



XAPP536 (v1.1) June 3, 2004

Gigabit System Reference Design

Author: Xilinx Systems Engineering Group

Summary

This application note describes the Gigabit System Reference Design (GSRD). The GSRD leverages the techniques outlined in Xilinx application note XAPP535 (available online soon) to demonstrate a high-performance Gigabit Ethernet reference system using a Xilinx Virtex-II Pro™ Field-Programmable Gate Array (FPGA). This application note contains four major sections.

The “[GSRD EDK Reference System](#)” section discusses how the GSRD EDK Reference System (`m1300_gsr_d_gemac_tft`) included in the design files, is capable of booting the Linux operating system, and running Netperf over the Gigabit Ethernet link. The reference system has been built and verified using the Xilinx ML300 Evaluation Platform, Xilinx ISE FPGA tools, and the Xilinx Embedded Development Kit (EDK). The GSRD EDK Reference System (`m1300_gsr_d_gemac_tft`) consists of the three main elements:

- Multi Port Memory Controller (MPMC)
- Communications DMA Controller (CDMAC)
- LocalLink GMAC Peripheral

The MPMC is a quad port memory controller that is used to provide memory access for the PPC405 and DMA engines. The PPC405 CPU is a Harvard architecture CPU; therefore, it provides separate Processor Local Bus (PLB) ports for the instructions and data. GSRD connects the Instruction and Data MB ports to two of the ports on the MPMC.

The CDMAC uses two ports of the MPMC to provide two full duplex channels of DMA. XAPP535 describes the MPMC and CDMAC system components in greater detail, and the benefits of a multi-ported memory controller.

The LocalLink GMAC Peripheral provides a Gigabit Ethernet Interface. This peripheral uses a streaming interface – the Xilinx LocalLink interface. LocalLink is a lightweight streaming interface for communication devices that provides a simple protocol to transfer data in a single direction. Full duplex communication devices such as the GMAC Peripheral utilize two LocalLink interfaces. The LocalLink interface specification can be found by registering for the [Aurora Reference Design](#).

The “[GSRD Hardware Peripheral Data Sheets](#)” section contains data sheets for each hardware peripheral in GSRD.

The “[GSRD Software Components](#)” section provides an overview of the software provided with GSRD. GSRD provides two software applications as performance metrics: bare metal Ethernet and Netperf on Linux. These applications will help to explore the boundaries of performance that exist in various use models.

The “[Building the GSRD Reference System under EDK](#)” section contains instructions for using EDK to build the GSRD Reference System, run simulations, and run applications on real hardware ([Xilinx ML300 Evaluation Platform](#)).

© 2004 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information “as is.” By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Introduction

The six key ideas upon which GSRD relies for high performance are outlined below:

- Parallel architecture leveraged from XAPP535
- The PPC 405 does not process payload data
- Support for Arbitrary Alignment of DMA Buffers using byte re-alignment
- Transport layer checksum offload
- Jumbo frames (9KB)
- Interrupt Moderation

GSRD alleviates the PPC 405 CPU from processing payload data by eliminating the need for the processor to move data buffers and perform checksum calculations over the payload data. The PPC405 processes only the headers of TCP/IP and related protocols.

Byte re-alignment is a feature of the CDMAC described in XAPP535. It is significant for GSRD because Ethernet and TCP/IP are byte wide protocols. The CDMAC contains byte-shifting logic to support the movement from/to any memory byte offset. This feature eliminates the requirement for the CPU to copy buffers before DMA occurs, relieving pressure on memory bandwidth and freeing the processor to do other work.

The LocalLink GMAC Peripheral is capable of performing the checksum calculation in hardware as the payload data is DMA'd from memory to the peripheral (and vice versa). The combination of the support for byte re-alignment and checksum offload serves to completely remove the PPC405 CPU from the high-speed Ethernet data path.

Standard Ethernet uses 1518 byte frames. Jumbo frame enabled Ethernet uses larger frames – typically 9KB. The support of Jumbo frames reduces the number of Ethernet frames per datum. Utilizing Jumbo frames, more data is transferred on the network with the same PPC405 header-processing rate.

Instead of interrupting the processor on every Ethernet frame, Interrupt Moderation waits until there are several frames to process before interrupting the processor, amortizing the interrupt overhead across multiple Ethernet frames. If too many interrupts are generated, the CPU will spend all of its time processing these interrupts, and no time executing application code. This is a state known as *live lock*. Using jumbo frames helps to eliminate the occurrence of live lock.

GSRD EDK Reference System

This section describes the contents of the GSRD Reference System and provides information about how the system is organized, implemented, and verified. The information presented introduces many aspects of the reference system, but additional and more detailed information about the software, tools, peripherals, and interface protocols exists in the “[GSRD Hardware Peripheral Data Sheets](#)” section and the “[GSRD Software Components](#)” section, as well as in XAPP535.

The GSRD EDK Reference System can be found in the ZIP file under the following directory.

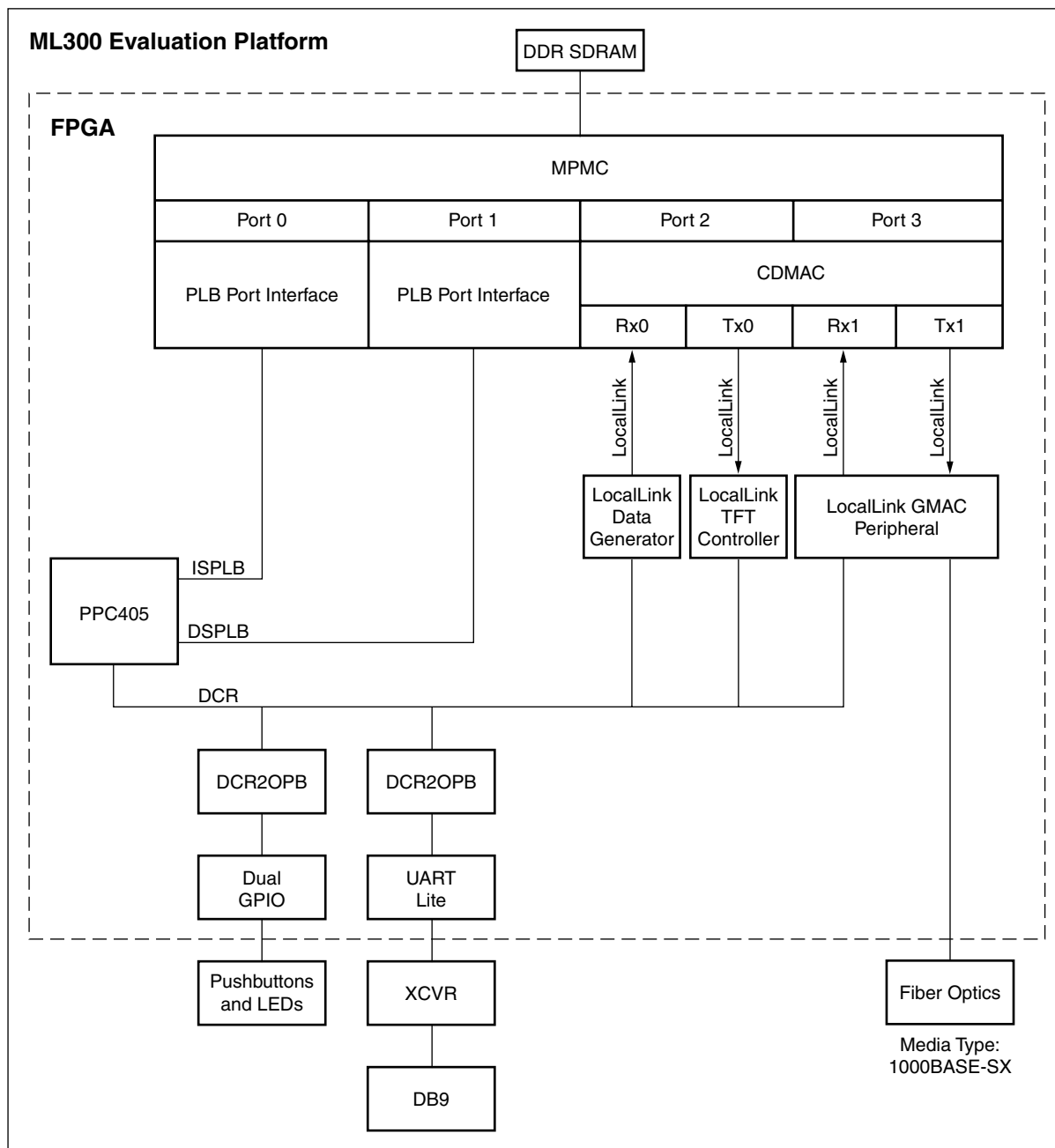
```
gsrd/projects/ml300_gsr_d_gemac_tft/
```

Hardware

[Figure 1](#) provides a high-level view of the hardware components of the system. This design demonstrates a system built around the MPMC coupled with 32-bit DDR SDRAM memory. A dual engine CDMAC connects to two ports of the MPMC. The instruction and data side PPC405 ports connect to the other two MPMC ports via PLB-to-MPMC Interface modules. LocalLink interfaces connect the CDMAC to the Gigabit Ethernet MAC, Color Bar Data Generator, and TFT Controller. LocalLink is a protocol specification optimized for high-performance streaming communications applications such as Gigabit Ethernet.

Lower bandwidth devices such as the Universal Asynchronous Receiver Transmitter (UART), Interrupt Controller, and the General Purpose Input Output (GPIO) are attached to the CPU DCR bus. DCR is an IBM CoreConnect bus primarily used with control and status registers

where simplicity is desired. Refer to the *DCR CoreConnect Architecture Specifications* for more information. The use of DCR for peripherals reduces the loading on the high bandwidth MPMC ports while minimizing FPGA resource utilization since large bus bridges can be avoided.



X536_01_051204

Figure 1: GSRD EDK Reference System High-Level Block Diagram

MPMC

The MPMC allows the 32-bit DDR SDRAM memory resource to be shared over 4 independent interface ports. More information about the MPMC and CDMAC components is available in XAPP535.

Two MPMC ports are connected to the two PLB ports of the PPC405 via PLB to MPMC Interface modules. The PLB to MPMC Interfaces translates transactions from the Instruction

and Data side PLB ports of the PPC405 into MPMC transactions. It handles all the necessary handshaking signals and clock synchronization between the PLB and MPMC interfaces. The remaining two MPMC ports attach to the CDMAC. This permits the CDMAC to manage the flow of two bi-directional streams of data to and from memory.

CDMAC

The CDMAC manages the flow of data between peripherals and memory. It supports variable packet sizes and can transfer data from/to arbitrary aligned memory addresses (byte resolution). CDMAC control and status registers are accessible by the PPC405 via the DCR interface.

The CDMAC is configured so that the Data Generator and TFT Controller will not have errors generated when the DMA engine reaches a descriptor with the *completed* bit set. However, if the DMA engine reaches a descriptor with the *completed* bit set on the GMAC Peripheral ports, an error is generated.

LocalLink Devices

LocalLink is a protocol for a point-to-point connection infrastructure optimized for streaming communications applications. The protocol supports flow control from the source or destination side of the data transfer. It also includes additional control signals to mark the start and end of frames and data payloads.

The GMAC Peripheral, a TFT Controller, and Data Generator are connected to the CDMAC in this reference system.

DCR

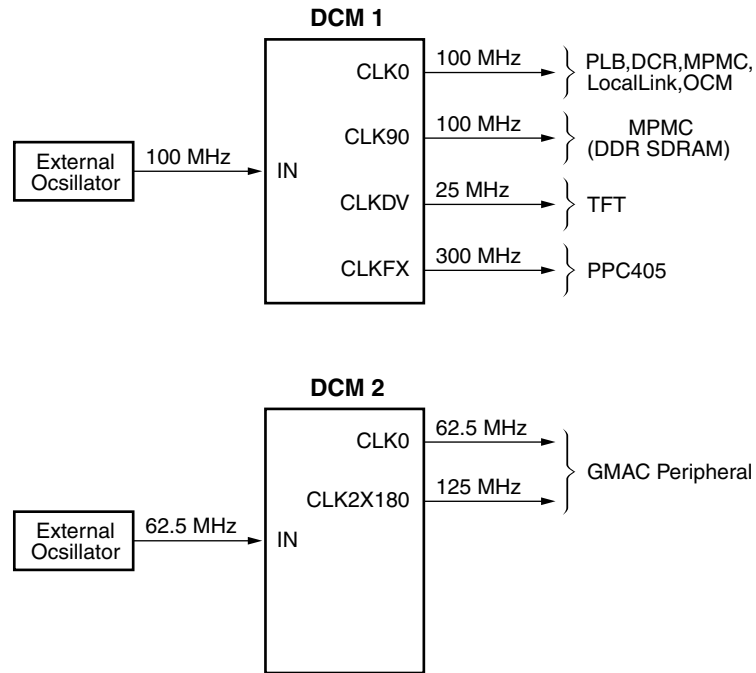
The DCR offers a very simple interface protocol and is used for accessing control and status registers in various devices. It allows for register access to various devices without loading down the high-speed interfaces. Since DCR devices are generally accessed infrequently and do not have high-performance requirements, they are used throughout the reference design for functions such as error status registers, interrupt controllers, and device initialization logic.

Interrupts

An interrupt controller for non-critical interrupts is controlled through the DCR. It allows multiple edge or level-sensitive interrupts from peripherals to be OR'ed together back to the CPU.

Clock Generation and Distribution

Virtex-II Pro FPGAs have abundant clock management and global clock buffer resources. The reference system uses these capabilities to generate a variety of different clocks. Figure 2 illustrates the use of the Digital Clock Managers (DCMs) for generating the main clocks in the design. A 100 MHz input clock is used to generate the main system clock that drives the PLB, MPMC, LocalLink, and OCM components. The CLK90 output of the DCM produces a 100 MHz clock that is phase shifted by 90 degrees for use by the MPMC. The main 100 MHz clock is divided down by four to create a 25 MHz TFT video clock. The CPU clock is multiplied up from the PLB clock to 300 MHz. A second DCM uses the 62.5 MHz input reference clock to generate 62.5 MHz and 125 MHz clocks with the necessary phase relationship for the GMAC Peripheral.



X536_02_051204

Figure 2: GSRD Reference System Clock Generation

CPU Debug via JTAG

The CPU can be debugged via JTAG with a variety of software development tools. The preferred method of communicating with the CPU via JTAG is to combine the CPU JTAG chain with the FPGA's main JTAG chain, which is also used to download bit files. Sharing the same JTAG chain for CPU debug and FPGA programming simplifies the number of cables needed since a single JTAG cable (like the Xilinx Parallel IV Cable) can be used for bit file download as well as CPU software debugging.

Other Devices

The GSRD Reference System contains 16KB Instruction-Side and 16KB Data-Side OCM modules. The OCM consists of Block RAM directly connected to the CPU. They allow the CPU fast access to memory and are useful for providing instructions and/or data directly to the CPU, bypassing the caches.

IP Version and Source

Table 1 summarizes the IP cores making up the reference design. The sum of the cores used in this design are shipped with the EDK product. Others are part of the reference design.

Table 1: IP Cores in the GSRD EDK Reference System

Hardware IP	Version	Source
bram_block	1.00.a	Local EDK Installation
cdmac	1.00.a	"gsrd_lib" Library
clk_rst_startup	1.00.a	Local "pcores" Directory
dcr_intc	1.00.b	Local EDK Installation
dcr_v29	1.00.a	Local EDK Installation
dcr2opb_bridge	1.00.a	"gsrd_lib" Library
dcr2opb_bridge	1.00.a	Local EDK Installation
dsbram_if_cntrl	2.00.a	Local EDK Installation
dsocm_v10	1.00.b	Local EDK Installation
isbram_if_cntrl	2.00.a	Local EDK Installation
isocm_v10	1.00.b	Local EDK Installation
ll_data_gen	1.00.a	"gsrd_lib" Library
ll_gmac_periph	1.00.a	"gsrd_lib" Library
ll_tft_cntrl	1.00.a	"gsrd_lib" Library
misc	1.00.a	Local "pcores" Directory
mpmc	1.00.a	"gsrd_lib" Library
my_jtag_logic	1.00.a	Local "pcores" Directory
opb_gpio	2.00.a	Local EDK Installation
opb_uartlite	1.00.b	Local EDK Installation
opb_v20	1.10.b	Local EDK Installation
plb_m1s1	1.00.a	"gsrd_lib" Library
plb_mpmc_if	1.00.a	"gsrd_lib" Library
ppc_trace	1.00.a	Local "pcores" Directory
ppc405	2.00.c	Local EDK Installation

Simulation and Verification

Simulation Overview

For simulation, the main test bench module `testbench.v` instantiates the FPGA (`system.v`) as the device under test and includes behavioral models for the FPGA to interact with. In addition to behavioral models for memory devices, clock oscillators, and external peripherals, the test bench also instantiates a CoreConnect bus monitor to observe the DCR bus for protocol violations. The test bench can also preload some of the memories in the system for purposes such as loading software for the CPU to execute. The MGT transmit and receive signal pairs are tied together in a loopback configuration. The `sim_params.v` file is designed to be modified by the user to customize various simulation options. These options include message display options, maximum simulation time, and clock frequency. The user should edit this file to reflect personal simulation preferences.

SWIFT and BFM CPU Models

The reference system demonstrates two different simulation methods to help verify designs using the PPC405 CPU. One method uses a full simulation model of the CPU based on the actual silicon. The second method employs Bus Functional Models (BFMs) to generate processor bus cycles from a command scripting language. These two methods offer different trade-offs between behavior in real hardware, ease of generating bus cycles, and the amount of real time to simulate a given clock cycle.

A SWIFT model can be used to simulate the CPU executing software instructions. In this scenario, the executable binary images of the software are preloaded into memory from which the CPU can boot up and run the code. Though this is a relatively slow way to exercise the design, it more accurately reflects the actual behavior of the system.

The SWIFT model is most useful for helping to bring up software and for correlating behavior in real hardware with simulation results. The reference system demonstrates the SWIFT model simulation flow by allowing the user to write a C program that is compiled into an executable binary file. This executable (in ELF format) is then converted into BRAM initialization commands using a tool called Data2MEM.

Note: Data2MEM can also generate memory files for the Verilog command **readmemh** to use to initialize external DDR memory.

When a simulation begins and reset is released, the PPC405 SWIFT model fetches the instructions from BRAM (which is mapped to the boot vector) and begins running the program. The user can then observe the bus cycles generated by the PPC405 CPU or any other signal in the design. For debugging purposes, the values of the PPC405 internal program counter, general-purpose registers, and special-purpose registers are available for display during simulation.

Generating a desired sequence of bus operations from the CPU might require a lot of software setup or simulation time. For early hardware bring-up or IP development, a bus functional model can be used to speed up simulation cycles and avoid having to write software. A model of the CPU is available in which two PLB master BFMs and one DCR BFM are instantiated to drive the CPU's PLB/DCR ports. These BFMs are provided in the CoreConnect toolkit and allow the user to generate bus operations by writing a script written in the Bus Functional Language (BFL). The reference design provides a sample BFL script that exercises many of the peripherals in the system. Refer to the *CoreConnect Toolkit* documentation for more information.

Since the PPC405 CPU SWIFT model and BFM model both have the same set of port interfaces, users can switch between the two simulation methods by compiling the appropriate set of files without having to modify the system's design source files. Users may need to modify their test benches to take into account the model being used.

Behavioral Models

The reference system includes some behavioral models to help exercise the devices and peripherals in the FPGA. Many of these models are freely available from various manufacturers and include interface protocol-checking features. The behavioral models and features included in the reference design are as follows.

- DDR memory models for testing the memory controllers
 - ◆ These models can also be preloaded with data for simulations
- Pull-ups connected to the GPIO for reading and driving outputs without getting unknown values
- Terminal interface connected to the UART for sending and receiving serial data
 - ◆ The terminal allows a user to interact with the simulation in real time
 - ◆ Characters sent out by the UART are displayed on a terminal, while characters typed into the terminal program are serialized and sent to the UART

- ◆ A simple file I/O mechanism passes data between the hardware simulator and the terminal program
- MGT transmit and receive pairs connected together in a loopback configuration

Synthesis and Implementation

The reference system can be synthesized and placed/routed into a Virtex-II Pro FPGA under the EDK tools. In particular, the ML300 board is targeted (although the design can be adapted to other boards). A basic set of timing constraints for the design is provided to allow the design to go through place and route.

Design Flow Environment

The EDK provides an environment to help manage the design flow including simulation, synthesis, implementation, and software compilation. EDK offers a Graphical User Interface (GUI) or command line interface to run these tools as part of the design flow. Consult the EDK documentation for more information.

Memory Map

This section diagrams the system memory map. It also documents the location of the DCR devices. The memory map reflects the default location of the system devices as defined in the `system.mhs` file.

Table 2: DCR Device Map

Device	Address Boundaries		Size
	Lower	Upper	
UART Lite	0x000	0x007	32B
Dual GPIO	0x008	0x00B	16B
Data Generator	0x010	0x017	32B
GMAC Peripheral	0x030	0x037	32B
TFT Controller	0x080	0x081	8B
Built-In ISOCM Controller	0x100	0x103	16B
CDMAC	0x140	0x17F	256B
Built-In DSOCM Controller	0x200	0x203	16B
INTC	0x3F0	0x3F7	32B

Table 3: Memory Maps

Device	Address Boundaries		Size	Comment
	Lower	Upper		
DDR SDRAM	0x00000000	0x07FFFFFF	128MB	
DDR SDRAM Shadow Memory	0x08000000	0x0FFFFFFF	128MB	Shadow memory allows TFT video memory to be accessed as an uncached region.
Data Side OCM Space	0xFE000000	0xFE003FFF	16KB	16KB address spaces wrap over 16MB region of 0xFE000000 to 0xFEFFFFFF.
Instruction Side OCM Space	0xFFFFC000	0xFFFFFFFF	16KB	16KB address spaces wrap over 16MB region of 0xFF000000 to 0xFFFFFFFF.

GSRD Hardware Peripheral Data Sheets

DCR to OPB Bridge

See XAPP535.

LocalLink TFT Controller

See XAPP535.

LocalLink Data Generator

See XAPP535.

LocalLink Gigabit Ethernet Media Access Controller (GMAC) Peripheral

Overview

The LocalLink GMAC Peripheral incorporates the Xilinx 1-Gigabit Ethernet MAC Core to provide a 1-Gigabit per second full duplex Ethernet Interface. The block diagram of the Peripheral is shown in Figure 3. Data is communicated via DMA operations over the LocalLink interfaces. Configuration and control of the peripheral is communicated via the DCR interface. The PHY interface is implemented as 1000BASE-X Physical Coding Sublayer (PCS) and Physical Medium Attachment (PMA), but the hardware can be tailored to accommodate external Ethernet PHYs using the Gigabit Media Independent Interface (GMII) Interface.

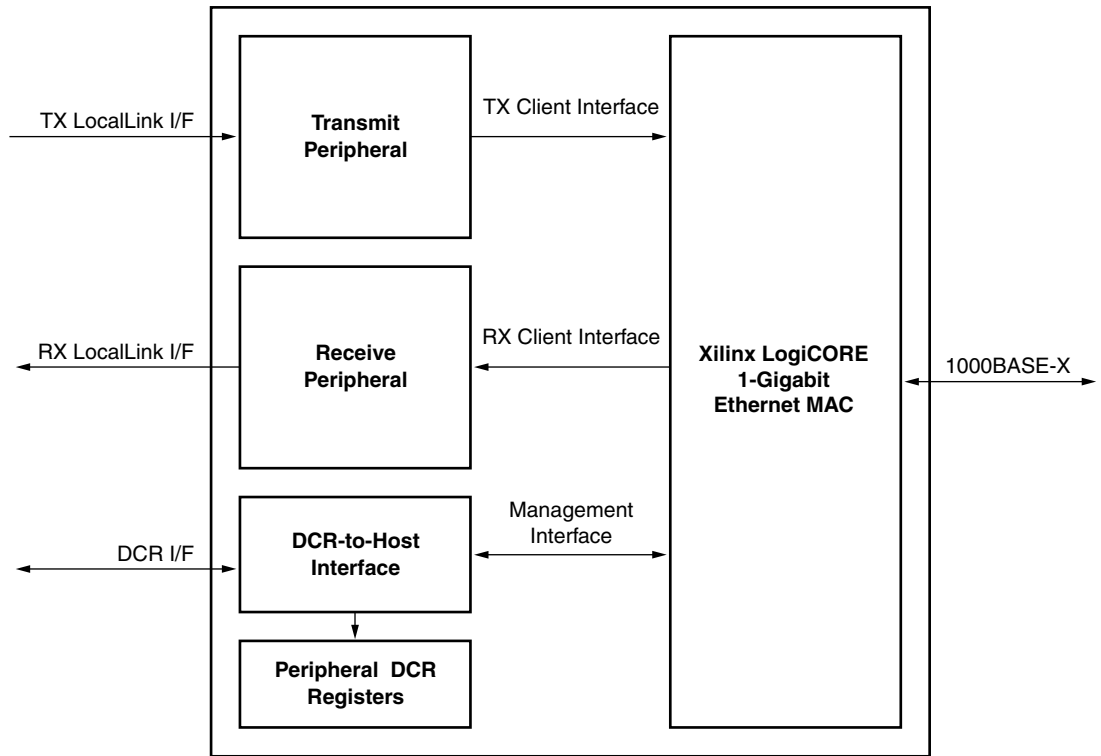


Figure 3: LocalLink Gigabit Ethernet MAC Peripheral Block Diagram

More information regarding the Xilinx LogiCORE 1-Gigabit Ethernet MAC can be found here:

<http://www.xilinx.com/systemio/gmac/index.htm>

Features

Summary of LogiCORE 1-Gigabit Ethernet MAC Features

- Single-speed, Full Duplex 1-Gigabit Ethernet MAC
- Designed to IEEE 802.3-2002 specification
- Full-Duplex Physical Coding Sublayer (PCS) with Physical Medium Attachment (PMA) for 1000BASE-X
- PCS supports Auto-Negotiation for information exchange with a link partner
- Support of VLAN frames to specification IEEE 802.3-2002
- Configurable flow control through MAC Control pause frames
- Configurable support of *jumbo frames* of any length
- Configurable inter-frame gap adjustment
- Available under terms of the SignOnce IP License

LocalLink Peripheral Specific Features

- 32-bit LocalLink transmit and receive interfaces to Communications DMA Controller
- Configuration/Status registers accessible over DCR bus
- Filtering of bad or truncated frames to reduce processor and memory utilization
- 16KB Transmit and Receive Buffers
- Transport Layer (UDP and TCP) checksum hardware assist
- All Valid Ethernet frames are passed to the software

Peripheral Design Facts

Table 4: Peripheral Design Facts

PCORE Specifics	
Supported Device(s)	Virtex-II Pro
Version	ll_gmac_periph_v1_00_a
Design File Format	Verilog
Resource Utilization	
LUTs	
FFs	
Block RAMs	18
MGTs	1
Design Tool Requirements	
Xilinx Implementation Tools	ISE 6.2i
EDK	Version 6.2
Simulation	ModelSim SE/EE 5.7b
Synthesis	XST

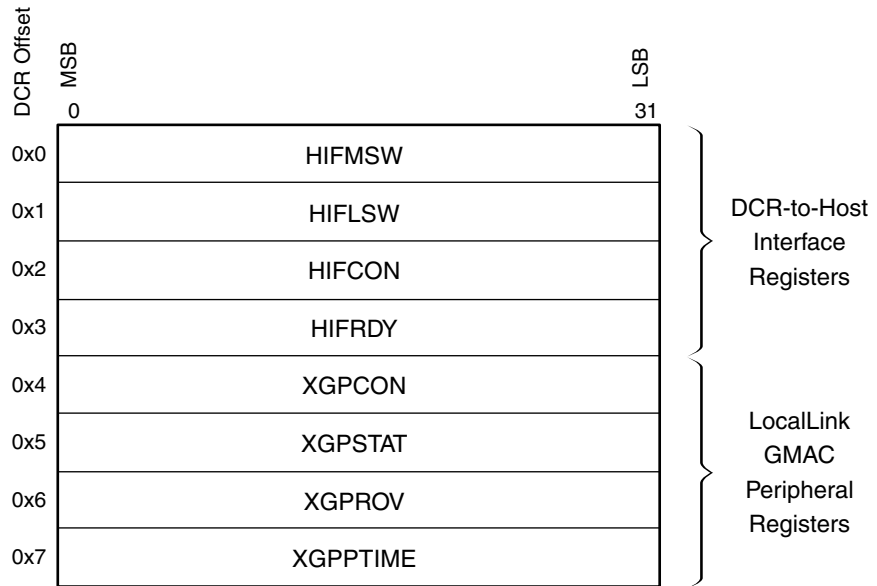
Programming Model

The Xilinx LogiCORE GMAC within the peripheral can be configured and controlled through a set of DCR registers. These registers act as a bridge between the GMAC Core internal registers and the DCR bus. This interface is described in the [“DCR-to-GMAC Host Interface”](#) section. Additional DCR registers provide control/status of the LocalLink Peripheral. They are described in the [“Peripheral Registers”](#) section

The Data Flow of the LocalLink GMAC Peripheral is controlled through buffer descriptors passed along from the DMA Controller. This is described further in the [“Transmit Data Flow - DMA Operation”](#) and [“Receive Data Flow - DMA Operation”](#) sections.

DCR Registers

[Figure 4](#) illustrates the DCR Register Model of the LocalLink GMAC Peripheral. The first four registers allow access to the internal LogiCORE GMAC core registers through a DCR to Host Interface block in the hardware. Registers four through seven allow control and monitoring of the Peripheral portion of the design.



X536_04_051204

Figure 4: LocalLink GMAC Peripheral Register Model

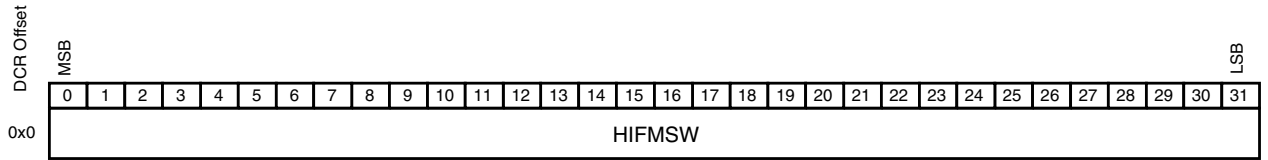
DCR-to-GMAC Host Interface

The LogiCORE GMAC core provides a processor independent hardware Management Interface. The interface is used to:

- configure the MAC core
- access optional statistics counters
- access PCS Sublayer registers via MII Data Input Output (MDIO)

The DCR-to-GMAC Host Interface block translates DCR reads and writes into transactions on the GMAC processor independent Management Interface. The DCR-to-GMAC Host Interface Registers are defined below.

HIFMSW Register



bits [0:31] **HIFMSW:** Host Interface Most Significant Data Word Register

access: read/write

default value: undefined

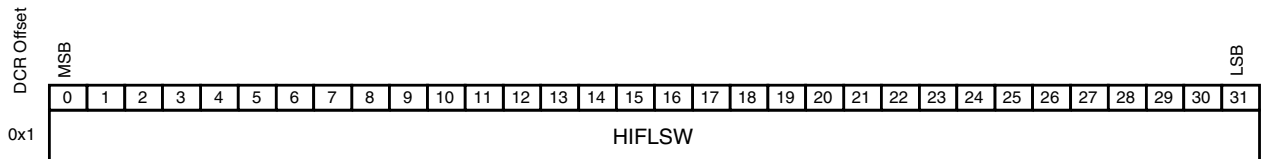
X536_05_051204

Figure 5: Host Interface Most Significant Word (HIFMSW) Register Definition

The HIFMSW register is used for GMAC Management Interface reads/writes when the value written is larger than a 4-byte quantity (32-bit). For example, Ethernet MAC address values are 48-bit values. If an Ethernet MAC address is to be written or read, the 16 most significant bits of the 48-bit MAC address would be placed in this register.

In the example above, the upper 16-bits of the MAC address are placed in the right-most portion of the HIFMSW register – [16:31].

HIFLSW Register



bits [0:31] **HIFLSW:** Host Interface Least Significant Data Word Register

access: read/write

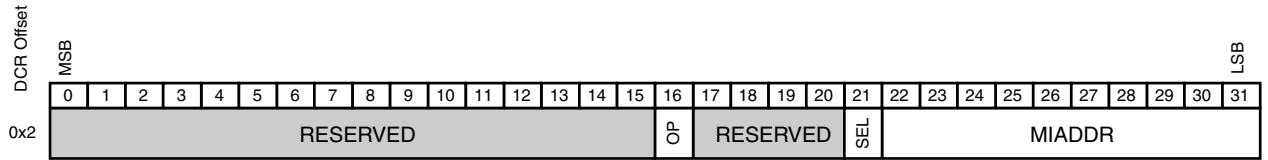
default value: undefined

X536_06_051204

Figure 6: Host Interface Least Significant Word (HIFLSW) Register Definition

The HIFLSW register is used for GMAC Management Interface reads/writes. Data is placed here by software prior to a Management Interface write. Data is placed here by the DCR-to-Host Interface during a Management Interface read.

HIFCON Register



- bits [0:15] **UNIMPLEMENTED:** Read as 0x0000

- bit 16 **OP:** Operation
 0 = Initiate a read transaction to the Management Interface
 1 = Initiate a write transaction to the Management Interface

 access: write

 default value: 0

- bits [17:20] **RESERVED:** Read as 0x0

- bit 21 **SEL:** GMAC Select - this bit should always be set to 0

 access: write

 default value: 0

- bits [22:31] **MIADDR:** Management Interface Address
 This address is used for the transaction being initiated

 access: write

 default: 0x000

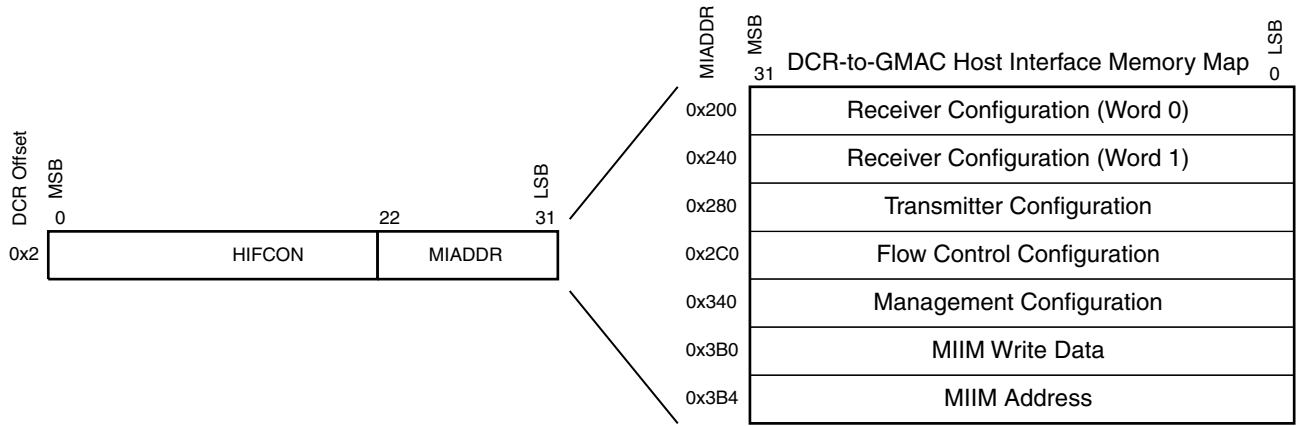
X536_07_051204

Figure 7: Host Interface Control (HIFCON) Register Definition

This register controls the operation of the DCR-to-Host Interface bridging function. Software writes this DCR register to initiate a read or a write. All values in this register should be written during the same DCR write because any write to this register initiates an operation on the DCR-to-Host Interface Bridge. See [Figure 9](#) for more details.

DCR-to-GMAC Host Interface Address Map

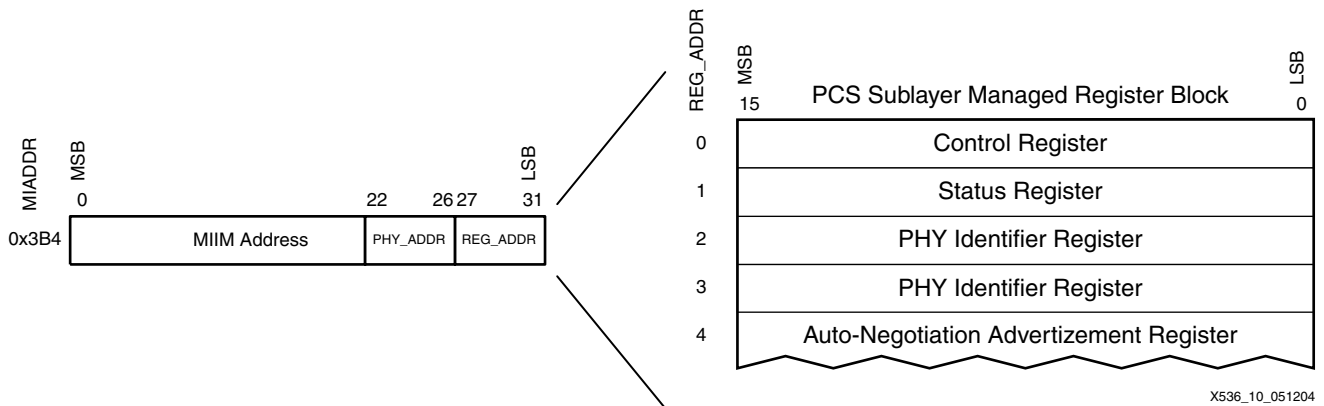
The MIADDR field of the HIFCON register provides a keyhole (or window) into a set of registers used for bridging between the DCR bus and the Management Interface of the LogiCORE GMAC core. This is depicted in Figure 9. The “MIIM Write Data” and “MIIM Address” registers are used for PCS Sublayer register block reads and writes. The remaining registers are directly mapped to the LogiCORE GMAC core Management registers – see the *1-Gigabit Ethernet MAC Core Data Sheet* for detailed bit maps of these registers.



X536_09_051204

Figure 9: DCR-to-GMAC Host Interface Memory Map

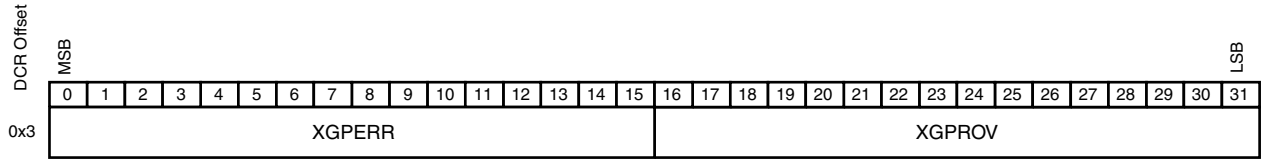
In order to write any of the PCS Sublayer registers defined in the *1-Gigabit Ethernet MAC Core Data Sheet*, the data must be written into the “MIIM Write Data” register shown in Figure 9. Then the PHY address and Register number are written to the “MIIM Address” register. The mapping of the “MIIM Address” is shown in Figure 11. The PHY Address (PHY_ADDR) is the 5-bit address of the PHY – the PHY_ADDR is set to 0b00001 for GSRD. The Register Address (REG_ADDR) is the PCS Sublayer register number to be accessed. In this way, the “MIIM Address” register is a window into the PCS Sublayer Register Block. This relationship is illustrated in Figure 9.



X536_10_051204

Figure 10: MIIM Address Register used to Access PCS Sublayer Register Block

XGPROV Register



bits [0:15] **XGPERR:** Peripheral RX Frame Error Count
 Incremented if incoming frame contains an error. Saturates at 0xFFFF.
 access: read/write
 default value: 0x0000

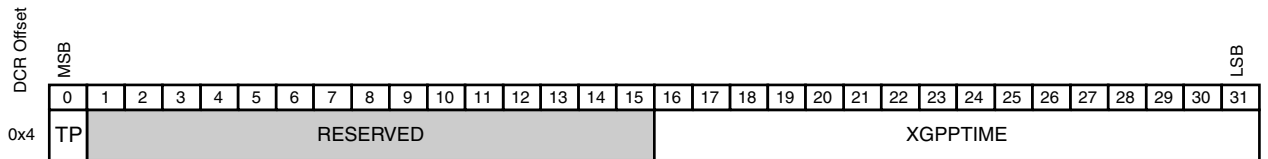
bits [8:31] **XGPROV:** Peripheral RX FIFO Overflow Count
 Incremented if incoming frame fills the RX FIFO. Saturates at 0xFFFF.
 access: read/write
 default value: 0x0000

X536_12_051204

Figure 12: Peripheral RX FIFO Overflow (XGPROV) Register Definition

The XGPROV register provides a count of all RX FIFO overflow events. This register does not count the number of frames received during a FIFO full state, only the frame that caused the full condition to occur. The XGPERR register provides a count of all incoming frames with errors (including truncated frames). All truncated and bad frames are silently discarded by the peripheral.

XGPPTIME Register



bit 0 **TP:** Transmit PAUSE Frame
 0 = No action taken
 1 = Initiates the transmission of a PAUSE Frame
 Note: This bit is self clearing
 access: read/write
 default value: 0

bits [1:15] **RESERVED:** Read as 0b0000000000000000

bits [8:31] **XGPPTIME:** PAUSE Time
 Sets the PAUSE Time field for subsequent PAUSE frames
 access: read/write
 default value: 0x0000

X536_13_051204

Figure 13: XGPPTIME Register Definition

The XGPPTIME register is used to set the 16-bit PAUSE value that should be transmitted with a flow control frame. This register can also be used to force a flow control frame to be transmitted. Transmit side Flow Control is not implemented at this time.

Initialization and Control

The LocalLink Peripheral does not require initialization.

The default implementation of GSRD uses the PCS/PMA version of the LogiCORE GMAC. The PCS/PMA Sublayer (or PHY) is implemented as part of the FPGA using the Xilinx MGT to directly drive optics transceivers onboard the ML300. Upon power-up the PCS Sublayer is electrically isolated from the data path of the GMAC. Software must clear the Isolate bit located in the PCS Sublayer Managed Register Block. Refer to the *1-Gigabit Ethernet MAC Core Data Sheet* for more details regarding this and other features of the LogiCORE. The required steps to clear this Isolate bit are summarized below:

Enable MDIO and set up MDIO clock divider via the Management Configuration Word.

For a system clock of 100MHz:

```
Management Configuration Word <= 0x00000034
```

Write the PCS Sublayer Control Register (Register 0) with the Isolate bit cleared:

```
PCS Sublayer Control Reg <= 0x00001140
```

See the *1-Gigabit Ethernet MAC Core Data Sheet* for detailed descriptions of these registers.

DCR-to-Host Interface Use Model

Software cannot directly access the internal registers of the LogiCORE GMAC core. These registers must be accessed through the DCR-to-GMAC Host Interface. The following sections describe the details of reading and writing the internal registers of the LogiCORE GMAC core.

For these discussions, we will assume the DCR Base Address for the Peripheral is 0x30 – the default implementation of GSRD.

Writing LogiCORE GMAC Management and PCS Sublayer Registers

Writing to these registers is a multi-step process. The steps to write to a GMAC Management Register are outlined below:

1. Write HIFLSW DCR register with desired data.
2. Write HIFCON DCR register with address in MIADDR field, bit 16 set, and 21 cleared.
3. Poll HIFRDY DCR register until the write is complete (optional).

Writing to the PCS Sublayer Registers requires an extra step because there is one more level of address indirection as described in the [“DCR-to-GMAC Host Interface Address Map”](#) section. The process is outlined below:

1. Write HIFLSW with desired data destined for the PCS Sublayer Register.
2. Write HIFCON with the address of the “MIIM Write Data” register (0x000083B0).
3. Write HIFLSW with a concatenation of the 5-bit PHY address and the PCS Sublayer Register offset.
4. Write HIFCON with the address of the “MIIM Address” register (0x000083B4).

See [Figure 11](#) for the bit map of the MIIM Address.

Let's revisit the example shown in the "Initialization and Control" section where we showed the steps to clear the Isolate bit inside the PCS Sublayer. Recall that MDIO must be enabled first, and the clock divider must be set via the Management Configuration Word. Next, the PCS Sublayer Control Register (Register 0) must be written, clearing the Isolate bit. The following code segment shows DCR moves for this example. Notice that bit 16 is set when writing the HIFCON register indicating a write operation.

Note: Assume DCR Base Address is 0x30.

```
// Management Configuration Word <= 0x00000034
mtdcr(0x30 + 1, 0x00000034);
mtdcr(0x30 + 2, 0x00008340);

// PCS Sublayer Control Reg <= 0x00001140
// PHY Address = 0b00001
// PCS Register Number = 0

mtdcr(0x30 + 1, 0x00001140);
mtdcr(0x30 + 2, 0x000083B0);

mtdcr(0x30 + 1, 0x00000020);
mtdcr(0x30 + 2, 0x000083B4);

while ( !(mfdcr(0x30 + 3) & 0x00000004) ) ;
```

Reading LogiCORE GMAC Management and PCS Sublayer Registers

Reading these registers is a multi-step process similar to writing. The steps are outlined below:

1. Write HIFCON DCR register with address in MIADDR field, and bits 16 and 21 cleared.
2. Read HIFLSW DCR register to initiate a read.

Reading from the PCS Sublayer Registers requires additional steps. This process is outlined below:

1. Write HIFLSW DCR register with MIIM Address (PHY_ADDR+REG_ADDR).
2. Write HIFCON DCR register with address of MIIM Address register (0x000003B4).
3. Poll the HIFRDY register until the read is complete.
4. Read HIFLSW DCR register to get requested data.

An example of a read from the PCS Sublayer register 0 is shown below:

Note: Assume DCR Base Address is 0x30 and that the MDIO has already been enabled.

```
// PHY Address = 0b00001
// PCS Register Number = 0

mtdcr(0x30 + 1, 0x00000020);
mtdcr(0x30 + 2, 0x000003B4);

while ( !(mfdcr(0x34) & 0x00000002) ) ;

pcsReg = mfdcr(0x30 + 1);
```

Transmit Data Flow - DMA Operation

The software interface to the Ethernet Transmit Data Path is through DMA Descriptors. The LocalLink GMAC Peripheral usage of the Transmit DMA Descriptor is shown in Figure 14. Words three, four, and five are used to communicate Checksum control information to the hardware. The USR0 and USR1 Words are available to the software driver to store state information. These fields are preserved after the DMA transaction is complete.

Note: Note: The TXCON, CSUMSTART, CSUMINSERT, and CSUMINIT fields of the first descriptor describing an Ethernet frame are sent to the peripheral. Subsequent descriptors for that frame will not be passed to the peripheral.

32-bit Word Count	Byte Address	MSB	LSB
0	0x00	NEXT DESCRIPTOR POINTER	
1	0x04	BUFFER ADDRESS	
2	0x08	BUFFER LENGTH	
3	0x0C	STATUS	RESERVED TXCON
4	0x10	CSUMSTART CSUMINSERT	
5	0x14	RESERVED	CSUMINIT
6	0x18	USR0	
7	0x1C	USR1	

X536_14_051204

Figure 14: Transmit Specific DMA Descriptor

TXCON Descriptor Field

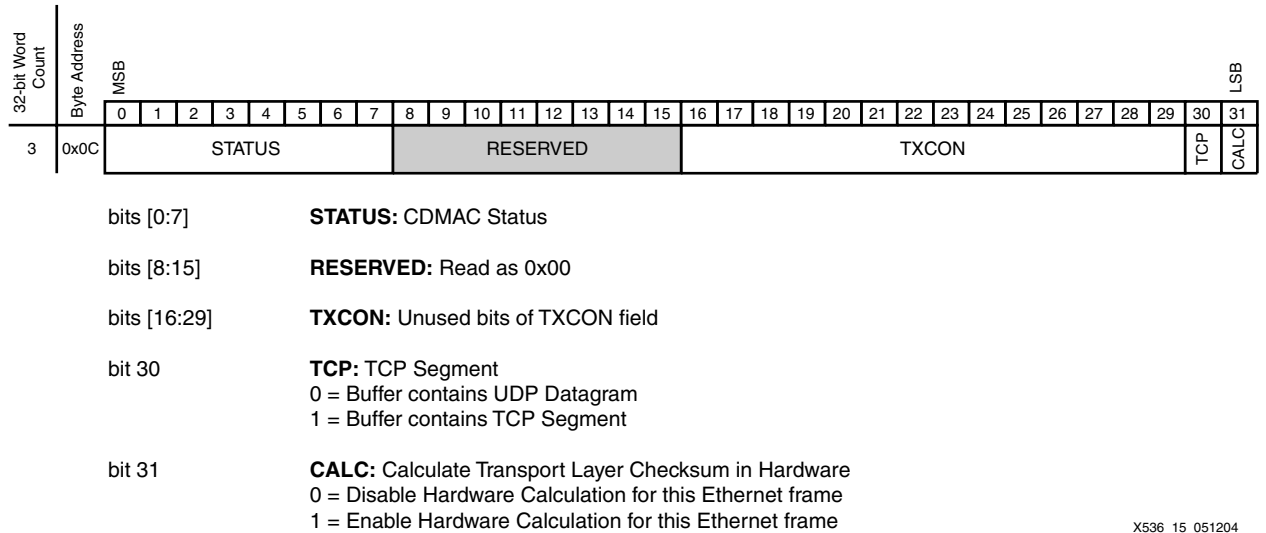


Figure 15: TX DMA Descriptor - STATUS and TXCON Field Definition

Note: This 32-bit Word in the DMA Descriptor is shared between the DMA Controller and the GMAC Peripheral.

CSUMSTART and CSUMINSERT Descriptor Field

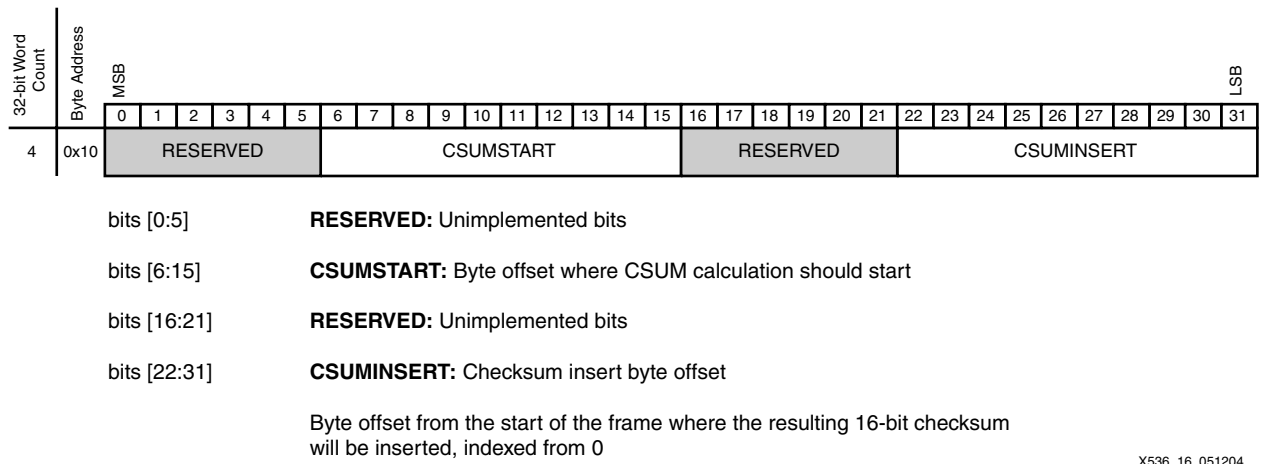
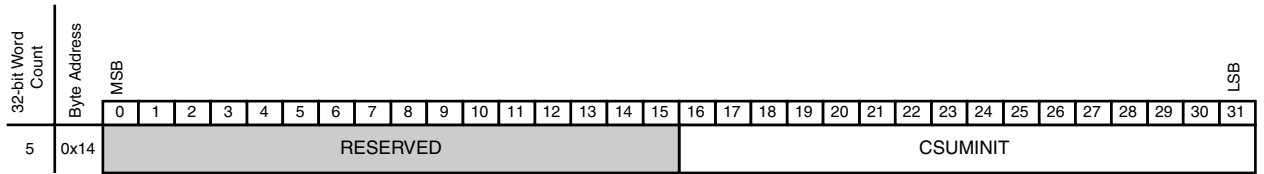


Figure 16: TX DMA Descriptor - CSUMSTART and CSUMINSERT Field Definition

CSUMINIT Descriptor Field



bits [0:5] **RESERVED:** Unimplemented bits

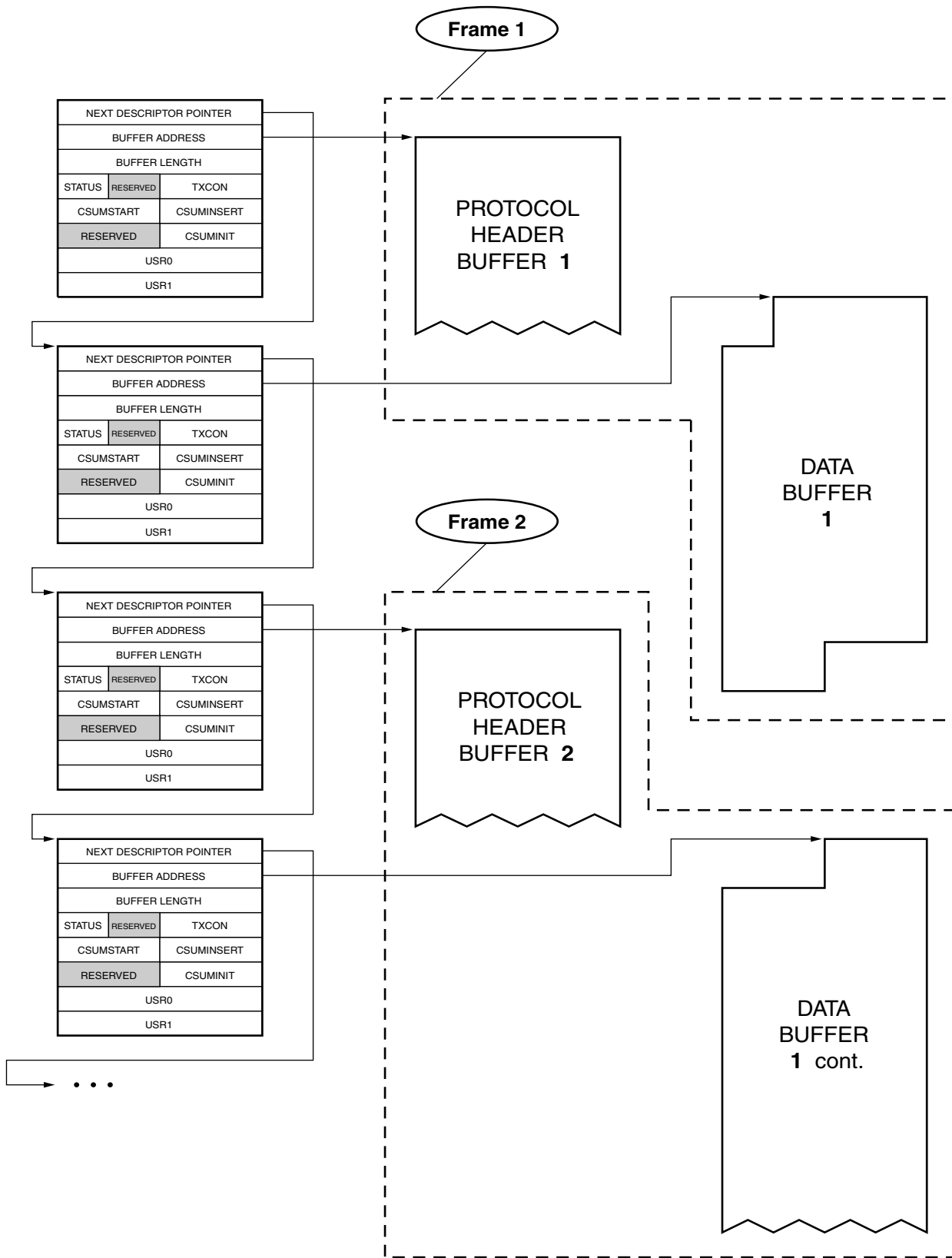
bits [16:31] **CSUMINIT:** Initial value to be loaded into checksum accumulate register

This value is used to initialize the checksum hardware engine with a seed value. This can be used to write the result of the software-calculated pseudo header 16-bit ones complement addition. If not used, the field must be set to zero.

X536_17_051204

Figure 17: TX DMA Descriptor - CSUMINIT Field Definition

The Descriptor shown in [Figure 14](#) can be chained together to gather a single Ethernet frame that has been built spanning multiple memory buffers. Let's examine a case where we have a large contiguous data buffer. The application sends this buffer to the TCP stack for transmission. The protocol stack will span this large buffer across multiple Ethernet frames as needed to adhere to the TCP protocol. The protocol stack builds as many Ethernet, IP, and TCP headers (protocol headers) as needed to send the contents of the data buffer. To illustrate this scenario graphically, consider a data buffer that must be transmitted using 2 Ethernet frames (2 TCP segments). [Figure 18](#) shows the DMA Descriptor chain that the software driver must set up and pass along to the DMA Controller.



X536_18_051204

Figure 18: Example Transmit Descriptor Chain (2 frames shown)

Receive Data Flow - DMA Operation

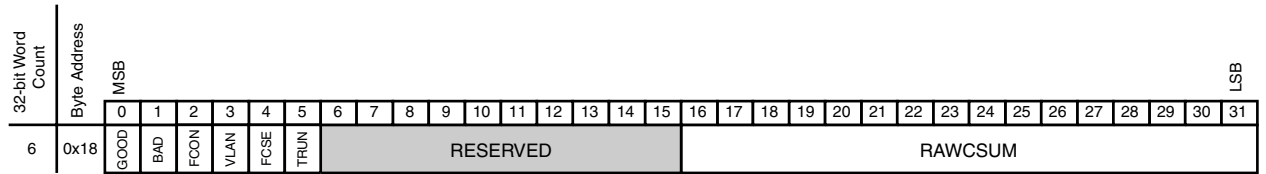
The software interface to the Receive Peripheral data path is through DMA Descriptors. The Receive DMA Descriptor layout is shown in Figure 19. The fields carry status communicated from the GMAC Peripheral to the software driver. The last descriptor describing an Ethernet frame carries valid RXSTAT and FRAMELEN fields. All other descriptors for that frame do not have valid RXSTAT and FRAMELEN fields.

32-bit Word Count	Byte Address	MSB		LSB		
		0			31	
0	0x00	NEXT DESCRIPTOR POINTER				
1	0x04	BUFFER ADDRESS				
2	0x08	BUFFER LENGTH				
3	0x0C	STATUS	RESERVED			
4	0x10	RESERVED				
5	0x14	RESERVED				
6	0x18	RXSTAT				
7	0x1C	FRAMELEN				

X536_19_051204

Figure 19: Receive Specific DMA Descriptor

RXSTAT Descriptor Field



- bit 0 **GOOD:** Good Frame
1 = Good Frame
- bit 1 **BAD:** Bad Frame
1 = Bad frame
- bit 2 **FCOON:** Flow Control Frame
0 = not a flow control frame
1 = flow control frame
- bit 3 **VLAN:** VLAN Frame
0 = frame does not contain a VLAN Tag
1 = frame contains a VLAN Tag
- bit 4 **FCSE:** Frame Check Sequence (CRC) Error
0 = frame passed FCS validation
1 = frame failed FCS validation
- bit 5 **TRUN:** Frame Truncated
0 = frame not truncated
1 = frame truncated due to FIFO overflow
- bits [6:15] **RESERVED:** unimplemented in hardware
- bits [16:31] **RAWCSUM:** Raw 16-bit Ones Complement Checksum

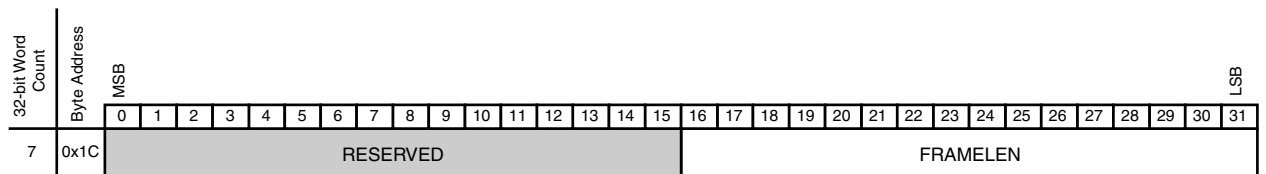
This field carries the 16-bit ones complement checksum calculated by the peripheral over the entire Ethernet frame.

X536_20_051204

Figure 20: RX DMA Descriptor - RXSTAT Field Definition

Note: This field is valid in the last descriptor of a chain that describes an Ethernet frame.

FRAMELEN Descriptor Field



- bits [0:15] **RESERVED:** unimplemented in hardware (filled with zero)
- bits [16:31] **FRAMELEN:** Length of Frame in bytes

X536_21_051204

Figure 21: RX DMA Descriptor - FRAMELEN Field Definition

Note: This field is valid in the last descriptor of a chain that describes an Ethernet frame.

The Receive Descriptor shown in [Figure 19](#) can be chained together to scatter a single Ethernet Frame over multiple memory buffers, and many frames can be chained together. Note, only one, or a fraction of one Ethernet frame can be associated with a single descriptor. [Figure 22](#) shows a typical chain of Receive Descriptors. The chain illustrated contains 4 Ethernet Frames after the DMA Controller has moved them into the data buffers.

Note: Both the DMAC_START_OF_PACKET and DMAC_END_OF_PACKET bits are set in each descriptor. This indicates that the frame starts and ends in each descriptor – 1 frame per buffer/descriptor.

