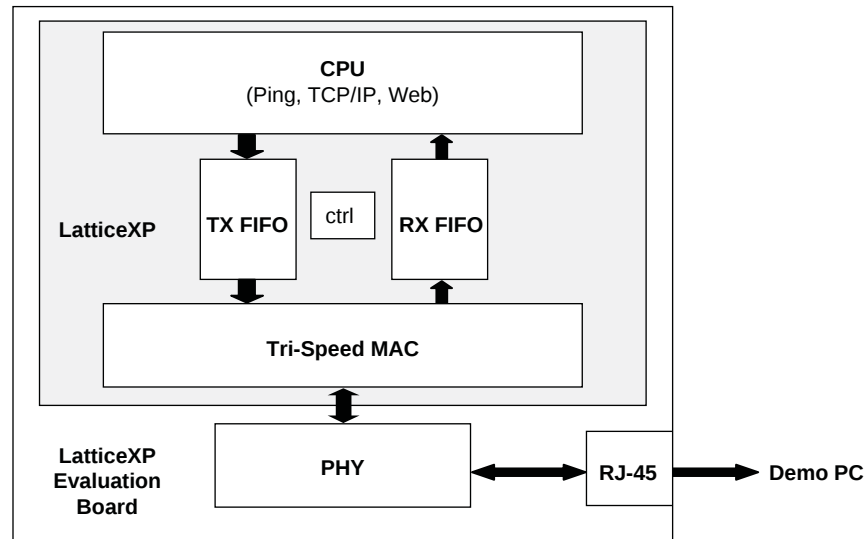


## Introduction

The following user's guide describes the Lattice Tri-Speed Ethernet Media Access Controller (TSMAC) IP demo. The demo shows the capability of the TSMAC core to function in a real network environment. The demo is designed to be simple and easy to use and requires no test equipment, lengthy setup or complex explanation. An application familiar to everyone, and practically synonymous with Ethernet and networking, is surfing the web. This demo uses a web server, entirely contained in a LatticeXP™ FPGA, to demonstrate the TSMAC IP core. This not only demonstrates the TSMAC IP, but it also shows the ability to embed a CPU in a Lattice FPGA and have an entire system in a chip.

The demo runs on a LatticeXP Advanced Evaluation Board connected to another host computer (demo PC) on the same network (or via a cross-over cable). The client demo PC will connect to the LatticeXP Advanced Evaluation Board, over a standard 100 Base-T network, using a web browser. The set-up is depicted in Figure 1.

**Figure 1. Tri-Speed Ethernet MAC Demo Setup**



This simple demo shows the ability of the TSMAC to be configured with a MAC address, receive 802.3 frames (both physical and broadcast), filter these packets and pass them to higher protocol software. It also shows the TSMAC receiving IP packets from higher layer software and transmitting them over the Ethernet.

TSMAC features demonstrated include:

- Real-world 802.3 Ethernet frames received and transmitted
- Accepts broadcast messages (DEST\_MAC = ff:ff:ff:ff:ff:ff)
- Accepts packets addressed to its physical MAC address
- Error counters and statistics

## Application Space

This demo, while not itself a truly useful application, could be used as the basis for real applications such as those described below.

### TSMAC Applications

- Control plane interface for low-end, networked devices (10/100 Base-T embedded systems)
- Packet validation and encryption functions in switches and firewalls
- Multiple TSMACs could be used to form a gigabit Ethernet switching solution. This would require a more powerful FPGA (LatticeSC™) and the use of external RAM and a powerful control plane processor

### LatticeXP Applications

- Networking device – this demo code could be used as a basis for constructing a networked device (see [www.openTCP.org](http://www.openTCP.org) or uIP for more information on TCP/IP stack usage considerations in embedded devices)
- Small system solution in an FPGA – this demo shows possibilities for using an 8-bit CPU, EBRs, custom IP and standard IP to produce a complete system solution in a LatticeXP device

### Limitations

The demo is limited in functionality because of the CPU and software resource constraints in the LatticeXP. The major issue being that EBR memory must hold all program instructions, data, web pages, variables, FIFOs, etc. A complete, robust TCP/IP stack is therefore not possible to fit in a LatticeXP-10 with 20kB left to the CPU. This would not be an issue in a larger LatticeXP device, but for this demo the following limitations apply:

- Recommend communication with one client at a time (demo PC)
- Supports a very basic TCP/IP stack with limited error handling, limited gateway support, no buffering or retransmit on collision
- Network configuration hard-coded – IP address, MAC address, gateway, etc.
- No dynamic web content; only hard-coded, static pages are displayed
- 100BaseT full-duplex operation only (no auto-negotiation)
- Expect low throughput, with dropped packets, on a busy network

### Background Knowledge

This demo assumes the user is familiar with basic TCP/IP networking concepts and has experience connecting equipment to networks and basic network trouble shooting skills. In particular, the user should be familiar with IP addressing, MAC addresses, the Ping utility, using a hub or swapped cable and configuring addresses on a PC.

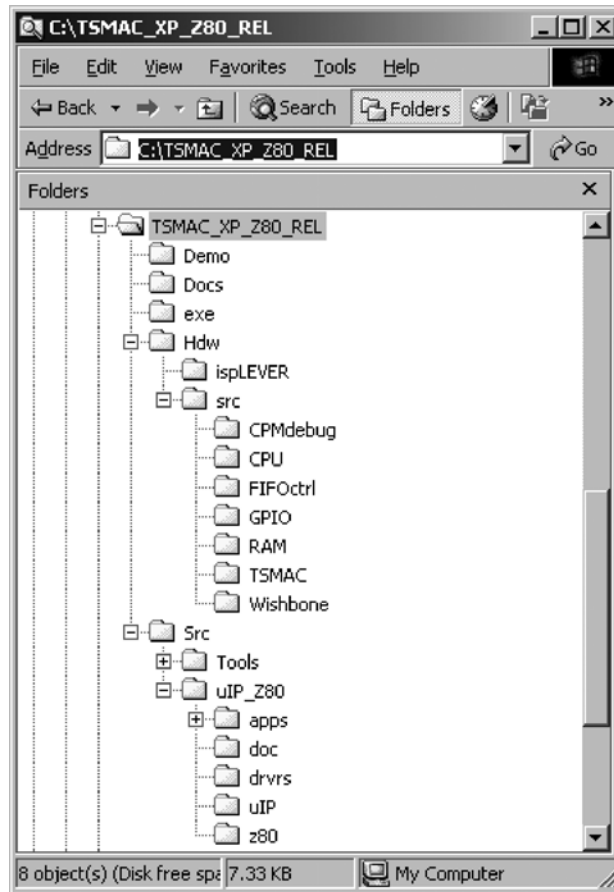
### Installation Guide

The TSMAC LatticeXP demo package is released by Lattice in a zipped file. Simply download the file and unzip it. The pre-built bitstream is included and must be loaded into the LatticeXP device. Other Lattice tools that may be needed are ispVM® System software to download the bitstream, and ispLEVER® design tools if any changes to the design are to be made.

### Installation

The LatticeXP TSMAC demo is distributed in a zipped file. Simply unzip the file and the directory structure shown in Figure 2 will be created.

Figure 2. Demo Directory Structure



- **Demo/** – Pre-built bitstream and ispVM configuration ready for download into the LatticeXP
- **Docs/** – User's guide
- **exe/** – Windows executable utilities used with bitstream and network parameter changes
- **Hdw/islever/** – Project and preference files to synthesize, map, place, route and create the demo bitstream
- **Hdw/Src/** – The Verilog HDL of the demo design
- **Src/uIP\_Z80/** – The demo application source code: TCP/IP stack, web server, pages, etc.
- **Src/Tools/** – Source code for Windows applications used in project development

## Environment Setup

The default demo bitstream is provided in the Demo/ directory. Nothing needs to be configured to use this default bitstream. If you plan to change the network parameters of the LatticeXP Advanced Evaluation Board, you will need to run a setup script to install certain environment variables and update the PATH to find the utilities.

*NOTE: ispVM System software and a Lattice ispDOWNLOAD<sup>®</sup> cable are required to download the bitstream into the LatticeXP device. Without these two tools, there is no way to load changes made to the demo operations. If you don't have an ispDOWNLOAD cable, you can skip the remainder of this section.*

*islever is needed to rebuild the bitstream if changes are made to the hardware design or if changes to the software (IP parameters) are to be permanently saved in the bitstream. If you do not have isplever installed, you cannot change the bitstream.*

**setup.bat**

The setup.bat program, located at the top level of the demo installation, creates the necessary environment variables and adds the appropriate directories to the search path. This file should be looked at and edited if the tools (ispVM, ispLEVER) are installed in places other than their default. Once edited to match the system setup, run setup.bat from a DOS window and then complete all work in that DOS window.

**Environment Variables**

The following environmental variables are created by setup.bat, or could be manually added to your environment.

- **ISPVM\_DIR** – Path to installation of ispVM System software. By default, this is c:\ispTOOLS\ispvmsystem\
- **ISPVM\_DEV** – If using something other than the USB pod. Default is EzUSB-0
- **ISP\_BIN** – Set to the location of the ispLEVER tools (the directory that holds memedit.exe, bitgen.exe, etc.) This is usually: c:\ispTOOLS5\_1\ispfpga\bin\nt\. The last '\ ' is required. An alternate method is to specify the location of the ispLEVER tools in your PATH environment variable

**PATH**

The PATH environment variable is modified by setup.bat to include access to the utilities used in changing the IP address in the bitstream and to locate DLLs used when communicating via the ispVM JTAG download cable.

In addition, if the application software is to be recompiled, add the path to the Z80 compiler tools z88dk with the command.:

```
C:> set PATH=%PATH%;c:\z88dk\bin
```

**The Demo**

The demo includes three separate pieces that together form the complete demonstration:

1. The hardware design – TSMAC IP, other IP, ispLEVER, ispVM
2. The application software – Z80 CPU, TCP/IP stack, web server
3. Network administration – Configuring IP addresses, testing networks, connecting equipment

The simplest way to use the demo is to load the pre-built bitstream from the Demo/ directory into a LatticeXP device on the evaluation board, and connect to the evaluation board with a hub or swapped cable.

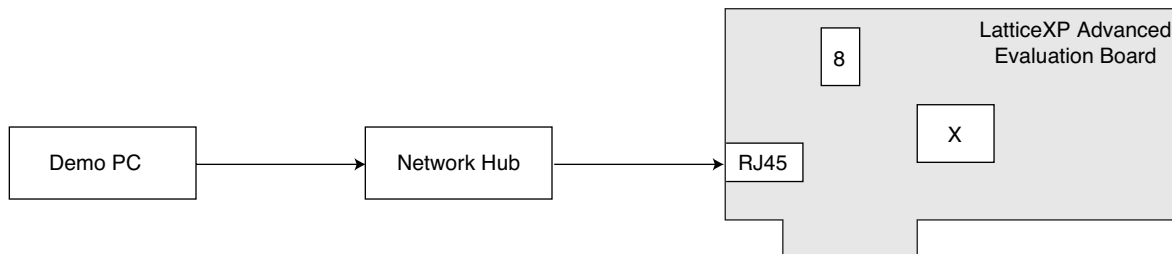
The next easiest method is to change the default IP address of the board and connect it to a real network. This requires having ispLEVER loaded so the utilities in Demo/ can edit.

A more complex method requires rebuilding the bitstream. This also requires ispLEVER to be installed and the TSMAC\_XP\_Hdw project to be loaded and synthesized.

The most complex way to use the demo is to change the application code (web pages, etc.). This requires installing the Z80 software development kit and other utilities to rebuild the application and embedding the CPU instructions and data into the EBR contents of the bitstream.

**Demo Setup**

The client demo PC connects to the LatticeXP Advanced Evaluation Board over a standard 100 Base-T network. The setup is depicted in the Figure 3.

**Figure 3. Connecting the LatticeXP Advanced Evaluation Board to a Network**

The demo can be set up in one of two modes. The evaluation board can be connected to a network and accessed by any computer on this network. Or, the evaluation board can be directly connected to the demo PC using a swapped network cable or a stand-alone hub. The direct connection provides quicker set-up and does not require a valid IP address for the evaluation board. The network operation method requires extra work since the LatticeXP bitstream must be modified with a new IP address, but it does show the TSMAC responding to real network traffic.

The board must have a valid address to communicate with other systems. The board's bitstream has a built-in default address of 192.168.33.175.

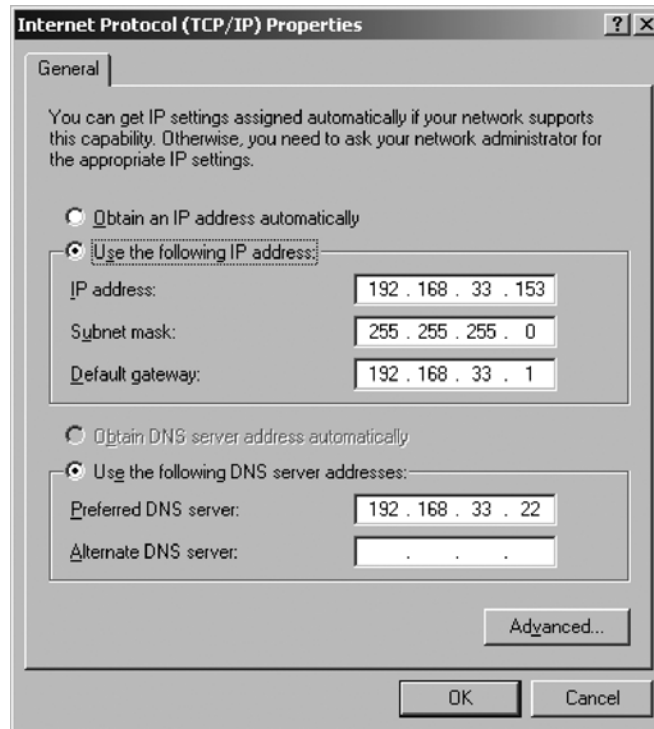
#### **Direct Connect**

To directly connect a PC, such as a laptop, to the evaluation board, a swapped cable or a hub can be used between the two systems. The PC and the evaluation board are then on their own network. Therefore, the LatticeXP demo default IP address can be used. The default LatticeXP IP address is 192.168.33.175. The PC must be changed to an address on the 192.168.33.0 network. Any IP address can be chosen (except .175). The example below shows 192.168.33.153.

**WARNING! Do not use this method and also connect to the Lattice network. This may conflict with an existing machine assigned an IP address at Lattice.**

In Windows, changing the PC's IP address is done through:

Start->Settings-> Network and Dial-up Connections->Local Area Connection->Properties->Internet Protocol (TCP/IP)->Properties-> Fixed IP address

**Figure 4. Internet Protocol (TCP/IP) Properties Window**

You may need to log into your PC using the machine's domain rather than a network domain login account. This ensures that you will have the proper permissions to change your network settings. See your network administrator if you have permission issues.

### Installation on a Network

To install the evaluation board on an existing network, it will need a valid IP address for that network. This should be a fixed IP address on the network. See your network administrator for an address. The IP address then needs to be programmed into the TCP/IP software running on the embedded CPU. The network's gateway and net mask must also be known. (As of this writing, only Class C addresses have been tested, without subnetting - unsure if the TCP/IP stack supports anything else).

All of the methods that change the board's IP address requires use of the ispVM tool at a minimum.

### Application Change:

The binary application code that executes on the internal CPU has fixed networking parameters in the code. These values can be inspected and modified using the netchange.exe program.

```
C:\TSMAC_XP_Z80_REL\Demo>netchange
```

```
netchange <Z80.bin> <Z80.map> [param=val ...]
```

Change network parameter values in the Z80 binary file 'Z80.bin' using 'Z80.map' to locate the global variables to be changed. This is useful to modify the IP address of a TSMAC XP demo board without recompiling the code. Not specifying parameters to change causes the existing values to be displayed but not changed.

Network Parameters:  
i=dd.dd.dd.dd change IP address

```
g=dd.dd.dd.dd change gateway IP address
m=dd.dd.dd.dd change netmask value
e=dd (0-31) modify TSMAC Ethernet address
```

For example, to change the IP address run:

```
C:\TSMAC_XP_Z80_REL\Demo>netchange demo.bin demo.map i=10.0.0.22
```

Once the network parameters have been set to the correct values in demo.bin, use the download utility to transfer the executable program into the LatticeXP EBR using the ispVM System software. The LatticeXP device must already have a valid tsmac\_xp\_hdw bitstream loaded and running.

```
C:\TSMAC_XP_Z80_REL\Demo>ispLoad demo.bin
LatticeXP TSMAC Demo CPU Download Program
built: Mar 16 2006 17:07:49
Creating access to the XP via JTAG ispVM
ISPVM_DIR found. Using location: c:\ispTOOLS\ispvmsystem
ISPVM_DEV found. Using: EzUSB-0
. . .
```

The CPU will reboot after downloading and the new network parameters will be used as long as the device is not reprogrammed (PROGRAM button, power-cycled or evaluation IP timeout).

#### To Edit the Bitstream

1. Use the netchange program to modify the application network parameters as described above
2. Run bin2hdw to convert the binary into EBR memory format and move it into the Hdw/ispLEVER/ director.
3. From the Hdw/ispLEVER/ directory, run add\_prog.bat and then isp\_bitgen.bat
4. Use ispVM System software to load the tsmac\_xp\_hdw.bit file into the LatticeXP device

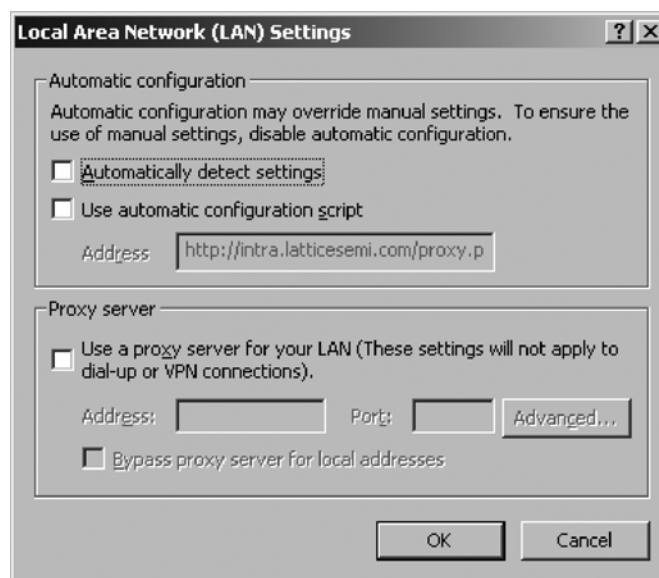
#### To Recompile the Application Source

1. In Src/uiP\_Z80/z80/, change the network parameters in uip\_opt.h
2. In Src/uiP\_Z80/z80/, run makeweb.bat
3. From the Hdw/ispLEVER/ directory, run add\_prog.bat and then isp\_bitgen.bat
4. Use ispVM System software to load the tsmac\_xp\_hdw.bit file into the LatticeXP device

#### **Additional Configurations**

The proxy server for the web browser must be disabled since the browser will connect directly to the evaluation board rather than through a proxy server. If your system displays the message shown in Figure 5, follow these steps in Windows for Internet Explorer to disable the proxy:

Tools->Internet Options->Connections->LAN Settings-> Disable Proxy Server

**Figure 5. Message Error Shown if Proxy Server is Enabled****Figure 6. Local Area Network (LAN) Settings Window**

## Running the Demo

This section guides the user through operating the demo. The network connection method and IP address should be configured using one of the methods in the previous section. The user should pay attention to the LEDs to ensure proper operation of the hardware.

*Note: The TSMAC IP evaluation core allows operation for one hour. After one hour, the entire board will lock up. The bitstream will need to be reloaded using ispVM System software, or if programmed into the Flash memory, the PROGRAM button will need to be pushed.*

## Visual Indicators

The demo uses the LEDs on the evaluation board to provide visual feedback of the demo operation. An important indicator of software operation is the steady blinking of LED1. LED8 should be lit once connected to a network, indicating a 100Mbps Ethernet link is established. When packets are being sent over the network, LED7 will blink. The 7-Segment display will increment whenever a valid packet is received. LED2 to LED5 will light up depending on the types of packets received and transmitted.

### Discrete LEDs

The eight individual green LEDs at the edge of the board, above the DIP switches, are used to indicate various conditions. The LEDs are numbered on the board with D1 at the top and D8 at the bottom (PCI fingers). Their functions are:

1. heart beat – Software blinks at every pass through the main loop when looking for packets
2. rx\_packet – Set whenever a packet has been received from the TSMAC
3. tx\_packet – Set whenever a packet has been sent to the TSMAC
4. ARP – Set whenever an ARP protocol packet has been received
5. Finger – Set whenever a finger protocol packet has been received
6. HTTP – Set whenever an http (web page) protocol packet has been received
7. Link Activity – Hard wired from PHY to indicate network activity
8. LINK Good – Hard wired from PHY to indicate a good 100Mbit link is detected

The software controlled LEDs have a two-second persistence.

### Segment Display

The display indicates the number of packets received by the software from the TSMAC. The decimal point will light if an error condition has been detected, such as a CRC error, FIFO overflow errors, etc. This display is completely controlled by the software.

### RJ-45 Jack

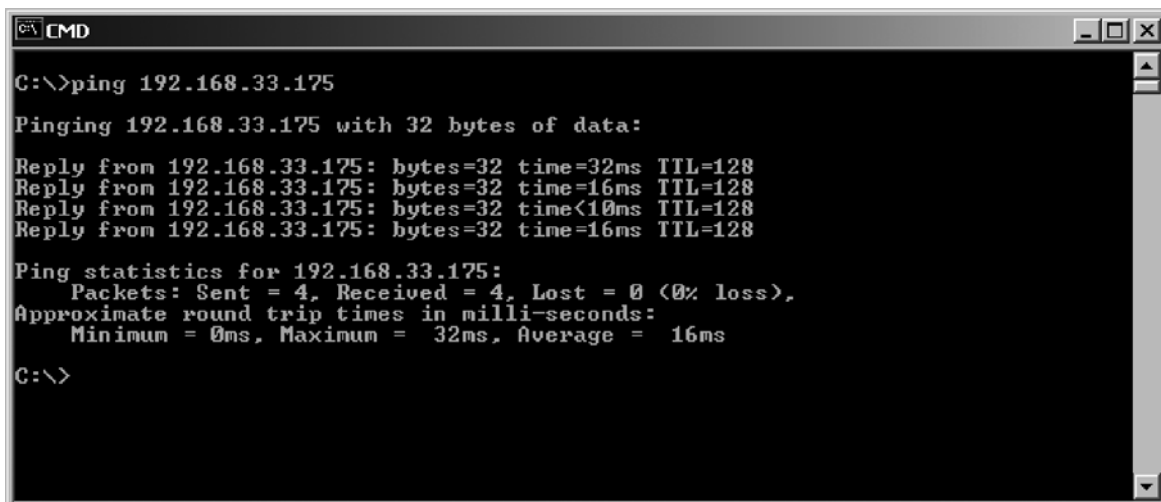
The yellow LED on the rear of the RJ-45 jack (closest to the PCI fingers) indicates network activity. This LED appears to be controlled by the PHY and/or connector. It is not controlled by the MAC or software, but works only after the PHY has been configured. This is a good indicator of traffic on the network.

### **Ping the Board**

The evaluation board responds to standard network ping commands from a host computer (demo PC). Ping shows network connectivity to the evaluation board, which indicates a properly functioning TSMAC. The first step should always be to ping the board to ensure network connectivity.

If you cannot ping the board, the other demos will not work. Ping must work before continuing.

**Figure 7. Ping the Board**



```
CMD

C:\>ping 192.168.33.175

Pinging 192.168.33.175 with 32 bytes of data:

Reply from 192.168.33.175: bytes=32 time=32ms TTL=128
Reply from 192.168.33.175: bytes=32 time=16ms TTL=128
Reply from 192.168.33.175: bytes=32 time<10ms TTL=128
Reply from 192.168.33.175: bytes=32 time=16ms TTL=128

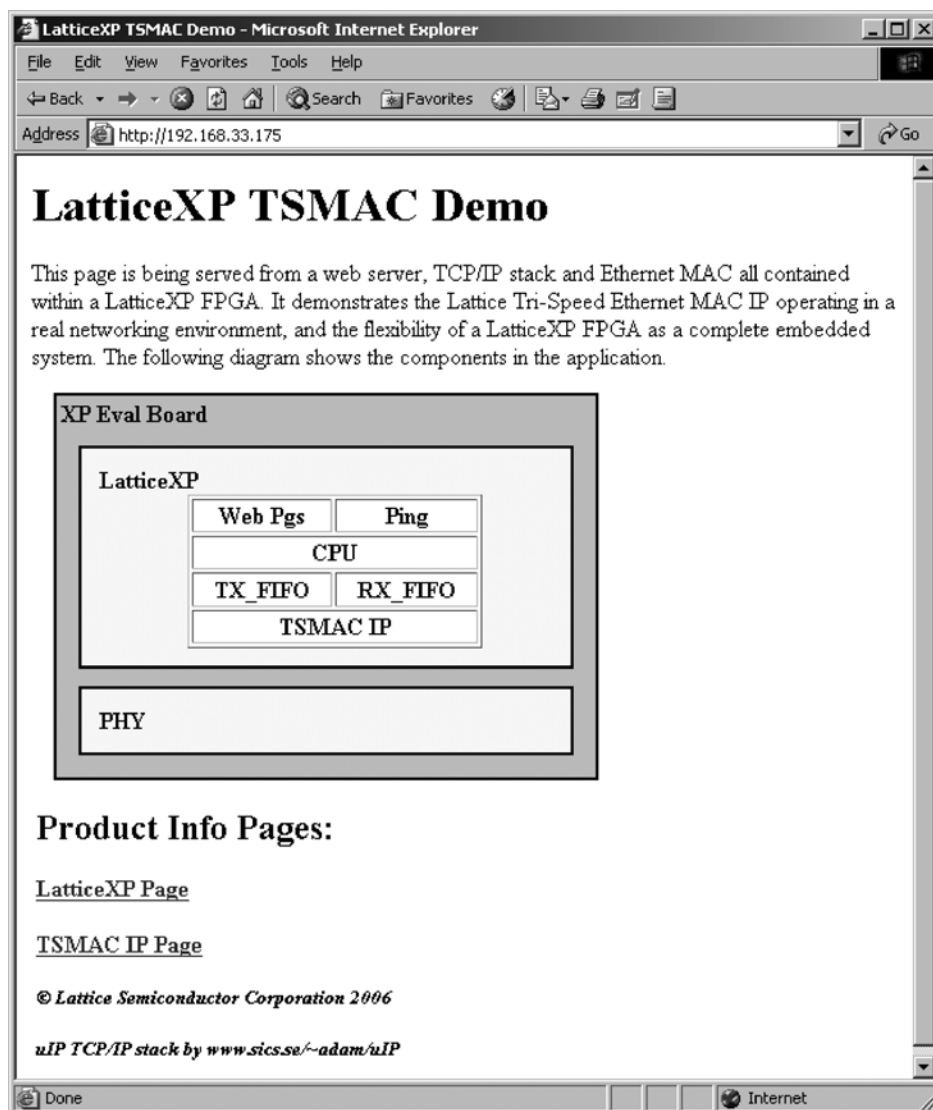
Ping statistics for 192.168.33.175:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 32ms, Average = 16ms

C:\>
```

### Demo Web Pages

The software running on the CPU inside the LatticeXP device contains a web server task that will serve pages to any computer that connects to it. The user can view the following pages in their web browser (remember to disable any use of a proxy, and choose to connect directly to the Internet). Open Internet Explorer and enter the board's IP address. The home page introduces the demo.

**Figure 8. LatticeXP TSMAC Demo Home Page**



There are two additional links at the bottom of the page. They open the pages shown in Figures 9 and 10.

Figure 9. LatticeXP Page

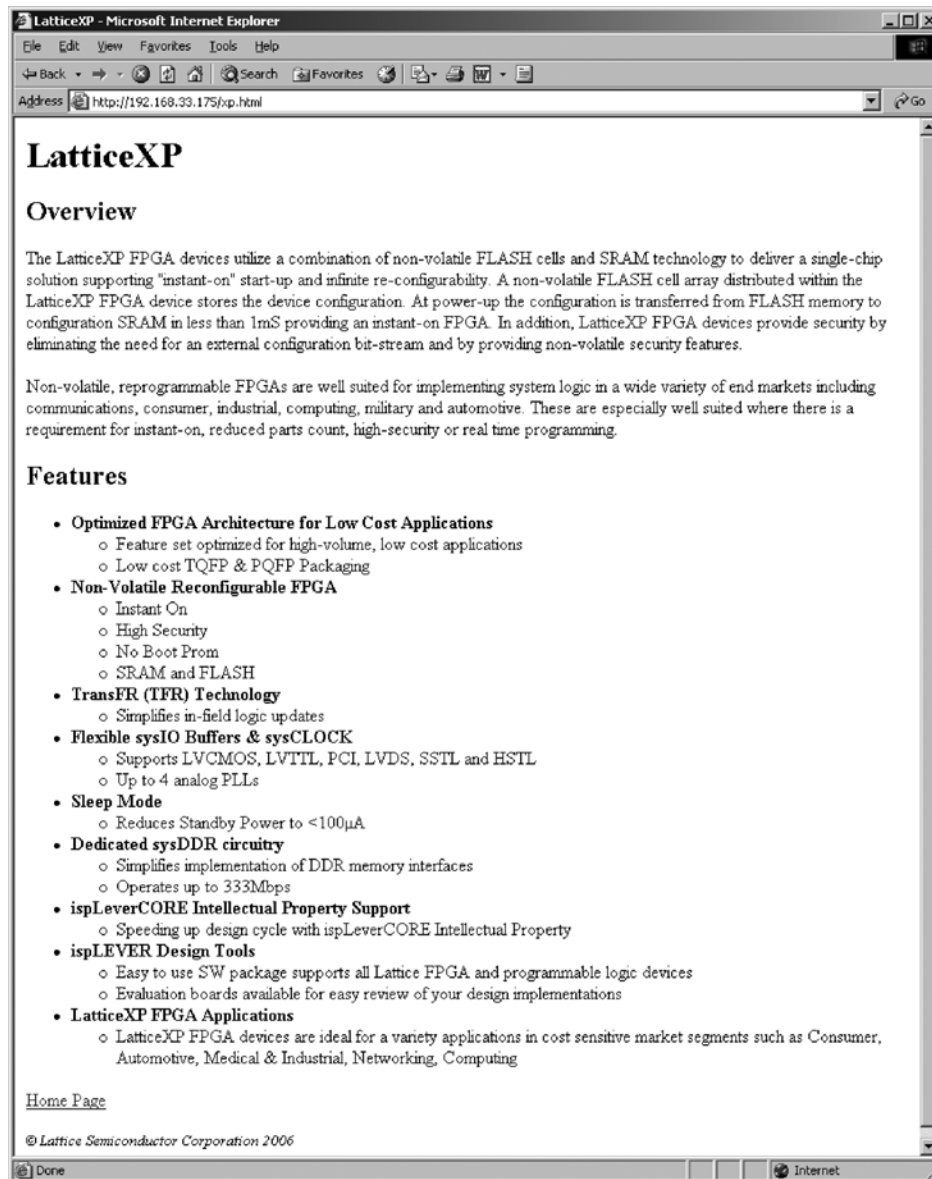
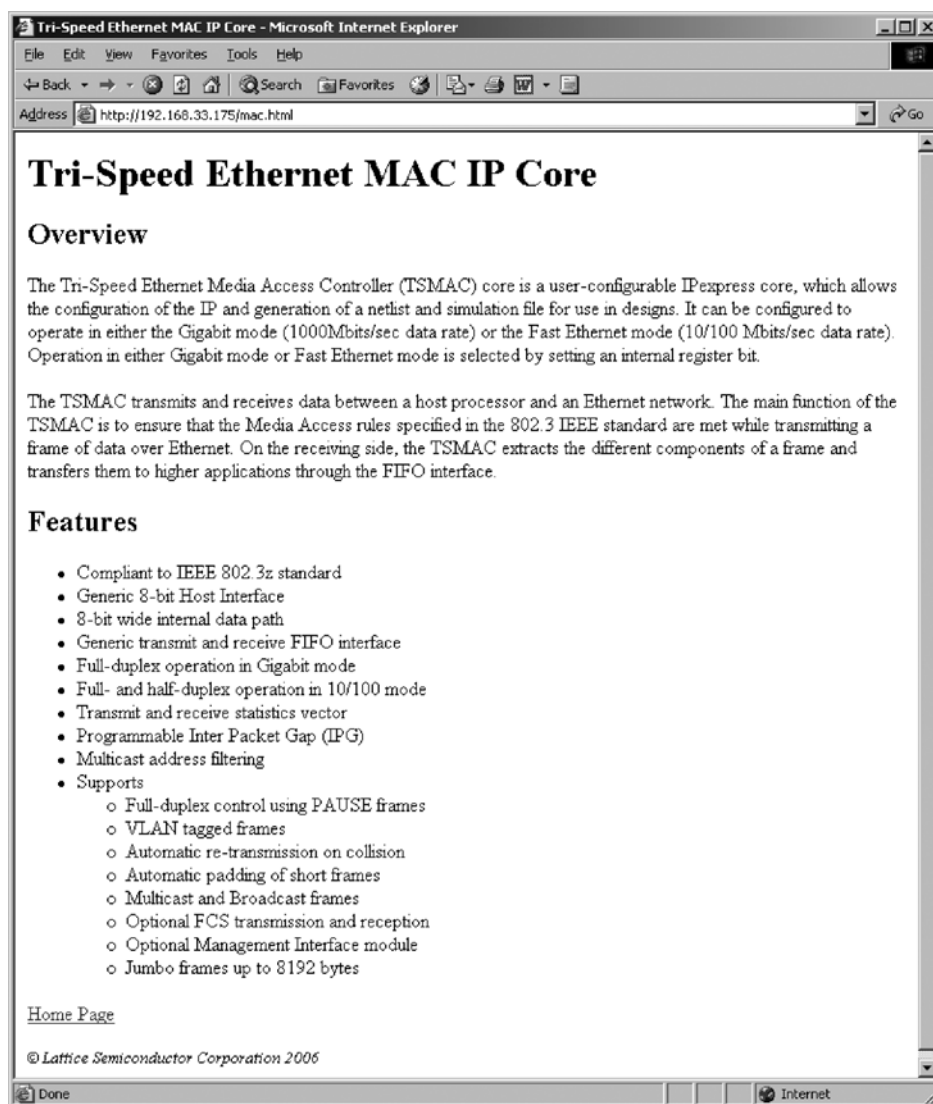


Figure 10. TSMAC IP Page



### TSMAC Statistics

Use the TCP protocol Finger to return statistics and configuration settings from the board.

The demo PC issues the following finger commands and the operating information is returned and displayed for the user. There are two finger command options:

- `finger stats@<ip>`
- `finger mac@<ip>`

**Figure 11. Finger for Statistics and Configuration Settings**

```
C:\>finger stats@192.168.33.175

[192.168.33.175]
IPNetwork Stats
MAC: 00:50:c2:58:c1:00 IP: c0.a8.21.af
RX pkts: 0x0011 errs: 0x0000
TX pkts: 0x0006 drop: 0x0001
d

C:\>finger mac@192.168.33.175

[192.168.33.175]
TSMAC Stats
RX pkts: 0x0073 drop: 0x0023 CRC: 0x0000
TX pkts: 0x0021 err: 0x0000
d

C:\>_
```

The stats option displays network configuration and statistics as seen by the TCP/IP and application software.

- The MAC address is the 6-byte address written into the TSMAC registers by the initialization software
- The IP address is the address the board is using
- Rx pkts is the 16-bit count of good Ethernet packets read from the Rx FIFO by the driver software
- Rx errs is the 16-bit count of software errors encountered during reading a packet, such as bad length (>1024), no EOF, and other hardware/driver interactions
- Tx pkts is the 16-bit count of Ethernet packets written into the Tx FIFO by the driver software
- Tx errs is the 16-bit count of software errors encountered during writing a packet, such as bad length (>1024) or if the FIFO still had a packet in it (only one allowed at a time in this demo)

The mac option displays TSMAC statistics, as counted by hardware signals from the core, such as mac\_rx\_eof, mac\_rx\_error, mac\_tx\_done, etc.

- Rx pkts is the 16-bit count of good Ethernet packets read from the Rx FIFO by the driver software
- Rx drop is the 16-bit count of packets dropped by the TSMAC because the rx FIFO had a packet and the mac\_ignore\_pkt signal was asserted. This is due to slow software and the demo allowing only one packet in the FIFO at a time
- Rx CRC is the 16-bit count of packets detected to have a bad CRC and dropped by the TSMAC
- Tx pkts is the 16-bit count of Ethernet packets read from the Tx FIFO by the TSMAC
- Tx errs is the 16-bit count of errors encountered during transmitting a packet, such as collisions or FIFO under-run

See the *Tri-Speed Ethernet Media Access Controller User's Guide* for more details on error statistics reported by the TSMAC.

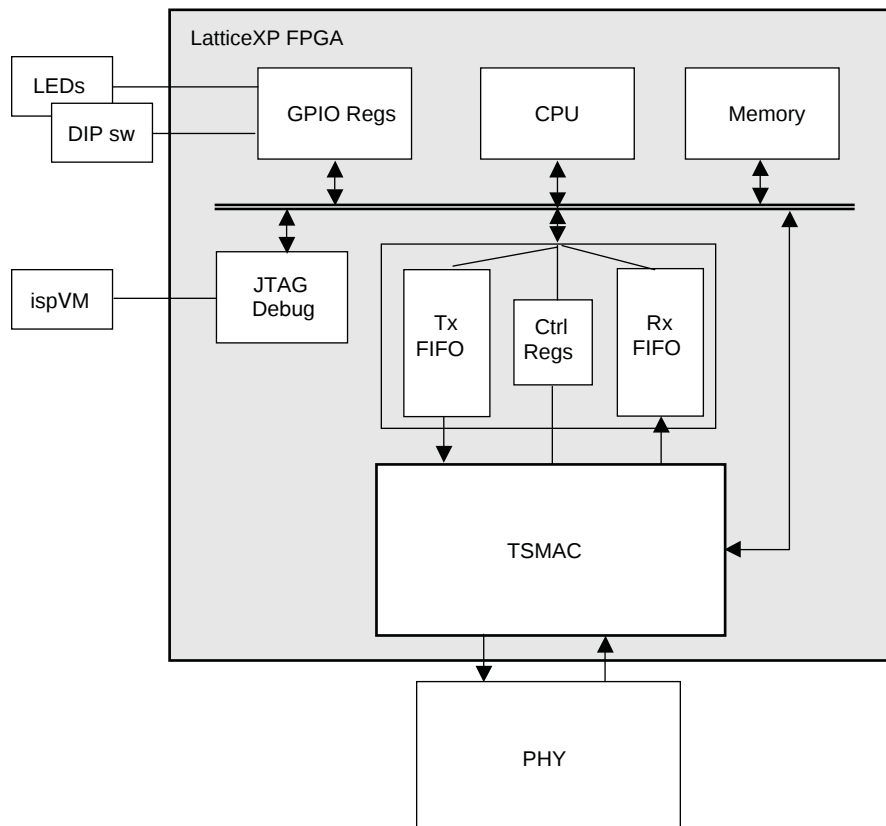
## Design Details

This section provides technical information on the demo system design. The supporting IP around the TSMAC core is described in detail. The software design is also described.

### Hardware

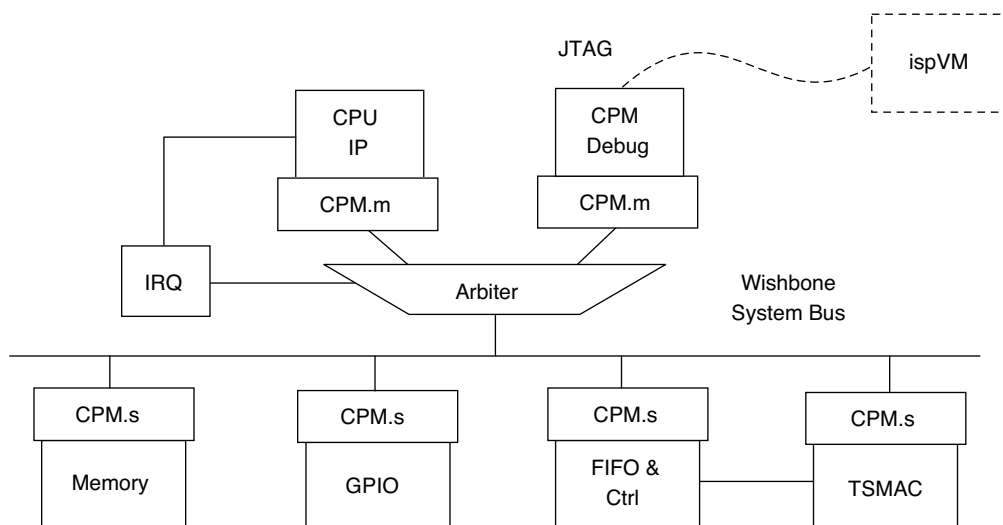
The following block diagram shows a high-level view of the IP blocks within the LatticeXP device and the external PHY device.

**Figure 12. Block Diagram of IP Blocks Contained in the LatticeXP**



### IP Interconnection

The IP modules are interconnected using the Wishbone bus and CPM Master and Slave interfaces. The Wishbone bus is an open standard that defines a microprocessor bus suitable for use in FPGAs. The demo provides a proof-of-concept implementation of the bus, using the proposed CPM interface to bridge Lattice IP to the Wishbone bus.

**Figure 13. Bridging IP with a Wishbone Bus Block Diagram**

The Wishbone bus specification can be found at [www.opencores.org](http://www.opencores.org). The CPM is a method used in this design to connect Wishbone Masters and Wishbone Slaves to the same bus.

### TSMAC Details

Details of the Tri-Speed MAC IP core can be found in the *Tri-Speed Ethernet Media Access Controller User's Guide* located on the Lattice web site at [www.latticesemi.com](http://www.latticesemi.com).

The TSMAC host interface signals are connected to the Wishbone bus via a CPM.slave module. The TSMAC internal registers are then accessible to the CPU. The TSMAC is located at address 0x8200 on the Wishbone bus. The following TSMAC registers are accessed by the CPU:

**Table 1. TSMAC Register Map**

| Register Description                       | Mnemonic     | Offset    | POR Value |
|--------------------------------------------|--------------|-----------|-----------|
| Mode register                              | MODE         | 00H - 01H | 0000H     |
| Transmit and Receive Control register      | TX_RX_CTL    | 02H - 03H | 0000H     |
| Maximum Packet Size register               | MAX_PKT_SIZE | 04H - 05H | 05EEH     |
| Inter Packet Gap register                  | IPG_VAL      | 08H - 09H | 0048H     |
| Tri-Speed MAC Address register 0           | MAC_ADDR_0   | 0AH - 0BH | 0000H     |
| Tri-Speed MAC Address register 1           | MAC_ADDR_1   | 0CH - 0DH | 0000H     |
| Tri-Speed MAC Address register 2           | MAC_ADDR_2   | 0EH - 0FH | 0000H     |
| Transmit and Receive Status                | TX_RX_STS    | 12H - 13H | 0000H     |
| GMII Management Interface Control register | GMII_MNG_CTL | 14H - 15H | 0000H     |
| GMII Management Data register              | GMII_MNG_DAT | 16H - 17H | 0000H     |
| VLAN Tag Length/type register              | VLAN_TAG     | 32H - 33H | 0000H     |
| Multicast_table_0                          | MLT_TAB_0    | 22H - 23H | 0000H     |
| Multicast_table_1                          | MLT_TAB_1    | 24H - 25H | 0000H     |
| Multicast_table_2                          | MLT_TAB_2    | 26H - 27H | 0000H     |
| Multicast_table_3                          | MLT_TAB_3    | 28H - 29H | 0000H     |
| Multicast_table_4                          | MLT_TAB_4    | 2AH - 2BH | 0000H     |
| Multicast_table_5                          | MLT_TAB_5    | 2CH - 2DH | 0000H     |
| Multicast_table_6                          | MLT_TAB_6    | 2EH - 2FH | 0000H     |
| Multicast_table_7                          | MLT_TAB_7    | 30H - 31H | 0000H     |
| Pause_opcode                               | PAUS_OP      | 34H - 35H | 0080H     |

The TSMAC and PHY are configured by the demo software to operate only in the following modes:

- 100 Base-T Mode (no auto-negotiation)
- Full duplex (no need to run half-duplex)
- Flow control enabled (not used)
- Strip preamble, SFD on received packets
- Keep CRC in received buffer
- MAX\_FRAME\_SIZE = 1024 bytes (due to limited EBR memory)
- Disable receive short frames
- Disable promiscuous mode

The following TSMAC registers (and PHY registers) are written by the software to configure the above modes:

```
// put TSMAC in IDLE state (disable Tx & Rx)
TSMAC->mode.reg8.lo = 0x00; // everything disabled

// TSMAC Configuration
TSMAC->tx_rx_ctl.reg8.lo= 0x88; // full-duplex, no multicast, pause OK, keep CRC, no promiscous
TSMAC->tx_rx_ctl.reg8.hi= 0x00; // no frames shorter than 64 bytes
TSMAC->max_pkt_size.reg8.lo = (UIP_BUFSIZE & 0xff); // max FIFO space in this demo
TSMAC->max_pkt_size.reg8.hi = ((UIP_BUFSIZE>>8) & 0xff);
TSMAC->ipg_val.reg8.lo = 0x08; // default Inter-packet gap in units of bit time
TSMAC->mac_addr_0.reg8.lo = uip_ethaddr.addr[1]; // Fixed MAC address defined in uipopt.h
TSMAC->mac_addr_0.reg8.hi = uip_ethaddr.addr[0]; // byte order defined in TSMAC users guide.
TSMAC->mac_addr_1.reg8.lo = uip_ethaddr.addr[3]; // Its not straight-forward.
TSMAC->mac_addr_1.reg8.hi = uip_ethaddr.addr[2];
TSMAC->mac_addr_2.reg8.lo = uip_ethaddr.addr[5];
TSMAC->mac_addr_2.reg8.hi = uip_ethaddr.addr[4];

// Setup the National 83865 PHY using the GMII Ctrl and Data registers
PHY_write(0, 0x2100); // 100 MBit mode, full duplex, no auto-negotiation
```

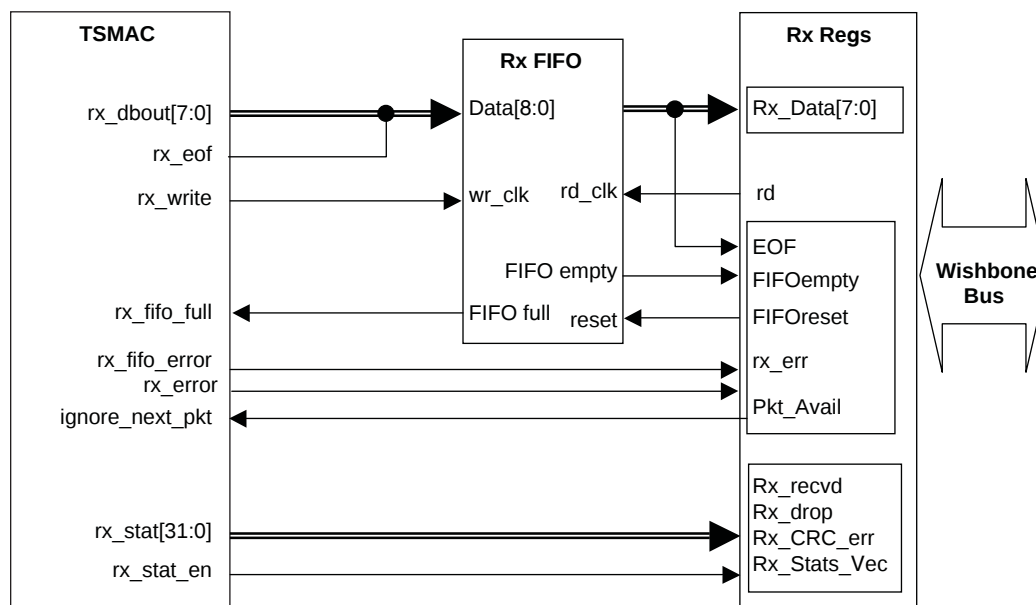
### FIFO Interface IP

The FIFO Interface IP connects the TSMAC packet data to the CPU bus. The FIFOs provide the storage for IP packets received and processed by the TSMAC, destined for the application. They also provide storage for IP packets generated by the application, ready for transmission by the TSMAC. This section details the hardware design and the functional operation. The TSMAC is located at address 0x8200 on the Wishbone bus.

### Receive Design

Figure 14 diagrams the components, data flow and control flow for receiving an IP packet from the TSMAC and making it available to the CPU.

Figure 14. Receive Block Diagram



**Module Descriptions:** The TSMAC module receives the Ethernet frame and writes the received bytes directly into the Rx FIFO. Only the receive signals of the TSMAC are shown in the diagram above. These signals control clocking received bytes into the FIFO and provide information about the status of the packet received.

The FIFO module provides the interface between the receive side of the TSMAC and the host CPU TCP/IP stack. It is a LatticeXP EBR Dual Clock 9x1024 FIFO library element. The received byte is written into bits 0 to 7 and the End of Frame (EOF) indicator is written into bit 8. The CPU reads from a register, which transfers the data bytes sequentially out of the FIFO, and stores the bytes for TCP/IP processing.

The Rx Regs module provides the register interface to the CPU. The CPU monitors bits in the control registers to discover when a new packet has been received and reads the received IP packet data bytes from another register. Receive status information is also stored in numerous registers in this module. The CPU reads the Data byte register to get the data contents of the received packet.

**Rx Regs Signal Descriptions:** These signals are the interface to the CPU and software and govern the hardware/software interaction. Signal descriptions for the TSMAC and FIFO can be found in their user manuals.

Table 2. Rx Regs Signal Descriptions

| Signal       | Type   | Description                                                                                            |
|--------------|--------|--------------------------------------------------------------------------------------------------------|
| Rx_Data      | Input  | TSMAC rx_dbout bits 7:0, even byte of packet word                                                      |
| EOF          | Input  | End of Frame marker, set for last byte of packet                                                       |
| Rd           | Output | Controls clocking next byte from FIFO                                                                  |
| FIFO_reset   | Output | Controls resetting the RX_FIFO. Allows software ability to clear FIFO during initialization.           |
| FIFO_empty   | Input  | Indicates when FIFO empty. Tells software no more RX_DATA reads are needed.                            |
| Pkt_Avail    | Input  | Indicates when a complete packet has been loaded. This is TSMAC rx_eof latched in for the CPU to read. |
| Rx_err       | Input  | Indicates when a packet error was detected. This is TSMAC rx_error latched in for the CPU to read.     |
| Rx_rcvd      | Input  | 16 bit counter of number of good packets received. Incremented by RX_IP logic.                         |
| Rx_drop      | Input  | 16 bit counter of number of packets dropped when FIFO was full. Incremented by RX_IP logic.            |
| Rx_CRC_err   | Input  | 16 bit counter of number of packets dropped because of rx_error signal. Incremented by RX_IP logic.    |
| Rx_Stats_Vec | Input  | 32 bit register holding last TSMAC rx_stat_vec. Latched in by rx_stat_en.                              |

Receive Registers

The TSMAC is located at address 0x8200 on the Wishbone bus.

**Table 3. Rx Interface Register Map**

| Offset    | Name         | Description                                                                              |
|-----------|--------------|------------------------------------------------------------------------------------------|
| 0         | Rx_Data      | Receive data byte                                                                        |
| 1         | Not used     |                                                                                          |
| 2         | Rx_Ctrl      | Bits control RX FIFO                                                                     |
| 3         | Rx_Status    | Bits monitor RX FIFO                                                                     |
| 4-5       | Rx_recvd     | Total number of good packets received                                                    |
| 6-7       | Rx_drop      | Total number of packets dropped because FIFO still full (CPU too slow)                   |
| 8-9       | Rx_CRC_err   | Total number of packets dropped because CRC check failed                                 |
| 0x0a-0x0d | Rx_Stats_Vec | 32-bit receive statistics vector from TSMAC core. Valid for current received packet only |

**Table 4. Register Name: Rx\_Data, Address: 0x8100**

| Bits | Name    | R/W | Description                                            |
|------|---------|-----|--------------------------------------------------------|
| 7:0  | Rx_Data | R   | Receive byte – Read to get next byte from receive FIFO |

**Table 5. Register Name: Rx\_Ctrl, Address: 0x8102**

| Bits | Name       | R/W | Description                                                                                                                                |
|------|------------|-----|--------------------------------------------------------------------------------------------------------------------------------------------|
| 7:4  | reserved   | R   | Reads back 0's                                                                                                                             |
| 2    | FIFO_reset | R/W | Write a 1 to force the FIFO to reset. Used for error recovery and driver initialization. Should not be written when TSMAC is storing bytes |
| 1:0  | reserved   | R   | Reads back 0's                                                                                                                             |

**Table 6. Register Name: Rx\_Stats, Address: 0x8103**

| Bits | Name         | R/W | Description                                                                                                   |
|------|--------------|-----|---------------------------------------------------------------------------------------------------------------|
| 7:6  | reserved     | R   | Reads back 0's                                                                                                |
| 5    | FIFO_full    | R   | Set when Rx FIFO is full, indicating FIFO overflow error condition                                            |
| 4    | FIFO_empty   | R   | Set when Rx FIFO empty, indicating CPU read last byte                                                         |
| 3    | MAC_fifo_err | R   | Set when the TSMAC detected an error in a received packet                                                     |
| 2    | Gig_en       | R   | Set if 1000baseT mode enabled. This should be always be 0                                                     |
| 1    | Rx_EOF       | R   | Read along with Rx_Data to see if more bytes are still in RX FIFO. 0= more bytes, 1 = last byte, end of frame |
| 0    | Pkt_Avail    | R   | Set when a new packet is available in the RX FIFO. Cleared when FIFO empty (CPU read every byte)              |

**Table 7. Register Name: Rx\_recvd\_hi, Address: 0x8104**

| Bits | Name        | R/W | Description                                               |
|------|-------------|-----|-----------------------------------------------------------|
| 7:0  | Rx_recvd_hi | R   | High byte of total good received packets (16-bit counter) |

**Table 8. Register Name: Rx\_recvd\_lo, Address: 0x8105**

| Bits | Name        | R/W | Description                                              |
|------|-------------|-----|----------------------------------------------------------|
| 7:0  | Rx_recvd_lo | R   | Low byte of total good received packets (16-bit counter) |

**Table 9. Register Name: Rx\_drop\_hi, Address: 0x8106**

| Bits | Name       | R/W | Description                                                         |
|------|------------|-----|---------------------------------------------------------------------|
| 7:0  | Rx_drop_hi | R   | High byte of total dropped packets (16-bit counter) cause FIFO busy |

**Table 10. Register Name: Rx\_drop\_lo, Address: 0x8107**

| Bits | Name       | R/W | Description                                                        |
|------|------------|-----|--------------------------------------------------------------------|
| 7:0  | Rx_drop_lo | R   | Low byte of total dropped packets (16-bit counter) cause FIFO busy |

**Table 11. Register Name: Rx\_CRC\_err\_hi, Address: 0x8108**

| Bits | Name          | R/W | Description                                                   |
|------|---------------|-----|---------------------------------------------------------------|
| 7:0  | Rx_CRC_err_hi | R   | High byte of total dropped CRC error packets (16-bit counter) |

**Table 12. Register Name: Rx\_CRC\_err\_lo, Address: 0x8109**

| Bits | Name          | R/W | Description                                                  |
|------|---------------|-----|--------------------------------------------------------------|
| 7:0  | Rx_CRC_err_lo | R   | Low byte of total dropped CRC error packets (16-bit counter) |

**Table 13. Register Name: rx\_stats\_vec[0], Address: 0x810a**

| Bits | Name            | R/W | Description                                                                                                                                     |
|------|-----------------|-----|-------------------------------------------------------------------------------------------------------------------------------------------------|
| 7    | Long Frame      | R   | <b>Long Frame.</b> This bit is set when a frame longer than specified in the MAX_FRAME_LENGTH register is received                              |
| 6    | Short frame     | R   | <b>Short Frame.</b> This bit is set when a frame shorter than the value specified in the MIN_FRAME_LENGTH register is received                  |
| 5    | IPG violation   | R   | <b>IPG Violation.</b> This bit is set when a frame is received before the IPG timer runs out                                                    |
| 4    | Preamble shrink | R   | <b>Preamble Shrink.</b> This bit is asserted if the number of Preamble bytes received is not equal to seven                                     |
| 3    | Prev. Carrier   | R   | <b>Carrier Event Previously Seen</b>                                                                                                            |
| 2    | Packet ignored  | R   | <b>Packet Ignored.</b> When set, this indicates the incoming packet is to be ignored                                                            |
| 1    | CRC error       | R   | <b>CRC Error.</b> This bit is set when a frame is received with an error in the CRC field                                                       |
| 0    | Length error    | R   | <b>Length Check Error.</b> This bit is set if the number of data bytes in the incoming frame matches the value in the length field of the frame |

**Table 14. Register Name: rx\_stats\_vec[1], Address: 0x810b**

| Bits | Name           | R/W | Description                                                                                                                                                                  |
|------|----------------|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 7    | Recv OK        | R   | <b>Receive OK.</b> This bit is set if the frame is received without any error                                                                                                |
| 6    | Multicast Addr | R   | <b>Multicast Address.</b> This bit is set to indicate the received frame contains a Multicast Address                                                                        |
| 5    | Broadcast Addr | R   | <b>Broadcast Address.</b> This bit is set to indicate the received frame contains a Broadcast Address                                                                        |
| 4    | Dribble nibble | R   | <b>Dribble Nibble.</b> This bit is set when only 4 bits of the data presented on the RS interface are valid                                                                  |
| 3    | Bad opcode     | R   | <b>Unsupported Opcode.</b> This bit is set if the received control frame has an unsupported opcode. In this version of the IP, only the opcode for PAUSE frame is supported. |
| 2    | Control Frame  | R   | <b>Control Frame.</b> This bit is set to indicate that a Control frame was received                                                                                          |
| 1    | PAUSE frame    | R   | <b>PAUSE Frame.</b> This bit is set when the received Control frame contains a valid PAUSE opcode                                                                            |
| 0    | VLAN Tag       | R   | <b>VLAN Tag Detected.</b> This bit is set when the Tri-Speed MAC receives a VLAN Tagged frame                                                                                |

**Table 15. Register Name: rx\_stats\_vec[2], Address: 0x810c**

| Bits | Name         | R/W | Description                   |
|------|--------------|-----|-------------------------------|
| 7:0  | ByteCount_hi | R   | High byte of frame byte count |

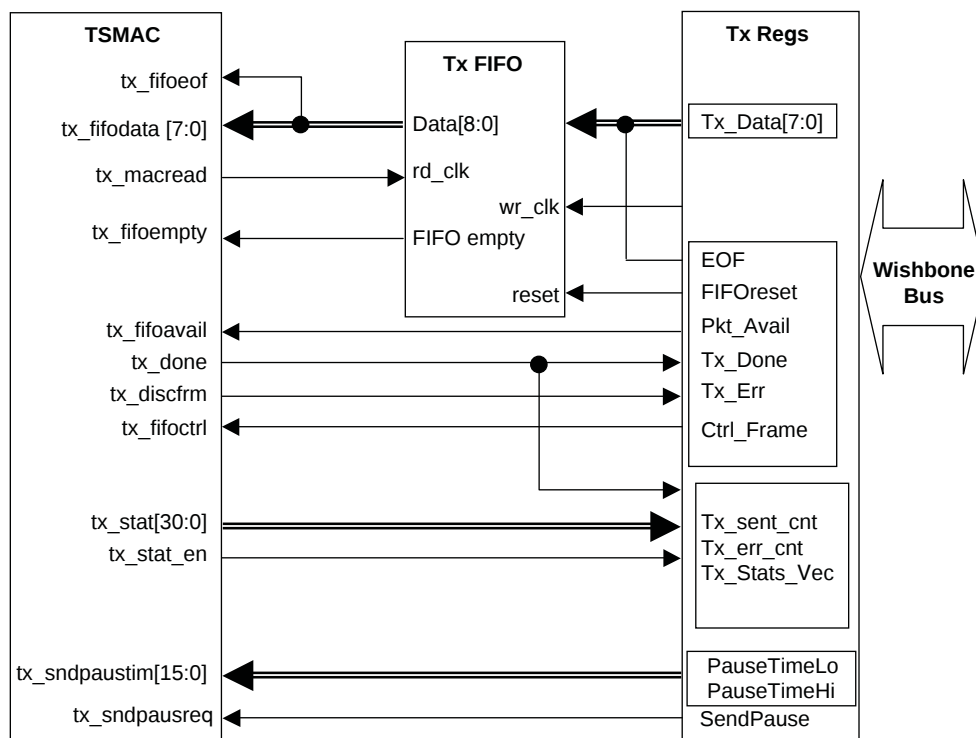
**Table 16. Register Name: rx\_stats\_vec[3], Address: 0x810d**

| Bits | Name         | R/W | Description                  |
|------|--------------|-----|------------------------------|
| 7:0  | ByteCount_lo | R   | Low byte of frame byte count |

Transmit Design

The following figure diagrams the components, data flow and control flow for transferring an IP packet, assembled by the CPU, into the TSMAC for transmit.

**Figure 15. Transmit Block Diagram**



**Module Descriptions:** The TSMAC module transmits an Ethernet frame by reading bytes from the Tx FIFO when signaled with the tx\_fifoavail signal. Only the transmit signals of the TSMAC are shown in the above figure. These signals provide information about the status of the packet being transmitted and control clocking bytes out of the FIFO for transmit. The TSMAC reads the bytes directly from a LatticeXP EBR Dual Clock FIFO library element.

The Tx FIFO module is a standard 9x1024 Dual Clock FIFO generated with IPexpress™ for a LatticeXP. Each 9-bit entry holds the byte to transmit (8 bits) and the tx\_fifoeof signal (1 bit). The tx\_fifoeof bit is used to indicate the last byte of a packet. It is 0 for all bytes except the last byte to transmit, in which case it is set to 1. This is done by the CPU when loading the FIFO.

The Tx Regs module provides the register interface to the CPU. The CPU writes the data bytes to be transmitted into TX\_Data and the EOF bit, which are then clocked into the FIFO. Transmit status information is also stored in other registers in this module.

**Tx Regs Signal Descriptions:** These signals are the interface between the software the Tx FIFO logic and govern the hardware/software interaction. Signal descriptions for the TSMAC and FIFO can be found in their respective user manuals.

**Table 17. Tx Regs Signal Descriptions**

| Signal                     | Type      | Description                                                                                                                                |
|----------------------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------|
| Tx_Data                    | Output    | TSMAC tx_fifo data bits 7:0                                                                                                                |
| EOF                        | Output    | End of Frame indicator paired with each byte. This bit is sent unchanged to the TSMAC with each corresponding data byte read from the FIFO |
| Wr                         | Output    | Controls clocking next word into the FIFO. Activated by writing Tx_Data. Values in Tx_Data and EOF are written into the FIFO               |
| FIFO_reset                 | Output    | Controls resetting the TX_FIFO. Allows software ability to clear FIFO during initialization                                                |
| Pkt_Avail                  | Output    | Indicates to logic a complete packet has been loaded into the FIFO. The TX_IP logic can then signal the TSMAC to start transmitting        |
| Tx_Done                    | Input     | Indicates packet has been sent. This FIFO empty signal. Once set the CPU can load new packet into FIFO                                     |
| Tx_Err                     | Input     | Indicates when a packet error was detected. This is TSMAC rx_error latched in for the CPU to read                                          |
| Ctrl_frame                 | Output    | Set by software to indicate frame type: 0=normal, 1= control (PAUSE)                                                                       |
| Tx_sent                    | Input     | 16-bit counter of number of good packets sent. Incremented by TX_IP logic. Based on status of TSMAC tx_done signal                         |
| Tx_err                     | Input     | 16-bit counter of number of packets error. Incremented by TX_IP logic. Based on status of TSMAC tx_discfrm signal                          |
| Tx_Stats_Vec               | Input     | 32-bit register holding last TSMAC tx_stat_vec. Latched in by tx_stat_en                                                                   |
| PauseTimeLo<br>PauseTimeHi | Output    | (2) 8-bit registers holding the PAUSE frame Timer value                                                                                    |
| SendPause                  | SendPause | Bit signaling TSMAC to send a PAUSE frame                                                                                                  |

*Note: The PAUSE signal is not used in demo operation. The core supports the ability to handle PAUSE frames, but the demo does not actively use them for flow control to keep things simple.*

#### Transmit Registers

The TSMAC is located at address 0x8200 on the Wishbone bus.

**Table 18. Tx Interface Register Map**

| Offset  | Name         | Description                                                                               |
|---------|--------------|-------------------------------------------------------------------------------------------|
| 0x10    | Tx_Data      | Transmit byte – Write to get byte into transmit FIFO                                      |
| 0x12    | Tx_Ctrl      | Bits to control TSMAC and TX FIFO                                                         |
| 0x13    | Tx_Stats     | Bits to monitor TSMAC and TX FIFO                                                         |
| 0x14-15 | Tx_sent      | Total number of packets transmitted without errors                                        |
| 0x16-17 | Tx_err       | Total number of packets dropped because of transmit errors, collisions                    |
| 0x18-1b | Tx_Stats_Vec | 32-bit transmit statistics vector from TSMAC core. Valid for last transmitted packet only |
| 0x1c-1d | PAUSE timer  | 16-bit value to send in a PAUSE control frame                                             |

**Table 19. Register Name: Tx\_Data, Address: 0x8110**

| Bits | Name    | R/W | Description                         |
|------|---------|-----|-------------------------------------|
| 7:0  | Tx_Data | W   | Byte to load into FIFO for transmit |

**Table 20. Register Name: Tx\_Ctrl, Address: 0x8112**

| Bits | Name       | R/W | Description                                                                                                                                 |
|------|------------|-----|---------------------------------------------------------------------------------------------------------------------------------------------|
| 7:4  | reserved   |     | Write as 0's, reads back what was written (0 at power-up)                                                                                   |
| 4    | SendPause  | R/W | 1=send pause frame, 0=normal operation                                                                                                      |
| 3    | Ctrl_frame | R/W | 1=control frame, 0=normal IP frame (default)                                                                                                |
| 2    | FIFO_reset | R/W | Write a 1 to force the FIFO to reset. Used for error recovery and driver initialization. Should not be written when TSMAC is storing bytes. |
| 1    | EOF        | R/W | 0 = more bytes to follow, 1 = last byte - End Of Frame                                                                                      |
| 0    | Pkt_Avail  | R/W | Set when a new packet is available in the TX FIFO. Start TSMAC transmitter when set to 1                                                    |

**Table 21. Register Name: Tx\_Stats, Address: 0x8113**

| Bits | Name         | R/W | Description                                                 |
|------|--------------|-----|-------------------------------------------------------------|
| 7:4  | reserved     |     | Write as 0's, reads back what was written (0 at power-up)   |
| 3    | TxFIFO_empty | R   | Set when TX FIFO is empty                                   |
| 2    | TxFIFO_Full  | R   | Set when TX FIFO has no more room                           |
| 1    | Tx_err       | R   | Set when the TSMAC detected an error in a transmitted frame |
| 0    | Tx_done      | R   | Set when TSMAC done transmitting the frame                  |

**Table 22. Register Name: Tx\_sent\_hi, Address: 0x8114**

| Bits | Name       | R/W | Description                                                  |
|------|------------|-----|--------------------------------------------------------------|
| 7:0  | Tx_sent_hi | R   | High byte of total good transmitted packets (16-bit counter) |

**Table 23. Register Name: Tx\_sent\_lo, Address: 0x8115**

| Bits | Name       | R/W | Description                                                 |
|------|------------|-----|-------------------------------------------------------------|
| 7:0  | Tx_sent_lo | R   | Low byte of total good transmitted packets (16-bit counter) |

**Table 24. Register Name: Tx\_err\_hi, Address: 0x8116**

| Bits | Name      | R/W | Description                                                    |
|------|-----------|-----|----------------------------------------------------------------|
| 7:0  | Tx_err_hi | R   | High byte of total dropped packets (16-bit counter) collisions |

**Table 25. Register Name: Tx\_err\_lo, Address: 0x8117**

| Bits | Name      | R/W | Description                                                   |
|------|-----------|-----|---------------------------------------------------------------|
| 7:0  | Tx_err_lo | R   | Low byte of total dropped packets (16-bit counter) collisions |

**Table 26. Register Name: tx\_stats\_vec[0] (bits 31-24), Address: 0x8118**

| Bits | Name                | R/W | Description                                                  |
|------|---------------------|-----|--------------------------------------------------------------|
| 7    | reserved            | R   | Reads as 0                                                   |
| 6    | Short Frame         | R   | FCS generation is disabled and a short frame was transmitted |
| 5:2  | Collision Count     | R   | 4-bit count of number of early collisions                    |
| 1    | Excessive Collision | R   | Excessive collision                                          |
| 0    | Late collision      | R   | Late collision                                               |

**Table 27. Register Name: `tx_stats_vec[1]` (bits 23-16), Address: `0x8119`**

| Bits | Name               | R/W | Description                                 |
|------|--------------------|-----|---------------------------------------------|
| 7    | Excessive Deferred | R   | Excessive Deferred transmission             |
| 6    | Deferred           | R   | Deferred transmission                       |
| 5:0  | ByteCount_hi       | R   | Upper six bits of transmit frame byte count |

**Table 28. Register Name: `tx_stats_vec[2]` (bits 15-8), Address: `0x811a`**

| Bits | Name         | R/W | Description                                   |
|------|--------------|-----|-----------------------------------------------|
| 7:0  | ByteCount_lo | R   | Lower eight bits of transmit frame byte count |

**Table 29. Register Name: `tx_stats_vec[3]` (bits 7-0), Address: `0x811b`**

| Bits | Name          | R/W | Description                                          |
|------|---------------|-----|------------------------------------------------------|
| 7    | VLAN tagged   | R   | VLAN tagged frame transmitted                        |
| 6    | PAUSE frame   | R   | PAUSE frame transmitted                              |
| 5    | FIFO underrun | R   | FIFO ran empty during transmission, incomplete frame |
| 4    | JUMBO frame   | R   | Jumbo frame transmitted                              |
| 3    | Bad FCS       | R   | Bad FCS detected during transmission                 |
| 2    | BroadCast     | R   | Broadcast frame transmitted                          |
| 1    | MultiCast     | R   | Multicast frame transmitted                          |
| 0    | UniCast       | R   | UniCast frame transmitted                            |

**Table 30. Register Name: `PAUSE_Time_hi`, Address: `0x811c`**

| Bits | Name     | R/W | Description                              |
|------|----------|-----|------------------------------------------|
| 7:0  | PAUSE_hi | R   | High byte of PAUSE time register setting |

**Table 31. Register Name: `PAUSE_Time_lo`, Address: `0x811d`**

| Bits | Name     | R/W | Description                             |
|------|----------|-----|-----------------------------------------|
| 7:0  | PAUSE_lo | R   | Low byte of PAUSE time register setting |

### CPU IP

An embedded 8-bit CPU IP core is used to execute the software TCP/IP stack and networking applications. The core is a Verilog synthesizable 8-bit Z80 microprocessor. This core, the tv80, was obtained from [www.OpenCores.org](http://www.OpenCores.org). The following quote describes its functionality:

The tv80 core was created as a Verilog port of the VHDL T80 core, Wallner, D., VHDL T80 Core, .[1], for use as a maintenance processor inside an ASIC. The tv80 has been modified since then for better synthesis timing/area results, and to incorporate several bug-fixes.

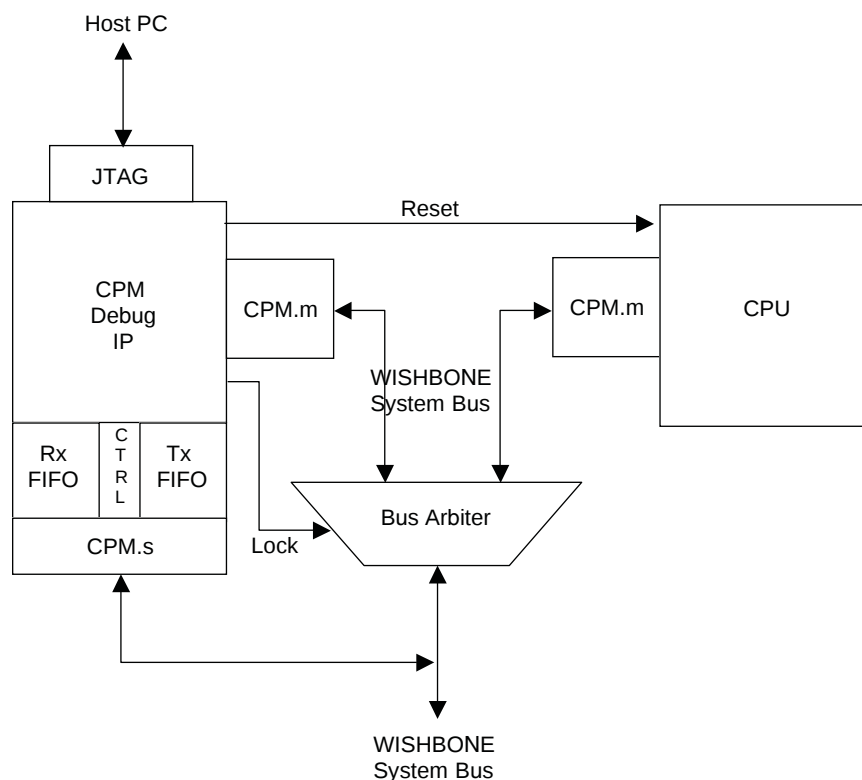
The T80, and the tv80 derived from it, attempt to maintain the original cycle timings of the Z80, but have radically different internal designs and timings. With its target being ASIC and embedded applications, the tv80 does not attempt to maintain the original pinout of the Z80.

### Memory IP

The CPU uses the EBR memories for program instructions, data storage and stack. The memories start at address 0 on the Wishbone system bus, which corresponds to the CPU's reset vector. The memory is constructed using IPexpress to instantiate a standard 20k x 8 bit array of SRAM.

**CPM Debug IP (JTAG)**

The CPM Debug IP module provides a method for performing bus operations on the Wishbone bus via a CPM master other than the CPU. The CPM Debug IP allows an external host (PC running an application connected to the evaluation board via ispVM System software), to read/write memory on the Wishbone bus. This allows a certain level of CPU and system debugging to be performed, by halting the CPU and inspecting memory contents. The program memory can also be loaded by writing directly into the EBR memories. The CPM Debug module also has registers that appear on the Wishbone to allow the CPU to write data back to, and read data from, the host PC. This allows the CPM Debug to act as a “stdio” conduit to allow program output (simple prints) to be directed to the PC where they can be displayed. The CPM Debug IP shares the Wishbone bus through the bus arbiter. The CPM Debug can lock the bus for itself, essentially freezing the CPU.

**Figure 16. Debug with CPM Debug IP****Description of Operations**

The CPM Debug IP receives commands from application software over the JTAG interface. The command is loaded into a user scan ring in JTAG and decodes to either a read or write operation. The target of the operation is an address on the WISHBONE system or an internal register. For this application, external addresses are from 0x0000 to 0xffff, and internal registers start at 0x10000 (bit A16 set).

An external address access performs a bus operation using the CPM Master interface. This is a normal bus operation (read or write), transferring data to/from the CPM slave device. The results should be exactly the same as when the CPU performs the operation to the CPM Slave device.

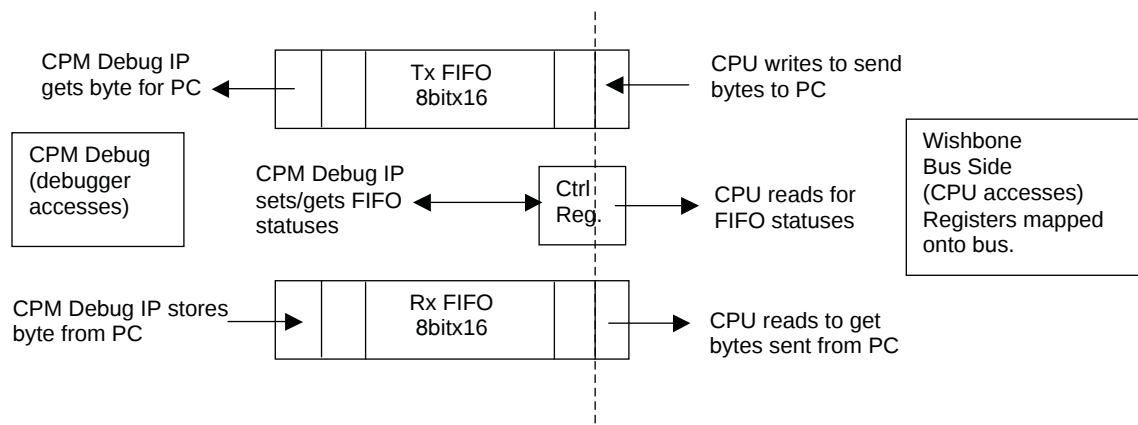
An internal address accesses registers inside the CPM Debug IP block. These registers are used for exchanging message data between the external application and the CPU. These registers also provide control of the CPU, by forcing it into reset, and the ability to lock the bus so CPM Debug has exclusive access.

Displaying Messages

Another use of the CPM Debug is to provide a stdio port for the program executing on the CPU. The CPM Debug IP has transmit and receive FIFOs that are accessible to a CPU on the Wishbone bus. The CPM Debug can be viewed as a UART. The following steps explain how this works.

1. The CPU wants to send a character string message to a “console” application on the PC attached to the ispVM JTAG module. It prepares to copy the characters one at a time into the transmit FIFO
2. The CPU reads the CPM Debug control register to see if the FIFO is full. The CPU writes a byte into the TX\_FIFO register and repeats, checking the fill status before each write. If the FIFO is full, the CPU waits before writing in any more bytes into the FIFO (driver's choice to drop characters or block program)
3. The console application on the PC continuously sends JTAG commands to read the TX\_FIFO status. When it detects it is no longer empty, it sends commands to read out bytes in the TX\_FIFO and display them. Reading of the TX\_FIFO stops whenever the FIFO is empty
4. Reversing the roles of the CPU and host application would allow data to be sent to the program running on the CPU

**Figure 17. CPM Debug IP as STDIO Port**

Registers and Addresses

**Table 32. CPM Debug Module Register Map**

| Address         | Name       | Description                                                                                           |
|-----------------|------------|-------------------------------------------------------------------------------------------------------|
| 0xffff - 0x0000 | Bus Access | Perform the Wishbone bus operation (read/write) to the target address. Bus access via CPM.master port |
| 0x10000         | Status     | FIFO status register                                                                                  |
| 0x10001         | Tx_FIFO    | Writes from CPU side, reads from JTAG side                                                            |
| 0x10002         | Rx_FIFO    | Reads from CPU side, writes from JTAG side                                                            |
| 0x10003         | Ctrl       | Control register                                                                                      |
| 0x10004         | Counts     | Tx/Rx FIFO level count register: Rx[7-4]; Tx[3-0]                                                     |

**Table 33. Register Name: Status, Address: 0x10000 (JTAG); 0x8400 (WISHBONE)**

| Bits | Name          | CPU R/W | Description                                                                                                                |
|------|---------------|---------|----------------------------------------------------------------------------------------------------------------------------|
| 7:4  | reserved      | R       | Reads 0's                                                                                                                  |
| 3    | RX_FIFO_FULL  | R       | Set when RX FIFO is full and cannot accept another byte from JTAG. CPU needs to read bytes out                             |
| 2    | RX_FIFO_EMPTY | R       | Set when RX FIFO has no bytes in it. First byte written in from JTAG clears this. CPU reads this to see if bytes available |
| 1    | TX_FIFO_FULL  | R       | Set when TX FIFO is full and cannot accept another byte from bus                                                           |
| 0    | TX_FIFO_EMPTY | R       | Set when TX FIFO has no bytes in it. First byte written in clears it                                                       |

**Table 34. Register Name: Tx\_FIFO, Address: 0x10001 (JTAG); 0x8401 (WISHBONE)**

| Bits | Name    | CPU R/W | Description                                                                                 |
|------|---------|---------|---------------------------------------------------------------------------------------------|
| 7:0  | Tx_FIFO | W       | CPU writes to send byte to host PC, CPM Debug reads to get byte destined for PC application |

**Table 35. Register Name: Rx\_FIFO, Address: 0x10002 (JTAG); 0x8402 (WISHBONE)**

| Bits | Name    | CPU R/W | Description                                                                             |
|------|---------|---------|-----------------------------------------------------------------------------------------|
| 7:0  | Rx_FIFO | R       | CPU reads to get a byte sent from PC, CPM Debug writes in to send bytes to CPU software |

**Table 36. Register Name: Counts, Address: 0x10003 (JTAG); 0x8403 (WISHBONE)**

| Bits | Name        | CPU R/W | Description                                                                                                              |
|------|-------------|---------|--------------------------------------------------------------------------------------------------------------------------|
| 7:4  | Rx_FIFO_CNT | R       | 4-bit count (0-15) of number of bytes in FIFO: 0 = none (if RX_FIFO_EMPTY bit is set) or 16 (if RX_FIFO_FULL bit is set) |
| 3:0  | Tx_FIFO_CNT | R       | 4-bit count (0-15) of number of bytes in FIFO: 0 = none (if TX_FIFO_EMPTY bit is set) or 16 (if TX_FIFO_FULL bit is set) |

**Table 37. Register Name: Ctrl, Address: 0x10004 (JTAG); (not visible from WISHBONE)**

| Bits | Name      | CPU R/W | Description                                                                                |
|------|-----------|---------|--------------------------------------------------------------------------------------------|
| 7:2  | reserved  | R       | Reads 0's                                                                                  |
| 1    | CPU reset | R/W     | When set, forces CPU into RESET. When clear, CPU runs                                      |
| 0    | Bus Lock  | R/W     | When set, CPM Debug owns bus and CPU busreq is asserted. When clear, CPU runs and uses bus |

**GPIO Register IP**

The GPIO registers are memory mapped addressable by the CPU. They start at address 0x8000 on the Wishbone bus. Software writes to these registers to blink LEDs to report status such as packet Tx/Rx, error packet, packet count, etc. DIP switch settings (not currently used) could be read to configure demo operation.

**Table 38. GPIO Register Map**

| Address | Description                |
|---------|----------------------------|
| 0x8000  | LED register               |
| 0x8001  | 7-segment display register |
| 0x8002  | DIP switch input           |

### 7-Segment LED Usage

The 7-segment display counts the number of received packets. The decimal point indicates an error condition has been detected with the MAC or the transmit/receive FIFOs.

### Discrete LEDs

- D1 = heart beat
- D2 = packet received
- D3 = packet transmit
- D4 = ARP request received
- D5 = finger request received
- D6 = http request received
- D7 = PHY activity LED - direct signal from PHY
- D8 = PHY 100Mbit Link good - direct signal from PHY

### DIP Switch Usage

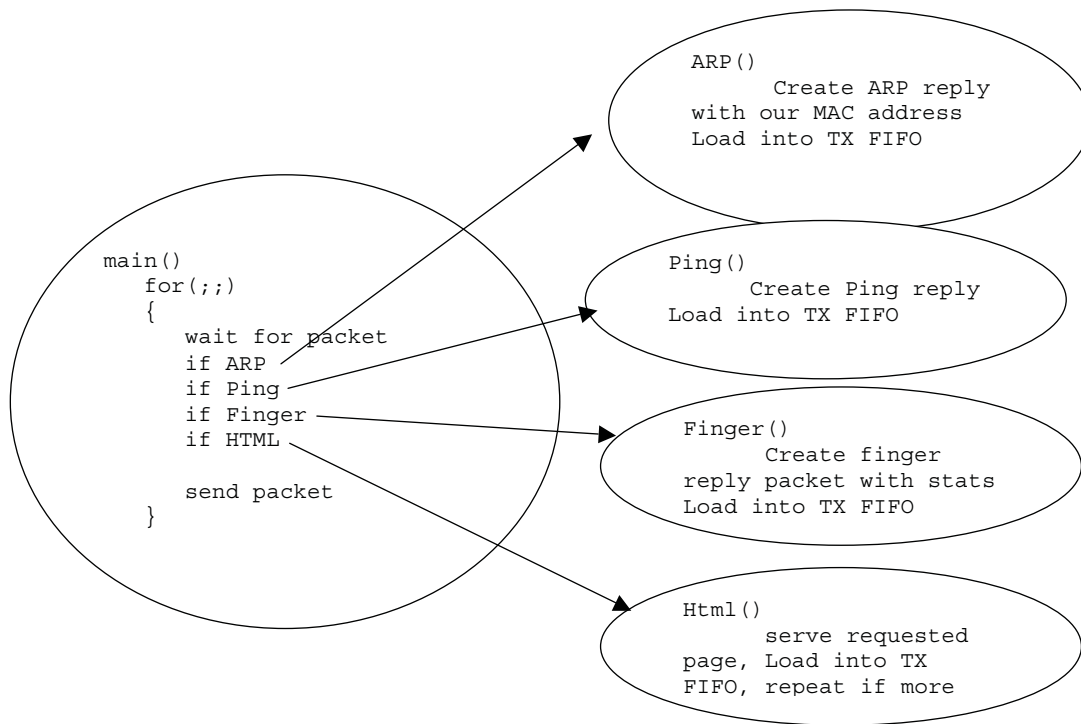
Not used.

## **Software**

The software to handle TCP/IP packets and serve the web pages runs on the embedded CPU inside the LatticeXP. The entire CPU, program memory, data and peripherals are all inside the LatticeXP. The software does the following tasks:

- On power-up/resets, initializes all devices
- Waits for an Ethernet packet
- Interrogates the packet
- Creates an ARP, ping, finger or web page response
- Sends the packet back to the client computer

Figure 18 shows the flow of the software. It is one forever loop that waits for a packet to come in and then takes action based on the packet protocol. The software never initiates a connection to another host system. It is always operating as a passive server, serving requests of the client (PC running the demo).

**Figure 18. Software Flow****Main Control Loop**

The entire control application exists in one forever loop. This is the standard approach to small control systems. Interrupts are not used, nor is an OS or executive. The program simply looks for a packet to come in, waits 0.5 seconds, then calls some time dependent routines in uIP and updates any LEDs and statuses. If a packet comes in it is transferred to the uIP packet buffer and control is given to uIP to process it. If uIP, or the application (finger or httpd) creates a packet to transmit back, uIP informs the control loop and the control loop calls the driver routine to transfer the packet to the MAC, and then returns to polling for a new packet.

**TSMAC Driver**

The software uses a driver to configure and control the TSMAC IP and the external PHY device via the host interface registers. The driver also handles the FIFO interface to send and receive Ethernet packets.

Configuration

- Initialize MAC
  - MAC Address
  - Mode
- Initialize PHY
  - Mode
- Clear Tx/Rx FIFOs

Receive Packet

- Poll pkt\_avail bit
- If set, read RxData register, save byte into uIP packet buffer
- Repeat while FIFO is not empty and not EOF
- When EOF or FIFO empty return packet length
- Clear pkt\_avail to notify IP that the FIFO is clear and ready for storage of a new arriving packet

### Transmit Packet

- uIP calls the net\_drvr to send bytes in the uIP packet buffer
- If tx\_done is not set then loop waiting for TX\_FIFO to be empty
- Set EOF bit to 0
- Copy byte[n] into TxData register, repeat until last byte
- At last byte set EOF to 1
- Write last byte into TxData, which clocks in to FIFO
- Set pkt\_avail to notify IP that the FIFO has a packet to transmit

### **TCP/IP Software Stack**

The software package uIP implements the IP, ICMP and TCP layers of the network stack. uIP is an open source light-weight TCP/IP stack specially designed for embedded 8-bit CPU systems. It has the following features:

- IP and TCP checksums
- IP fragment reassembly
- Multiple TCP connections
- TCP options
- Variable tCP MSS
- RTT estimation
- TCP flow control
- TCP urget data

The uIP package also comes with the following networking applications:

- ARP - reply to request
- ICMP - (echo == ping)
- HTML server

Information on uIP can be found at: [www.sics.se/~adam/uip/](http://www.sics.se/~adam/uip/)

## **Development Environment**

This section discusses how to build the complete demo system. This section is of value to applications engineers, field application engineers and users who want to see the steps involved in running the Lattice tools (ispLEVER, IPexpress™, etc.) to create a working system. This is an example design that shows one possible method of implementing the TSMAC IP core in a working system. It is not intended to be a production-ready, complete system design.

### **Hardware Components**

This section discusses the resources and procedures required to rebuild the design bitstream from the Verilog sources and IP.

### **Development Tools**

- ispLEVER – Synthesize, simulate IP cores and generate bitstream
- ispVM – Download bitstream into LatticeXP device

It is assumed that ispLEVER has been installed at the default location: C:>ispTOOLS5\_1.

---

### TSMAC Demo Design Files

The Verilog source files and ispLEVER project files are located in TSMAC\_XP\_Z80/Hdw/ directory. The source code for the custom designed modules that implement the demo hardware are located under Src/ directory. The source code is automatically compiled into the project. The tsmac\_xp\_hdw ispLEVER project configuration is in the tsmac\_xp\_hdw.syn file.

### TSMAC IP Core

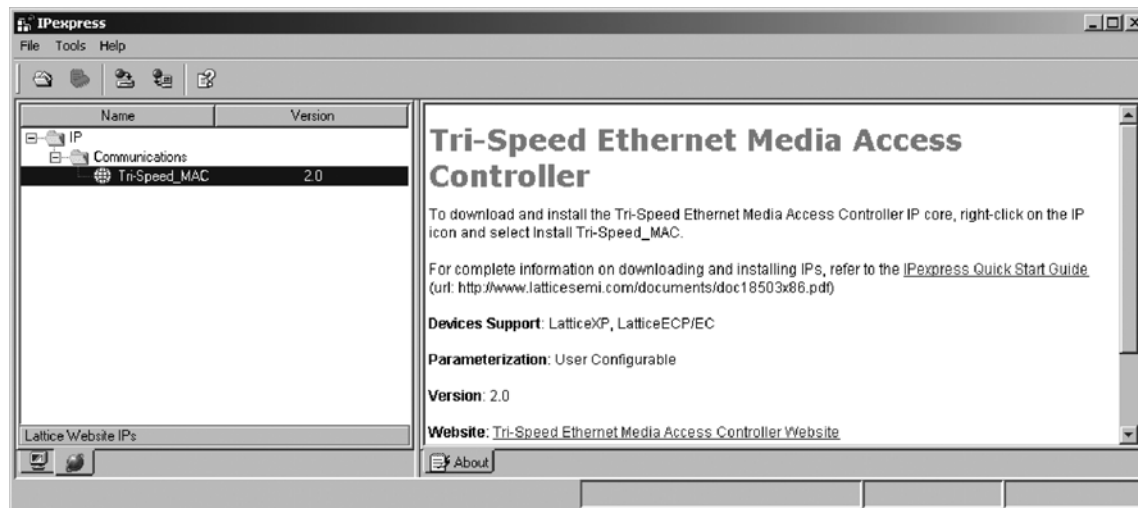
A demo version of the TSMAC core is included in the ispLEVER directory of the project. It will build with all the other RTL source files of the project to create the working bitstream. For demo purposes, there is no need to obtain the complete TSMAC IP core package from Lattice.

The TSMAC core can be obtained using the IPexpress flow in the ispLEVER tool. IPexpress downloads the TSMAC core from the Lattice web site. This is an evaluation version of the core that allows synthesis and operation in LatticeXP hardware for a demo period of one hour. A license is required for unrestricted use.

To download the TSMAC core, run the IPexpress tool from ispLEVER. Click on the **Globe** tab at the bottom of the left pane. This will launch a connection to Lattice IPs server. You may need to configure your Internet connection options. See **Tools -> Preferences...** if you need to use a proxy to gain access to the Internet.

For complete information on downloading and installing IPs, refer to the *IPexpress Quick Start Guide* (url: [www.latticesemi.com/documents/doc18503x86.pdf](http://www.latticesemi.com/documents/doc18503x86.pdf)).

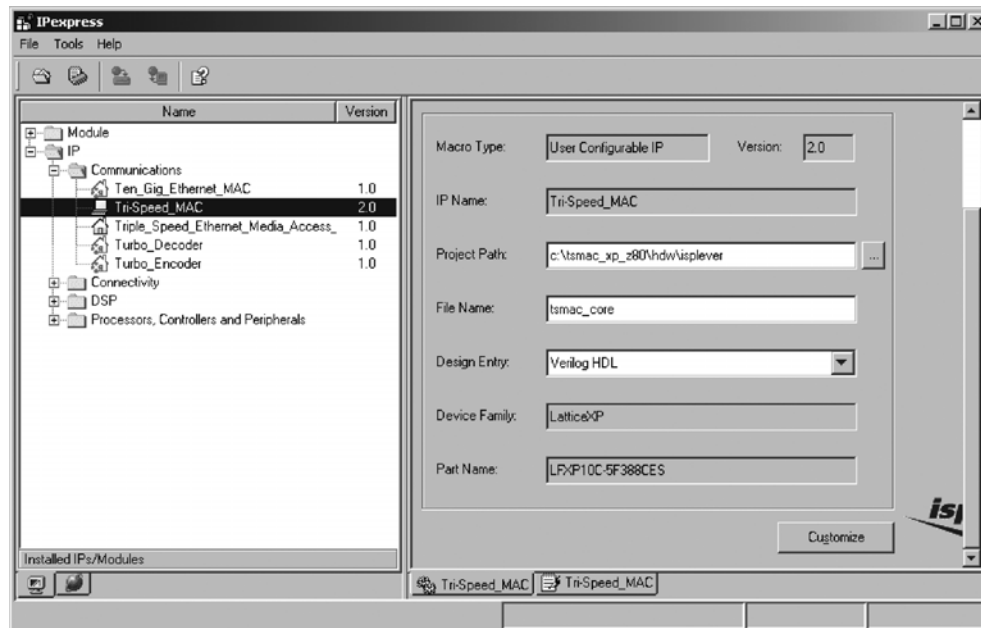
**Figure 19. Lattice IP Server Tab for IPexpress User Configurable IP Cores**



Highlight the TriSpeed MAC core, right-click and choose to download. It will save an exe file onto your PC that you later run to install the core software into ispLEVER. After downloading the exe file, close ispLEVER and run the exe file to install the TSMAC core.

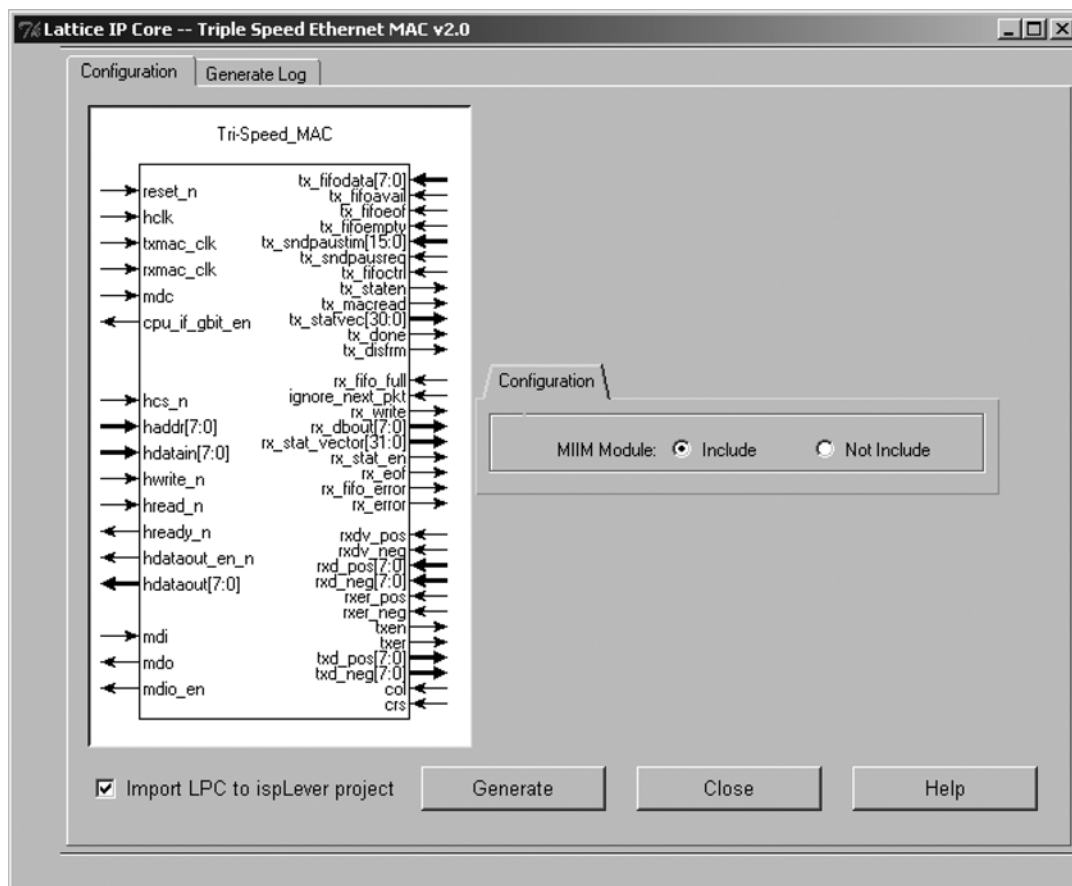
After installing the TSMAC IP package, use ispLEVER and IPexpress again to generate a TSMAC module from the installed IP.

Figure 20. Installed TSMAC IP Core in IPexpress



Name the core tarmac\_core because this is the name expected in the TSMAC\_XP\_Z80 Demo project. Press the **Customize** button and the following screen will appear.

Figure 21. TSMAC IP Parameters GUI



Make sure the MIIM module is included so that the mdio management signals are provided to access and configure the PHY. Press the **Generate** button and Synplify® will be used to synthesize the IP core into a NGO file that will be pulled into the demo project during place and route. A black-box Verilog file is also generated for inclusion in the project.

A ts\_mac\_eval directory will be created in the ispLEVER project directory. This directory contains all the TSMAC resources. A readme file explains the directory contents and how to run the simulation test bench that is included under the ts\_mac\_eval/testbench/ directory. For the purposes of this demo, no files need to be touched in this directory tree.

The IPexpress tool creates the black-box Verilog file tsmac\_core\_bb.v which contains the port list of the TSMAC module for connection with other project modules. The demo project uses this file for the IP module interface definitions. The actual netlist of the IP is in the file tsmac\_core.ngo and will be combined with the project during map and PAR.

### Synthesizing the Demo Design

A number of warnings are generated when synthesizing. These warnings can be ignored. They fall into several categories:

- Synthesis directives – The tool does not ignore certain options, targeted for other synthesis tools, in the IPexpress generated modules
- Unconnected signals
  - Wishbone bus – Not using interrupts or all of the select signal lines
  - Z80 signals – Certain interface signals are not used (interrupts, memory refresh, etc.)
  - JTAG signals – Not all are needed
- CPU implementation warnings – These are in the Z80 CPU core, and it works despite them
- Carry signals not connected – Not used on most counters in the design

### Simulating the Demo Design

No test bench is available yet.

## Software Components

This section discusses the resources and procedures required to rebuild the demo application from the C source files.

### Development Tools

- uIP – Freeware, embedded TCP/IP stack and web server
- z88dk – Freeware Z80 C compiler, assembler, executable generator - binary code proven to run on CPU in Verilog simulator and emulator
- bin2ebr – Utility to convert the application binary program into a format that can be loaded into EBRs

Need to have install location of z88dk in your path to build Z80 application source code.

```
C:> set PATH=%PATH%;c:\z88dk\bin
```

### TCP/IP Stack and Web Page Application

This section describes how to compile the source code that runs on the embedded Z80 CPU into a binary executable. Include this executable in the bitstream and then download the bitstream into the LatticeXP. The source code for the drivers, TCP/IP stack and demo application can be found under the Src/uIP\_Z80/ directory.

The first step is compiling the source code. The source code consists of the device drivers for the TSMAC, the TCP/IP stack, the applications including the web pages. The source is built from the Src/uIP\_Z80/z80/ directory with makeweb.bat. Building uses the Z80 z88dk tool set to compile all the code into a binary file in one step.

```
C:> makeweb.bat
```

The resulting files are demo.bin and demo.map. The batch file converts demo.bin into an EBR friendly format and stores the resulting prog.mem file in the Hdw/ispLEVER/ directory. The EBRs are then initialized using this file.

Pay careful attention to the output of the compiler/assembler. It is not always easy to see warning and error messages in the output. The compiler uses intermediate file names for the source it is compiling. The batch file will not stop on an error. You must manually verify that all source files have compiled correctly.

There are two ways to initialize the EBRs with the program contents. One method uses IPexpress; the other uses ispLEVER command line utilities. The IPexpress method requires re-opening the ebr\_mem module in IPexpress and re-generating the resulting Verilog file. This populates the EBR contents with the new prog.mem file, but requires re-synthesizing, mapping, placing, routing and creating a bitstream.

The utility method operates only on the post-PAR .ncd file and replaces the initialization contents of the EBRs with the prog.mem contents and then manually runs the bitgen tool to produce a bitstream. This is the preferred method since it does not change the design. You must install ispLEVER at c:\ispTOOLS5\_1 or edit the setup.bat file to point to where your ispLEVER tools are installed. This method requires an existing .ncd file. If it is not present, you must start at the beginning of the flow.

```
C:> add_prog.bat
```

```
C:> isp_bitgen
```

### Web Pages

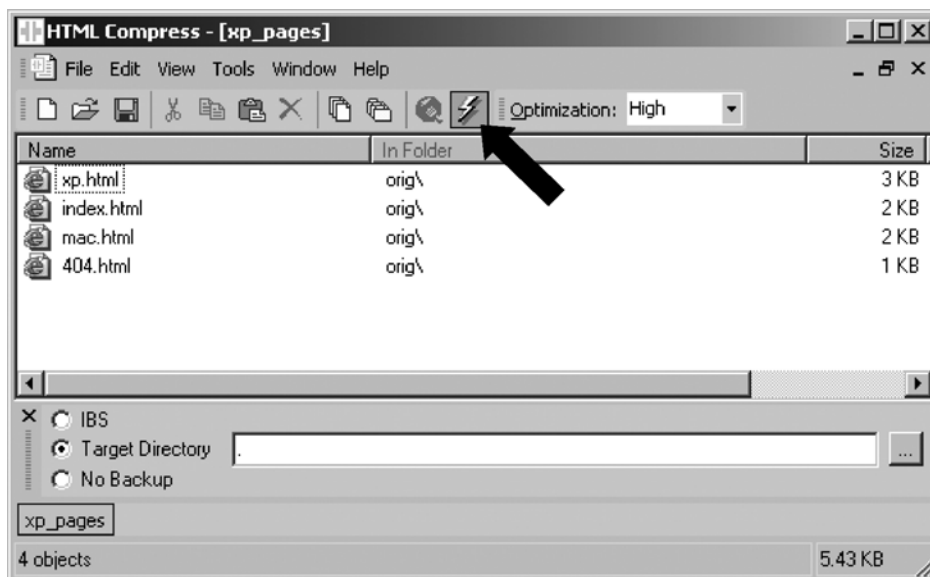
This section describes how to create/edit the HTML pages and embed them into the source code. The pages are edited, then compressed to remove extraneous white space and codes, and then compiled into the binary executable. The following tools are required to build and include the web pages:

- HTML Compress utility
- C compiler such as Dev-Cpp or Cygwin gcc

The first step is to edit the actual HTML files. These are located in the Src/uIP\_Z80/apps/httpd/pages/orig/ directory. There are four files that are included into the source. Index.html is the opening page. Mac.html describes the TSMAC core and xp.html provides information on the LatticeXP FPGA. 404.html is an error page sent by the server when an unknown request is received. When editing these files, pay careful attention to size. Hand editing the HTML files is recommended. Do not include images or other content.

After editing, the files are compressed using the HTML Compress utility. The project file is located in Src/uIP\_Z80/apps/httpd/pages/. Double-click on xp\_pages.ls5 to start the compress utility.

Figure 22. HTML Compress Utility Window



Click on the yellow lightning bolt to compress all the selected HTML files in the orig/ directory. The resulting compressed HTML files are in this directory (Src/uIP\_Z80/apps/httpd/pages/). The original files are not modified.

Check that the files were compressed correctly by opening them in a web browser (double click on index.html and it should be viewed normally, links to the other pages should work and they should be viewable.)

The next step is to convert the four HTML files into a pseudo file-system that can be processed by the uIP httpd server. The text pages are converted into Z80 assembly and arranged as data bytes in memory. A file system directory entry is created that references the start of each file based on the file name. A custom tool generates the file-system in this format so it can be compiled with the Z80 z88dk tool set. The tool is makepages.exe and a copy exists in the Src/uIP\_Z80/apps/httpd/pages/ directory. The source for the tool is located in Src/uIP\_Z80/apps/httpd/page\_tools/. If the pre-built executable does not run on your Windows system, rebuild the tool from the source in page\_tools/.

The makepages.exe utility creates the files fsdata.c and dir\_tbl.h. fsdata.c contains all the web page data and the file system structure. This file is assembled into binary data and embedded into the demo executable. The dir\_tbl.h contains a definition of the file entries in the file system directory structure. Both of these need to be moved up on directory level for inclusion during compiling.

Run the tool:

```
C:> makepages files.txt
C:> copy fsdata.c ..
C:> copy dir_tbl.h ..
```

## Troubleshooting

### Running the Demo

**The board/demo does nothing:**

- Make sure the software heartbeat (LED1) is blinking
- Reload the bitstream or run ispLoad with a known working design (i.e. the demo bitstream)
- Press the PROGRAM button, the eval timer may have expired and locked up the board

**Cannot ping the board:**

- Make sure LED8 is on, indicating a good connection
- Look for LED7 and the yellow LED on the RJ-45 jack to blink, indicating network activity (ping packets being sent from the PC)
- Look for the counter LED to increment, indicating ping packets are being received by the software
- Make sure the network settings are correct for the board on the network and that the IP address is valid

**Web browser will not display pages:**

- Ping the board and look for network activity
- Make sure the proxy (if used) is disabled on the browser
- Extreme network activity resulting in demo software losing packets, use direct connect to the board

**Problems changing the IP address:**

- Make sure setup.bat was run to configure access to the utilities and ispVM and ispLEVER
- Make sure ispVM and a download cable are installed
- Make sure ispLEVER is installed in order to create a new bitstream

**Test Points**

The following signals are brought out to the bank of headers next to the 7-segment LED. They provide a means to monitor real-time activity of the TSMAC core with the Tx/Rx FIFOs and the PHY.

**Table 39. Signals for Real-Time Monitoring**

| TP | Pin | Signal           |
|----|-----|------------------|
| 1  | A2  | txmac_clk_src    |
| 2  | A3  | mac_tx_fifoavail |
| 3  | A4  | mac_tx_fifoeof   |
| 4  | B3  | mac_tx_macread   |
| 5  | B4  | mac_tx_done      |
| 6  | C3  | mac_tx_discfrm   |
| 7  | C4  | mac_tx_fifoempty |
| 8  | B12 | tx_en            |
| 9  | C5  | rxmac_clk_src    |
| 10 | D3  | mac_rx_write     |
| 11 | D8  | mac_rx_eof       |
| 12 | D9  | mac_rx_error     |
| 13 | A12 | mac_ignore_pkt   |
| 14 | A13 | col              |
| 15 | A18 | crs              |
| 16 | B13 | tx_er            |
| 17 | R2  | rx_dv            |
| 18 | T2  | rx_er            |

## Third-Party Software

Table 40 lists the third-party software tools, source code, packages, etc. that are used in the TSMAC XP demo. These packages are all freely available to the general public from various web sites on the Internet. Notices or claims restricting their use on the TSMAC demo project or prohibit distributing them as part of the demo package were not found.

**Table 40. Third Party Software Tools**

| Software          | Source/Author                   | License                    | Description                                                                  |
|-------------------|---------------------------------|----------------------------|------------------------------------------------------------------------------|
| UIP               | Adam Dunkles                    | Open Source BSD            | C source code for TCP/IP stack and networking protocols                      |
| tv80              | Guy Hutchison, OpenCores.org    | GPL                        | Verilog Z80 CPU IP                                                           |
| as80              | Frank A. Vorstenbosch           | Freeware                   | Z80 assembler                                                                |
| dasmz80           | ???                             | Freeware, legacy           | Z80 disassembler                                                             |
| Hex2v             | Thomas Coonan                   | GPL                        | Utility to create verilog mem file for simulations                           |
| z80em             | Gunter Woigk                    | MIT                        | Z80 CPU emulator core source files                                           |
| z88dk             | Dominic Morris, SourceForge.net | Clarified Artistic License | Z80 C development environment: compiler, assembler, linker, libraries, etc.  |
| HTML Compress 5.5 | FreeSoft                        | Freeware                   | HTML Compress compresses HTML by removing unnecessary white space characters |
| Dev-Cpp           | BloodshedSoftware.net           | Freeware                   | C++ compiler/IDE for Windows                                                 |

## Reference Information

The following documents provide more information on topics discussed throughout this guide:

- LatticeXP Data Sheet
- Tri-Speed Ethernet Media Access Controller User's Guide
- National Semiconductor DP83865 Gig PHYTER Data Sheet
- LatticeXP Advanced Evaluation Board

## Technical Support Assistance

Hotline: 1-800-LATTICE (North America)  
+1-503-268-8001 (Outside North America)

e-mail: [techsupport@latticesemi.com](mailto:techsupport@latticesemi.com)

Internet: [www.latticesemi.com](http://www.latticesemi.com)