XAPP807 (v1.3) January 17, 2007

Minimal Footprint Tri-Mode Ethernet MAC Processing Engine

Author: Jue Sun, Harn Hua Ng, and Peter Ryser

Summary

The Tri-Mode Ethernet MAC (TEMAC) UltraController-II module is a minimal footprint, embedded network processing engine based on the PowerPC™ 405 (PPC405) processor core and the TEMAC core embedded within a Virtex™-4 FX Platform FPGA. The TEMAC UltraController-II module connects to an external PHY through Gigabit Media Independent Interface (GMII) and Management Data Input/Output (MDIO) interfaces and supports tri-mode (10/100/1000 Mb/s) Ethernet. Software running from the processor cache reads and writes through an On-Chip Memory (OCM) interface to two FIFOs that act as buffers between the different clock domains of the PPC405 OCM and the TEMAC.

The TEMAC UltraController-II module uses minimal resources: one PPC405, one TEMAC, two Virtex-4 FIFOs, 20 slice flip-flops, and 18 look-up tables (LUTs). Because of the minimal footprint design, a greater number of FPGA logic resources remain available to the user.

The xapp807.zip file includes software with a web server demonstration to initialize the TEMAC and to receive and send frames. A web server application running on top of the uIP TCP/IP stack demonstrates the software drivers. The accompanying reference design includes VHDL, Verilog, and C source code. The software and hardware for this reference design was developed and tested using a Xilinx ML403 embedded system development platform. [Ref 1]

Introduction

The TEMAC UltraController-II module provides a simple way to receive and send Ethernet frames. It utilizes a modified version of the UltraController-II design and a TEMAC controller module, as shown in Figure 1.

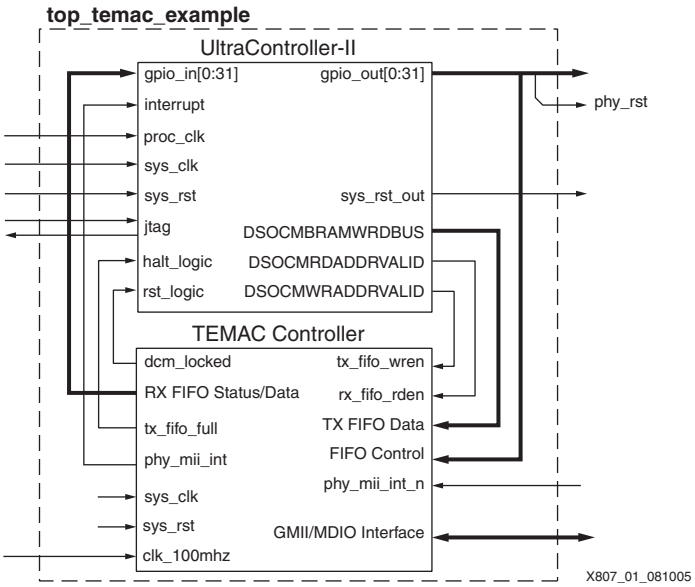


Figure 1: **TEMAC UltraController-II Module High-Level Block Diagram**

Required Hardware/Tools

- Xilinx ML403 development board
- Xilinx ISE
- Xilinx Platform Studio

TEMAC UltraController-II Features

- Supports 10/100/1000 Mb/s network link rates
- Performs auto-negotiation of a network link
- Supports full-duplex operations
- Supplies a Device Control Register (DCR) interface to the TEMAC for configuration
- Uses a simple software driver
- Utilizes minimal logic resources

The design consists of an embedded TEMAC, embedded processor, and minimal glue logic. The glue logic includes the following:

- A receive (RX) Virtex-4 FIFO to buffer inbound frames from the TEMAC to the PPC405
- A transmit (TX) Virtex-4 FIFO to buffer outbound frames from the PPC405 to the TEMAC
- Control logic to handle read and write operations on the FIFOs

The reference design C code includes functions to:

- Initialize the TEMAC and perform auto-negotiation of the network link
- Receive frames from the RX FIFO
- Send frames to the TX FIFO

All software runs from the 16 KB instruction-side and 16 KB data-side cache memory of the PPC405.

Data Flow

Figure 2 is a high-level block diagram that shows the flow of data through the TEMAC UltraController-II module. The numbers in the diagram correspond to the numbered descriptions that follow.

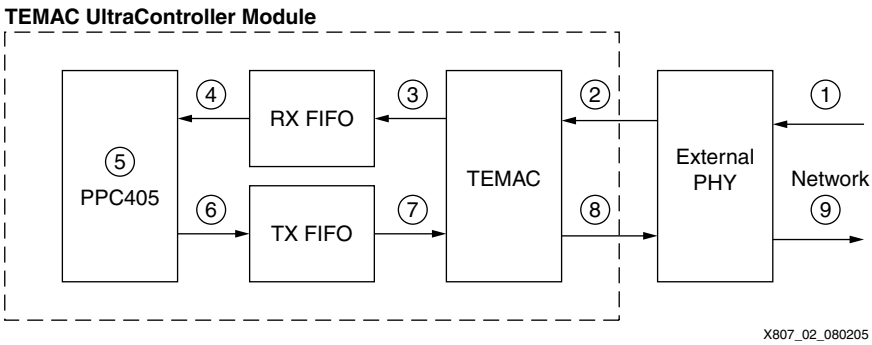


Figure 2: **Data Flow Block Diagram**

1. The external PHY receives a frame from the network.
2. The PHY processes the frame and sends it to the TEMAC through the GMII interface.
3. The TEMAC processes the received frame and passes it to the RX FIFO.
4. The RX FIFO buffers the received frame, synchronizes the different clock domains of the PPC405 and TEMAC, and allows the PPC405 to read the frame.

5. The PPC405 reads the inbound frame, processes it, and writes an outbound frame to the TX FIFO.
6. The TX FIFO buffers outbound frames and synchronizes the different clock domains of the PPC405 and TEMAC.
7. When the logic in the fabric detects that a frame has been written into the TX FIFO, it reads the TX FIFO and sends the frame to the TEMAC.
8. The TEMAC processes the data and sends the frame to the external PHY through the GMII interface.
9. The PHY sends the frame to the network.

Hardware Overview

Figure 3 is a simplified block diagram showing the logic within the TEMAC UltraController-II black-box design. It does not show the buffers, clocks, and registers that are used to buffer the inputs and outputs of the TEMAC block.

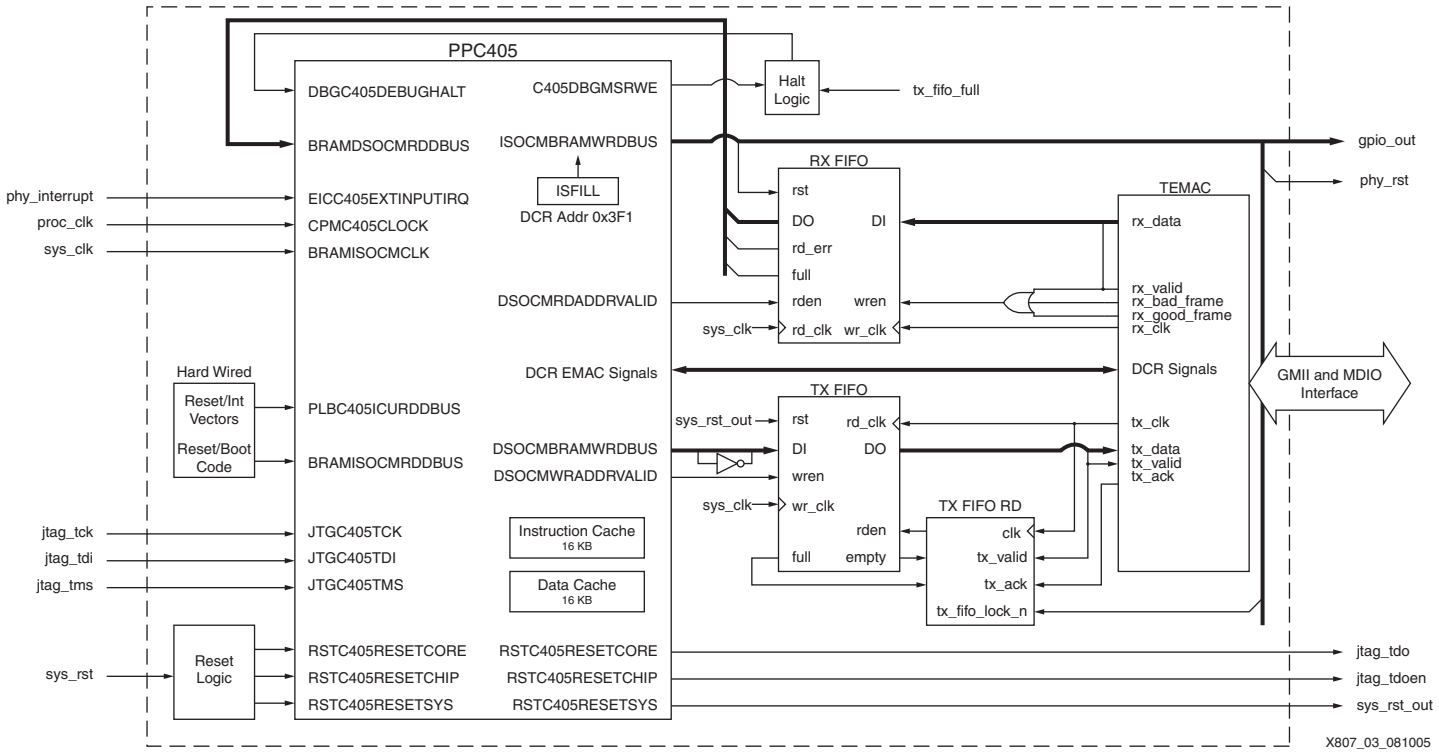


Figure 3: **TEMAC UltraController-II Black Box Block Diagram**

PPC405

The PPC405 is instantiated within the UltraController-II module. It uses data-side OCM (DSOCM) and instruction-side OCM (ISOCM) signals to transfer data between the FIFOs. The PPC405 block's DCR EMAC signals directly connect to the TEMAC block to enable the DCR interface between the TEMAC and PPC405. The PHY interrupt is sourced from the external interrupt port to trigger an interrupt whenever a change in the link is detected by the PHY.

TEMAC

The TEMAC block is initialized in temac_controller.v(.vhd). It consists of a GMII and MDIO interface to the PHY, a DCR interface to the PPC405, and other signals that interface to the TX FIFO and RX FIFO. The temac_controller.v(.vhd) file also instantiates a DCM to generate a 125-MHz gtx_clk signal for the TEMAC. For more on TEMAC clocking, refer to UG074. [Ref 2]

Signal Descriptions

Table 1 lists the TEMAC UltraController-II signals.

Table 1: **TEMAC UltraController-II Signals**

Signal	Definition
rx_bad_frame	An output of the TEMAC that is asserted for one clock cycle after a frame has been transmitted on rx_data indicating the transmitted frame is corrupt.
rx_clk	The TEMAC receive clock.
rx_data[7:0]	The data bus from the TEMAC to the RX FIFO.
rx_fifo_full	A signal from the RX FIFO to the PPC405 (through BRAMDSOCMRDDBUS) that indicates if RX FIFO is overflowing and needs to reset.
rx_fifo_rden	The RX FIFO read enable signal that is connected to DSOCMRDADDRVALID of the PPC405 block.
rx_fifo_rderr	A signal from the RX FIFO to the PPC405 (through BRAMDSOCMRDDBUS) that indicates whether the PPC405 was reading from an empty RX FIFO.
rx_fifo_rst	A signal from the PPC405 (through ISOCMBRAMWRDBUS) that resets the RX FIFO when it is overflowing.
rx_fifo_wren	The RX FIFO write enable signal that is asserted when rx_valid, rx_good_frame, or rx_bad_frame signals are asserted, enabling the valid rx_data[7:0] to be written.
rx_good_frame	An output of the TEMAC that is asserted for one clock cycle after a frame has been transmitted on rx_data indicating that the transmitted frame is good.
rx_valid	An output of the TEMAC that indicates there is valid data on rx_data[7:0].
tx_ack	A signal from TEMAC that is asserted for one clock cycle after tx_valid is asserted and the first byte of the frame is read by TEMAC. It acknowledges to the TX FIFO that TEMAC is ready to read the rest of the frames.
tx_clk	The TEMAC transmit clock.
tx_data[7:0]	The data bus from the TX FIFO to the TEMAC.
tx_fifo_empty	An output of the TX FIFO that is used by the TX FIFO RD logic to prevent it from reading TX FIFO when its empty.
tx_fifo_full	An output of the TX FIFO that halts the PPC405 when the TX FIFO is full to prevent overflow.
tx_fifo_lock_n	A signal from the PPC405 (through ISOCMBRAMWRDBUS) that indicates to the TX FIFO that it should not start reading a new frame because the PPC405 is current writing a frame to TX FIFO. It deasserts when PPC405 is writing to the TX FIFO.
tx_fifo_rden	An input to TX FIFO from the TX FIFO RD logic block that enables the TEMAC to read a frame from the TX FIFO to the TEMAC.
tx_fifo_wren	An output of the TX FIFO that is connected to the DSCOMWRADDVALID port of the PPC405 block.
tx_valid	A signal to the TEMAC indicating that rx_data[7:0] is a valid byte in a frame.

Web Server Application

This application note uses a web server demonstration based on a uIP TCP/IP stack. An implementation of the TCP/IP protocol stack, uIP is intended for small 8-bit and 16-bit microcontrollers. It provides the necessary protocols for Internet communication, with a very small code footprint and RAM requirements. The uIP code size is on the order of a few kilobytes, and RAM usage is on the order of a few hundred bytes. [Ref 3]

Software Functions and Routines

This section explains how the software interfaces with the hardware block through a few functions and routines. Some hardware details are also included to illustrate why software routines are written this way. All software routines covered in this section are located in the xapp807.zip file at sw/standalone/src/ethernet/emac.c. A file containing more generic routines for reference is located at sw/standalone/src/ethernet/original_emac.c.

Initialization

The `tapdev_init()` initialization function must be called in the beginning of a C program before calling any other functions or routines. This function first initializes exception handlers in order for interrupts to work. It then sets TEMAC registers accordingly through the DCR interface, enabling auto-negotiation and interrupts, and waits for an interrupt from the PHY to indicate an active network link.

Interrupt

The `Ext_Exception_Handler()` and `configure_emac()` interrupt functions must be included in the software. These two functions are called when an interrupt from the PHY is asserted. A PHY interrupt indicates that a connection has been established, changed, or shut down. These functions set the speed for the TEMAC accordingly.

Note: The EMAC registers are accessed through the DCR buses from the PPC405. Although the DCR buses are not connected in VHDL or Verilog files, they are connected through hard connections within the FPGA.

Receiving a Frame

In hardware, a 9-bit x 2k RX FIFO is used to buffer the inbound frame. Figure 4 shows the timing between the TEMAC and the RX FIFO.

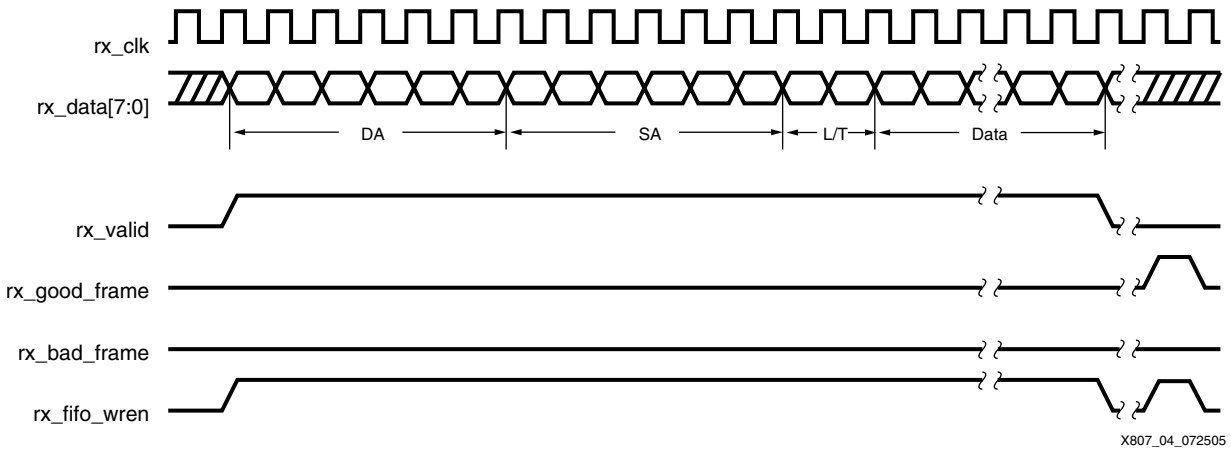


Figure 4: **RX Timing**

Table 2 shows how the rx_valid signal and rx_data[7:0] bus from the TEMAC are written into the RX FIFO.

Table 2: **RX FIFO Storage Map**

DIP 0	DI 7	DI 6	DI 5	DI 4	DI 3	DI 2	DI 1	DI 0
rx_valid	rx_data[7]	rx_data[6]	rx_data[5]	rx_data[4]	rx_data[3]	rx_data[2]	rx_data[1]	rx_data[0]

To receive a frame:

1. The rx_valid signal is asserted when there is valid data on the rx_data[7:0] bus from the TEMAC.

2. The rx_good_frame and rx_bad_frame signals are asserted for one clock cycle after a frame has been transmitted on rx_data[7:0].
3. The rx_fifo_wren signal is asserted when rx_valid, rx_good_frame, or rx_bad_frame signals are asserted. This writes all the valid rx_data[7:0] and, at the end, a value of all zeros into the RX FIFO. A falling edge on rx_valid in software, therefore, indicates the end of a frame. The design does not differentiate a good frame from a bad frame in the fabric, thus it relies on the software to check whether the frame is corrupt.

The software can read from the RX FIFO at any time during its execution using the following instruction:

```
input = lwz(FIFOPORT);
```

The **`lwz()`** function in xpseudo_asm.h is used to read from the OCM. FIFOPORT is the address map for the OCM port. This function allows the user to read the current data in the RX FIFO and all the other signals on the BRAMDSOCMRDBUS. DSOCMRDADDRVALID on the PPC405 block is asserted automatically when the **`input = lwz(FIFOPORT)`** instruction is executed.

The following routine reads the frame in the RX FIFO into an array specified by **`uip_buf[]`** and returns the length of the frame.

```
int tapdev_read(){
Xuint32 input;
u16_t x, time_out=0;
do {
  if(time_out>65530){return 0;}//The webserver application requires the routine to timeout and return 0
when there is no frame in RX FIFO for a long time
  x=0;                                                  //Initialize frame byte counter
  time_out++;                                          //Increment the time out counter whenever there
are no frames to read in RX FIFO
  do {                                                  //This is the loop to read a frame
    input = lwz(FIFOPORT);                              //read from RX FIFO
    uip_buf[x] = (u8_t)(input & RX_DATA_FILTER);        //get the data portion of input
    if (!(input & RX_EMPTY_FILTER)) {x++;}             //increment the byte counter if PPC were not
reading from an empty RX FIFO
  } while (((input & RX_VALID_FILTER) || (input & RX_EMPTY_FILTER&& x!=0)) && (!(input &
RX_FULL_FILTER)));
                                                        //stop looping if no frame is in RX FIFO,
                                                        //or if RX FIFO is full, or if RX VALID
                                                      //is not high anymore indicating end of a frame
  if ((input & RX_FULL_FILTER))                         //If RX FIFO was full
    {reset_rx_fifo();}                                  //Reset the RX FIFO
  } while ((input & RX_FULL_FILTER) || (x==0));    //Loop again if RX FIFO was full or if RX FIFO
did not have a frame to be read
return (x-1);                                           //Return the length of the frame
}
```

If the network traffic is received at a faster rate than the rate at which frames are retrieved from the RX FIFO, the RX FIFO overflows. Upon overflow, the software resets the RX FIFO and frames are lost. If the RX FIFO is reset in the middle of writing a frame, then the next frame that the software retrieves will not be complete. This design relies on the uIP TCP/IP stack to resend lost frames. A checksum discards incomplete frames. Table 5, page 13 shows how resetting the RX FIFO is affected by different utilization levels of the bandwidth.

Sending a Frame

In hardware, a 9-bit x 2k TX FIFO is used to buffer the outbound frame. Figure 5, page 7 shows the timing between the TEMAC and the TX FIFO.

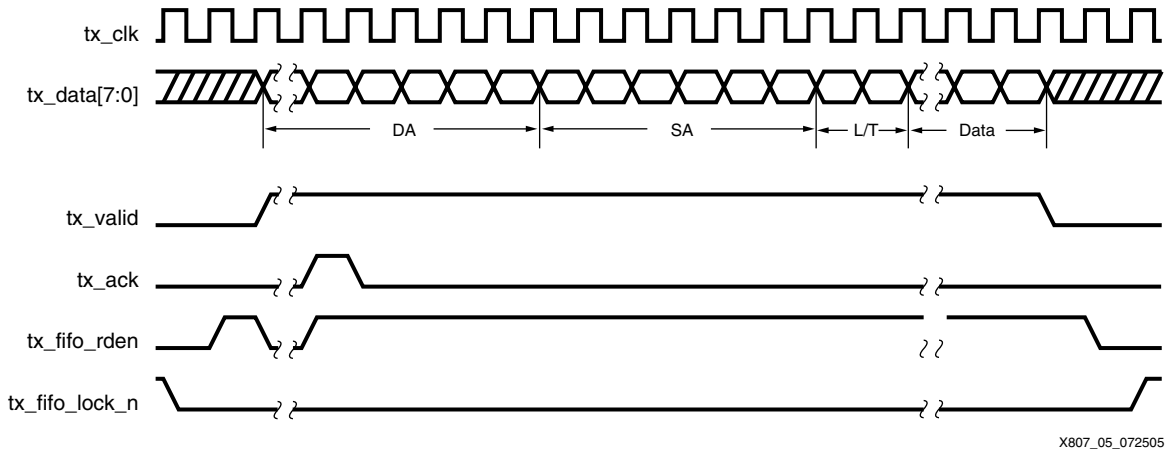


Figure 5: **TX Timing**

Table 3 shows the TX FIFO storage map. The DI of TX FIFO is connected to the DSOCMBRAMWRDBUS port of the PPC405.

Table 3: **TX FIFO Storage Map**

DIP 0	DI 7	DI 6	DI 5	DI 4	DI 3	DI 2	DI 1	DI 0
tx_valid	tx_data[7]	tx_data[6]	tx_data[5]	tx_data[4]	tx_data[3]	tx_data[2]	tx_data[1]	tx_data[0]

To send a frame:

1. The tx_valid signal is asserted by the TX FIFO while sending a frame. It must be asserted with the first byte of the frame until tx_ack is asserted by the TEMAC.
2. When tx_ack is asserted by the TEMAC, TX FIFO must send the second byte of the frame to the TEMAC.
3. After tx_ack is asserted, the TX FIFO sends one byte on each tx_clk cycle to the TEMAC frame, asserting the tx_valid until the last byte of the frame is sent.

This application note does not detail the logic between the TX FIFO and the TEMAC, however, Figure 6 shows how the tx_fifo_rden signal is generated.

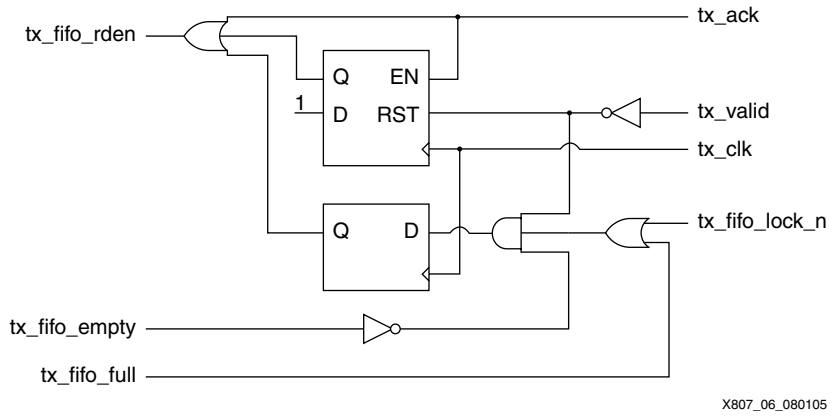


Figure 6: **TX FIFO Read Logic**

If the network traffic is transmitted at a faster rate than the rate at which frames are sent to the TX FIFO, the TX FIFO underflows. To prevent underflow, the user must call the `lock_tx_fifo()` function. With the TX FIFO locked, the TEMAC will not begin to read the

frame that the PPC405 is currently writing, thus preventing the TX FIFO from emptying in the middle of sending a frame to the TEMAC.

In software, the following instruction is used to write a byte into the TX FIFO:

```
stw(FIFOPORT, ((Xuint32) (frame[x]) ) );
```

The **stw()** function is used to write to the OCM bus in xpseudo_asm.h. This function allows the PPC405 to write the value of **frame[x]** to the TX FIFO while tx_valid is asserted (Logic 1). To send the frame, the user must write the first byte of the frame twice. To end the frame and cause tx_valid to be deasserted (Logic 0), the user must write 0x00000100 twice. The user then calls **unlock_tx_fifo()** to allow the TEMAC to start reading the outbound frame.

The function used to send data in the web server demonstration is shown as below. It sends data from two buffers, **uip_buf[]** and **uip_appdata[]**, with a length of **uip_len**.

```
void tapdev_send(){
u16_t x, first_array, second_array;
lock_tx_fifo();                              // lock fifo so that TEMAC will not read from it while
it's writing a frame
stw(FIFOPORT, ((Xuint32) (uip_buf[0])));     // store the first byte
first_array = UIP_LLH_LEN + 40;              // this is the length of the first array, uip_buf[]
second_array = uip_len - UIP_LLH_LEN-40;     // this is the length of data in the second array,
uip_appdata[]
if(uip_len > first_array) {                  // if the frame to be sent exists in both buffers
  for(x = 0; x < first_array; ++x) {         // send the data in the first array, uip_buf[] first
    stw(FIFOPORT, ((Xuint32) (uip_buf[x])));
  }
  for(x=0; x < second_array; ++x) {         // then send the data in the second array, uip_appdata[]
    stw(FIFOPORT, ((Xuint32) (uip_appdata[x])));
  }
} else {                                   // if the frame is short enough to be in just the uip_buf[]
    for(x = 0; x < uip_len; ++x) {          // send the frame from just uip_buf[]
    stw(FIFOPORT, ((Xuint32) (uip_buf[x]) ));
  }
}
stw(FIFOPORT, 0x00000100);                  // two dummy writes into TX FIFO with tx_valid being 0
stw(FIFOPORT, 0x00000100);
unlock_tx_fifo();                          // unlock FIFO to enable TEMAC to start reading this frame
}
```

Clocks

Table 4 lists the clocks used in the TEMAC UltraController-II module. See UG018 and UG074 listed in "References," page 15 for additional details on clocking requirements.

Table 4: **TEMAC UltraController-II Clocking**

Clock Name	Default Frequency	Source	Used By	Permitted Range
sys_clk_in	100 MHz	Generated onboard	Various DCMs	N/A
sys_clk	150 MHz	DCM	TEMAC logic and OCM interface	100-200 MHz
proc_clk	300 MHz	DCM	Processor block	100-350 MHz
gtx_clk	125 MHz	DCM	TEMAC	Fixed
rx_clk	1.25 MHz @ 10 Mb/s 12.5 MHz @ 100 Mb/s 125 MHz @ 1000 Mb/s	TEMAC	TEMAC logic	N/A

Table 4: **TEMAC UltraController-II Clocking *(Continued)***

Clock Name	Default Frequency	Source	Used By	Permitted Range
tx_clk	1.25 MHz @ 10 Mb/s 12.5 MHz @ 100 Mb/s 125 MHz @ 1000 Mb/s	TEMAC	TEMAC logic	N/A

Notes:

1. sys_clk and proc_clk must be integer multiples.
2. The integer multiple of the two clocks must be specified in the uc2.v(.vhd) module using the PPC405 ports DSCNTLVALUE and ISCNTLVALUE. For details, refer to UG018.
3. DCMs are used to generate gtx_clk, sys_clk, and proc_clk.
4. DCMs for sys_clk and proc_clk are instantiated in top_temac_example.v; the DCMs for gtx_clk is instantiated in temac_controller.v.

Quick Start

This Quick Start section provides step-by-step instructions for bringing up the TEMAC UltraController-II module and for demonstrating the uIP web server application. Pregenerated files are provided in the **demos** folder if the user wishes to bypass the steps for generating binary files.

Ensure that ISE and EDK are properly installed before proceeding. Extract the xapp807.zip file to a local directory. Check the `readme.txt` for required versions of the Xilinx ISE and EDK tool sets.

Generating a Hardware Bitstream

1. In ISE, launch Project Navigator and browse to the **projnav** folder.
2. Select **File** → **Open Project** → **top_temac_example_verilog.ise**.
3. Generate a bitstream for top_temac_example.v(.vhd):
   a. In the Sources in Project window, right-click **top_temac_example** to select the module.
   b. In the Processes for Source window, select **Generate Programming File** → <right-click> → **Rerun All**.
   c. The generated NCD file appears in the **projnav** folder.

Generating a Software ELF in Xilinx Platform Studio

The xapp807.zip includes files for generating an ELF. The drivers adopted for the uIP application are located in sw/standalone/src/ethernet/emac.c. A general software driver is located in sw/standalone/src/ethernet/emac_original.c.

1. Launch Xilinx Platform Studio (XPS) and select **File** → **Open Project** → **temac_ultracontroller_module.xmp**.
2. Select **File** → **Open** → **sw/standalone/src/uip-0.9/unix/uipopt.h** and specify the desired IP (line 132) and MAC address (line 182) for the ML403 board.

   Note: The first three bytes of the IP address should match the IP address of the network to which the board is to be connected. The default IP is 192.168.1.4.
3. In the Applications window, right-click on the project **webserver** and select **Build project** to generate an ELF executable.

Both the hardware bitstream and software ELF file are now ready for generating an RBT or MCS file.

Generating RBT and MCS Files

Use the top_temac_example.ncd file and executable ELF file and generate an RBT file or an MCS file. Refer to XAPP719 for more information on the generation of these files and cache configuration using the USR_ACCESS_VIRTEX4 register.

1. Use a command window and browse to the **generate_rbt_mcs** folder.
2. Using a command line, type:

   ```
   xilperl genmcs.pl ../projnav/top_temac_example.ncd
   ../ppc405_0/code/executable.elf demo
   ```

   This command generates two files:

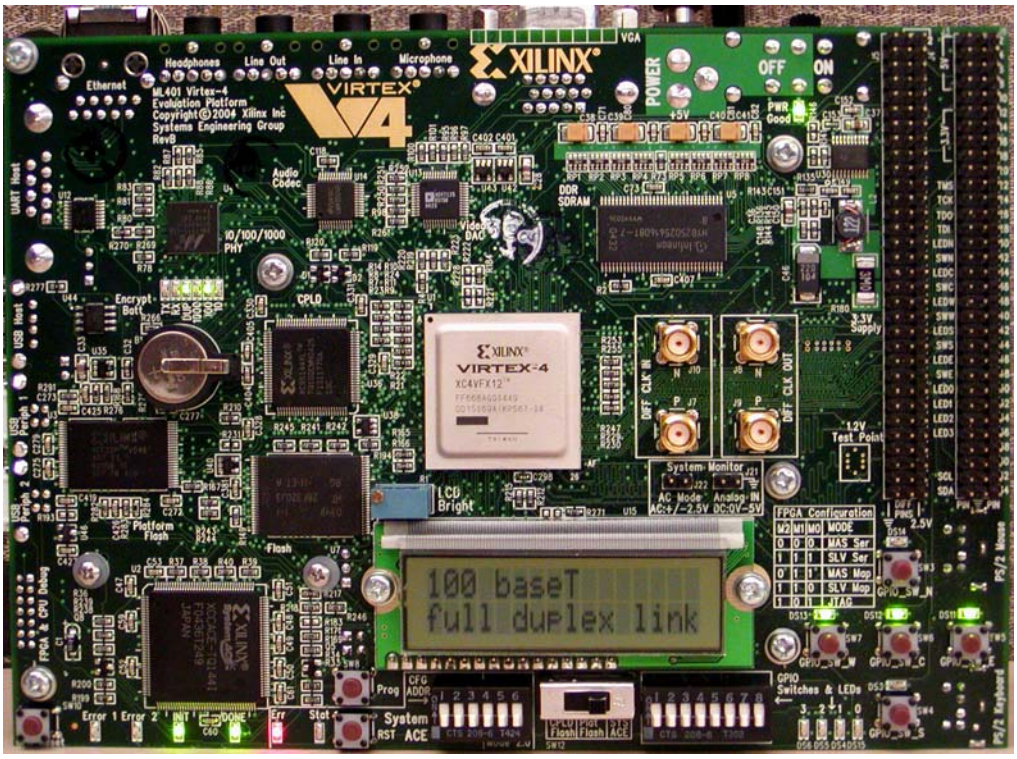
   - The demo.mcs is used if programming a PROM to load the bitstream
   - The demo.rbt file is used if programming by way of a JTAG interface

Setting Up the Board and Loading the Bitstream

The uIP application provided with the TEMAC UltraController-II module targets a Virtex-4 ML403 board based on a Virtex-4 FX12 FPGA. The target board configures the FPGA and loads the cache in a single step.

1. Connect power to the target board.
2. Connect a Xilinx Parallel Cable IV (PC4) JTAG cable between the host computer and target board.
3. Connect a cable from the target board to the host computer or the network.
   a. Use a crossover cable if connecting the board to a computer.
   b. Use an Ethernet cable if connecting the board to a network.
   c. Make sure that the network card on the computer is set to auto-negotiation.
4. From the ISE project's Process for Source window, select **Configure Device (iMPACT)** → **Run**. Assign demo.mcs to the PROM or assign demo.rbt to the FPGA to program the board in Boundary-scan mode. The user must set Startup Clock (FPGA) in **Edit** → **Preferences** → **iMPACT Configuration Preferences** to Ignore Setting.

   Initially, the LCD shows that the board is waiting for a connection. The board should then make a connection and display the connection speed on the LCD. Figure 7, page 11 shows how a working board should look after a connection has been established.

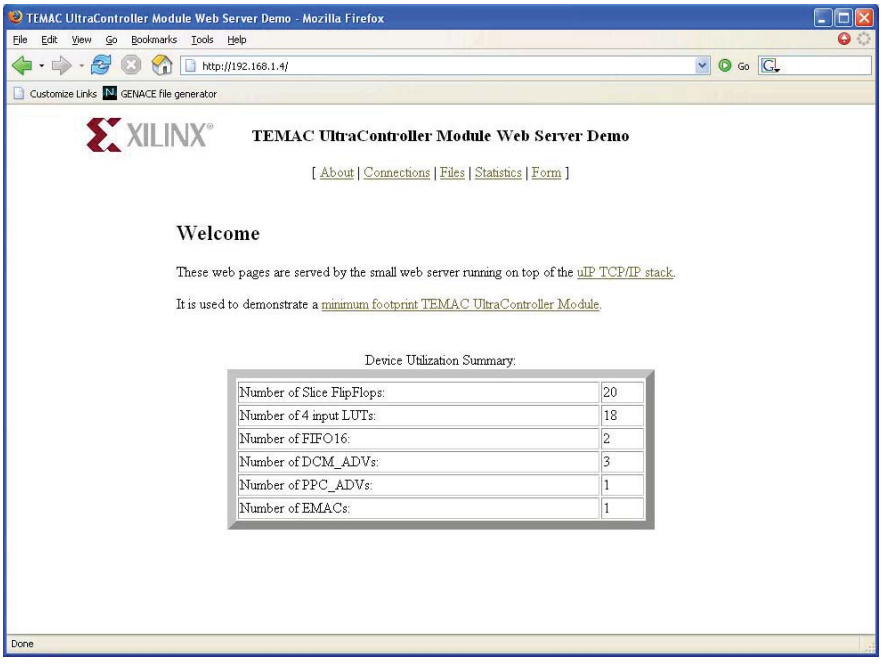


X807_07_080205

Figure 7: **ML403 Target Board Running a Network Connection**

5. Try pinging the ML403 board after connecting. In a Web browser, type the IP address of the ML403 board to see the sample web server page as shown in Figure 8.

   Note: This demonstration was tested in Internet Explorer v6.0 and Firefox v1.0.1.



X807_08_081105

Figure 8: **The uIP Web Server Welcome Page**

(Optional) Exporting from EDK to ISE

To change uc2.mhs and convert it to uc2.v, in XPS click **Tools** → **Generate Netlist**. A new uc2.v is generated in the **hdl** folder.

TEMAC Configuration

The Ethernet wrapper is generated from the CORE Generator™ tool, with the settings shown in Figure 9 and Figure 10. See UG074 for more information on the embedded tri-mode Ethernet MAC.

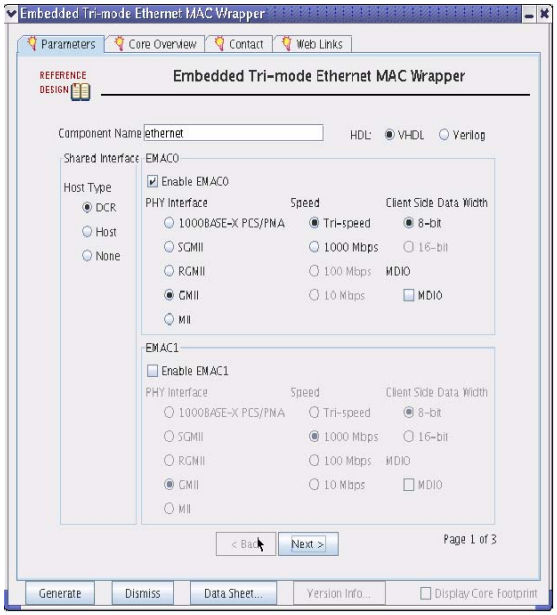


Figure 9: **Embedded Tri-Mode Ethernet MAC Wrapper (Page 1)**

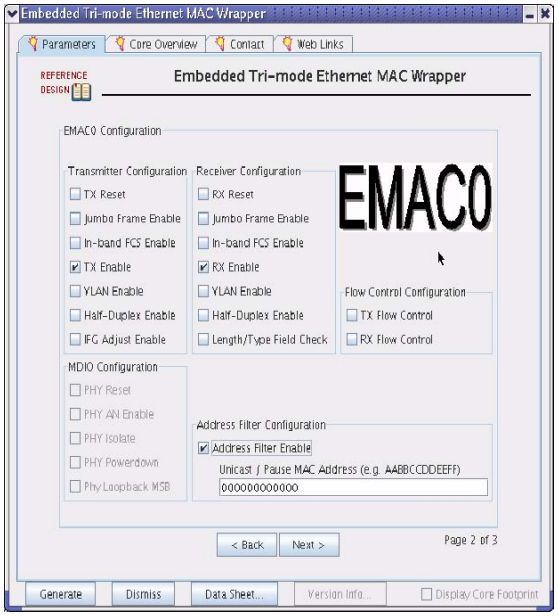


Figure 10: **Embedded Tri-Mode Ethernet MAC Wrapper (Page 2)**

Performance

The performance test was done with a SmartBits 600 (SMB-600) network performance analysis system. The network speed was measured by sending a ping request (60 bytes) to the ML403 board with the software uIP loaded.

The display indicates `% Utilization`, which shows the percentage of the bandwidth that is being used. For an example, 100% utilization means sending frames back to back; 50% utilization means sending a frame and waiting for a period of time equivalent to the time it took to send the frame.

Table 5 summarizes the network bandwidth utilization at speeds of 1000, 100, and 10 Mb/s.

Table 5: **Bandwidth Utilization**

	1000 Mb/s	100 Mb/s	10 Mb/s
Number of frames that can be handled in 100% utilization before the RX FIFO resets	36	310	Infinite
Max percent utilization without causing the RX FIFO to reset	9%	89%	100%
Percent of frames lost at 5% utilization	0%	0%	0%
Percent of frames lost at 10% utilization	30%	0%	0%
Percent of frames lost at 25% utilization	68%	0%	0%
Percent of frames lost at 50% utilization	84%	0%	0%
Percent of frames lost at 75% utilization	89%	0%	0%
Percent of frames lost at 100% utilization	92%	10%	0%

Notes:

1. Utilization numbers based on settings of proc_clk running at 350 MHz and sys_clk running at 175 MHz.

Resource Utilization

Table 6 shows the resource utilization of the TEMAC UltraController-II module implemented in an XC4VFX12 device.

Table 6: **Resource Utilization on an XC4VFX12 Device**

Resource	Used	Available	% Utilization
Slice Flip-Flops	20	10,944	<1%
LUTs (4-input)	18	10,944	<1%
Logic Distribution			
Occupied Slices	49	5,472	<1%
Total LUTs (4-input)	40	10,944	<1%
Bonded IOBs	41	320	12%
BUFG/BUFGCTRLs	8	32	25%
FIFO16/RAMB16s	2	36	5%
DCM_ADVs	3	4	75%
PPC405_ADVs	1	1	100%
EMACs	1	1	100%

Notes:

1. The table does not include utilization numbers for cache loading through the USR_ACCESS_VIRTEX4 register.

Reference Design

Figure 11 lists some of the reference design files contained in the zip file located at http://www.xilinx.com/bvdocs/appnotes/xapp807.zip. Figure 11 shows the Verilog structure only, although the VHDL structure is nearly identical.

```
xapp807/
        | -- temac_ultracontroller_module.xmp (XPS project file)
        | -- generate_rbt_mcs/
        |                    | -- genmcs (perl script used to generate .rbt and .mcs files from .mcd and .elf files)
        | -- hdl/
        |       | --uc2.v         (Verilog file with UltraController-II module, instantiating PPC405 block)
        | -- implementation/      (black boxes for hardware modules)
        | -- pcores/              (pcores used for the design)
        | -- ppc405_0/
        |             | -- code/
        |             |         | -- executable.elf          (pre-generated software executable for uIP)
        | -- projnav/
        |             | -- top_temac_example.v               (Top demo module for TEMAC UltraController-II Module)
        |             | -- top_temac_example_ML403.ucf       (Constraints File)
        |             | -- temac.v                           (module containing TEMAC instantiation)
        |             | -- top_temac_example.ise             (Project Navigator project file)
        |             | -- temac_controller.v     (verilog file with temac_controller module, instantiating TEMAC and
        |             |                           its supporting logic)
        |             | --uar_load.v   (verilog file with uar_load module used for configuring the cache through the
        |             |                 USR_ACCESS_VIRTEX4)
        | -- sw/
        |       | -- standalone/
        |       |               | -- linker_scripts/
        |       |               |                   | -- uc2_linker_script (linker script file)
        |       |               | -- src/
        |       |               |       | -- common/
        |       |               |       |           | -- crt0.S
        |       |               |       |           | -- gpio.c          (gpio driver)
        |       |               |       |           | -- lcs_ml403.c     (LCD driver)
        |       |               |       |           | -- sim_sleep.c     (timer)
        |       |               |       |           | -- xexception_l.c  (interrupt handlers)
        |       |               |       |           | -- xvectors.S
        |       |               |       | -- ethernet/                   (low-level drivers)
        |       |               |       |           | -- emac.c          (software handler for uIP)
        |       |               |       |           | -- emac_original.c (software handler for other applications)
        |       |               |       |           | -- xgpemac_l       (low-level driver for accessing EMAC registers)
        |       |               |       | -- uip-0.9/                    (source codes from uIP software)
        |       |               |       |           | -- apps/
        |       |               |       |           |         |-- httpd/
        |       |               |       |           |         |         | -- cgi.c
        |       |               |       |           |         |         | -- cgi.h
        |       |               |       |           |         |         | -- fs.c
        |       |               |       |           |         |         | -- fs.h
        |       |               |       |           |         |         | -- fsdata.c
        |       |               |       |           |         |         | -- fsdata.h
        |       |               |       |           |         |         | -- httpd.c
        |       |               |       |           |         |         | -- httpd.h
        |       |               |       |           |         |         | -- makefsdata (perl file that generates fsdata.c
        |       |               |       |           |         |         |                from html
        |       |               |       |           |         |         |               files in fs folder)
        |       |               |       |           |         |         | -- fs/ (folder containing html templates
        |       |               |       |           |         |         |        for the web server)
        |       |               |       |           | -- unix/
        |       |               |       |           |         |-- main.c
        |       |               |       |           |         |-- uip_arch.c
        |       |               |       |           |         |-- uipopt.h
        |       |               |       |           | -- uip /
        |       |               |       |           |         |-- uip.h
        |       |               |       |           |         |-- uip.c
        |       |               |       |           |         |-- uip_arch.h
        |       |               |       |           |         |-- uip_arp.c
        |       |               |       |           |         |-- uip_arp.h
        | -- demos/
        |         | -- executable.elf                  (pregenerated software ELF file)
        |         | -- executable.bin                  (pregenerated software BIN file)
        |         | -- top_temac_example.ncd           (pregenerated hardware NCD file)
        |         | -- demo.rbt                        (pregenerated RBT file for programming FPGA through JTAG)
        |         | -- demo.mcs                        (pregenerated MCS file for programming PROM)
```

Figure 11: **Reference Design Directory and Files**

Conclusion

The TEMAC UltraController-II module provides a convenient and inexpensive way of adding Ethernet functionality to a design. It uses very few FPGA resources, and the software can be run from the embedded PPC405 caches. Simple interfaces allow a shorter user development time. The web server application is used to demonstrate the design, but other solutions can be derived from these techniques.

References

1. Virtex-4 ML403 embedded development platform website
   http://www.xilinx.com/ml403
2. UG074, Virtex-4 Embedded Tri-Mode Ethernet MAC User Guide
   http://www.xilinx.com/bvdocs/userguides/ug074.pdf
3. For information about the uIP web server software used in this application note, see
   http://www.sics.se/~adam/uip/
4. UG018, PowerPC™ 405 Processor Block Reference Guide
   http://www.xilinx.com/bvdocs/userguides/ug018.pdf
5. UG071, Virtex-4 Configuration Guide
   http://www.xilinx.com/bvdocs/userguides/ug071.pdf
6. UG082, ML40*x* Reference Design User Guide
   http://www.xilinx.com/bvdocs/userguides/ug082.pdf
7. XAPP571, DEBUGHALT Controller for PowerPC Boot and Reset Operations
   http://www.xilinx.com/bvdocs/appnotes/xapp571.pdf
8. XAPP575, UltraController-II: Minimal Footprint Embedded Processing Engine
   http://www.xilinx.com/bvdocs/appnotes/xapp575.pdf
9. XAPP719, PowerPC Cache Configuration Using the USR_ACCESS_VIRTEX4 Register
   http://www.xilinx.com/bvdocs/appnotes/xapp719.pdf
10. For tutorials covering software ELF generation, see
    http://www.xilinx.com/ultracontroller
11. For information on the SmartBits 600 chassis, see
    http://www.spirentcom.com/documents/611.pdf

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
08/15/05	1.0	Initial Xilinx release.
10/04/05	1.0.1	Minor correction to step 1 of "Generating a Software ELF in Xilinx Platform Studio," page 9. Added reference to XAPP719.
01/17/06	1.1	Updated to ISE 7.1i Service Pack 4.
03/02/06	1.2	• Updated to ISE 8.1.02i and EDK 8.1.01i. • Added clock feedbacks on DCM1 and DCM2. • Removed sys_rst_out signal from top-level port list in top_temac_example.vhd. • Added script to convert ELF files to binary files on all platforms.
01/17/07	1.3	Updated xapp807.zip. See readme.txt for details.