASIC3E 17 – Fusion 20 – IGLOO 21 – IGLOOe
FULLDUPLEX	0 to 1	0	This controls the core's support of half-duplex operation. 0 – Half- and full-duplex operation supported 1 – Full-duplex only When set to '1', the collision and backoff logic required to support half-duplex operation is omitted, reducing the size of the core.

Table 3-2 · Core10100_AHBAPB Parameters (continued)

Parameter	Values	Default Value	Description
ENDIANESS	0 to 2	1	Sets the endianess of the core. 0 – Programmable by software 1 – Little 2 – Big When set to nonzero, the size of the core is reduced.
ADDRFILTER	0 to 1	1	Enables the internal address filter RAM. 0 – Internal address filter RAM disabled 1 – Internal address filter RAM enabled
AHB_AWIDTH	20 to 32	32	Sets the width of the AHB address bus used to interface to the system memory.
AHB_DWIDTH	8, 16, 32	32	Sets the width of the AHB data bus used to interface to the system memory.
APB_DWIDTH	8, 16, 32	32	Sets the width of the APB data bus used to access the registers within the core.
TCDEPTH	1 to 4	1	Defines the maximum number of frames that can reside in the transmit FIFO at one time. The maximum number of frames that reside in the TX FIFO at one time is $2^{TCDEPTH}$.
RCDEPTH	1 to 4	2	Defines the maximum number of frames that can reside in the receive FIFO at one time. The maximum number of frames that reside in the RX FIFO at one time is $2^{RCDEPTH}$ -1.
TFIFODEPTH	7 to 12	9	Sets the size of the internal FIFO used to buffer transmit data. The size is $2^{TFIFODEPTH}$ × AHB_DWIDTH / 8 bytes. The transmit FIFO size must be greater than TCDEPTH times the maximum permitted frame size.
RFIFODEPTH	7 to 12	10	Sets the size of the internal FIFO used to buffer receive data. The size is $2^{RFIFODEPTH}$ × AHB_DWIDTH / 8 bytes. The receive FIFO size must be greater than RCDEPTH times the maximum permitted frame size.
RMII	0, 1	0	When set to 1, the core supports RMII interface. When set to 0, the core supports MII interface.

CSR Interface Signals

Table 3-3 lists the signals included on the Core10100 core.

Table 3-3 · Core10100 Signals

Name	Type	Polarity	Description
Control and Status Register Interface			
CLKCSR	In	Rise	CSR clock
CSRREQ_	In	HIGH	This signal is set by a host to request a data transfer on the CSR interface. It can be a read or a write request, depending on the value of the CSRRW signal.
CSRRW	In	HIGH	This signal indicates the type of request on the CSR interface. Setting CSRRW indicates a read operation, and clearing it indicates a write operation.
CSRBE	In	CSRWIDTH/8	This signal is the data byte enable to indicate which byte lanes of CSRDATAI or CSRDATAO are the valid data bytes. Each bit of the CSRBE controls a single byte lane. All CSRBE signal combinations are allowed.
CSRDATAI	In	CSRWIDTH	The write data is provided by the system on the CSRDATAI inputs during the write request.
CSRADDR	In	8	The CSRADDR receives the address of an individual CSR data transaction. The meaning of CSRADDR depends on the CSRWIDTH parameter. For CSRWIDTH = 32 (32-bit interface), only the CSRADDR bits from 6 down to 2 are significant. The addresses are longword-aligned (32-bit) in this mode. For CSRWIDTH = 16 (16-bit interface), the CSRADDR bits from 6 down to 1 are significant. The addresses are word-aligned (16-bit) in this mode. For CSRWIDTH = 8 (8-bit interface), all bits of CSRADDR are significant. The addresses are byte-aligned (8-bit) in this mode.
CSRACK	Out	HIGH	The CSRACK signal indicates either that valid data is present on the CSRDATAO outputs during a read request or that the CSRDATAI inputs have been sampled during a write request. The current version of Core10100 has the CSRACK signal statically tied to logic 1—Core10100 responds to reads and writes immediately.
CSRDATAO	Out	CSRWIDTH	The CSRDATAO signal provides the read data in response to a read request.
Data Interface			
CLKDMA	In	Rise	Data clock
DATAACK	In	HIGH	The DATAACK input is an acknowledge signal supplied by the host in response to the MAC's request. In the case of a read operation, DATAACK indicates valid data is on the DATAI input. The DATAI input must be stable while DATAACK is set. In the case of a write operation, setting DATAACK indicates that the host is ready to fetch the data supplied by Core10100 on the DATAO output. Regardless of the current transaction type (write or read), a data transfer occurs on every rising edge of CLKDMA on which both DATAREQ and DATAACK are set. The DATAACK signal can be asserted or deasserted at any clock cycle, even in the middle of a burst transfer.
DATAI	In	DATAWIDTH	The read data must be provided on the DATAI input by the system in response to a read request.

Table 3-3 · Core10100 Signals (continued)

Name	Type	Polarity	Description
DATAREQ_	Out	HIGH	This signal is set by Core10100 to put a request for the data transfer on the interface. While DATAREQ remains active, the DATARW signal is stable—there is no transition on DATARW.
DATARW	Out	HIGH	The DATARW output indicates the type of request on the data interface. When set, it indicates a read operation; when cleared, it indicates a write operation.
DATAEOB	Out	HIGH	The DATAEOB output is an “end-of-burst” signal used for burst transactions. When set, it indicates the last data transfer for a current burst; when cleared, it indicates that there will be more data transfers.
DATAO	Out	DATAWIDTH	Data to be written is provided by Core10100 on DATAO during a write request.
DATAADDR	Out	DATADEPTH	This signal addresses the external memory space for a data transaction. The meaning of the DATAADDR bits depends on the DATAWIDTH parameter. For DATAWIDTH = 32 (32-bit interface), only DATAADDR bits DATADEPTH–1 down to 2 are significant. The addresses are longword-aligned (32-bit) in this mode. For DATAWIDTH = 16 (16-bit interface), the DATAADDR bits from DATADEPTH–1 down to 1 are significant. The addresses are word-aligned (16-bit) in this mode. For DATAWIDTH = 8 (8-bit interface), all bits of DATAADDR are significant. The addresses are byte-aligned (8-bit) in this mode.

Common Interface Signals

The following signals are included on both the Core10100 and Core10100_AHBAPB cores.

Table 3-4 · Signals Included in Core10100 and Core10100_AHBAPB

Name	Type	Polarity / Bus Size	Description
General Host Interface Signal			
RSTCSR	In	HIGH	Host-side reset
INT	Out	HIGH	Interrupt
RSTTCO	Out	HIGH	Transmit side reset
RSTRCO	Out	HIGH	Receive side reset
TPS	Out	HIGH	Transmit process stopped
RPS	Out	HIGH	Receive process stopped
Serial ROM Interface			
SDI	In	1	Serial data
SCS	Out	1	Serial chip select
SCLK	Out	1	Serial clock output
SDO	Out	1	Serial data output
External Address Filtering Interface			
MATCH	In	HIGH	External address match When HIGH, indicates that the destination address on the MATCHDATA port is recognized by the external address-checking logic and that the current frame must be received by Core10100. When LOW, indicates that the destination address on the MATCHDATA port is not recognized and that the current frame should be discarded. Note that the match signal should be valid only when the MATCHVAL signal is HIGH.
MATCHVAL	In	HIGH	External address match valid When HIGH, indicates that the MATCH signal is valid.
MATCHEN	Out	HIGH	External match enable When HIGH, indicates that the MATCHDATA signal is valid. The MATCHEN output should be used as an enable signal for the external address-checking logic. It is HIGH for at least four CLKR clock periods to allow for the latency of external address-checking logic.
MATCHDATA	Out	48	External address match data The MATCHDATA signal represents the 48-bit destination address of the received frame. Note that the MATCHDATA signal is valid only when the MATCHEN signal is HIGH.
RMII/MII PHY Interface			
CLKT	In	Rise	Clock for transmit operation This must be a 25 MHz clock for a 100 Mbps operation or a 2.5 MHz clock for a 10 Mbps operation. This input is only used in MII mode. In RMII mode, this input will be grounded by SmartDesign.

Table 3-4 · Signals Included in Core10100 and Core10100_AHBAPB (continued)

Name	Type	Polarity / Bus Size	Description
CLKR	In	Rise	Clock for receive operation This must be a 25 MHz clock for a 100 Mbps operation or a 2.5 MHz clock for a 10 Mbps operation. This input is only used in MII mode. In RMII mode, this input will be grounded by SmartDesign.
RX_ER	In	HIGH	Receive error If RX_ER is asserted during Core10100 reception, the frame is received and status of the frame is updated with RX_ER. The RX_ER signal must be synchronous to the CLKR receive clock.
RX_DV	In	HIGH	Receive data valid signal The PHY device must assert RX_DV when a valid data nibble is provided on the RXD signal. The RX_DV signal must be synchronous to the CLKR receive clock.
COL	In	HIGH	Collision detected This signal must be asserted by the PHY when a collision is detected on the medium. It is valid only when operating in a half-duplex mode. When operating in a full-duplex mode, this signal is ignored by Core10100. The COL signal is not required to be synchronous to either CLKR or CLKT. The COL signal is sampled internally by the CLKT clock.
CRS	In	HIGH	Carrier sense This signal must be asserted by the PHY when either a receive or transmit medium is non-idle. The CRS signal is not required to be synchronous with either CLKR or CLKT.
MDI	In	1	MII management data input The state of this signal can be checked by reading the CSR9.19 bit.
RXD	In	4	Receive data recovered and decoded by PHY The RXD[0] signal is the least significant bit. The RXD bus must be synchronous to the CLKR in MII mode. In RMII mode, RXD[1:0] is used and RXD[3:2] will be grounded by SmartDesign. In RMII mode, RXD[1:0] is synchronous to RMII_CLK.
TX_EN	Out	HIGH	Transmit enable When asserted, indicates valid data for the PHY on the TXD port. The TX_EN signal is synchronous to the CLKT transmit clock.
TXER	Out	HIGH	Transmit error The current version of Core10100 has the TXER signal statically tied to logic 0 (no transmit errors).
MDC	Out	Rise	MII management clock This signal is driven by the CSR9.16 bit.
MDO	Out	1	MII management data output This signal is driven by the CSR9.18 bit.
MDEN	Out	HIGH	MII management buffer control

Table 3-4 · Signals Included in Core10100 and Core10100_AHBAPB (continued)

Name	Type	Polarity / Bus Size	Description
TXD	Out	4	Transmit data The TXD[0] signal is the least significant bit. In RMII mode TXD[1:0] is used. In RMII mode, TXD[1:0] is synchronous to RMII_CLK. The TXD bus is synchronous to the CLKT in MII mode.
RMII_CLK	In	Rise	50 MHz ± 50 ppm clock source shared with RMII PHY. This input is used only in RMII mode. In MII mode, this input will be grounded by SmartDesign.
CRS_DV	In	High	Carrier sense/receive data valid for RMII PHY

AHB/APB Interface Signals

Table 3-5 lists the signals included in the Core10100_AHBAPB core.

Table 3-5 · Core10100_AHBAPB Signals

Name	Type	Description
APB Interface (CPU register access)		
PCLK	In	APB clock
PRESETN	In	APB reset (active low and asynchronous)
PSEL	In	APB select
PENABLE	In	APB enable
PWRITE	In	APB write
PADDR	In [7:0]	APB address
PWDATA	In [APB_DWIDTH–1:0]	APB write data
PRDATA	Out [APB_DWIDTH–1:0]	APB read data
AHB Interface (memory access)		
HCLK	In	AHB clock
HRESETN	In	AHB reset (active low and asynchronous)
HWRITE	Out	AHB write
HADDR	Out [AHB_AWIDTH–1:0]	AHB address
HREADY	In	AHB ready
HTRANS	Out [1:0]	AHB transfer type
HSIZE	Out [2:0]	AHB transfer size
HBURST	Out [2:0]	AHB burst size
HPROT	Out [3:0]	AHB protection; set to '0000'
HRESP	In [1:0]	AHB response
HWDATA	Out [AHB_DWIDTH–1:0]	AHB data out
HRDATA	In [AHB_DWIDTH–1:0]	AHB data in
HBUSREQ_	Out	AHB bus request
HGRANT	In	AHB bus grant

All signals listed in Table 3-5 conform to the AMBA specification rev. 2.0.

4 Software Interface

Register Maps

Control and Status Register Addressing

The Control and Status registers are located physically inside Core10100 and can be accessed directly by a host via an 8-, 16- or 32-bit interface. All the CSRs are 32 bits long and quadword-aligned. The address bus of the CSR interface is 8 bits wide, and only bits 6–0 of the location code shown in Table 4-1 are used to decode the CSR register address.

Table 4-1 · CSR Locations

Register	Address	Reset Value	Description
CSR0	00H	FE000000H	Bus mode
CSR1	08H	00000000H	Transmit poll demand
CSR2	10H	00000000H	Receive poll demand
CSR3	18H	FFFFFFFFH	Receive list base address
CSR4	20H	FFFFFFFFH	Transmit list base address
CSR5	28H	F0000000H	Status
CSR6	30H	32000040H	Operation mode
CSR7	38H	F3FE0000H	Interrupt enable
CSR8	40H	E0000000H	Missed frames and overflow counters
CSR9	48H	FFF483FFH	MII management
CSR10	50H	00000000H	Reserved
CSR11	58H	FFFE0000H	Timer and interrupt mitigation control

Note: CSR9 bits 19 and 2 reset values are dependent on the MDI and SDI inputs. The above assumes MDI is high and SDI is low.

CSR Definitions

Table 4-2 · Bus Mode Register (CSR0)

Bits 31:24								
Bits 23:16				DBO	TAP			
Bits 15:8			PBL					
Bits 7:0	BLE	DSL					BAR	SWR

Note: The CSR0 register has unimplemented bits (shaded). If these bits are read, they will return a predefined value. Writing to these bits has no effect.

Table 4-3 · Bus Mode Register Bit Functions

Bit	Symbol	Function
CSR0.20	DBO	Descriptor byte ordering mode: 1 – Big-endian mode used for data descriptors 0 – Little-endian mode used for data descriptors
CSR0.(19..17)	TAP	Transmit automatic polling If TAP is written with a nonzero value, Core10100 performs an automatic transmit descriptor polling when operating in suspended state. When the descriptor is available, the transmit process goes into running state. When the descriptor is marked as owned by the host, the transmit process remains suspended. The poll is always performed at the current transmit descriptor list position. The time interval between two consecutive polls is shown in Table 4-4 on page 29.
CSR0.(13..8)	PBL	Programmable burst length Specifies the maximum number of words that can be transferred within one DMA transaction. Values permissible are 0, 4, 8, 16 and 32. When the value 0 is written, the bursts are limited only by the internal FIFO's threshold levels. The width of the single word is equal to the CSRWIDTH generic parameter; i.e., all data transfers always use the maximum data bus width. Note that PBL is valid only for the data buffers. The data descriptor burst length depends on the DATAWIDTH parameter. The rule is that every descriptor field (32-bit) is accessed with a single burst cycle. For DATAWIDTH = 32, the descriptors are accessed with a single 32-bit word transaction; for DATAWIDTH = 16, a burst of two 16-bit words; and for DATAWIDTH = 8, a burst of four 8-bit words.
CSR0.7	BLE	Big/little endian Selects the byte-ordering mode used by the data buffers. 1 – Big-endian mode used for the data buffers 0 – Little-endian mode used for the data buffers
CSR0.(6..2)	DSL	Descriptor skip length Specifies the number of longwords between two consecutive descriptors in a ring structure.
CSR0.1	BAR	Bus arbitration scheme 1 – Transmit and receive processes have equal priority to access the bus. 0 – Intelligent arbitration, where the receive process has priority over the transmit process
CSR0.0	SWR	Software reset Setting this bit resets all internal flip-flops. The processor should write a '1' to this bit and then wait until a read returns a '0', indicating that the reset has completed. This bit will remain set for several clock cycles.

Table 4-4 · Transmit Automatic Polling Intervals

CSR0.(19..17)	10 Mbps	100 Mbps
000	TAP disabled	TAP disabled
001	819 µs	81.9 µs
010	2,450 µs	245 µs
011	5,730 µs	573 µs
100	51.2 µs	5.12 µs
101	102.4 µs	10.24 µs
110	153.6 µs	15.36 µs
111	358.4 µs	35.84 µs

Table 4-5 · Transmit Poll Demand Register (CSR1)

Bits 31:24	TPD(31..24)
Bits 23:16	TPD(23..16)
Bits 15:8	TPD(15..8)
Bits 7:0	TPD(7..0)

Table 4-6 · Transmit Poll Demand Bit Functions

Bit	Symbol	Function
CSR1.(31..0)	TPD	Writing this field with any value instructs Core10100 to check for frames to be transmitted. This operation is valid only when the transmit process is suspended. If no descriptor is available, the transmit process remains suspended. When the descriptor is available, the transmit process goes into the running state.

Table 4-7 · Receive Poll Demand Register (CSR2)

Bits 31:24	RPD(31..24)
Bits 23:16	RPD(23..16)
Bits 15:8	RPD(15..8)
Bits 7:0	RPD(7..0)

Table 4-8 · Receive Poll Demand Bit Functions

Bit	Symbol	Function
CSR2.(31..0)	RPD	Writing this field with any value instructs Core10100 to check for receive descriptors to be acquired. This operation is valid only when the receive process is suspended. If no descriptor is available, the receive process remains suspended. When the descriptor is available, the receive process goes into the running state.

Table 4-9 · Receive Descriptor List Base Address Register (CSR3)

Bits 31:24	RLA(31..24)
Bits 23:16	RLA(23..16)
Bits 15:8	RLA(15..8)
Bits 7:0	RLA(7..0)

Table 4-10 · Receive Descriptor List Base Address Register Bit Functions

Bit	Symbol	Function
CSR3.(31..0)	RLA	Start of the receive list address Contains the address of the first descriptor in a receive descriptor list. This address must be longword-aligned (RLA(1..0) = 00).

Table 4-11 · Transmit Descriptor List Base Address Register (CSR4)

Bits 31:24	TLA(31..24)
Bits 23:16	TLA(23..16)
Bits 15:8	TLA(15..8)
Bits 7:0	TLA(7..0)

Table 4-12 · Transmit Descriptor List Base Address Register Bit Functions

Bit	Symbol	Function
CSR4.(31..0)	TLA	Start of the transmit list address Contains the address of the first descriptor in a transmit descriptor list. This address must be longword-aligned (TLA(1..0) = 00).

Table 4-13 · Status Register (CSR5)

Bits 31:24								
Bits 23:16		TS			RS			NIS
Bits 15:8	AIS	ERI			GTE	ETI		RPS
Bits 7:0	RU	RI	UNF			TU	TPS	TI

Note: The CSR5 register has unimplemented bits (shaded). If these bits are read, they will return a predefined value. Writing to these bits has no effect.

Table 4-14 · Status Register Bit Functions

Bit	Symbol	Function
CSR5.(22..20)	TS	Transmit process state (read-only) Indicates the current state of a transmit process: 000 – Stopped; RESET or STOP TRANSMIT command issued 001 – Running, fetching the transmit descriptor 010 – Running, waiting for end of transmission 011 – Running, transferring data buffer from host memory to FIFO 100 – Reserved 101 – Running, setup packet 110 – Suspended; FIFO underflow or unavailable descriptor 111 – Running, closing transmit descriptor
CSR5.(19..17)	RS	Receive process state (read-only) Indicates the current state of a receive process: 000 – Stopped; RESET or STOP RECEIVE command issued 001 – Running, fetching the receive descriptor 010 – Running, waiting for the end-of-receive packet before prefetch of the next descriptor 011 – Running, waiting for the receive packet 100 – Suspended, unavailable receive buffer 101 – Running, closing the receive descriptor 110 – Reserved 111 – Running, transferring data from FIFO to host memory
CSR5.16	NIS	Normal interrupt summary This bit is a logical OR of the following bits: CSR5.0 – Transmit interrupt CSR5.2 – Transmit buffer unavailable CSR5.6 – Receive interrupt CSR5.11 – General-purpose timer overflow CSR5.14 – Early receive interrupt Only the unmasked bits affect the normal interrupt summary bit. The user can clear this bit by writing a 1. Writing a 0 has no effect.
CSR5.15	AIS	Abnormal interrupt summary This bit is a logical OR of the following bits: CSR5.1 – Transmit process stopped CSR5.5 – Transmit underflow CSR5.7 – Receive buffer unavailable CSR5.8 – Receive process stopped CSR5.10 – Early transmit interrupt Only the unmasked bits affect the abnormal interrupt summary bit. The user can clear this bit by writing a 1. Writing a 0 has no effect.

Table 4-14 · Status Register Bit Functions (continued)

Bit	Symbol	Function
CSR5.14	ERI	Early receive interrupt Set when Core10100 fills the data buffers of the first descriptor. The user can clear this bit by writing a 1. Writing a 0 has no effect.
CSR5.11	GTE	General-purpose timer expiration Gets set when the general-purpose timer reaches zero value. The user can clear this bit by writing a 1. Writing a 0 has no effect.
CSR5.10	ETI	Early transmit interrupt Indicates that the packet to be transmitted was fully transferred into the FIFO. The user can clear this bit by writing a 1. Writing a 0 has no effect.
CSR5.8	RPS	Receive process stopped RPS is set when a receive process enters a stopped state. The user can clear this bit by writing a 1. Writing a 0 has no effect.
CSR5.7	RU	Receive buffer unavailable When set, indicates that the next receive descriptor is owned by the host and is unavailable for Core10100. When RU is set, Core10100 enters a suspended state and returns to receive descriptor processing when the host changes ownership of the descriptor. Either a receive-poll-demand command is issued or a new frame is recognized by Core10100. The user can clear this bit by writing a 1. Writing a 0 has no effect.
CSR5.6	RI	Receive interrupt Indicates the end of a frame receive. The complete frame has been transferred into the receive buffers. Assertion of the RI bit can be delayed using the receive interrupt mitigation counter/timer (CSR11.NRP/CSR11.RT). The user can clear this bit by writing a 1. Writing a 0 has no effect.
CSR5.5	UNF	Transmit underflow Indicates that the transmit FIFO was empty during a transmission. The transmit process goes into a suspended state. The user can clear this bit by writing a 1. Writing a 0 has no effect.
CSR5.2	TU	Transmit buffer unavailable When set, TU indicates that the host owns the next descriptor on the transmit descriptor list; therefore, it cannot be used by Core10100. When TU is set, the transmit process goes into a suspended state and can resume normal descriptor processing when the host changes ownership of the descriptor. Either a transmit-poll-demand command is issued or transmit automatic polling is enabled. The user can clear this bit by writing a 1. Writing a 0 has no effect.

Table 4-14 · Status Register Bit Functions (continued)

Bit	Symbol	Function
CSR5.1	TPS	Transmit process stopped TPS is set when the transmit process goes into a stopped state. The user can clear this bit by writing a 1. Writing a 0 has no effect.
CSR5.0	TI	Transmit interrupt Indicates the end of a frame transmission process. Assertion of the TI bit can be delayed using the transmit interrupt mitigation counter/timer (CSR11.NTP/CSR11.TT). The user can clear this bit by writing a 1. Writing a 0 has no effect.

Table 4-15 · Operation Mode Register (CSR6)

Bits 31:24		RA						
Bits 23:16		TTM	SF					
Bits 15:8	TR		ST				FD	
Bits 7:0	PM	PR		IF	PB	HO	SR	HP

Note: The CSR6 register has unimplemented bits (shaded). If these bits are read, they will return a predefined value. Writing to these bits has no effect.

Table 4-16 · Operation Mode Register Bit Functions

Bit	Symbol	Function
CSR6.30	RA	Receive all When set, all incoming frames are received, regardless of their destination address. An address check is performed, and the result of the check is written into the receive descriptor (RDES0.30).
CSR6.22	TTM	Transmit threshold mode 1 – Transmit FIFO threshold set for 100 Mbps mode 0 – Transmit FIFO threshold set for 10 Mbps mode In RMII mode, this bit is also used to select the frequency of both transmit and receive clocks between 2.5 MHz and 25 MHz. This bit can be changed only when a transmit process is in a stopped state.
CSR6.21	SF	Store and forward When set, the transmission starts after a full packet is written into the transmit FIFO, regardless of the current FIFO threshold level. This bit can be changed only when the transmit process is in the stopped state.
CSR6.(15..14)	TR	Threshold control bits These bits, together with TTM, SF, and PS, control the threshold level for the transmit FIFO.

Table 4-16 · Operation Mode Register Bit Functions (continued)

Bit	Symbol	Function
CSR6.13	ST	Start/stop transmit command Setting this bit when the transmit process is in a stopped state causes a transition into a running state. In the running state, Core10100 checks the transmit descriptor at a current descriptor list position. If Core10100 owns the descriptor, then the data starts to transfer from memory into the internal transmit FIFO. If the host owns the descriptor, Core10100 enters a suspended state. Clearing this bit when the transmit process is in a running or suspended state instructs Core10100 to enter the stopped state. Core10100 does not go into the stopped state immediately after clearing the ST bit; it will finish the transmission of the frame data corresponding to current descriptor and then moves to stopped state. The status bits of the CSR5 register should be read to check the actual transmit operation state. Before giving the Stop Transmit command, the transmit state machine in CSR5 can be checked. If the Transmission State machine is in SUSPENDED state, the Stop Transmit command can be given so that complete frame transmission by MAC is ensured.
CSR6.9	FD	Full-duplex mode: 0 – Half-duplex mode 1 – Forcing full-duplex mode Changing of this bit is allowed only when both the transmitter and receiver processes are in the stopped state.
CSR6.7	PM	Pass all multicast When set, all frames with multicast destination addresses will be received, regardless of the address check result.
CSR6.6	PR	Promiscuous mode When set, all frames will be received regardless of the address check result. An address check is not performed.
CSR6.4	IF	Inverse filtering (read-only) If this bit is set when working in a perfect filtering mode, the receiver performs an inverse filtering during the address check process. The “filtering type” bits of the setup frame determine the state of this bit.
CSR6.3	PB	Pass bad frames When set, Core10100 transfers all frames into the data buffers, regardless of the receive errors. This allows the runt frames, collided fragments, and truncated frames to be received.
CSR6.2	HO	Hash-only filtering mode (read-only) When set, Core10100 performs an imperfect filtering over both the multicast and physical addresses. The “filtering type” bits of the setup frame determine the state of this bit.

Table 4-16 · Operation Mode Register Bit Functions (continued)

Bit	Symbol	Function
CSR6.1	SR	Start/stop receive command Setting this bit enables the reception of the frame by Core10100 and the frame is written into the receive FIFO. If the bit is not enabled, then the frame is not written into the receive FIFO. Setting this bit when the receive process is in a stopped state causes a transition into a running state. In the running state, Core10100 checks the receive descriptor at the current descriptor list position. If Core10100 owns the descriptor, it can process an incoming frame. When the host owns the descriptor, the receiver enters a suspended state and also sets the CSR5.7 (receive buffer unavailable) bit. Clearing this bit when the receive process is in a running or suspended state instructs Core10100 to enter a stopped state after receiving the current frame. Core10100 does not go into the stopped state immediately after clearing the SR bit. Core10100 will finish all pending receive operations before going into the stopped state. The status bits of the CSR5 register should be read to check the actual receive operation state.
CSR6.0	HP	Hash/perfect receive filtering mode (read-only) 0 – Perfect filtering of the incoming frames is performed according to the physical addresses specified in a setup frame. 1 – Imperfect filtering over the frames with the multicast addresses is performed according to the hash table specified in a setup frame. A physical address check is performed according to the CSR6.2 (HO, hash-only) bit. When both the HO and HP bits are set, an imperfect filtering is performed on all of the addresses. The “filtering type” bits of the setup frame determine the state of this bit.

Table 4-17 lists all possible combinations of the address filtering bits. The actual values of the IF, HO, and HP bits are determined by the filtering type (FT1–FT0) bits in the setup frame, as shown in Table 4-37 on page 50. The IF, HO, and HP bits are read-only.

Table 4-17 · Receive Address Filtering Modes Summary

PM CSR6.7	PR CSR6.6	IF CSR6.4	HO CSR6.2	HP CSR6.0	Current Filtering Mode
0	0	0	0	0	16 physical addresses – perfect filtering mode
0	0	0	0	1	One physical address for physical addresses and 512-bit hash table for multicast addresses
0	0	0	1	1	512-bit hash table for both physical and multicast addresses
0	0	1	0	0	Inverse filtering
x	1	0	0	x	Promiscuous mode
0	1	0	1	1	Promiscuous mode
1	0	0	0	x	Pass all multicast frames
1	0	0	1	1	Pass all multicast frames

Table 4-18 lists the transmit FIFO threshold levels. These levels are specified in bytes.

Table 4-18 · Transmit FIFO Threshold Levels (bytes)

CSR6.21	CSR6.15..14	CSR6.22 = 1	CSR6.22 = 0
0	00	64	128
0	01	128	256
0	10	128	512
0	11	256	1024
1	xx	Store and forward	Store and forward

Table 4-19 · Interrupt Enable Register (CSR7)

Bits 31:24								
Bits 23:16								NIE
Bits 15:8	AIE	ERE			GTE	ETE		RSE
Bits 7:0	RUE	RIE	UNE			TUE	TSE	TIE

Note: The CSR7 register has unimplemented bits (shaded). If these bits are read, they will return a predefined value. Writing to these bits has no effect.

Table 4-20 · Interrupt Enable Register Bit Function

Bit	Symbol	Function
CSR7.16	NIE	Normal interrupt summary enable When set, normal interrupts are enabled. Normal interrupts are listed below: CSR5.0 – Transmit interrupt CSR5.2 – Transmit buffer unavailable CSR5.6 – Receive interrupt CSR5.11 – General-purpose timer expired CSR5.14 – Early receive interrupt
CSR7.15	AIE	Abnormal interrupt summary enable When set, abnormal interrupts are enabled. Abnormal interrupts are listed below: CSR5.1 – Transmit process stopped CSR5.5 – Transmit underflow CSR5.7 – Receive buffer unavailable CSR5.8 – Receive process stopped CSR5.10 – Early transmit interrupt
CSR7.14	ERE	Early receive interrupt enable When both the ERE and NIE bits are set, early receive interrupt is enabled.

Table 4-20 · Interrupt Enable Register Bit Function (continued)

Bit	Symbol	Function
CSR7.11	GTE	General-purpose timer overflow enable When both the GTE and NIE bits are set, the general-purpose timer overflow interrupt is enabled.
CSR7.10	ETE	Early transmit interrupt enable When both the ETE and AIE bits are set, the early transmit interrupt is enabled.
CSR7.8	RSE	Receive stopped enable When both the RSE and AIE bits are set, the receive stopped interrupt is enabled.
CSR7.7	RUE	Receive buffer unavailable enable When both the RUE and AIE bits are set, the receive buffer unavailable is enabled.
CSR7.6	RIE	Receive interrupt enable When both the RIE and NIE bits are set, the receive interrupt is enabled.
CSR7.5	UNE	Underflow interrupt enable When both the UNE and AIE bits are set, the transmit underflow interrupt is enabled.
CSR7.2	TUE	Transmit buffer unavailable enable When both the TUE and NIE bits are set, the transmit buffer unavailable interrupt is enabled.
CSR7.1	TSE	Transmit stopped enable When both the TSE and AIE bits are set, the transmit process stopped interrupt is enabled.
CSR7.0	TIE	Transmit interrupt enable When both the TIE and NIE bits are set, the transmit interrupt is enabled.

Table 4-21 · Missed Frames and Overflow Counter Register (CSR8)

Bits 31:24										OCO	FOC(10..7)				
Bits 23:16	FOC(6..0)														MFO
Bits 15:8	MFC(15..8)														
Bits 7:0	MFC(7..0)														

Note: The CSR8 register has unimplemented bits (shaded). If these bits are read they will return a predefined value. Writing to these bits has no effect.

Table 4-22 · Missed Frames and Overflow Counter Bit Functions

Bit	Symbol	Function
CSR8.28	OCO	Overflow counter overflow (read-only) Gets set when the FIFO overflow counter overflows. Resets when the high byte (bits 31:24) is read.
CSR8.(27..17)	FOC	FIFO overflow counter (read-only) Counts the number of frames not accepted due to the receive FIFO overflow. The counter resets when the high byte (bits 31:24) is read.
CSR8.16	MFO	Missed frame overflow Set when a missed frame counter overflows. The counter resets when the high byte (bits 31:24) is read.
CSR8.(15..0)	MFC	Missed frame counter (read-only) Counts the number of frames not accepted due to the unavailability of the receive descriptor. The counter resets when the high byte (bits 31:24) is read. The missed frame counter increments when the internal frame cache is full and the descriptors are not available.

Table 4-23 · MII Management and Serial ROM Interface Register (CSR9)

Bits 31:24								
Bits 23:16					MDI	MDEN	MDO	MDC
Bits 15:8								
Bits 7:0					SDO	SDI	SCLK	SCS

Note: The CSR9 register has unimplemented bits (shaded). If these bits are read they will return a predefined value. Writing to these bits has no effect.

Table 4-24 · MII Management and Serial ROM Register Bit Functions

Bit	Symbol	Function
CSR9.19	MDI	MII management data in signal (read-only) This bit reflects the sample on the mdi port during the read operation on the MII management interface.
CSR9.18	MDEN	MII management operation mode 1 – Indicates that Core10100 reads the MII PHY registers 0 – Indicates that Core10100 writes to the MII PHY registers This register bit directly drives the top-level MDEN pin. It is intended to be the active low tristate enable for the MDIO data output.
CSR9.17	MDO	MII management write data The value of this bit drives the mdo port when a write operation is performed.
CSR9.16	MDC	MII management clock The value of this bit drives the mdc port.

Table 4-24 · MII Management and Serial ROM Register Bit Functions (continued)

Bit	Symbol	Function
CSR9.3	SDO	Serial ROM data output The value of this bit drives the sdo port of Core10100.
CSR9.2	SDI	Serial ROM data input This bit reflects the sdi port of Core10100.
CSR9.1	SCLK	Serial ROM clock The value of this bit drives the sclk port of Core10100.
CSR9.0	SCS	Serial ROM chip select The value of this bit drives the scs port of Core10100.

The MII management interface can be used to control the external PHY device from the host side. It allows access to all of the internal PHY registers via a simple two-wire interface. There are two signals on the MII management interface: the MDC (Management Data Clock) and the MDIO (Management Data I/O). The IEEE 802.3 indirection tristate signal defines the MDIO. Core10100 uses four unidirectional external signals to control the management interface. For proper operation of the interface, the user must connect a tristate buffer with an active low enable (inside or outside the FPGA), as shown in Figure 4-1. The Serial ROM interface can be used to access an external Serial ROM device via CSR9. The user can supply an external Serial ROM device, as shown in Figure 4-2 on page 39. The Serial ROM can be used to store user data, such as Ethernet addresses. Note that all access sequences and timing of the Serial ROM interface are handled by the software.

If the Serial ROM interface is not used, the sdi input port should be connected to logic 0 and the output ports (SCS, SCLK, and SDO) should be left unconnected.

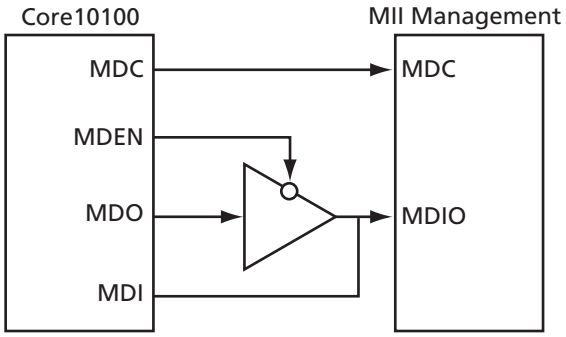

Figure 4-1 · I/O Tristate Buffer Connections

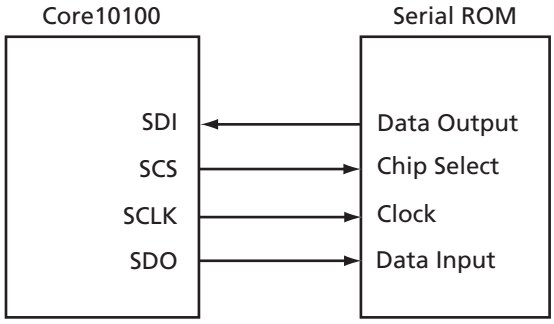

Figure 4-2 · External Serial ROM Connections

Table 4-25 · General-Purpose Timer and Interrupt Mitigation Control Register (CSR11)

Bits 31:24	CS	TT	NTP
Bits 23:16	RT	NRP	CON
Bits 15:8	TIM(15..8)		
Bits 7:0	TIM(7..0)		

Table 4-26 · General-Purpose Timer and Interrupt Mitigation Control Bit Functions

Bit	Symbol	Function
CSR11.31	CS	Cycle size Controls the time units for the transmit and receive timers according to the following: 1 – MII 100 Mbps mode – 5.12 µs MII 10 Mbps mode – 51.2 µs 0 – MII 100 Mbps mode – 81.92 µs MII 10 Mbps mode – 819.2 µs
CSR11.(30..27)	TT	Transmit timer Controls the maximum time that must elapse between the end of a transmit operation and the setting of the CSR5.TI (transmit interrupt) bit. This time is equal to TT × (16 × CS). The transmit timer is enabled when written with a nonzero value. After each frame transmission, the timer starts to count down if it has not already started. It is reloaded after every transmitted frame. Writing 0 to this field disables the timer effect on the transmit interrupt mitigation mechanism. Reading this field gives the actual count value of the timer.
CSR11.(26..24)	NTP	Number of transmit packets Controls the maximum number of frames transmitted before setting the CSR5.TI (transmit interrupt) bit. The transmit counter is enabled when written with a nonzero value. It is decremented after every transmitted frame. It is reloaded after setting the CSR5.TI bit. Writing 0 to this field disables the counter effect on the transmit interrupt mitigation mechanism. Reading this field gives the actual count value of the counter.

Table 4-26 · General-Purpose Timer and Interrupt Mitigation Control Bit Functions (continued)

Bit	Symbol	Function
CSR11.(23..20)	RT	Receive timer Controls the maximum time that must elapse between the end of a receive operation and the setting of the CSR5.RI (receive interrupt) bit. This time is equal to RT × CS. The receive timer is enabled when written with a nonzero value. After each frame reception, the timer starts to count down if it has not already started. It is reloaded after every received frame. Writing 0 to this field disables the timer effect on the receive interrupt mitigation mechanism. Reading this field gives the actual count value of the timer.
CSR11.(19..17)	NRP	Number of receive packets Controls the maximum number of received frames before setting the CSR5.RI (receive interrupt) bit. The receive counter is enabled when written with a nonzero value. It is decremented after every received frame. It is reloaded after setting the CSR5.RI bit. Writing 0 to this field disables the timer effect on the receive interrupt mitigation mechanism. Reading this field gives the actual count value of the counter.
CSR11.16	CON	Continuous mode 1 – General-purpose timer works in continuous mode 0 – General-purpose timer works in one-shot mode This bit must always be written before the timer value is written.
CSR11.(15..0)	TIM	Timer value Contains the number of iterations of the general-purpose timer. Each iteration duration is as follows: MII 100 Mbps mode – 81.92 µs MII 10 Mbps mode – 819.2 µs

Frame Data and Descriptors

Descriptor / Data Buffer Architecture Overview

A data exchange between the host and Core10100 is performed via the descriptor lists and data buffers, which reside in the system shared RAM. The buffers hold the host data to be transmitted or received by Core10100. The descriptors act as pointers to these buffers. Each descriptor list should be constructed by the host in a shared memory area and can be of an arbitrary size. There is a separate list of descriptors for both the transmit and receive processes.

The position of the first descriptor in the descriptor list is described by CSR3 for the receive list and by CSR4 for the transmit list. The descriptors can be arranged in either a chained or a ring structure. In a chained structure, every descriptor contains a pointer to the next descriptor in the list. In a ring structure, the address of the next descriptor is determined by CSR0.(6..2) (DSL—descriptor skip length). Every descriptor can point to up to two data buffers. When using descriptor chaining, the address of the second buffer is used as a pointer to the next descriptor; thus, only one buffer is available. A frame can occupy one or more data descriptors and buffers, but one descriptor cannot exceed a

single frame. In a ring structure, the descriptor operation may be corrupted if only one descriptor is used. Additionally, in the ring structure, at least two descriptors must be set up by the host. In a transmit process, the host can give the ownership of the first descriptor to Core10100 and causes the data specified by the first descriptor to be transmitted. At the same time, the host holds the ownership of the second or last descriptor to itself. This is done to prevent Core10100 from fetching the next frame until the host is ready to transmit the data specified in the second descriptor. In a receive process, the ownership of all available descriptors, unless it is pending processing by the host, must be given to Core10100.

Core10100 can store a maximum of two frames in the Transmit Data FIFO, including the frame waiting inside the Transmit Data FIFO, the frame being transferred from the data interface into the Transmit Data FIFO, and the frame being transmitted out via the MII interface from the Transmit Data FIFO.

Core10100 can store a maximum of four frames in the Receive Data FIFO, including the frame waiting inside the Receive Data FIFO, the frame being transferred to the data interface from the Receive Data FIFO, and the frame being received via the MII interface into the Receive Data FIFO.

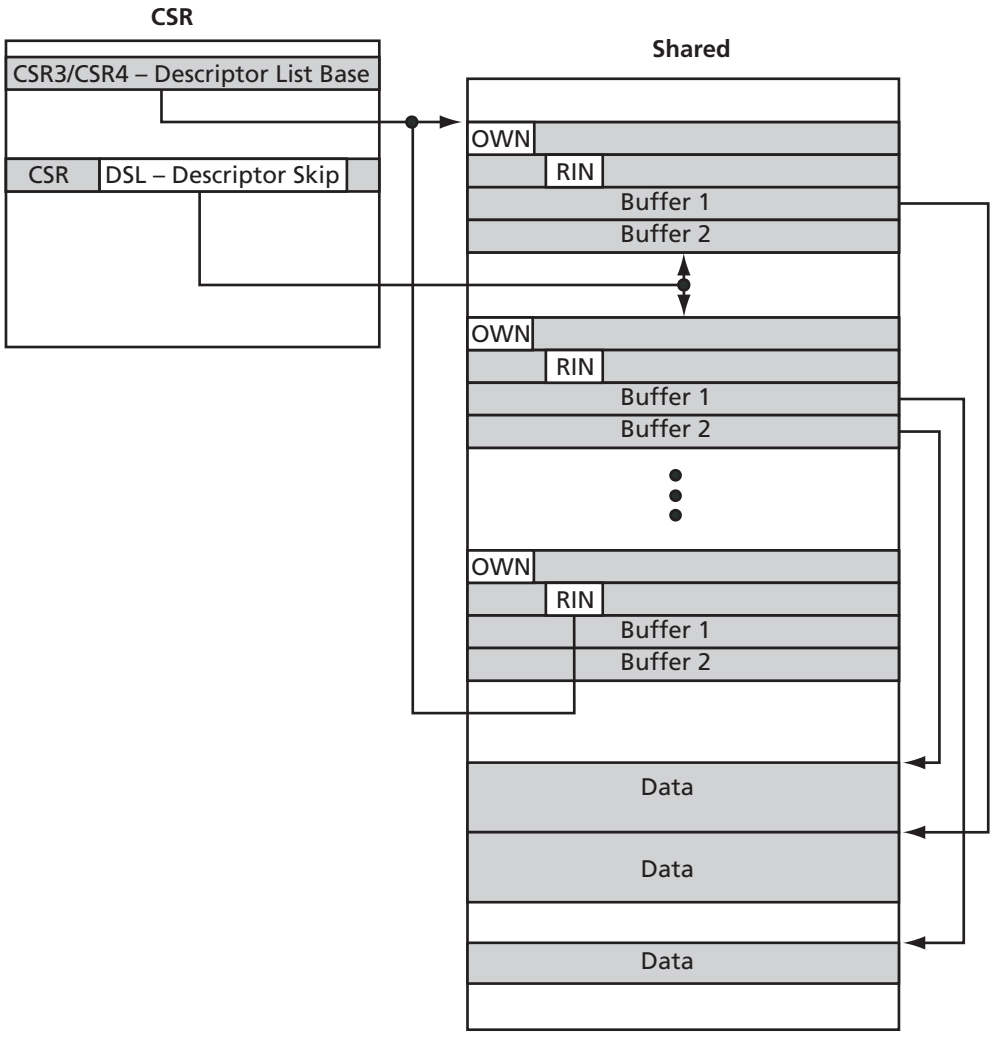

Figure 4-3 · Descriptors in Ring Structure

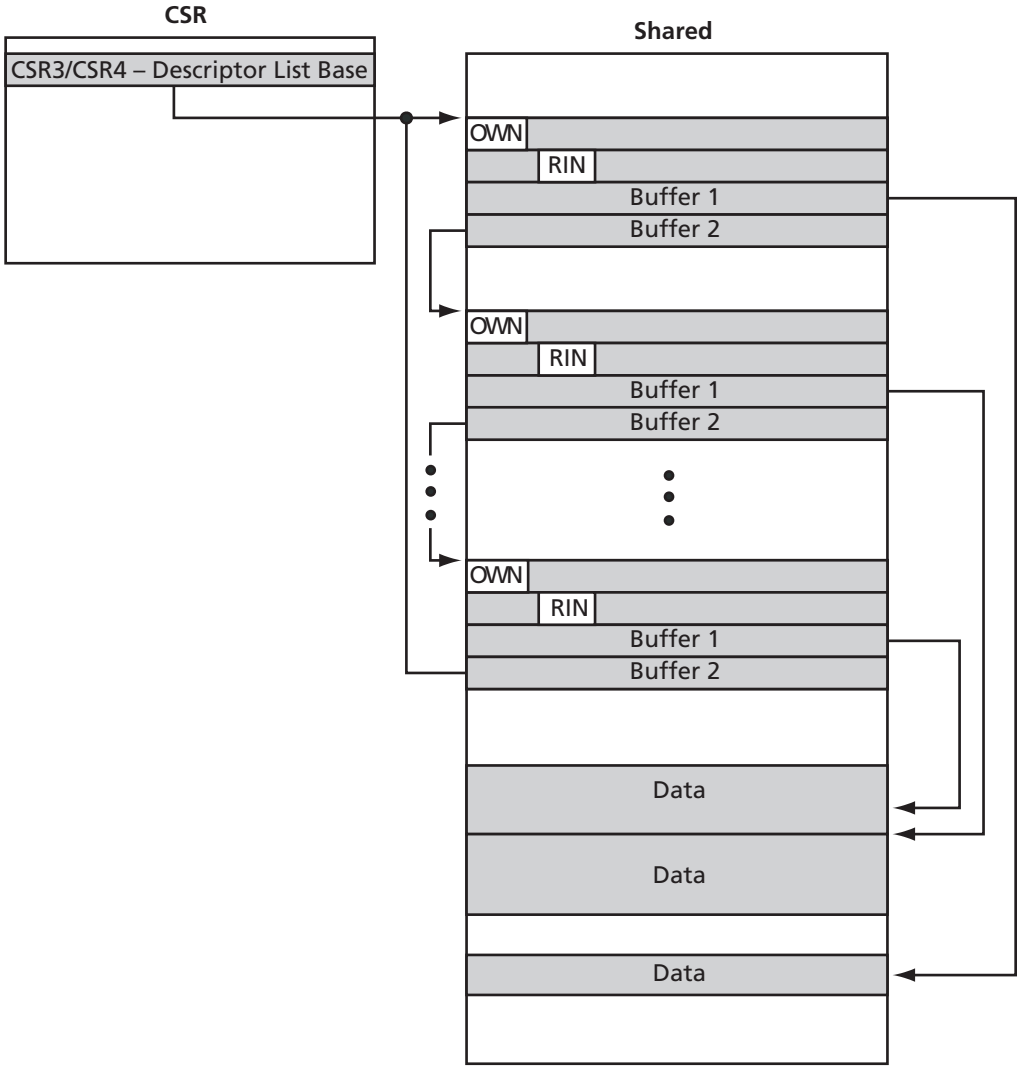

Figure 4-4 · Descriptors in Chained Structure

Table 4-27 · Receive Descriptors

RDES0	OWN	STATUS		
RDES1	CONTROL		RBS2	RBS1
RDES2	RBA1			
RDES3	RBA2			

Table 4-28 · STATUS (RDES0) Bit Functions

Bit	Symbol	Function
RDES0.31	OWN	Ownership bit 1 – Core10100 owns the descriptor. 0 – The host owns the descriptor. Core10100 will clear this bit when it completes a current frame reception or when the data buffers associated with a given descriptor are already full.
RDES0.30	FF	Filtering fail When set, indicates that a received frame did not pass the address recognition process. This bit is valid only for the last descriptor of the frame (RDES0.8 set), when the CSR6.30 (receive all) bit is set and the frame is at least 64 bytes long.
RDES0.(29..16)	FL	Frame length Indicates the length, in bytes, of the data transferred into a host memory for a given frame This bit is valid only when RDES0.8 (last descriptor) is set and RDES0.14 (descriptor error) is cleared.
RDES0.15	ES	Error summary This bit is a logical OR of the following bits: RDES0.1 – CRC error RDES0.6 – Collision seen RDES0.7 – Frame too long RDES0.11 – Runt frame RDES0.14 – Descriptor error This bit is valid only when RDES0.8 (last descriptor) is set.
RDES0.14	DE	Descriptor error Set by Core10100 when no receive buffer was available when trying to store the received data. This bit is valid only when RDES0.8 (last descriptor) is set.
RDES0.11	RF	Runt frame When set, indicates that the frame is damaged by a collision or by a premature termination before the end of a collision window. This bit is valid only when RDES0.8 (last descriptor) is set.
RDES0.10	MF	Multicast frame When set, indicates that the frame has a multicast address. This bit is valid only when RDES0.8 (last descriptor) is set.
RDES0.9	FS	First descriptor When set, indicates that this is the first descriptor of a frame.
RDES0.8	LS	Last descriptor When set, indicates that this is the last descriptor of a frame.

Table 4-28 · STATUS (RDES0) Bit Functions (continued)

Bit	Symbol	Function
RDES0.7	TL	Frame too long When set, indicates that a current frame is longer than maximum size of 1,518 bytes, as specified by 802.3. TL (frame too long) in the receive descriptor has been set when the received frame is longer than 1,518 bytes. This flag is valid in all receive descriptors when multiple descriptors are used for one frame.
RDES0.6	CS	Collision seen When set, indicates that a late collision was seen (collision after 64 bytes following SFD). This bit is valid only when RDES0.8 (last descriptor) is set.
RDES0.5	FT	Frame type When set, indicates that the frame has a length field larger than 1,500 (Ethernet-type frame). When cleared, indicates an 802.3-type frame. This bit is valid only when RDES0.8 (last descriptor) is set. Additionally, FT is invalid for runt frames shorter than 14 bytes.
RDES0.3	RE	Report on MII error When set, indicates that an error has been detected by a physical layer chip connected through the MII interface. This bit is valid only when RDES0.8 (last descriptor) is set.
RDES0.2	DB	Dribbling bit When set, indicates that the frame was not byte-aligned. This bit is valid only when RDES0.8 (last descriptor) is set.
RDES0.1	CE	CRC error When set, indicates that a CRC error has occurred in the received frame. This bit is valid only when RDES0.8 (last descriptor) is set. Additionally, CE is not valid when the received frame is a runt frame.
RDES0.0	ZERO	This bit is reset for frames with a legal length.

Table 4-29 · CONTROL and COUNT (RDES1) Bit

Bit	Symbol	Function
RDES1.25	RER	Receive end of ring When set, indicates that this is the last descriptor in the receive descriptor ring. Core10100 returns to the first descriptor in the ring, as specified by CSR3 (start of receive list address).
RDES1.24	RCH	Second address chained When set, indicates that the second buffer's address points to the next descriptor and not to the data buffer. Note that RER takes precedence over RCH.
RDES1.(21..11)	RBS2	Buffer 2 size Indicates the size, in bytes, of memory space used by the second data buffer. This number must be a multiple of four. If it is 0, Core10100 ignores the second data buffer and fetches the next data descriptor. This number is valid only when RDES1.24 (second address chained) is cleared.
RDES1.(10..0)	RBS1	Buffer 1 size Indicates the size, in bytes, of memory space used by the first data buffer. This number must be a multiple of four. If it is 0, Core10100 ignores the first data buffer and uses the second data buffer.

Table 4-30 · RBA1 (RDES2) Bit Functions

Bit	Symbol	Function
RDES2.(31..0)	RBA1	Receive buffer 1 address Indicates the length, in bytes, of memory allocated for the first receive buffer. This number must be longword-aligned (RDES2.(1..0) = '00').

Table 4-31 · RBA2 (RDES3) Bit Functions

Bit	Symbol	Function
RDES3.(31..0)	RBA2	Receive buffer 2 address Indicates the length, in bytes, of memory allocated for the second receive buffer. This number must be longword-aligned (RDES3.(1..0) = '00').

Table 4-32 · Transmit Descriptors

TDES0	OWN	STATUS		
TDES1	CONTROL		TBS2	TBS1
TDES2	TBA1			
TDES3	TBA2			

Table 4-33 · STATUS (TDES0) Bit Functions

Bit	Symbol	Function
TDES0.31	OWN	Ownership bit 1 – Core10100 owns the descriptor. 0 – The host owns the descriptor. Core10100 will clear this bit when it completes a current frame transmission or when the data buffers associated with a given descriptor are empty.
TDES0.15	ES	Error summary This bit is a logical OR of the following bits: TDES0.1 – Underflow error TDES0.8 – Excessive collision error TDES0.9 – Late collision TDES0.10 – No carrier TDES0.11 – Loss of carrier This bit is valid only when TDES1.30 (last descriptor) is set.
TDES0.11	LO	Loss of carrier When set, indicates a loss of the carrier during a transmission. This bit is valid only when TDES1.30 (last descriptor) is set.
TDES0.10	NC	No carrier When set, indicates that the carrier was not asserted by an external transceiver during the transmission. This bit is valid only when TDES1.30 (last descriptor) is set.
TDES0.9	LC	Late collision When set, indicates that a collision was detected after transmitting 64 bytes. This bit is not valid when TDES0.1 (underflow error) is set. This bit is valid only when TDES1.30 (last descriptor) is set.
TDES0.8	EC	Excessive collisions When set, indicates that the transmission was aborted after 16 retries. This bit is valid only when TDES1.30 (last descriptor) is set.
TDES0.(6..3)	CC	Collision count This field indicates the number of collisions that occurred before the end of a frame transmission. This value is not valid when TDES0.8 (excessive collisions bit) is set. This bit is valid only when TDES1.30 (last descriptor) is set.

Table 4-33 · STATUS (TDES0) Bit Functions (continued)

Bit	Symbol	Function
TDES0.1	UF	Underflow error When set, indicates that the FIFO was empty during the frame transmission. This bit is valid only when TDES1.30 (last descriptor) is set.
TDES0.0	DE	Deferred When set, indicates that the frame was deferred before transmission. Deferring occurs if the carrier is detected when the transmission is ready to start. This bit is valid only when TDES1.30 (last descriptor) is set.

Table 4-34 · CONTROL (TDES1) Bit Functions

Bit	Symbol	Function
TDES1.31	IC	Interrupt on completion Setting this flag instructs Core10100 to set CSR5.0 (transmit interrupt) immediately after processing a current frame. This bit is valid when TDES1.30 (last descriptor) is set or for a setup packet.
TDES1.30	LS	Last descriptor When set, indicates the last descriptor of the frame.
TDES1.29	FS	First descriptor When set, indicates the first descriptor of the frame.
TDES1.28	FT1	Filtering type This bit, together with TDES0.22 (FT0), controls a current filtering mode. This bit is valid only for the setup frames.
TDES1.27	SET	Setup packet When set, indicates that this is a setup frame descriptor.
TDES1.26	AC	Add CRC disable When set, Core10100 does not append the CRC value at the end of the frame. The exception is when the frame is shorter than 64 bytes and automatic byte padding is enabled. In that case, the CRC field is added, despite the state of the AC flag.
TDES1.25	TER	Transmit end of ring When set, indicates the last descriptor in the descriptor ring.
TDES1.24	TCH	Second address chained When set, indicates that the second descriptor's address points to the next descriptor and not to the data buffer. This bit is valid only when TDES1.25 (transmit end of ring) is reset.
TDES1.23	DPD	Disabled padding When set, automatic byte padding is disabled. Core10100 normally appends the PAD field after the INFO field when the size of an actual frame is less than 64 bytes. After padding bytes, the CRC field is also inserted, despite the state of the AC flag. When DPD is set, no padding bytes are appended.

Table 4-34 · CONTROL (TDES1) Bit Functions (continued)

Bit	Symbol	Function
TDES1.22	FT0	Filtering type This bit, together with TDES0.28 (FT1), controls the current filtering mode. This bit is valid only when the TDES1.27 (SET) bit is set.
TDES1.(21..11)	TBS2	Buffer 2 size Indicates the size, in bytes, of memory space used by the second data buffer. If it is zero, Core10100 ignores the second data buffer and fetches the next data descriptor. This bit is valid only when TDES1.24 (second address chained) is cleared.
TDES1.(10..0)	TBS1	Buffer 1 size Indicates the size, in bytes, of memory space used by the first data buffer. If it is 0, Core10100 ignores the first data buffer and uses the second data buffer.

Table 4-35 · TBA1 (TDES2) Bit Functions

Bit	Symbol	Function
TDES2.(31..0)	TBA1	Transmit buffer 1 address Contains the address of the first data buffer. For the setup frame, this address must be longword-aligned (TDES3.(1..0) = '00'). In all other cases, there are no restrictions on buffer alignment.

Table 4-36 · TBA2 (TDES3) Bit Functions

Bit	Symbol	Function
TDES3(31..0)	TBA2	Transmit buffer 2 address Contains the address of the second data buffer. There are no restrictions on buffer alignment.

MAC Address and Setup Frames

The setup frames define addresses that are used for the receive address filtering process. These frames are never transmitted on the Ethernet connection. They are used to fill the address filtering RAM. A valid setup frame must be exactly 192 bytes long and must be allocated in a single buffer that is longword-aligned. TDESI.27 (setup frame indicator) must be set. Both TDES1.29 (first descriptor) and TDES1.30 (last descriptor) must be cleared for the setup frame. The FT1 and FT0 bits of the setup frame define the current filtering mode.

Table 4-37 on page 50 lists all possible combinations. Table 4-38 on page 50 shows the setup frame buffer format for perfect filtering modes. Table 4-39 on page 51 shows the setup frame buffer for imperfect filtering modes. The setup should be sent to Core10100 when Core10100 is in stop mode. When a RAM with more than 192 bytes is used for the address filtering RAM, a setup frame with more than 192 bytes can be written into this memory to initialize its contents,

but only the first 192 bytes constitute the address filtering operation. While writing the setup frame buffer in the host memory, the buffer size must be twice the size of the setup frame buffer.

Table 4-37 · Filtering Type Selection

FT1	FT0	Description
0	0	Perfect filtering mode Setup frame buffer is interpreted as a set of sixteen 48-bit physical addresses.
0	1	Hash filtering mode Setup frame buffer contains a 512-bit hash table plus a single 48-bit physical address.
1	0	Inverse filtering mode Setup frame buffer is interpreted as a set of sixteen 48-bit physical addresses.
1	1	Hash only filtering mode Setup frame buffer is interpreted as a 512-bit hash table.

Table 4-38 · Perfect Filtering Setup Frame Buffer

Byte Number	Data Bits 31:16	Data Bits 15:0
1:0	{Physical Address [39:32],Physical Address [47:40]}	
3:2	{Physical Address [23:16],Physical Address [31:24]}	
5:4	{Physical Address [7:0],Physical Address [15:8]}	
15:12	xxxxxxxxxxxxxxxx	Physical Address 1 (15:00)
19:16	xxxxxxxxxxxxxxxx	Physical Address 1 (31:16)
23:20	xxxxxxxxxxxxxxxx	Physical Address 1 (47:32)
. . .	. . .	. . .
171:168	xxxxxxxxxxxxxxxx	Physical Address 14 (15:00)
175:172	xxxxxxxxxxxxxxxx	Physical Address 14 (31:16)
179:176	xxxxxxxxxxxxxxxx	Physical Address 14 (47:32)
183:180	xxxxxxxxxxxxxxxx	Physical Address 15 (15:00)
187:184	xxxxxxxxxxxxxxxx	Physical Address 15 (31:16)
191:188	xxxxxxxxxxxxxxxx	Physical Address 15 (47:32)

Table 4-39 · Hash Table Setup Frame Buffer Format

Byte Number	Data Bits 31:16	Data Bits 15:0
3:0	xxxxxxxxxxxxxxxx	Hash filter (015:000)
7:4	xxxxxxxxxxxxxxxx	Hash filter (031:016)
11:8	xxxxxxxxxxxxxxxx	Hash filter (047:032)
. . .	. . .	. . .
123:121	xxxxxxxxxxxxxxxx	Hash filter (495:480)
127:124	xxxxxxxxxxxxxxxx	Hash filter (511:496)
131:128	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	
135:132	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	
. . .	. . .	
159:156	xxxxxxxxxxxxxxxx	Physical Address (15:00)
163:160	xxxxxxxxxxxxxxxx	Physical Address (31:16)
167:164	xxxxxxxxxxxxxxxx	Physical Address (47:32)
171:168	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	
175:172	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	
. . .	. . .	
183:180	xxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxx
187:184	xxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxx
191:188	xxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxx

Internal Operation

The address bus width of the Receive/Transmit Data RAMs can be customized via the core parameters RFIFODEPTH and TFIFODEPTH (Table 3-1 on page 19). Those memory blocks must be at least as big as the longest frame used on a given network. Core10100 stops to request new frame data when there are two frames already in the Transmit Data RAM. It resumes the request for new frame data when there is either one or no frame in the Transmit Data RAM.

At any given time, the Receive Data RAM can hold no more than four frames, including frames currently under transfer.

DMA Controller

The DMA is used to control a data flow between the host and Core10100.

The DMA services the following types of requests from the Core10100 transmit and receive processes:

- Transmit request:

  Descriptor fetch

  Descriptor closing

  Setup packet processing

  Data transfer from host buffer to transmit FIFO

- Receive request:

Descriptor fetch

Descriptor closing

Data transfer from receive FIFO to host buffer

The key task for the DMA is to perform an arbitration between the receive and transmit processes. Two arbitration schemes are possible according to the CSR0.1 bit:

- 1 – Round-robin arbitration scheme in which receive and transmit processes have equal priorities
- 0 – The receive process has priority over the transmit process unless transmission is in progress. In this case, the following rules apply:

  The transmit process request should be serviced by the DMA between two consecutive receive transfers.

  The receive process request should be serviced by the DMA between two consecutive transmit transfers.

Transfers between the host and Core10100 performed by the DMA component are either single data transfers or burst transfers. For the data descriptors, the data transfer size depends on the core parameter DATAWIDTH. The rule is that every descriptor field (32-bit) is accessed with a single burst. For DATAWIDTH = 32, the descriptors are accessed with a single transaction; for DATAWIDTH = 16, the descriptors are accessed with a burst of two 16-bit words, and for DATAWIDTH = 8, the descriptors are accessed with a burst of four 8-bit words.

In the case of data buffers, the burst length is defined by CSR0.(13..8) (programmable burst length) and can be set to 0, 1, 2, 4, 8, 16, or 32. When set to 0, no maximum burst size is defined, and the transfer ends when the transmit FIFOs are full or the receive FIFOs are empty.

Transmit Process

The transmit process can operate in one of three modes: running, stopped, or suspended. After a software or hardware reset, or after a stop transmit command, the transmit process is in a stopped state. The transmit process can leave a stopped state only after the start transmit command.

When in a running state, the transmit process performs descriptor/buffer processing. When operating in a suspended or stopped state, the transmit process retains the position of the next descriptor, i.e., the address of the descriptor following the last descriptor being closed. After entering a running state, that position is used for the next descriptor fetch. The only exception is when the host writes the transmit descriptor base address register (CSR4). In that case, the descriptor address is reset and the fetch is directed to the first position in the list. Before writing to CSR4 the MAC must be in a stopped state.

When operating in a stopped state, the transmit process stopped (tps) output is HIGH. This output can be used to disable the clkt clock signal external to Core10100. When both the tps and receive process stopped (rps) outputs are HIGH, all clock signals except clkcsr can be disabled external to Core10100.

The transmit process remains running until one of the following events occurs:

- The hardware or software reset is issued. Setting the CSR0.0 (SWR) bit can perform the software reset. After the reset, all the internal registers return to their default states. The current descriptor's position in the transmit descriptor list is lost.
- A stop transmit command is issued by the host. This can be performed by writing 0 to the CSR6.13 (ST) bit. The current descriptor's position is retained.
- The descriptor owned by the host is found. The current descriptor's position is retained.
- The transmit FIFO underflow error is detected. An underflow error is generated when the transmit FIFO is empty during the transmission of the frame. When it occurs, the transmit process enters a suspended state. Transmit automatic polling is internally disabled, even if it is enabled by the host by writing the TAP bits. The current descriptor's position is retained.

Leaving a suspended state is possible in one of the following situations:

- A transmit poll demand command is issued. This can be performed by writing CSR1 with a nonzero value. The transmit poll demand command can also be generated automatically when transmit automatic polling is enabled.

Transmit automatic polling is enabled only if the CSR0(19..17) (TAP) bits are written with a nonzero value and when there was no underflow error prior to entering the suspended state.

- A stop transmit command is issued by the host. This can be performed by writing 0 to the CSR6.13 (ST) bit. The current descriptor's position is retained.

A typical data flow for the transmit process is illustrated in Figure 4-6 on page 53. The events for the transmit process typically happen in the following order:

1. The host sets up CSR registers for the operational mode, interrupts, etc.
2. The host sets up transmit descriptors/data in the shared RAM.
3. The host sends the transmit start command.
4. Core10100 starts to fetch the transmit descriptors.
5. Core10100 transfers the transmit data to Transmit Data RAM from the shared RAM.
6. Core10100 starts to transmit data on MII.

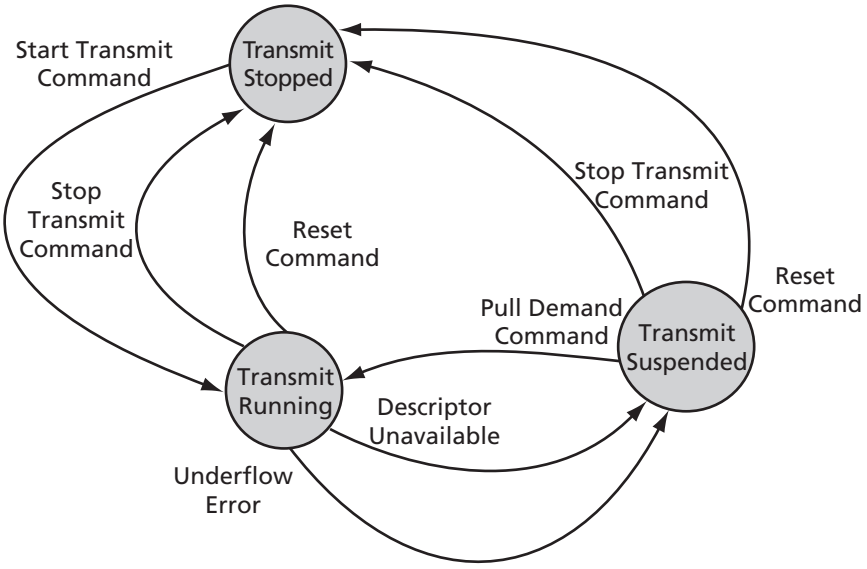

Figure 4-5 · Transmit Process Transitions

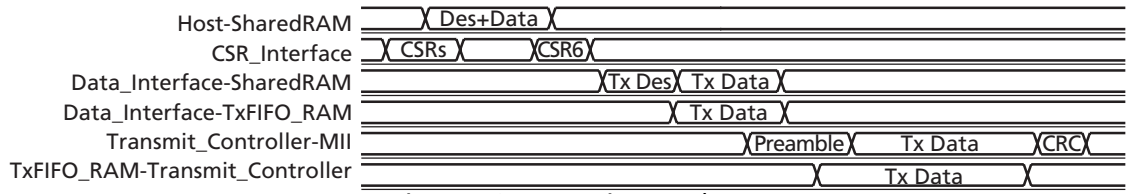

Figure 4-6 · Transmit Data Flow

Note: Refer to the *Core10100 User's Guide* for an example of transmit data timing.

Receive Process

The receive process can operate in one of three modes: running, stopped, or suspended. After a software or hardware reset, or after a stop receive command, the receive process is in the stopped state. The receive process can leave a stopped state only after a start receive command.

In the running state, the receiver performs descriptor/buffer processing. In the running state, the receiver fetches from the receive descriptor list. It performs this fetch regardless of whether there is any frame on the link. When there is no frame pending, the receive process reads the descriptor and simply waits for the frames. When a valid frame is recognized, the receive process starts to fill the memory buffers pointed to by the current descriptor. When the frame ends, or when the memory buffers are completely filled, the current frame descriptor is closed (ownership bit cleared). Immediately, the next descriptor on the list is fetched in the same manner, and so on.

When operating in a suspended or stopped state, the receive process retains the position of the next descriptor (the address of the descriptor following the last descriptor that was closed). After entering a running state, the retained position is used for the next descriptor fetch. The only exception is when the host writes the receive descriptor base address register (CSR3). In that case, the descriptor address is reset and the fetch is pointed to the first position in the list. Before writing to CSR3, the MAC must be in a stopped state.

When operating in a stopped state, the rps output is HIGH. This output allows for switching the receive clock clkr off externally. When both the rps and tps outputs are HIGH, all clocks except clkcsr can be externally switched off.

The receive process runs until one of the following events occurs:

- A hardware or software reset is issued by the host. A software reset can be performed by setting the CSR0.0 (SWR) bit. After reset, all internal registers return to their default states. The current descriptor's position in the receive descriptor list is lost.
- A stop receive command is issued by the host. This can be performed by writing 0 to the CSR6.1 (SR) bit. The current descriptor's position is retained.
- The descriptor owned by the host is found by Core10100 during the descriptor fetch. The current descriptor's position is retained.

Leaving a suspended state is possible in one of the following situations:

- A receive poll command is issued by the host. This can be performed by writing CSR2 with a nonzero value.
- A new frame is detected by Core10100 on a receive link.
- A stop receive command is issued by the host. This can be performed by writing 0 to the CSR6.1 (SR) bit. The current descriptor's position is retained.

The receive state machine goes into stopped state after the current frame is done if a STOP RECEIVE command is given. It does not go in to a stopped state immediately.

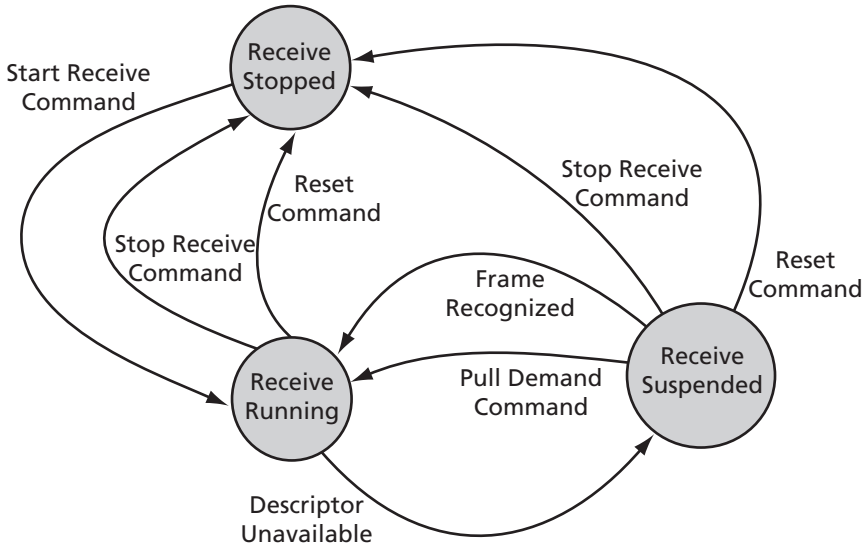

Figure 4-7 · Receive Process Transitions

Note: Refer to the *Core10100 User's Guide* for an example of receive timing.

A typical data flow in a receive process is illustrated in Figure 4-8 on page 55. The events for the receive process typically happen in the following order:

1. The host sets up CSR registers for the operational mode, interrupts, etc.
2. The host sets up receive descriptors in the shared RAM.
3. The host sends the receive start command.
4. Core10100 starts to fetch the transmit descriptors.
5. Core10100 waits for receive data on MII.
6. Core10100 transfers received data to the Receive Data RAM.
7. Core10100 transfers received data to shared RAM from Receive Data RAM.

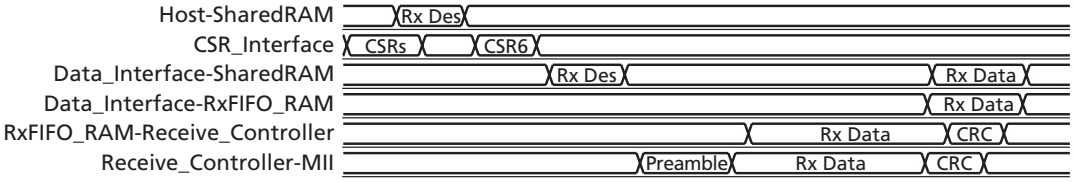

Figure 4-8 · Receive Data Flow

Interrupt Controller

The interrupt controller uses three internal Control and Status registers: CSR5, CSR7, and CSR11. CSR5 contains the Core10100 status information. It has 10 bits that can trigger an interrupt. These bits are collected in two groups: normal interrupts and abnormal interrupts. Each group has its own summary bit, NIS and AIS, respectively. The NIS and AIS bits directly control the int output port of Core10100. Every status bit in CSR5 that can source an interrupt can be individually masked by writing an appropriate value to CSR7 (Interrupt Enable register).

Additionally, an interrupt mitigation mechanism is provided for reducing CPU usage in servicing interrupts. Interrupt mitigation is controlled via CSR11. There are separate interrupt mitigation control blocks for the transmit and receive interrupts. Both of these blocks consist of a 4-bit frame counter and a 4-bit timer. The operation of these blocks is similar for the receive and transmit processes. After the end of a successful receive or transmission operation, an

appropriate counter is decremented and the timer starts to count down if it has not already started. An interrupt is triggered when either the counter or the timer reaches a zero value. This allows Core10100 to generate a single interrupt for a few received/transmitted frames or after a specified time since the last successful receive/transmit operation.

It is possible to omit transmit interrupt mitigation for one particular frame by setting the Interrupt on Completion (IC) bit in the last descriptor of the frame. If the IC bit is set, Core10100 sets the transmit interrupt immediately after the frame has been transmitted.

The int port remains LOW for a single clock cycle on every write to CSR5. This enables the use of both level- and edge-triggered external interrupt controllers.

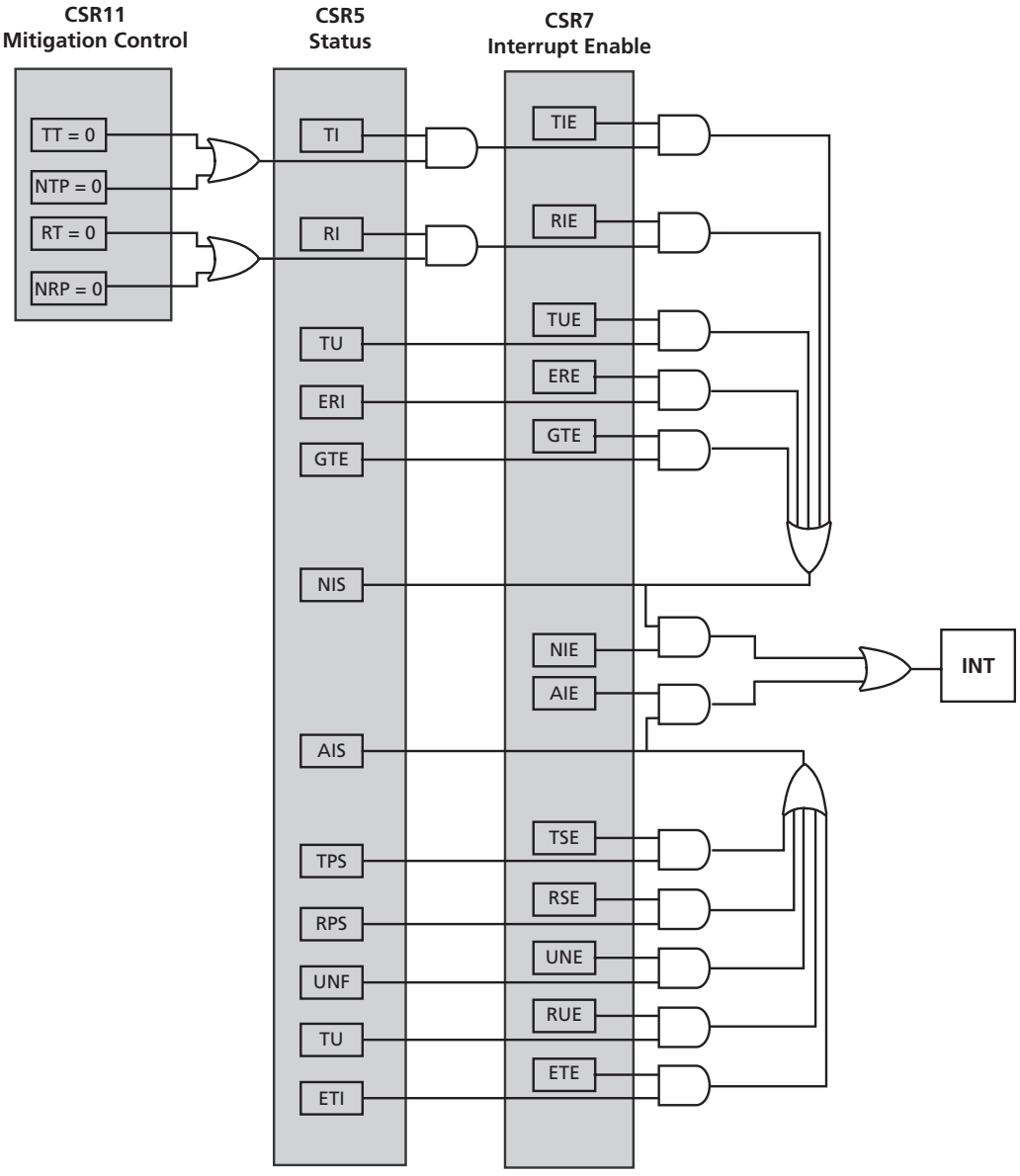

Figure 4-9 · Interrupt Scheme

General-Purpose Timer

Core10100 includes a 16-bit general-purpose timer to simplify time interval calculation by an external host. The timer operates synchronously with the transmit clock clkt generated by the PHY device. This gives the host the possibility of measuring time intervals based on actual Ethernet bit time.

The timer can operate in one-shot mode or continuous mode. In one-shot mode, the timer stops after reaching a zero value; in continuous mode, it is automatically reloaded and continues counting down after reaching a zero value.

The actual count value can be tested with an accuracy of ±1 bit by reading CSR11.(15..0). When writing CSR11.(15..0), the data is stored in the internal reload register. The timer is immediately reloaded and starts to count down.

Data Link Layer Operation

MII Interface

Core10100 uses a standard MII interface as defined in the 802.3 standard.

This interface can be used for connecting Core10100 to an external Ethernet 10/100 PHY device.

MII Interface Signals

Table 4-40 · External PHY Interface Signals

IEEE 802.3 Signal Name	Core10100 Signal Name	Description
RX_CLK	CLKR	Clock for receive operation This must be a 25 MHz clock for 100 Mbps operation or a 2.5 MHz clock for 10 Mbps operation.
RX_DV	RX_DV	Receive data valid signal The PHY device must assert RX_DV when a valid data nibble is provided on the RXD signal. The RX_DV signal must be synchronous to the CLKR receive clock.
RX_ER	RX_ER	Receive error If RX_ER is asserted during Core10100 reception, the frame is received and status of the frame is updated with RX_ER. The RX_ER signal must be synchronous to the CLKR receive clock.
RXD	RXD	Receive data recovered and decoded by PHY The RXD[0] signal is the least significant bit. The RXD bus must be synchronous to the CLKR receive clock.
TX_CLK	CLKT	Clock for transmit operation This must be a 25 MHz clock for 100 Mbps operation or a 2.5 MHz clock for 10 Mbps operation.
TX_EN	TX_EN	Transmit enable When asserted, indicates valid data for the PHY on TXD. The TX_EN signal is synchronous to the CLKT transmit clock.
TXD	TXD	Transmit data The TXD[0] signal is the least significant bit. The TXD bus is synchronous to the CLKT transmit clock.

Table 4-40 · External PHY Interface Signals (continued)

IEEE 802.3 Signal Name	Core10100 Signal Name	Description
COL	COL	Collision detected This signal must be asserted by the PHY when a collision is detected on the medium. It is valid only when operating in a half-duplex mode. When operating in a full-duplex mode, this signal is ignored by Core10100. The COL signal is not required to be synchronous to either CLKR or CLKT. The COL signal is sampled internally by the CLKT clock.
CRS	CRS	Carrier sense This signal must be asserted by the PHY when either a receive or a transmit medium is non-idle. The CRS signal is not required to be synchronous to either CLKR or CLKT.
TX_ER	TX_ER	Transmit error The current version of Core10100 has the TX_ER signal statically tied to logic 0 (no transmit errors).
MDC	MDC	MII management clock This signal is driven by the CSR9.16 bit.
MDIO	MDI	MII management data input The state of this signal can be checked by reading the CSR9.19 bit.
	MDO	MII management data output This signal is driven by the CSR9.18 bit.

MII Receive Operation

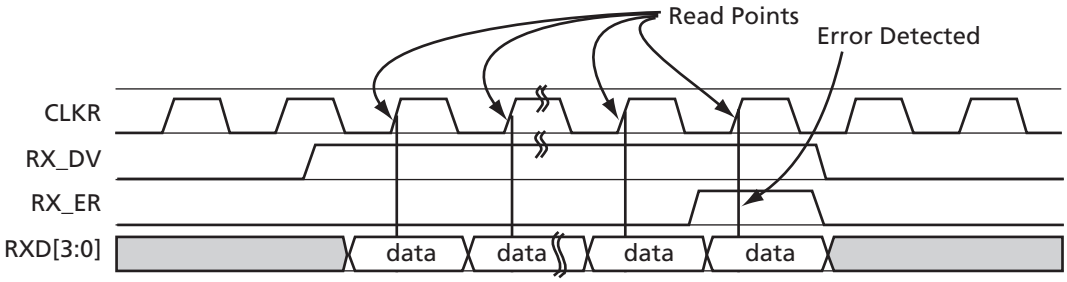

Figure 4-10 · MII Receive Operation

MII Transmit Operation

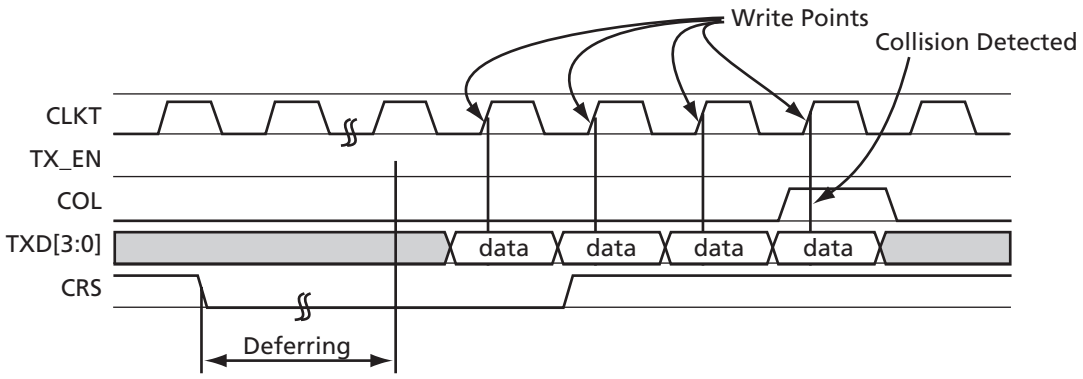

Figure 4-11 · MII Transmit Operation

Frame Format

Core10100 supports the Ethernet frame format shown in Figure 4-12 (“B” indicates bytes). The standard Ethernet frames (DIX Ethernet), as well as IEEE 802.3 frames, are accepted.

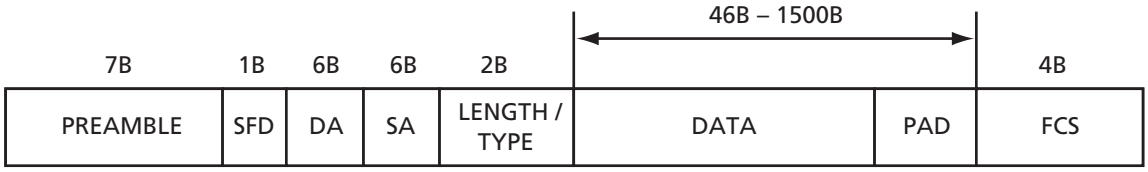

Figure 4-12 · Frame Format

Table 4-41 · Frame Field Usage

Field	Width (bytes)	Transmit Operation	Receive Operation
PREAMBLE	7	Generated by Core10100	Stripped from received data Not required for proper operation
SFD	1	Generated by Core10100	Stripped from received data
DA	6	Supplied by host	Checked by Core10100 according to current address filtering mode and passed to host
SA	6	Supplied by host	Passed to host
LENGTH/ TYPE	6	Supplied by host	Passed to host
DATA	0-1500	Supplied by host	Passed to host
PAD	0-46	Generated by Core10100 when CSR.23 (DPD) bit is cleared and data supplied by host is less than 64 bytes	Passed to host
FCS	4	Generated by Core10100 when CSR.26 bit is cleared	Checked by Core10100 and passed to host

Collision Handling

Collision detection is performed via the col input port. If a collision is detected before the end of the PREAMBLE/SFD, Core10100 completes the PREAMBLE/SFD, transmits the JAM sequence, and initiates a backoff computation. If a collision is detected after the transmission of the PREAMBLE and SFD, but prior to 512 bits being transmitted, Core10100 immediately aborts the transmission, transmits the JAM sequence, and then initiates a backoff. If a collision is detected after 512 bits have been transmitted, the collision is termed a late collision. Core10100 aborts the transmission and appends the JAM sequence. The transmit message is flushed from the FIFO. Core10100 does not initiate a backoff and does not attempt to retransmit the frame when a late collision is detected.

Core10100 uses a "truncated binary exponential backoff" algorithm for backoff computing, as defined in the IEEE 802.3 standard and outlined in Figure 4-13.

Backoff processing is performed only in half-duplex mode. In full-duplex mode, collision detection is disabled.

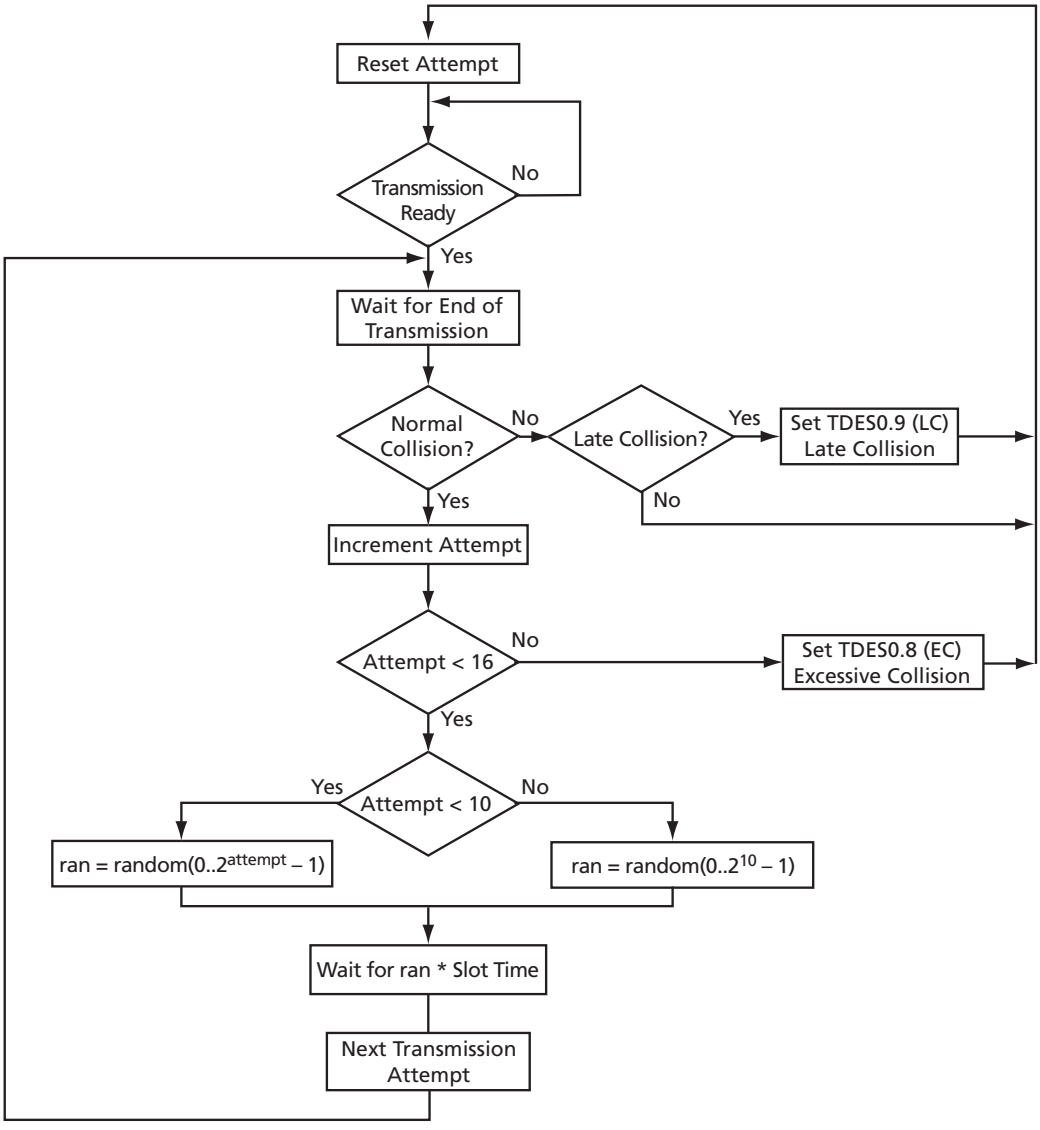

Figure 4-13 · Backoff Process Algorithms

Deferring

The deferment algorithm is implemented per the 802.3 specification and outlined in Figure 4-14. The InterFrame Gap (IFG) timer starts to count whenever the link is not idle. If activity on the link is detected during the first 60 bit times of the IFG timer, the timer is reset and restarted once activity has stopped. During the final 36 bit times of the IFG timer, the link activity is ignored.

Carrier sensing is performed only when operating in half-duplex mode. In full-duplex mode, the state of the CRS input is ignored.

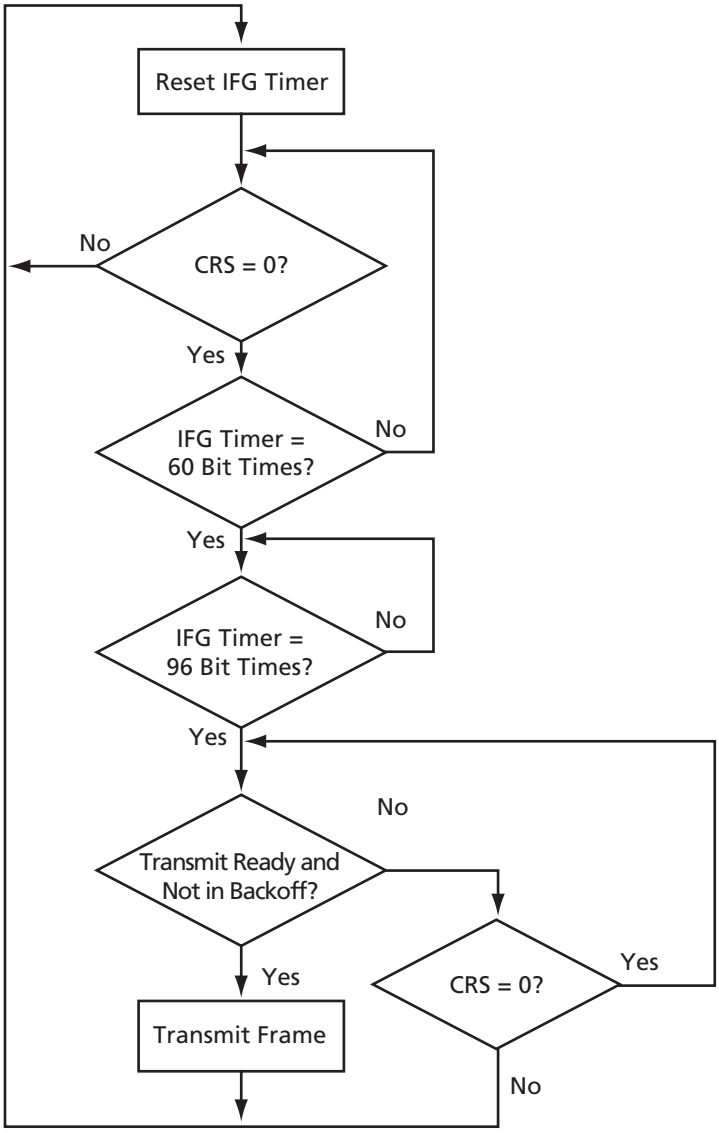

Figure 4-14 · Deferment Process Algorithms

Receive Address Filtering

There are three kinds of addresses on the LAN: the unicast addresses, the multicast addresses, and the broadcast addresses. If the first bit of the address (IG bit) is 0, the frame is unicast, i.e., dedicated to a single station. If the first bit is 1, the frame is multicast, i.e., destined for a group of stations. If the address field contains all ones, the frame is broadcast and is received by all stations on the LAN.

When Core10100 operates in perfect filtering mode, all frames are checked against the addresses in the address filtering RAM. The unicast, multicast, and broadcast frames are treated in the same manner.

When Core10100 operates in the imperfect filtering mode, the frames with the unicast addresses are checked against a single physical address. The multicast frames are checked using the 512-bit hash table. To receive the broadcast frame, the hash table bit corresponding to the broadcast address CRC value must be set. Core10100 applies the standard Ethernet CRC function to the first six bytes of the frame that contains a destination address. The least significant nine bits of the CRC value are used to index the table. If the indexed bit is set, the frame is accepted. If this bit is cleared, the frame is rejected. The algorithm is shown in Figure 4-15.

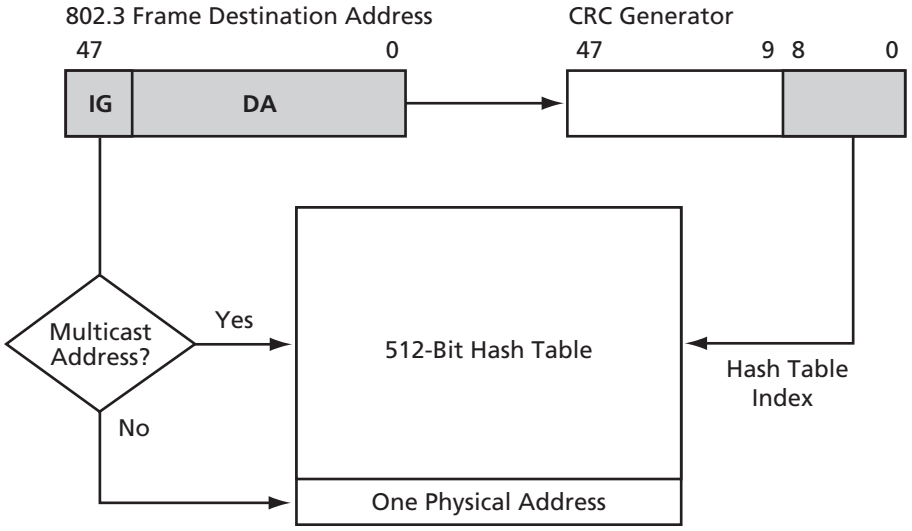

Figure 4-15 · Filtering with One Physical Address and the Hash Table

It is important that one bit in the hash table corresponds to many Ethernet addresses. Therefore, it is possible that some frames may be accepted by Core10100, even if they are not intended to be received. This is because some frames that should not have been received have addresses that hash to the same bit in the table as one of the proper addresses. The software should perform additional address filtering to reject all such frames. The receive address filtering RAM must be enabled using the ADDRFILTER core parameter to enable the above functionality.

Steps for Calculating CRC with Hash Filtering

Following are the steps the core is using, and Testbench/Software needs to follow. These are the steps for calculating CRC with which the hash filter logic of the DUT accepts the frames properly:

1. Initial value of the CRC is 0xFFFFFFFF.
2. XOR the incoming data bit with the 31st bit of the current CRC value.
3. Left shift the current CRC value by one bit.
4. Check the XORed value from step 2. If this value is 1'b1 then XOR the current CRC value with the generator polynomial (0x4C11DB7).
5. Insert the bit value result from step 2 at the 0th bit location of the current CRC value.
6. Repeat steps 2, 3, 4, and 5 until the CRC is calculated for all the bits of the data.MII_TO_RMII Internal Architecture

External Address Filtering Interface

An external address filtering interface is provided to extend the internal filtering capabilities of Core10100. The interface allows connection of external user-supplied address checking logic. All signals from the interface are synchronous to the clkr clock.

If the external address filtering is not used, all input ports of the interface must be grounded and all output ports must be left floating.

Table 4-42 · External Address Interface Description

Core10100 Signal Name	Type	Description
MATCH	In	External address match When HIGH, indicates that the destination address on the MATCHDATA port is recognized by the external address checking logic and that the current frame should be received by Core10100. When LOW, indicates that the destination address on the MATCHDATA port is not recognized and that the current frame should be discarded. Note that the MATCH signal should be valid only when the MATCHVAL signal is HIGH.
MATCHVAL	In	External address match valid When HIGH, indicates that the MATCH signal is valid.
MATCHEN	Out	External match enable When HIGH, indicates that the MATCHDATA signal is valid. The MATCHEN output should be used as an enable signal for the external address checking logic. It is HIGH for at least four CLKR clock periods to allow for latency of external address checking logic.
MATCHDATA	Out	External address match data The MATCHDATA signal represents the 48-bit destination address of the received frame. Note that the MATCHDATA signal is valid only when matchen signal is HIGH.

MII to RMII Interface

The 25 MHz transmit clock (CLKT) and receive clock (CLKR) are derived from the 50 MHz RMII_CLK (divide by 2 for 100 Mbps operation). The 2.5 MHz transmit clock (CLKT) and receive clock (CLKR) are derived from the 50 MHz RMII_CLK (divide by 20 for 10 Mbps operation). The internal clock net CLK_TX_RX must be assigned to a global clock network. The CSR6 bit 22, which is connected to the SPEED port in the MII_RMII block, will select the clock frequency as either 2.5 MHz or 25 MHz.

The data width on the MII interface is 4 bits for both transmit and receive. The data width on the RMII interface is 2 bits for both transmit and receive. The CRS and RX_DV signals are decoded from CRS_DV. The COL signal is derived from AND-ing together the TX_EN and the decoded CRS signal from the CRS_DV line in half duplex mode.

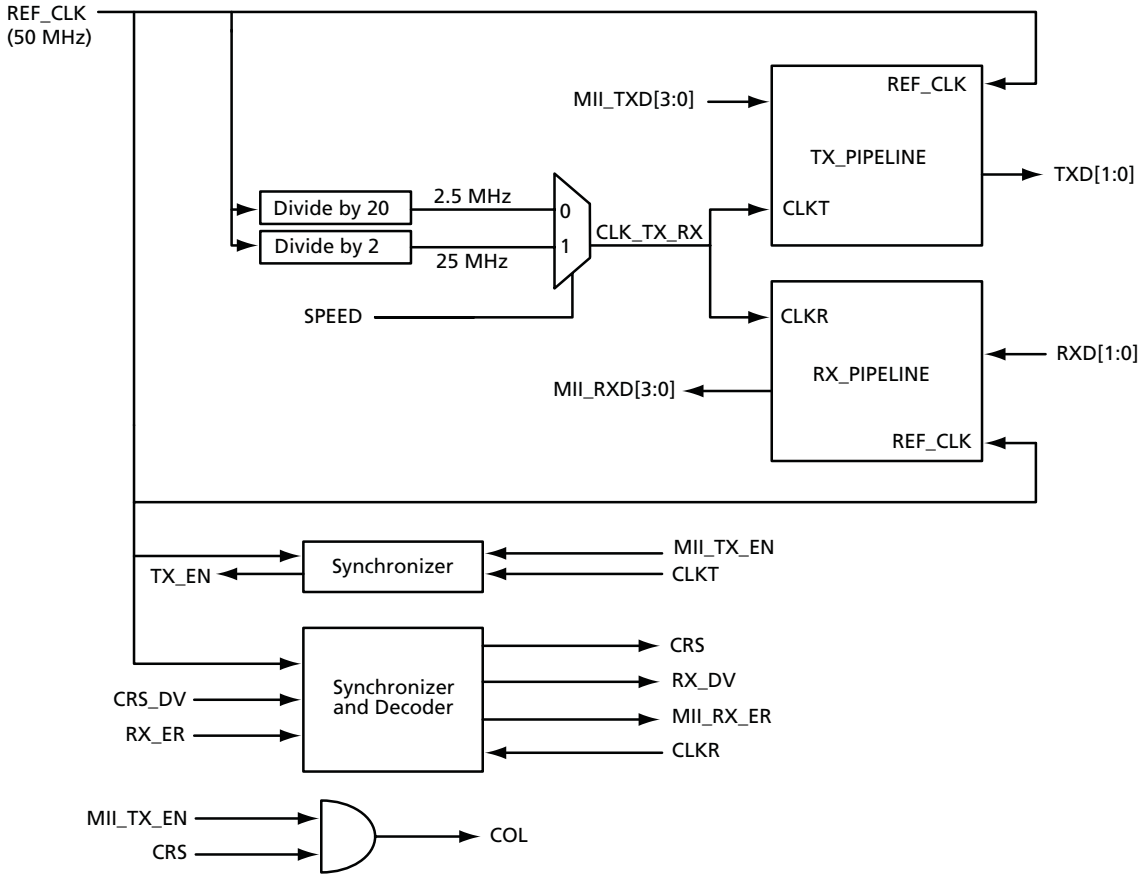

Figure 4-16 · MII_TO_RMII Internal Architecture

Interface Timing

Core10100—CSR Interface

CSR Read/Write Operation

The CSR read and write operations are synchronous to the positive edge of the CLKCSR signal and are illustrated in Figure 5-1. Read operations require that the data be read in the same clock cycle in which the csrreq signal is set to logic 1.

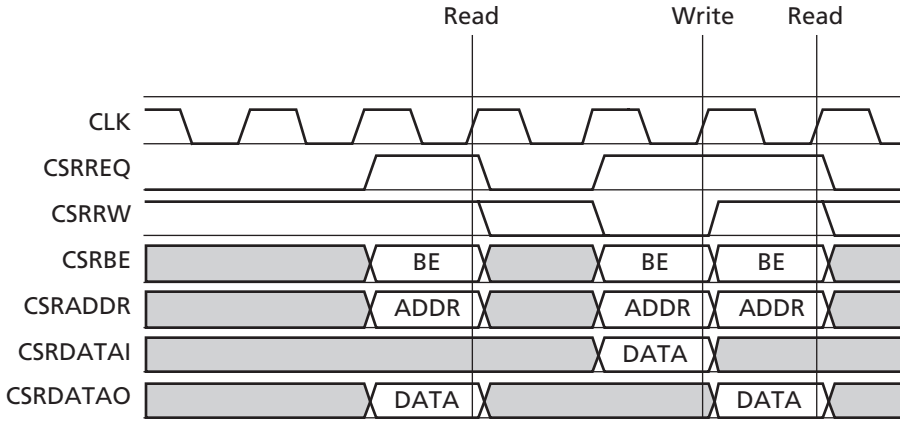

Figure 5-1 · CSR Read/Write Operation

Core10100—Data Interface

The data interface is used for data transfers between Core10100 and external shared system memory. It is a master via the DMA interface; i.e., Core10100 operates as an initiator on this data interface. The interface operates synchronously with the CLKDMA clock supplied by the system. The data width of the interface can be changed using the core parameter DATAWIDTH. Possible DATAWIDTH values are 8, 16, and 32. There are two data exchange types that can be initiated and performed by Core10100 via the DMA interface. The first data exchange type is the transmit and receive descriptors. These are set up by the host and fetched by the DMA interface to instruct Core10100 to exchange the Ethernet frame data in specified locations of shared RAM. The second data exchange type is the Ethernet data type.

Data Interface Write Operation

The data interface supports single or burst data transfer. The writes are operated on the positive edge of the clock CLKDMA. The write operation starts when the data interface sets DATAREQ to HIGH, and then the data interface waits until DATAACK from the host interface is set to HIGH (which indicates that the host is ready to receive the writes). A byte enable signal DATABE indicates the valid bytes on each write. The signal DATAOB indicates to the hosts that it is the end of a burst transfer. The signal DATAACK can be asserted or deasserted at any clock cycle; even in the middle of a burst transfer.

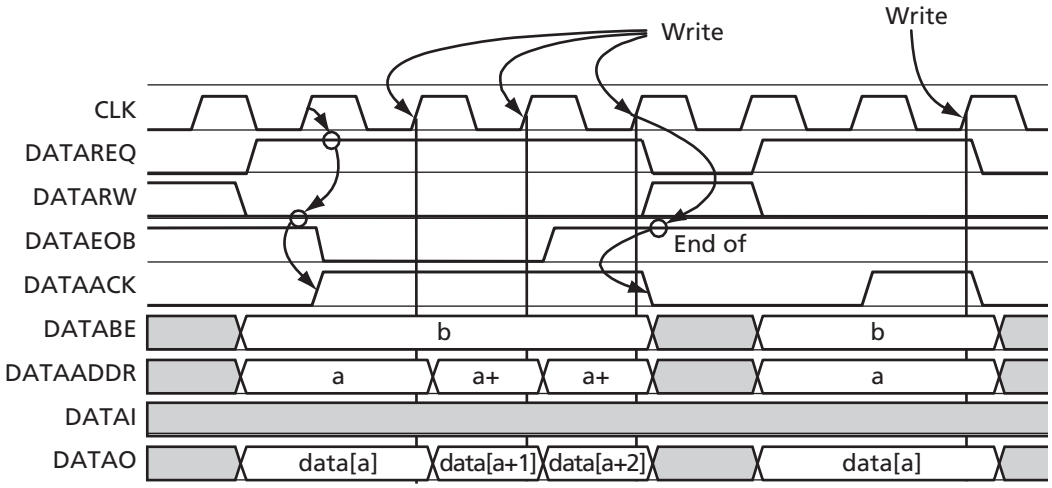

Figure 5-2 · Core10100 Host Data Write Operation

Data Interface Read Operation

The data interface supports single or burst data transfer. The reads are operated on the positive edge of the clock CLKDMA. The read operation starts when the data interface sets datareq to HIGH, and then the data interface waits until DATAACK from the host interface is set to HIGH (which indicates that the data is ready to be received by the data interface). A byte enable signal, databe, indicates the valid bytes on each read request. The signal DATAOB indicates to the hosts that it is the end of a burst transfer. dataack can be asserted or deasserted at any clock cycle, even in the middle of a burst transfer.

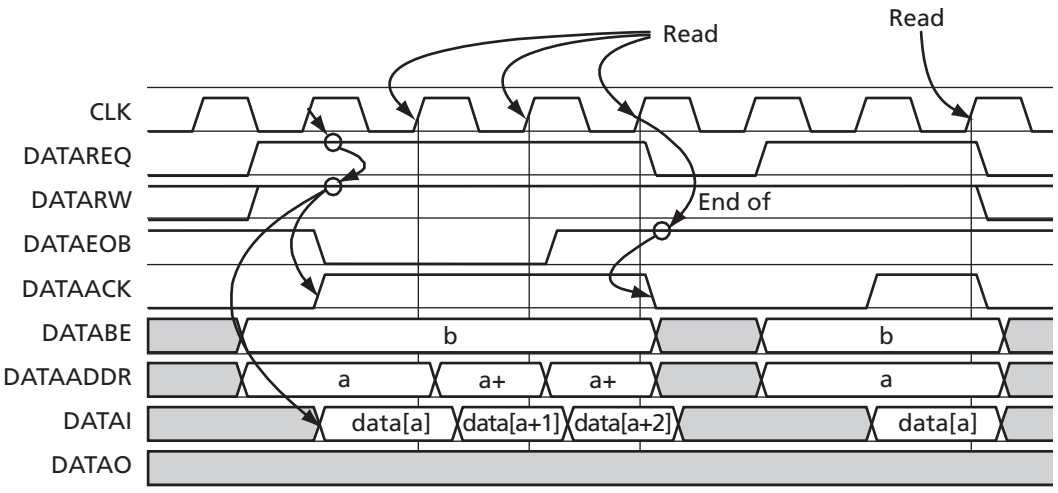

Figure 5-3 · Host Data Read Operation

Core10100_AHBAPB—APB Interface

Figure 5-4 and Figure 5-5 depict typical write cycle and read cycle timing relationships relative to the APB system clock, PCLK.

Figure 5-4 · Data Write Cycle

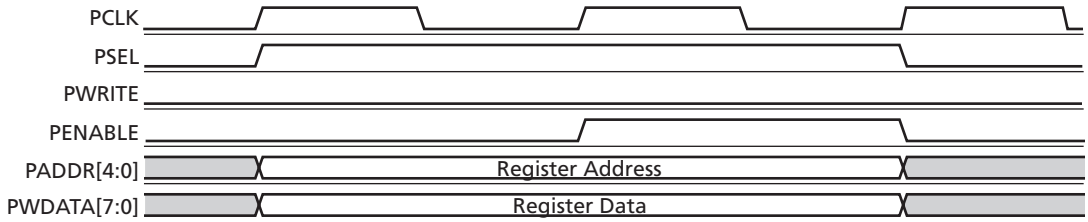

Figure 5-5 · Data Read Cycle

More detailed descriptions and timing waveforms can be found in the AMBA specification:

http://www.amba.com/products/solutions/AMBA_Spec.html.

Core10100_AHBAPB—AHB Interface

Core10100 implements an AMBA AHB–compliant master function on the core data interface, allowing the core to access memory for data storage. The AHB interface is compliant with the AMBA specification.

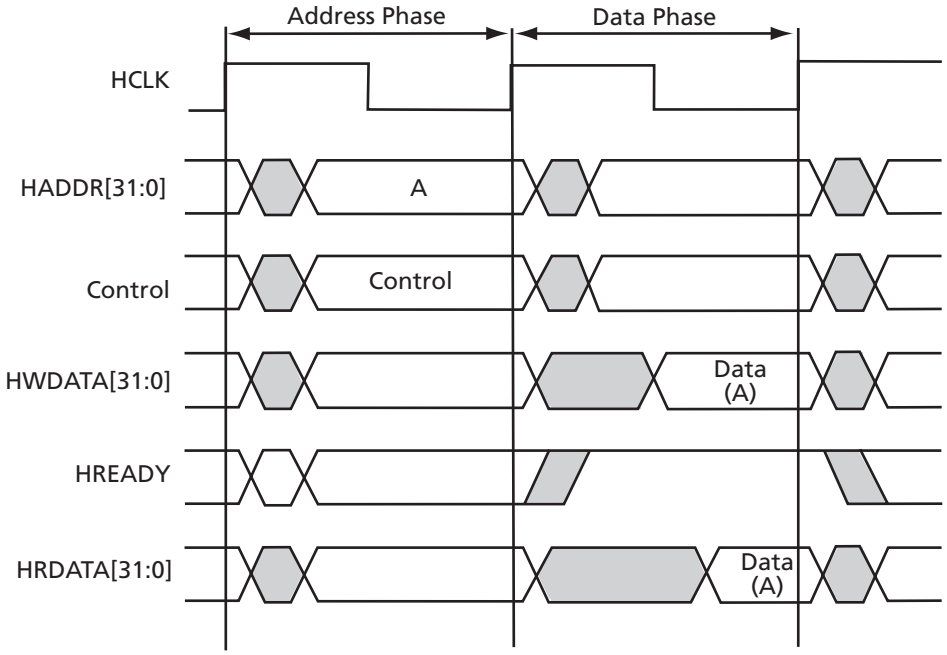

Figure 5-6 · Simple Transfer

More detailed descriptions and timing waveforms can be found in the AMBA specification:

http://www.amba.com/products/solutions/AMBA_Spec.html.

Core10100-RMII Interface

Core10100 implements the MII-to-RMII interface, which is compliant with the RMII specification. Full timing diagrams are available in the RMII specification:

http://www.national.com/appinfo/networks/files/rmii_1_2.pdf

Clock and Reset Control

Clock Controls

As shown in Figure 5-7 on page 69, there are four clock domains in the design:

- The TC and BD components operate synchronously with the CLKT clock supplied by the MII PHY device. This is a 2.5 MHz clock for 10 Mbps operation or a 25 MHz clock for 100 Mbps operation.
- The RC operates synchronously with the CLKR clock supplied by the MII PHY device. This is a 2.5 MHz clock for 10 Mbps operation or a 25 MHz clock for 100 Mbps operation.
- The TFIFO, RFIFO, TLSM, RLSM, and DMA components operate synchronously with the CLKDMA global clock supplied by the system.

- The CSR operates synchronously with the CLKCSR clock supplied by the system.

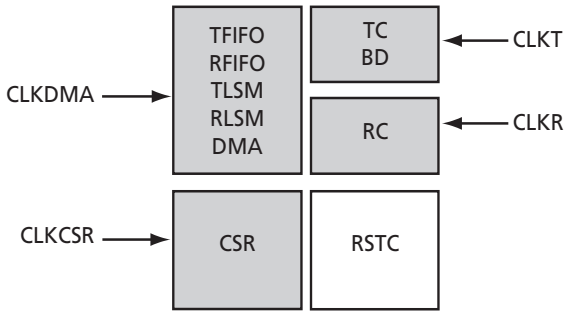

Figure 5-7 · Clock Domains and Reset

All clock signals are independent and can be asynchronous one to another. If needed, the CLKCSR and CLKDMA clock domains can be connected together with the same system clock signal in the user's system to consolidate global clock resources, or they can be from independent clock sources.

A minimum frequency of clock clkcsr is required for proper operation of the transmit, receive, and general-purpose timers. The minimum frequency for CLKCSR must be at least the clkt frequency divided by 64. For proper operation of the receive timer, the CLKCSR frequency must be at least the CLKR frequency divided by 64. If the clock frequency conditions described above are not met, do not use transmit interrupt mitigation control, receive interrupt mitigation control, or the general-purpose timer. Appropriate clocks should be also supplied when the hardware reset operation is performed.

Reset Control

Hardware Reset

Core10100 contains a single input RSTCSR signal. This signal is sampled in the RSTC component by clock CLKCSR. The RSTC component generates an internal asynchronous reset for every clock domain in Core10100. The internal reset is generated by the input RSTCSR and software reset. The internal reset remains active until the circuitry of all clock domains is reset.

The external reset signal must be active (HIGH) for at least one period of clock CLKCSR in the user's design. The minimum recovery time for a software reset is two CLKCSR periods plus one maximum clock period among CLKDMA, CLKT, and CLKR.

Software Reset

Software reset can be performed by setting the CSR0.0 (SWR) bit. The software reset will reset all internal flip-flops.

Timing Constraints

Actel recommends that correct timing constraints be used for the Synthesis and Layout stages of the design process. In particular, the cross-clock-domain paths must be constrained as follows:

- FROM "CLKDMA" TO "CLKT" uses clock period of CLKDMA
- FROM "CLKT" TO "CLKDMA" uses clock period of CLKT
- FROM "CLKDMA" TO "CLKR" uses clock period of CLKDMA
- FROM "CLKR" TO "CLKDMA" uses clock period of CLKR
- FROM "CLKCSR" TO "CLKT" uses clock period of CLKCSR
- FROM "CLKT" TO "CLKCSR" uses clock period of CLKT
- FROM "CLKCSR" TO "CLKR" uses clock period of CLKCSR
- FROM "CLKR" TO "CLKCSR" uses clock period of CLKR

Note: For Core10100_AHBAPB, CLKDMA should be replaced by HCLK and CLKSCR by PCLK.

6 Testbench Operation and Modification

User Testbench (Core10100)

An example user testbench is included with the Obfuscated and RTL releases of Core10100. The Obfuscated and RTL releases provide the precompiled Model*Sim* model, as well as the source code for the user testbench, to ease the process of integrating the Core10100 macro into a design and verifying it. A block diagram of the example user design and testbench is shown in Figure 6-1.

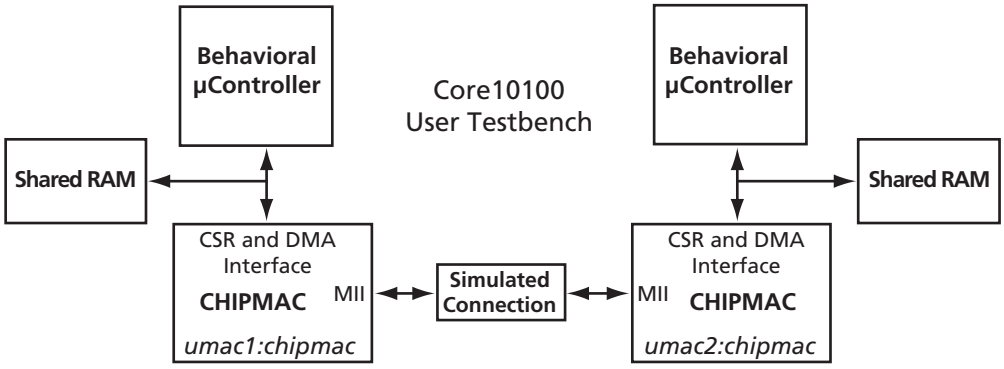

Figure 6-1 · Core10100 User Testbench

The user testbench includes a simple example design that serves as a reference for users who want to implement their own designs. RTL source code for the user testbench shown in Figure 6-1 is included in the source directory for the Obfuscated and RTL releases of Core10100.

The testbench for the example user design implements a subset of the functionality tested in the verification testbench, described in the previous chapter. Conceptually, as shown in Figure 6-1, two instantiations of the Core10100 core are connected via simulated connections in the user testbench. Example transmit and receive between the two Core10100 units is demonstrated by the user testbench so you can gain a basic understanding of how to use the core.

The source code for the user testbench contains the same example wrapper, CHIPMAC, used in the verification testbench. For details on the support routines (tasks for Verilog testbenches; functions and procedures for VHDL testbenches), see Appendix A: "User Testbench Support Routines" on page 75.

The user testbench consists of two cores: umac1 and umac2. In the example, umac1 transmits a 64-byte frame to umac2. To do so, the user testbench exercises the following steps:

For umac1:

1. Write several CSR registers to set up the operation mode.
2. Write two transmit descriptors into shared RAM (uram1).
3. Write the 64-byte data into shared RAM (uram1). The data consists of a sequence: 0, 1, 2, …, 63.
4. Turn on transmission.
5. Wait for the transmit interrupt.
6. Read the status register CSR5.
7. Clear the interrupt flags.

For umac2:

1. Write several CSR registers to set up the operation mode.
2. Write two receive descriptors into shared RAM (uram2).
3. Turn on receiving.
4. Wait for the receive interrupt.
5. Read the status register CSR5.
6. Check received data to match data sent by umac1.
7. Clear the interrupt flags.

The operations of umac1 and umac2 are concurrent.

AHBAPB User Testbench (Core10100_AHBAPB)

An example AHBAPB user testbench to exercise the AHB and APB interfaces on Core10100_AHBAPB is included with the Obfuscated and RTL releases of Core10100.

The Obfuscated and RTL releases provide the precompiled Model*Sim* model, as well as the source code for the user testbench, to ease the process of integrating the Core10100 macro into a design and verifying it.

A block diagram of the example user design and testbench is shown in Figure 6-2.

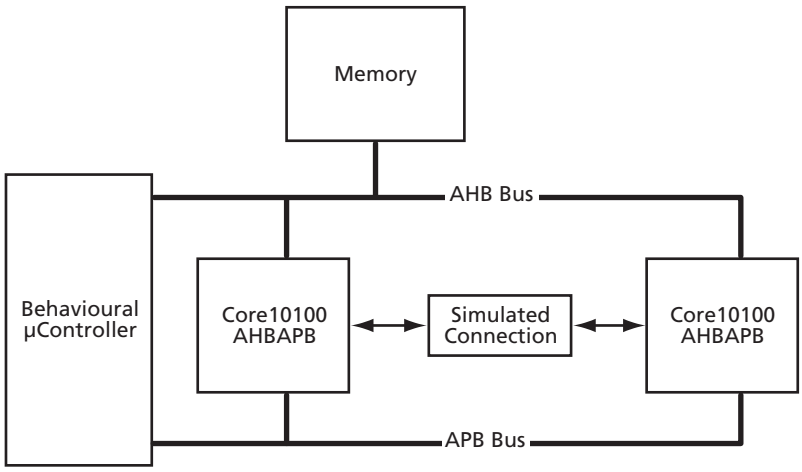

Figure 6-2 · Core10100_AHBAPB User Testbench

The testbench for the example user design implements the same test sequence as performed by the user testbench for Core10100. The difference is that the behavioral processor accesses memory via the AHB and accesses the core via the APB.

7 System Operation

This chapter provides various hints to ease the process of implementation and integration of Core10100 into your own design.

Usage with Cortex™-M1

Core10100 can also be used with Cortex-M1, the Actel soft IP version of the popular ARM® microprocessor that has been optimized for Actel FPGA devices. To create a design using Cortex-M1 and Core10100 (Figure 7-1 on page 73), you should use SmartDesign.

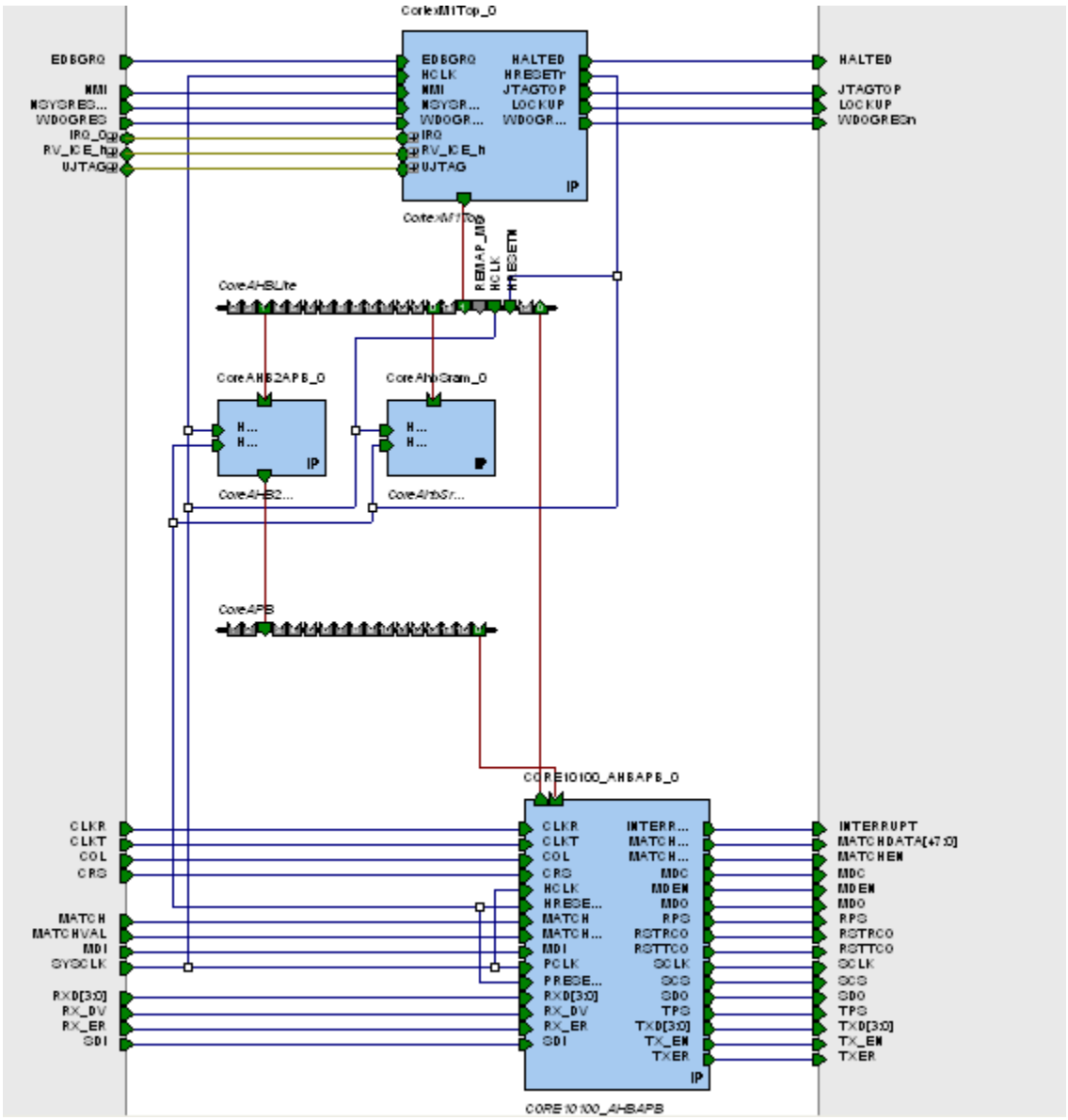

Figure 7-1 · Example System Using CoreMP7 and Core10100

A

User Testbench Support Routines

The verification and user testbenches for the Core10100 macro make use of various support routines, both in VHDL and Verilog. The various support routines are described in this appendix for the VHDL and Verilog testbenches.

VHDL Support

The VHDL support routines (procedures and functions) are provided within a package. The support routines are referenced from within the user testbenches, via library and use clauses.

Procedure Definitions

Procedure *print(arguments)*

Several *print* procedures are defined by overloading different argument types from string, integer, std_logic, and std_logic_vector.

Procedure *print_wt(arguments)*

Several *print_wt* procedures display information as the *print* procedure, but simulation time is added at the beginning of each display.

Procedure *print_tx_descriptor*

The procedure *print_tx_descriptor* displays detailed information about a transmit descriptor. It is defined below:

```
procedure print_tx_descriptor (
marks   : in STRING;
des0    : in integer;
des1    : in integer;
des2    : in integer;
des3    : in integer
) ;
```

The string *marks* is displayed at beginning of the information, and *des0, des1,des2*, and *des3* are the four 32-bit words of the transmit descriptor.

Procedure *print_rx_descriptor*

The procedure *print_rx_descriptor* displays detailed information about a receive descriptor. It is defined below:

```
procedure print_rx_descriptor (
marks   : in STRING;
des0    : in integer;
des1    : in integer;
des2    : in integer;
des3    : in integer
) ;
```

The string *marks* is displayed at beginning of the information, and *des0, des1,des2*, and *des3* are the four 32-bit words of the receive descriptor.

Procedure *print_csr5*

The procedure *print_csr5* displays detailed information on the CSR status register. It is defined below:

```
procedure print_csr5 (
marks   : in STRING;
csr     : in integer
);
```

The string *marks* is displayed at beginning of the information, and *csr* is the value of CSR register CSR5.

Procedure *write_csr*

The procedure *write_csr* writes a CSR register. It is defined below:

```
procedure write_csr (
signal  clk     : in std_logic;
signal  csrreq  : out std_logic;
signal  csrrw   : out std_logic;
signal  csrbe   : out std_logic_vector(CSRWIDTH/8-1 downto 0);
signal  csraddr : out std_logic_vector(CSRDEPTH-1 downto 0);
signal  csrdatai: out std_logic_vector(CSRWIDTH-1 downto 0);
signal  csrack  : in std_logic;
wa              : in integer;
wd              : in integer
)
```

The CLKCSR is *clk*. The value of the CSR register address is *wa*, and the value of the CSR register is *wd*.

Procedure *read_csr*

The procedure *read_csr* reads a CSR register. It is defined below:

```
procedure read_csr (
signal  clk     : in std_logic;
signal  csrreq  : out std_logic;
signal  csrrw   : out std_logic;
signal  csrbe   : out std_logic_vector(CSRWIDTH/8-1 downto 0);
signal  csraddr : out std_logic_vector(CSRDEPTH-1 downto 0);
signal  csrdatai: out std_logic_vector(CSRWIDTH-1 downto 0);
signal  csrack  : in std_logic;
ra              : in integer;
rd              : out integer
)
```

The clkcsr is *clk*. The value of the CSR register address is *ra*, and the value of the CSR register is *rd*.

Procedure *tb_write_data*

The procedure *tb_write_data* writes data into shared RAM, issued from the testbench. It is defined below:

```
procedure tb_write_data (
count           : in integer;
signal  clk     : in std_logic;
signal  we      : out std_logic;
signal  waddr   : out std_logic_vector(DATADEPTH-1 downto 0);
signal  wdata   : out std_logic_vector(DATAWIDTH-1 downto 0);
wa              : in integer;
wd              : in int_array
)
```

The CLKDMA is *clk, count* is number of the byte, *wa* is the beginning address of the sequence data, *wd* is an array storing the written data, *we* is the write enable issued from testbench, *waddr* is the write address to shared RAM issued from the testbench, and *wdata* is the write data bus issued from the testbench.

Procedure *tb_read_data*

The procedure *tb_read_data* reads data from shared RAM, issued from the testbench. It is defined below:

```
procedure tb_read_data (
count           : in integer;
signal  clk     : in std_logic;
signal  re      : out std_logic;
signal  raddr   : out std_logic_vector(DATADEPTH-1 downto 0);
signal  rdata   : out std_logic_vector(DATAWIDTH-1 downto 0);
ra              : in integer;
rd              : out int_array
)
```

The CLKDMA is *clk, count* is the number of bytes, *ra* is the beginning address of the sequence data, *rd* is an array storing the written data, *re* is the read enable issued from testbench, *raddr* is the read address to shared RAM issued from the testbench, and *rdata* is the read data to the testbench.

Procedure *tb_write_tx_descriptor*

The procedure *tb_write_tx_descriptor* writes a transmit descriptor into shared RAM, issued from the testbench. It is defined below:

```
procedure tb_write_tx_descriptor (
marks           : in STRING;
signal  clk     : in std_logic;
signal  we      : out std_logic;
signal  waddr   : out std_logic_vector(DATADEPTH-1 downto 0);
signal  wdata   : out std_logic_vector(DATAWIDTH-1 downto 0);
desaddr         : in integer;
des0            : in integer;
des1            : in integer;
des2            : in integer;
des3            : in integer
)
```

The string *marks* is displayed at beginning of the information, *clk* is the clkdma, *desaddr* is the beginning address of the descriptor, *we* is the write enable issued from the testbench, *waddr* is the write address to shared RAM issued from the testbench, *wdata* is the write data bus issued from the testbench, and *des0, des1,des2,* and *des3* are the four 32-bit words of the descriptor.

Procedure *tb_write_rx_descriptor*

The procedure *tb_write_rx_descriptor* writes a receive descriptor into shared RAM, issued from the testbench. It is defined below:

```
procedure tb_write_rx_descriptor (
marks           : in STRING;
signal  clk     : in std_logic;
signal  we      : out std_logic;
signal  waddr   : out std_logic_vector(DATADEPTH-1 downto 0);
signal  wdata   : out std_logic_vector(DATAWIDTH-1 downto 0);
desaddr         : in integer;
des0            : in integer;
des1            : in integer;
des2            : in integer;
des3            : in integer
)
```

The string *marks* is displayed at beginning of the information, *clk* is the CLKDMA, *desaddr* is the beginning address of the descriptor, *we* is the write enable issued from the testbench, *waddr* is the write address to shared RAM issued from the testbench, *wdata* is the write data bus issued from the testbench, and *des0, des1,des2,* and *des3* are the four 32-bit words of the descriptor.

Procedure *tb_read_descriptor*

The procedure *tb_read_descriptor* reads a receive descriptor into shared RAM, issued from the testbench. It is defined below:

```
procedure tb__descriptor (
signal  clk     : in std_logic;
signal  re      : out std_logic;
signal  raddr   : out std_logic_vector(DATADEPTH-1 downto 0);
signal  rdata   : out std_logic_vector(DATAWIDTH-1 downto 0);
desaddr         : in integer;
des0            : out integer;
des1            : out integer;
des2            : out integer;
des3            : out integer
)
```

The string *marks* is displayed at beginning of the information, *clk* is the CLKDMA, *desaddr* is the beginning address of the descriptor, *re* is the read enable issued from the testbench, *raddr* is the read address to shared RAM issued from the testbench, *rdata* is the read data to the testbench, and *des0, des1, des2,* and *des3* are the four 32-bit words of the descriptor.

Procedure *tb_read_check_descriptor*

The procedure *tb_read_check_descriptor* reads a receive descriptor from shared RAM, issued from the testbench and checked it against sequential data starting from 0 and incrementing with a step size of 1. It is defined below:

```
procedure tb_read_check_rx_data (
count           : in integer;
signal  clk     : in std_logic;
signal  re      : out std_logic;
signal  raddr   : out std_logic_vector(SHRAMDEPTH-1 downto 0);
signal  rdata   : in std_logic_vector(SHRAMWIDTH-1 downto 0);
ra              : in integer;
signal error    : inout integer
)
```

The CLKDMA is *clk, desaddr* is the beginning address of descriptor, *re* is the read enable issued from the testbench, *raddr* is the read address to shared RAM issued from the testbench, *rdata* is the read data to the testbench, *count* is the number of bytes, *ra* is the read address, and *error* is the error counter, which is incremented by the total number of mismatches.

Verilog Support

The Verilog versions of the testbenches make use of the following tasks, which are included within the top-level module of the user testbenches.

Verilog Tasks

Task Definitions

Task *print_tx_descriptor*

The task *print_tx_descriptor* displays detailed information on a transmit descriptor. It is defined below:

```
task print_tx_descriptor;
    input[STRINGSIZE-1:0] marks;
    input des0;
    integer des0;
    input des1;
    integer des1;
    input des2;
    integer des2;
    input des3;
    integer des3;
```

The string *marks* is displayed at beginning of the information, and *des0*, *des1*, *des2*, and *des3* are the four 32-bit words of the transmit descriptor.

Task *print_rx_descriptor*

The task *print_rx_descriptor* displays detailed information on a receive descriptor. It is defined below:

```
task print_rx_descriptor;
    input[STRINGSIZE-1:0] marks;
    input des0;
    integer des0;
    input des1;
    integer des1;
    input des2;
    integer des2;
    input des3;
    integer des3;
```

The string *marks* is displayed at beginning of the information, and *des0*, *des1*, *des2*, and *des3* are the four 32-bit words of the receive descriptor.

Task *print_csr5*

The task *print_csr5* displays detailed information on the CSR status register. It is defined below:

```
task print_csr5;
    input [STRINGSIZE-1 : 0] marks;
    input csr;
    integer csr;
```

The string *marks* is displayed at beginning of the information, and *csr* is the value of CSR register CSR5.

Task *u1_write_csr*

The task *u1_write_csr* writes a CSR register of MAC unit 1. It is defined below:

```
task u1_write_csr;
    input wa;
    integer wa;
    input wd;
    integer wd;
```

The variable *wa* is the value of the CSR register address, and *wd* is the value of the CSR register.

Task *u2_write_csr*

The task *u2_rite_csr* writes a CSR register of MAC unit 2. It is defined below:

```
task u2_write_csr;
    input wa;
    integer wa;
    input wd;
    integer wd;
```

The variable *wa* is the value of the CSR register address, and *wd* is the value of the CSR register.

Task *u1_read_csr*

The task *u1_read_csr* reads a CSR register in MAC unit 1. It is defined below:

```
task u1_read_csr;
    input ra;
    integer ra;
    output rd;
    integer rd;
```

The variable *ra* is the value of the CSR register address, and *rd* is the value of the CSR register.

Task *u2_read_csr*

The task *u2_read_csr* reads a CSR register in MAC unit 2. It is defined below:

```
task u2_read_csr;
    input ra;
    integer ra;
    output rd;
    integer rd;
```

The variable *ra* is the value of the CSR register address, and *rd* is the value of the CSR register.

Task *u1_write_data*

The task *u1_write_data* writes data into shared RAM unit 1, issued from the testbench. It is defined below:

```
task u1_write_data;
    input count;
    integer count;
    input wa;
    integer wa;
    input[MAX_DATA_ARRAY_SIZE-1:0] wd;
```

The variable *count* is number of bytes, *wa* is the beginning address of the sequence data, and *wd* is an array storing the written data.

Task *u2_write_data*

The task *u2_write_data* writes data into shared RAM unit 2, issued from the testbench. It is defined below:

```
task u2_write_data;
    input count;
    integer count;
    input wa;
    integer wa;
    input[MAX_DATA_ARRAY_SIZE-1:0] wd;
```

The variable *count* is number of bytes, *wa* is the beginning address of the sequence data, and *wd* is an array storing the written data.

Task *u1_read_data*

The task *u1_read_data* reads data from shared RAM unit 1, issued from the testbench. It is defined below:

```
task u1_read_data;
    input count;
    integer count;
    input ra;
    integer ra;
    input[MAX_DATA_ARRAY_SIZE-1:0] rd;
```

The variable *ra* is the beginning address of the sequence data, *rd* is an array storing the written data, and *re* is the read enable issued from the testbench.

Task *u2_read_data*

The task *u2_read_data* reads data from shared RAM unit 2, issued from the testbench. It is defined below:

```
task u2_read_data;
    input count;
    integer count;
    input ra;
    integer ra;
    input[MAX_DATA_ARRAY_SIZE-1:0] rd;
```

The variable *ra* is the beginning address of the sequence data, *rd* is an array storing the written data, and *re* is the read enable issued from the testbench.

Task *u1_write_tx_descriptor*

The task *u1_write_tx_descriptor* writes a transmit descriptor into shared RAM unit 1, issued from the testbench. It is defined below:

```
task u1_write_tx_descriptor;
```

```
    input [STRINGSIZE-1 : 0] marks;
    input desaddr;
    integer desaddr;
    input des0;
    integer des0;
    input des1;
    integer des1;
    input des2;
    integer des2;
    input des3;
    integer des3;
```

The string *marks* is displayed at the beginning of the information, and *des0, des1, des2,* and *des3* are the four 32-bit words of the transmit descriptor.

Task *u2_write_tx_descriptor*

The task *u2_write_tx_descriptor* writes a transmit descriptor into shared RAM unit 2, issued from the testbench. It is defined below:

```
task u2_write_tx_descriptor;
    input [STRINGSIZE-1 : 0] marks;
    input desaddr;
    integer desaddr;
    input des0;
    integer des0;
    input des1;
    integer des1;
    input des2;
    integer des2;
    input des3;
    integer des3;
```

The string *marks* is displayed at the beginning of the information, *desaddr* is the starting address of the descriptor, and *des0, des1, des2,* and *des3* are the four 32-bit words of the transmit descriptor.

Task *u1_write_rx_descriptor*

The task *u1_write_rx_descriptor* writes a receive descriptor into shared RAM unit 1, issued from the testbench. It is defined below:

```
task u1_write_rx_descriptor;
    input [STRINGSIZE-1 : 0] marks;
    input desaddr;
    integer desaddr;
    input des0;
    integer des0;
    input des1;
    integer des1;
    input des2;
    integer des2;
    input des3;
    integer des3;
```

The string *marks* is displayed at the beginning of the information, *desaddr* is the starting address of the descriptor, and *des0*, *des1*, *des2*, and *des3* are the four 32-bit words of the receive descriptor.

Task *u2_write_rx_descriptor*

The task *u2_write_rx_descriptor* writes a receive descriptor into shared RAM unit 2, issued from the testbench. It is defined below:

```
task u2_write_rx_descriptor;
    input [STRINGSIZE-1 : 0] marks;
    input desaddr;
    integer desaddr;
    input des0;
    integer des0;
    input des1;
    integer des1;
    input des2;
    integer des2;
    input des3;
    integer des3;
```

The string *marks* is displayed at the beginning of the information, *desaddr* is the starting address of the descriptor, and *des0*, *des1*, *des2*, and *des3* are the four 32-bit words of the receive descriptor.

Task *u1_read_rx_descriptor*

The task *u1_read_rx_descriptor* reads a receive descriptor from shared RAM unit 1, issued from the testbench. It is defined below:

```
task u1_read_rx_descriptor;
    input [STRINGSIZE-1 : 0] marks;
    input desaddr;
    integer desaddr;
    output des0;
    integer des0;
    output des1;
    integer des1;
    output des2;
    integer des2;
    output des3;
    integer des3;
```

The string *marks* is displayed at the beginning of the information, *desaddr* is the starting address of the descriptor, and *des0*, *des1*, *des2*, and *des3* are the four 32-bit words of the descriptor.

Task *u2_read_rx_descriptor*

The task *u2_read_rx_descriptor* reads a receive descriptor from shared RAM unit 2, issued from the testbench. It is defined below:

```
task u2_read_rx_descriptor;
    input [STRINGSIZE-1 : 0] marks;
    input desaddr;
    integer desaddr;
    output des0;
    integer des0;
    output des1;
    integer des1;
    output des2;
    integer des2;
    output des3;
    integer des3;
```

The string *marks* is displayed at the beginning of the information, *desaddr* is the starting address of the descriptor, and *des0*, *des1*, *des2*, and *des3* are the four 32-bit words of the descriptor.

Task *u1_read_check_descriptor*

The task *u1_read_check_descriptor* reads a receive descriptor from shared RAM unit 1, issued from the testbench and checked against a sequence of data starting from 0 and incrementing at a step size of 1. It is defined below:

```
task u1_read_check_rx_data;
    input count;
    integer count;
    input ra;
    integer ra;
```

The variable *ra* is the starting address, and *count* is the total number of bytes of the checked data.

Task *u2_read_check_descriptor*

The task *u2_read_check_descriptor* reads a receive descriptor from shared RAM unit 2, issued from the testbench and checked against a sequence of data starting from 0 and incrementing at a step size of 1. It is defined below:

```
task u2_read_check_rx_data;
    input count;
    integer count;
    input ra;
    integer ra;
```

The variable *ra* is the starting address, and *count* is the total number of bytes of the checked data.

Transmit and Receive Functional Timing Examples

Transmit Examples

Transmit Overview

A typical Core10100 transmit is shown in Figure B-1.

1. Host sends the transmit command and Core10100 enters the transmit process.
2. Core10100 starts to request the descriptors.
3. Core10100 starts to request frame data and write them into the transmit FIFO.
4. Core10100 starts to transmit a frame on the MII interface.

A typical transmit undergoes these four processes.

In this chapter, more detailed dataflow diagrams are provided to illustrate the timing information for the above four processes.

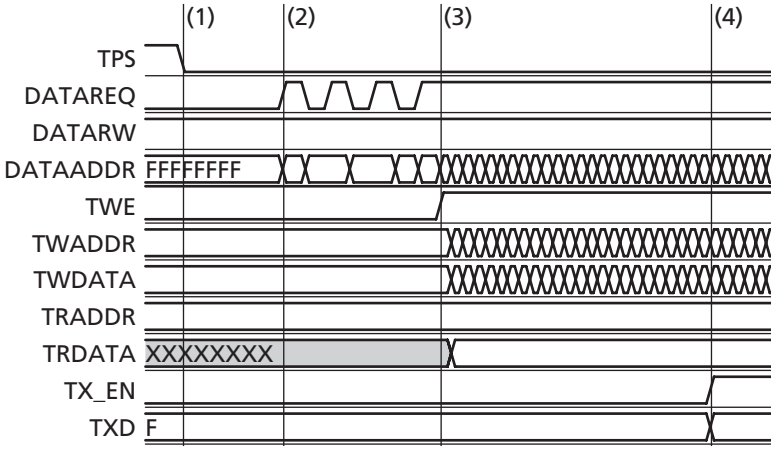

Figure B-1 · A Typical Transmit Dataflow

Core10100 Enters Transmit Process

The block CSR performs this operation.

1. Host sets the CSR register CSR6.13 ST to start transmit.
2. The tps signal goes LOW after one CLKCSR cycle, which indicates that Core10100 enters the transmit process.

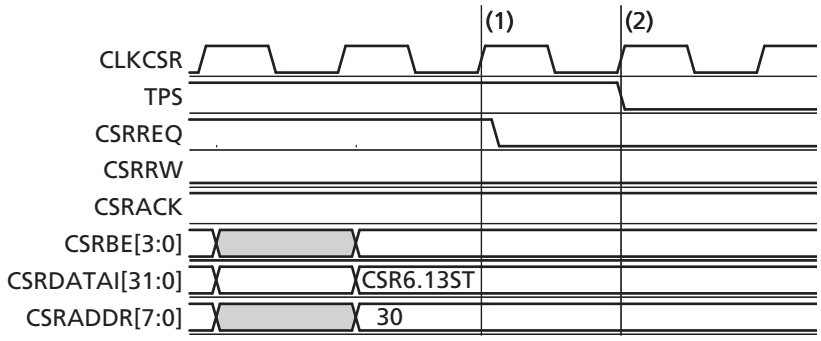

Figure B-2 · Enters Transmit Process

Core10100 Starts to Request Transmit Descriptors

Figure B-3 illustrates operations between TPS going LOW and a transmit descriptor start.

1. Host sends the transmit start command.
2. Core10100 starts to fetch the first descriptor.

Note: t0 = 4 × CLKDMA period + 3 × CLKCSR period + z.

Where z is 2 × CLKDMA period if CLKDMA period is greater than CLKCSR period, or z is 2 × CLKCSR period if CLKCSR period is greater than CLKDMA period. Delay z is the result of handshaking between CSR clock domain and other domains in the design.

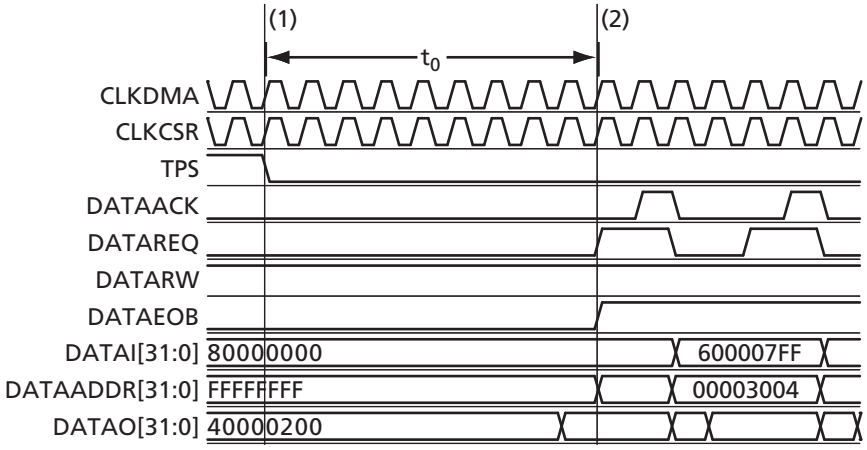

Figure B-3 · Core10100 Starts Transmit Descriptor Requests

Transmit Descriptor and Data Fetches

Transmit Descriptor Fetch in 32-Bit Mode

1. Read the first 32-bit word of transmit descriptor.
2. Read the second 32-bit word of transmit descriptor.
3. Read the third 32-bit word of transmit descriptor.
4. Read the first 32-bit data fetch and write into transmit FIFO.
5. Read the second 32-bit data fetch and write into transmit FIFO.

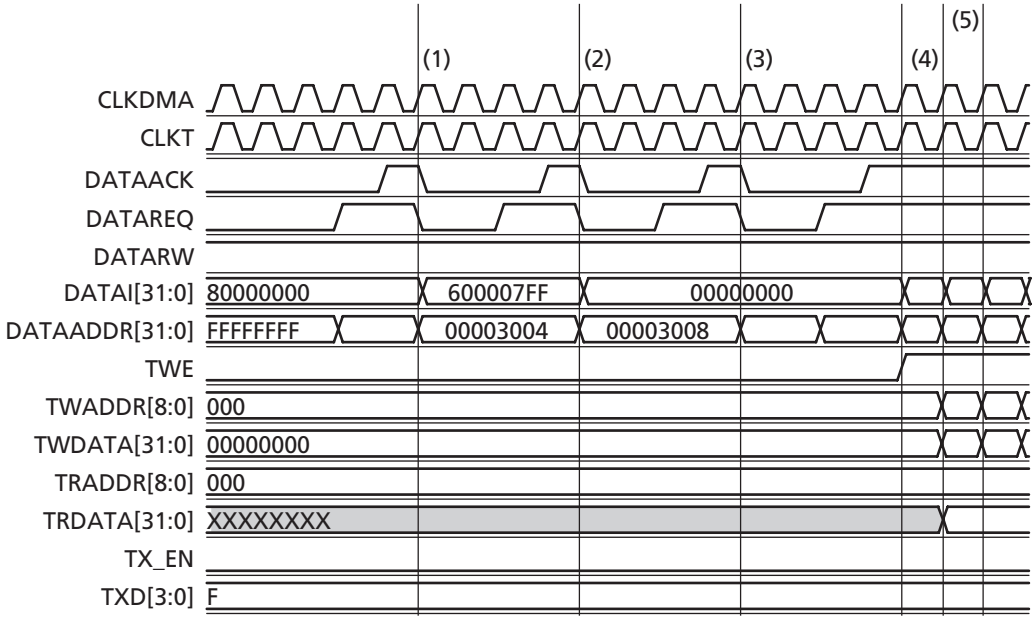

Figure B-4 · Transmit Descriptor Fetch in 32-Bit Mode

Note: An extra cycle is inserted between any two descriptor fetches.

Transmit Descriptor and Data Fetch in 16-Bit Mode

1. Read the first 16-bit word of transmit descriptor.
2. Read the second 16-bit word of transmit descriptor.
3. Read the third 16-bit word of transmit descriptor.
4. Read the fourth 16-bit word of transmit descriptor.
5. Read the fifth 16-bit word of transmit descriptor.
6. Read the sixth 16-bit word of transmit descriptor.
7. Read the first 16-bit data fetch and write into transmit FIFO.
8. Read the second 16-bit data fetch and write into transmit FIFO.
9. Read the third 16-bit data fetch and write into transmit FIFO.
10. Read the fourth 16-bit data fetch and write into transmit FIFO.

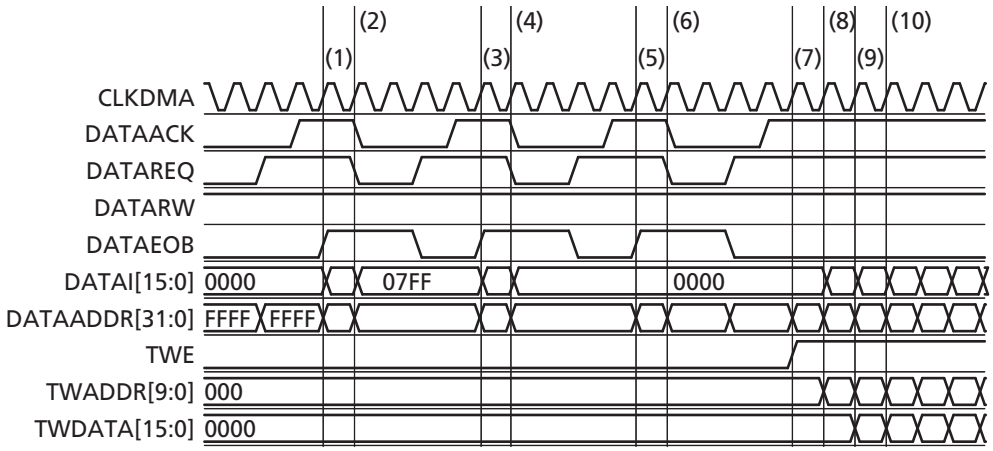

Figure B-5 · Transmit Descriptor Fetch in 16-Bit Mode

Transmit Descriptor and Data Fetch in 8-Bit Mode

1. Four reads of the first to fourth 8-bit words of the transmit descriptor.
2. Four reads of the fifth to eighth 8-bit words of the transmit descriptor.
3. Four reads of the ninth to twelfth 8-bit words of the transmit descriptor.
4. Read the first 8-bit data fetch and write into the transmit FIFO.
5. Read the second 8-bit data fetch and write into the transmit FIFO.
6. Read the third 8-bit data fetch and write into the transmit FIFO.
7. Read the fourth 8-bit data fetch and write into the transmit FIFO.

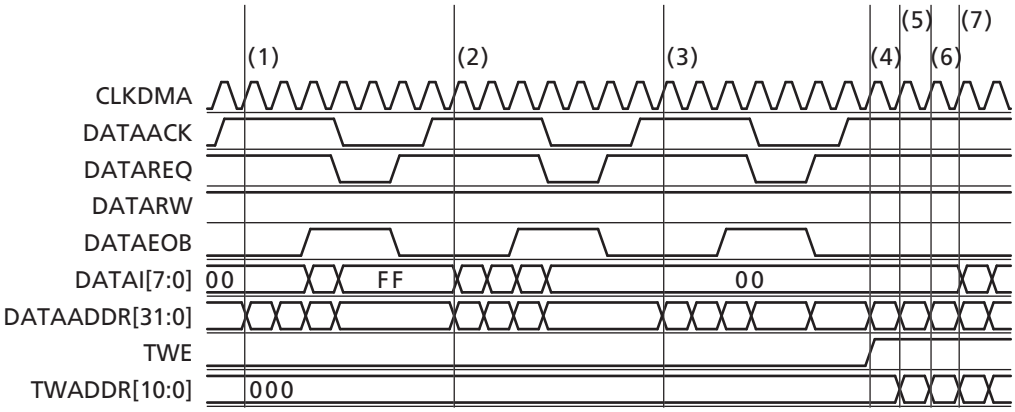

Figure B-6 · Transmit Descriptor Fetch in 8-Bit Mode

Core10100 Starts to Transmit on MII

1. Core10100 starts to write to the Transmit Data RAM.
2. Core10100 reaches the transmit FIFO level (see Table 4-11 on page 30). Figure B-7 on page 91 shows that the transmit FIFO threshold is set at 64 bytes, with sixteen 32-bit word writes.
3. Transmit starts on MII.

Note: t0 = CLKDMA period × FIFO threshold level / DATAWIDTH × 8 or
t0 = CLKDMA period × frame size / DATAWIDTH × 8 in store and forward mode, and
t1 = 3 × CLKDMA period + 5 × CLKT period.

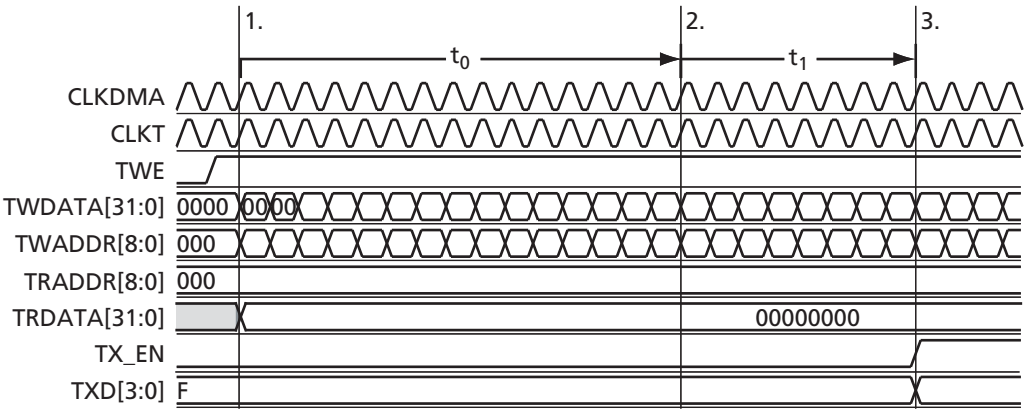

Figure B-7 · Transmit FIFO Threshold and Start of Transmit on MII

Transmit on MII

1. Core10100 starts to transmit the preamble and SFD.
2. Core10100 sends the read address to the External Transmit Data RAM.
3. Core10100 reads the first 32 bits of data.
4. Core10100 starts to transmit the data

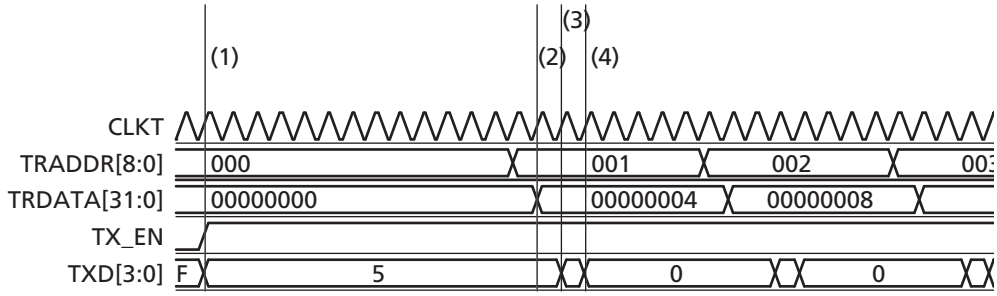

Figure B-8 · Transmit on MII

Transmit on MII with 32-Bit Transmit Data RAM

(1), (2) Core10100 sends out requested read addresses. t0 is eight cycles.

(3), (4) t1 is the time between Core10100 sending out a read address request and the appearance of the requested data on MII.

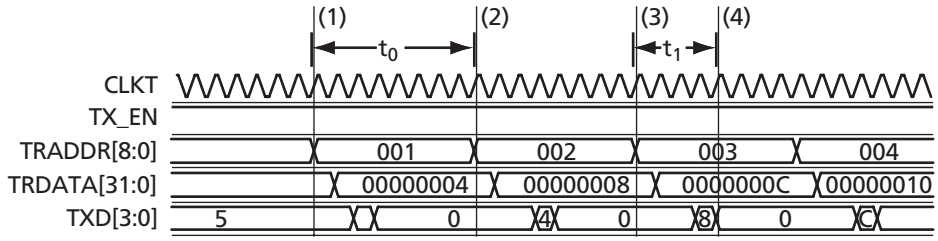

Figure B-9 · Transmit on MII with 32-Bit Transmit Data RAM

Transmit on MII with 16-Bit Transmit Data RAM

(1), (2) Core10100 sends out requested read addresses. t0 is four cycles.

(3), (4) t1 is the time between Core10100 sending out a read address request and the appearance of the requested data on MII.

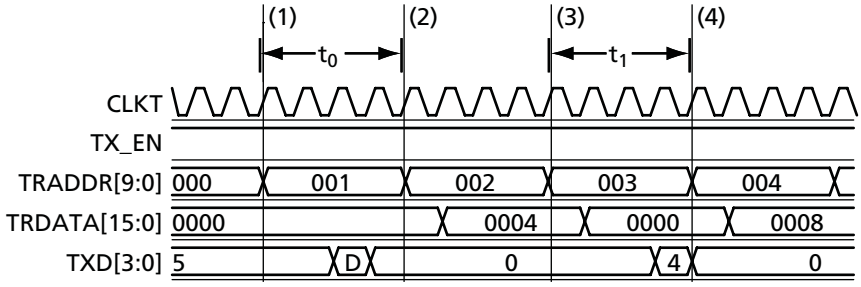

Figure B-10 · Transmit on MII with 16-Bit Transmit Data RAM

Transmit on MII with 8-Bit Transmit Data RAM

(1), (2) Core10100 sends out requested read addresses. t0 is two cycles.

(3), (4) t1 is the time between Core10100 sending out a read address request and the appearance of the requested data on MII.

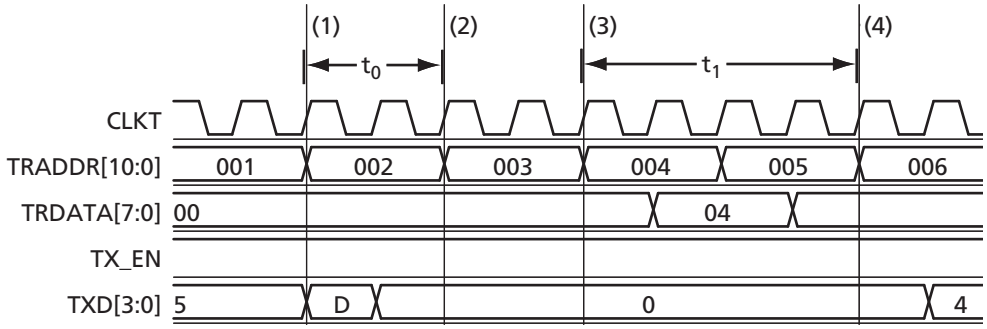

Figure B-11 · Transmit on MII with 8-Bit Transmit Data RAM

Receive Examples

Receive Dataflow Overview

Core10100 receives Ethernet data from the MII interface, and the Receive Controller writes the received data into the Receive Data RAM. The RFIFO Controller for Core10100 starts to transfer received data from the Receive Data RAM to the shared memory via the DMA unit when the data in the Receive Data RAM exceeds 64 bytes. Figure B-12 on page 93 illustrates the received data travelling through different Core10100 interfaces. A typical receive consists of the following steps (as shown in Figure B-12 on page 93):

1. Core10100 starts to receive the preamble and SFD.
2. Core10100 starts to write the receive data to the Receive Data RAM.
3. Core10100 writes the 64th byte of the received data to the receive FIFO.
4. Core10100 starts to transfer received data from the Received Data RAM to the shared RAM.

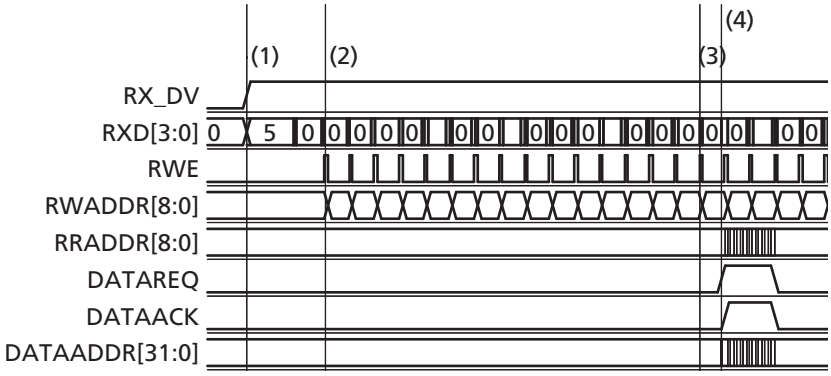

Figure B-12 · A Typical Receive Example

Core10100 Receives and Writes Receive Data RAM

Core10100 Receives and Writes 32-Bit Receive Data RAM

1. Core10100 starts to receive the preamble.
2. Core10100 starts to receive the packet.
3. Core10100 starts to write the first 32-bit word into the receive FIFO.
4. Core10100 starts to write the second 32-bit word into the receive FIFO.

Note: t0 = 16 × CLKR period, t1 = 8 × CLKR period.

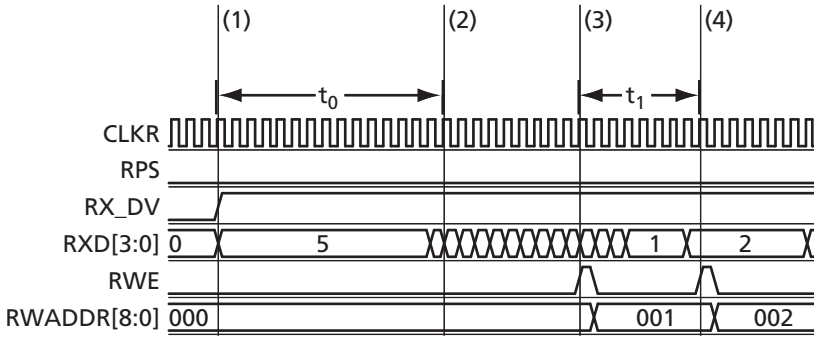

Figure B-13 · Core10100 Receives and Writes Receive Data RAM

Core10100 Receives and Writes 16-Bit Receive Data RAM

1. Core10100 starts to receive the preamble.
2. Core10100 starts to receive the packet.
3. Core10100 starts to write the first 16-bit word into the receive FIFO.
4. Core10100 starts to write the second 16-bit word into the receive FIFO.

Note: t0 = 16 × CLKR period, t1 = 4 × CLKR period

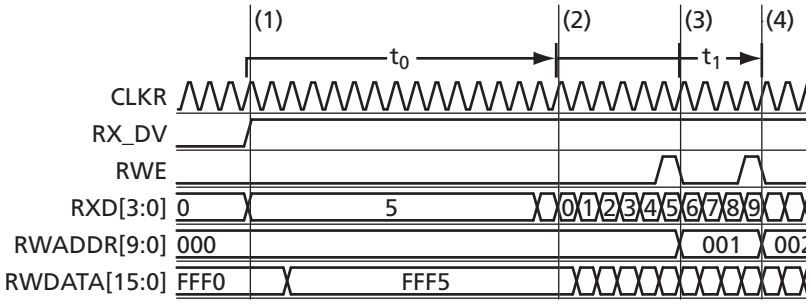

Figure B-14 · Core10100 Receives and Writes 16-Bit Receive Data RAM

Core10100 Receives and Writes 8-Bit Receive Data RAM

1. Core10100 starts to receive the preamble.
2. Core10100 starts to receive the packet.
3. Core10100 starts to write the first 8-bit word into the receive FIFO.
4. Core10100 starts to write the second 8-bit word into the receive FIFO.

Note: t0 = 16 × CLKR period, t1 = 2 × CLKR period.

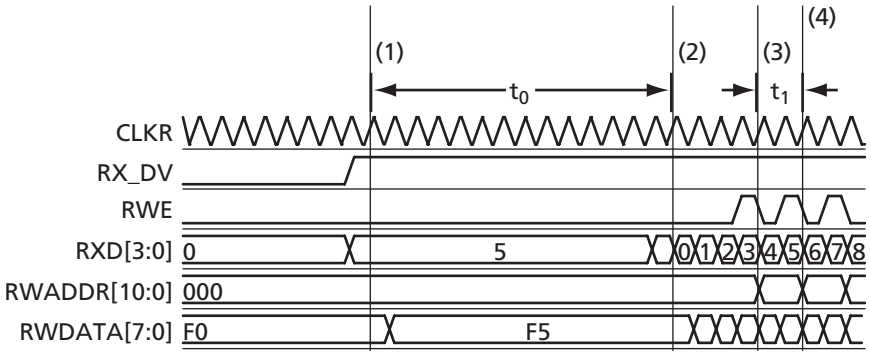

Figure B-15 · Core10100 Receives and Writes 8-Bit Receive Data RAM

Transfer Receive Data to Shared Memory

32-Bit Word Transfer from Receive Data RAM to Shared Memory

1. Core10100 writes the 64th byte of the frame into the Receive Data RAM.
2. Core10100 starts to send the data request to transfer received data into the shared memory.
3. The first 32-bit word is written into the shared memory via the data interface.
4. The 64th byte of the frame is written into the shared memory.

Note: t0 = 6 × CLKDMA period.

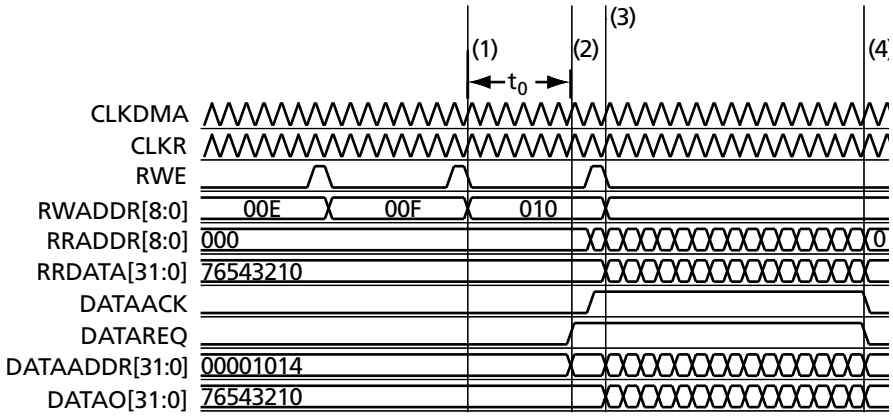

Figure B-16 · 32-Bit Word Transfer From Receive Data RAM to Shared Memory

16-Bit Word Transfer from Receive Data RAM to Shared Memory

1. Core10100 writes the 64th byte of the frame into the Receive Data RAM.
2. Core10100 starts to send the data request to transfer received data into the shared memory.
3. The first 32-bit word is written into the shared memory via the data interface.
4. The 64th byte of the frame is written into the shared memory.

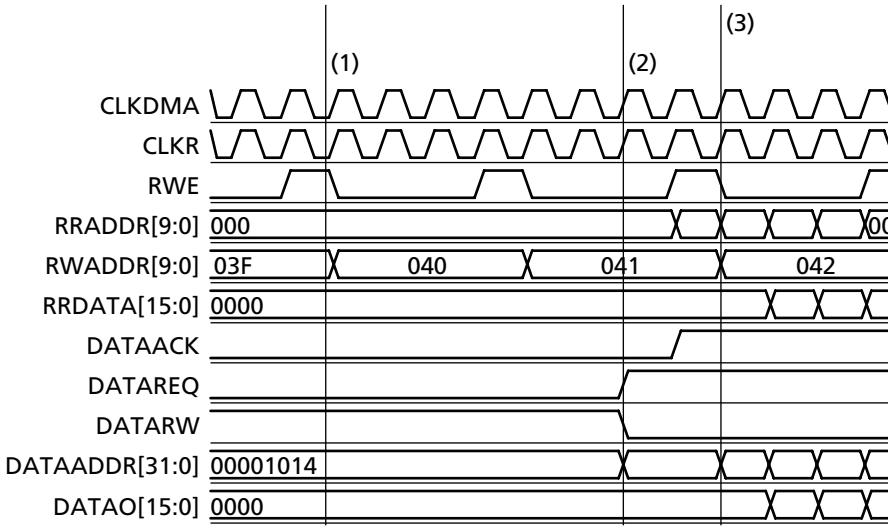

Figure B-17 · 16-Bit Word Transfer from Receive Data RAM to Shared Memory

8-Bit Word Transfer from Receive Data RAM to Shared Memory

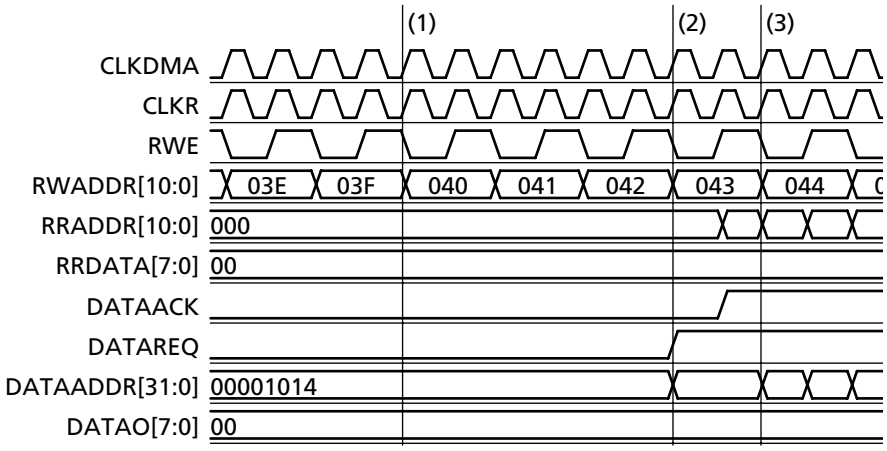

Figure B-18 · 8-Bit Word Transfer from Receive Data RAM to Shared Memory

Core10100 Receive Descriptor Fetch

The receive descriptor fetch timing is essentially the same as the transmit descriptor fetch timing. In reality, transmit descriptor fetches and receive descriptor fetches can happen mixed or alternately through the DMA interface. Refer to Figure B-4 on page 89, Figure B-5 on page 90, and Figure B-6 on page 90.

List of Document Changes

The following table lists critical changes that were made in the current version of the document.

Previous Version	Changes in Current Version (v4.0)	Page
v3.1	The core name Core10/100 was changed to Core10100, and Core10/100-AHB was changed to Core10100_AHB. The core version was changed from v3.2 to v4.0.	N/A
	Figure 1 · Core10100 Block Diagram and Figure 2 · Typical Core10100 Application were updated to change MII to RMII/MII.	5
	Figure 3 · ARM-Based System Using Core10100_AHBAPB was replaced.	6
	Instances of CoreConsole were changed to SmartDesign throughout the document.	N/A
	Table 1 · Core10100 Device Utilization and Performance for an 8-Bit Datapath through Table 6 · Core10100_AHBAPB Device Utilization and Performance for a 32-Bit Datapath were updated.	7–8
	Table 7 · Parameter Settings was revised to update the TFIFODEPTH and RFIFODEPTH values. A row was added for RMII. The DATADEPTH parameter value was changed to 20 for the 8-bit cores.	10
	Figure 1-1 · Core 10100 Architecture and Figure 1-2 · Core10100_AHBAPB Architecture were revised to add an MII to RMII block.	11–12
	The "CSR – Control/Status Register Logic" section was revised to remove reference to the power management functionality of Core10100.	13
	The "Licensing" section was revised to remove the Evaluation version. The CoreConsole section was removed. Figure 2-1 · Core10100 Configuration within SmartDesign and Figure 2-2 · Core10100_AHBAPB Configuration within SmartDesign replaced previous figures of configuration in CoreConsole. The "Importing into Libero IDE" section and "Simulation Flows" section were updated. The "Synthesis in Libero IDE" section is new.	15–18
	The introduction to the "Interface Descriptions" section was revised to list interfaces as CSR and AMBA instead of legacy, AHB, and APB.	19
	ProASIC3L was added to the values for the FAMILY parameter in Table 3-1 · Core10100 Parameters. A default value column was added. The acceptable values were revised for the following parameters: DATADEPTH, TCDEPTH, RCDEPTH, TFIFODEPTH, and RFIFODEPTH. The description was revised for TCDEPTH and RCDEPTH.	19
	A default value column was added to Table 3-2 · Core10100_AHBAPB Parameters. Acceptable values were revised for the following parameters: AHB_AWIDTH, TCDEPTH, RCDEPTH, TFIFODEPTH, and RFIFODEPTH. The description was revised for TCDEPTH and RCDEPTH.	20
	The section title "Legacy Interface Signals" was changed to "CSR Interface Signals". Table 3-3 · Core10100 Signals and Table 3-4 · Signals Included in Core10100 and Core10100_AHBAPB were revised to change signal names to all capital letters. The subheading "MII PHY Interface" in Table 3-4 was revised and is now "RMII/MII PHY Interface." Three signal names changed: rxer to RX_ER, rxdv to RX_DV, and txen to TX_EN. The descriptions were revised to include RMII for RMII_CLK and CRS_DV. The descriptions for CLKT, CLKR, RX_ER, RXD,TXD, and RMII_CLK were revised in Table 3-4 · Signals Included in Core10100 and Core10100_AHBAPB. Table 3-5 · Core10100_AHBAPB Signals was moved to the end of the chapter.	22–24
	The reset value for CSR9 was changed to FFF483FFH in Table 4-1 · CSR Locations.	27
	The permissible values were changed for PBL in Table 4-3 · Bus Mode Register Bit Functions. 1 and 2 are no longer permissible values.	28

Previous Version	Changes in Current Version (v4.0)	Page
v3.1 (cont'd)	The function for TTM was revised to add information about RMII mode in Table 4-16 · Operation Mode Register Bit Functions.	33
	In Table 4-23 · MII Management and Serial ROM Interface Register (CSR9), MII was changed to MDEN.	38
	The title of Figure 4-1 · I/O Tristate Buffer Connections was changed from "External Tristate Buffer Connections."	39
	The function for CON was revised to add information about when the bit should be written in Table 4-26 · General-Purpose Timer and Interrupt Mitigation Control Bit Functions.	40
	The description for RX_ER was revised in Table 4-40 · External PHY Interface Signals.	57
	The "MII to RMII Interface" section was revised to state the internal clock net CLK_TX_RX must be assigned to a global clock network. The CLK_TX_RX was added to Figure 4-16 · MII_TO_RMII Internal Architecture.	64
	Figure 5-6 · Simple Transfer is new.	68
	The "Core10100-RMII Interface" section is new.	68
	The Verification Testbench section was removed. References to the Evaluation release of Core10100 were removed from the "Testbench Operation and Modification" section.	71
	The "Usage with Cortex™-M1" section replaced the "Usage with CoreMP7" section.	73
	The "Software Drivers" chapter of the handbook was deleted. It contained only the following text: "Example software drivers are available from Actel for Core10100. Contact Actel Technical Support for information (tech@actel.com)."	N/A
	The "Verification Tests Description" Appendix was removed.	N/A
v3.0	All references to the Core100100 datasheet were removed, as it has been superseded by the Core10100 Handbook.	N/A
v2.3	The "Memory Blocks" section was updated to add information on the MII and RMII.	14
	The RMII parameter was added to Table 3-1 · Core10100 Parameters and Table 3-2 · Core10100_AHBAPB Parameters.	19, 20
	The RMII_CLK and CRS_DV signals were added to Table 3-4 · Signals Included in Core10100 and Core10100_AHBAPB.	24
	The descriptions for CSR6.13 and CSR6.1 were updated in Table 4-16 · Operation Mode Register Bit Functions.	33
	The description for CSR8.(15..0) was updated in Table 4-22 · Missed Frames and Overflow Counter Bit Functions.	38
	A new sentence was added to the end of the "MAC Address and Setup Frames" section regarding setup frame buffer size.	49
	The first three rows of Table 4-38 · Perfect Filtering Setup Frame Buffer were revised.	50
	The sentence, "Before writing to CSR4, the MAC must be in a stopped state" was added to the "Transmit Process" section. The sentence, "Before writing to CSR3, the MAC must be in a stopped state" was added to the "Receive Process" section. A sentence was also added to clarify when the receive state machine goes into a stopped state.	52, 54
	The following sentence was added to the "Receive Address Filtering" section: "To receive the broadcast frame, the hash table bit corresponding to the broadcast address CRC value should be set."	62
	The "Steps for Calculating CRC with Hash Filtering" section is new.	63
	The "MII to RMII Interface" section is new.	64
v2.2	The "Supported Device Families" section was added and the "Memory Requirements" section was updated to include ProASIC3L.	7, 10

Previous Version	Changes in Current Version (v4.0)	Page
v2.1	Core version changed from v3.1 to v3.2.	7
	The "RC – Receive Controller" section was updated to remove the words "using an external address RAM" from the sentence about internal address filtering.	14
	Table 3-1 · Core10100 Parameters and Table 3-2 · Core10100_AHBAPB Parameters were updated for the ADDRFILTER description.	19, 20
	In Table 4-1 · CSR Locations, the reset value for CSR9 was updated. A table note was added.	27
	A paragraph was added to the function description for the CSR0.0 bit in Table 4-3.	28
	Table 4-23 · MII Management and Serial ROM Interface Register (CSR9) was updated to add one column.	38
	Table 4-24 · MII Management and Serial ROM Register Bit Functions was updated to change the symbol for bit CSR9.18 to MDEN. An explanatory sentence was added to the function.	38
	Figure 4-1 · I/O Tristate Buffer Connections and the text preceding it were updated to indicate an active low enable with the tristate buffer.	39
	The "Receive Address Filtering" section was updated to add a sentence at the end describing how to enable the functionality discussed.	62
v2.0	Added version number to cover page.	Title
	Added the "Core Versions" section.	7
	Added the IGLOO/e family to Table 1, Table 2, Table 3, Table 4, Table 5, and Table 6.	7–9
	Added the IGLOO/e family to the "Memory Requirements" section.	10
	Changed the Reset Value for CSR1 and CSR2 in Table 4-1.	27
	Changed the left-hand column of Table 4-27 to RDES0–3.	43
	Changed the left-hand column of Table 4-32 to TDES0–3.	46
	Changed the datar signal to "datarw" in Figure 5-2 and Figure 5-3.	66

D

Product Support

Actel backs its products with various support services including Customer Service, a Customer Technical Support Center, a web site, an FTP site, electronic mail, and worldwide sales offices. This appendix contains information about contacting Actel and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From Northeast and North Central U.S.A., call **650.318.4480**
From Southeast and Southwest U.S.A., call **650. 318.4480**
From South Central U.S.A., call **650.318.4434**
From Northwest U.S.A., call **650.318.4434**
From Canada, call **650.318.4480**
From Europe, call **650.318.4252** or **+44 (0) 1276 401 500**
From Japan, call **650.318.4743**
From the rest of the world, call **650.318.4743**
Fax, from anywhere in the world **650.318.8044**

Actel Customer Technical Support Center

Actel staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Customer Technical Support Center spends a great deal of time creating application notes and answers to FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Actel Technical Support

Visit the Actel Customer Support website (www.actel.com/custsup/search.html) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the Actel web site.

Website

You can browse a variety of technical and non-technical information on Actel's home page, at www.actel.com.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. Several ways of contacting the Center follow:

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is tech@actel.com.

Phone

Our Technical Support Center answers all calls. The center retrieves information, such as your name, company name, phone number and your question, and then issues a case number. The Center then forwards the information to a queue where the first available application engineer receives the data and returns your call. The phone hours are from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. The Technical Support numbers are:

650.318.4460
800.262.1060

Customers needing assistance outside the US time zones can either contact technical support via email (tech@actel.com) or contact a local sales office. Sales office listings can be found at www.actel.com/contact/offices/index.html.

Index

Actel is the leader in low-power and mixed-signal FPGAs and offers the most comprehensive portfolio of system and power management solutions. Power Matters. Learn more at www.actel.com.

Actel Corporation *• 2061 Stierlin Court • Mountain View, CA 94043 • USA*
Phone 650.318.4200 • Fax 650.318.4600 • Customer Service: 650.318.1010 • Customer Applications Center: 800.262.1060

Actel Europe Ltd*. • River Court, Meadows Business Park • Station Approach, Blackwater • Camberley Surrey GU17 9AB • United Kingdom*
Phone +44 (0) 1276 609 300 • Fax +44 (0) 1276 607 540

Actel Japan *• EXOS Ebisu Building 4F • 1-24-14 Ebisu Shibuya-ku • Tokyo 150 • Japan*
Phone +81.03.3445.7671 • Fax +81.03.3445.7668 • http://jp.actel.com

Actel Hong Kong *• Room 2107, China Resources Building • 26 Harbour Road • Wanchai • Hong Kong*
Phone +852 2185 6460 • Fax +852 2185 6488 • www.actel.com.cn

50200077-6/2.09