

DDR Memory and LWIP LIB

Double Data Rate (DDR) Memory

- Maximize throughput:
 - High clock rate (133+ MHz)
 - DDR, 100 – 200 MHz; DDR2, 200 – 533 MHz; DDR3: 400 – 800 MHz
 - Transfer data on both edges
 - Burst reads and writes
 - Short access times
- Simple performance definition:
 - Frequency rate
 - 133 MHz, 166 MHz, etc.
 - CAS (Column Address Strobe) Latency (CL), in clock cycles
 - delay time between when a column access is requested, and the data is available
 - 2, 2.5(3), etc.

DDR2 SDRAM Interface Module

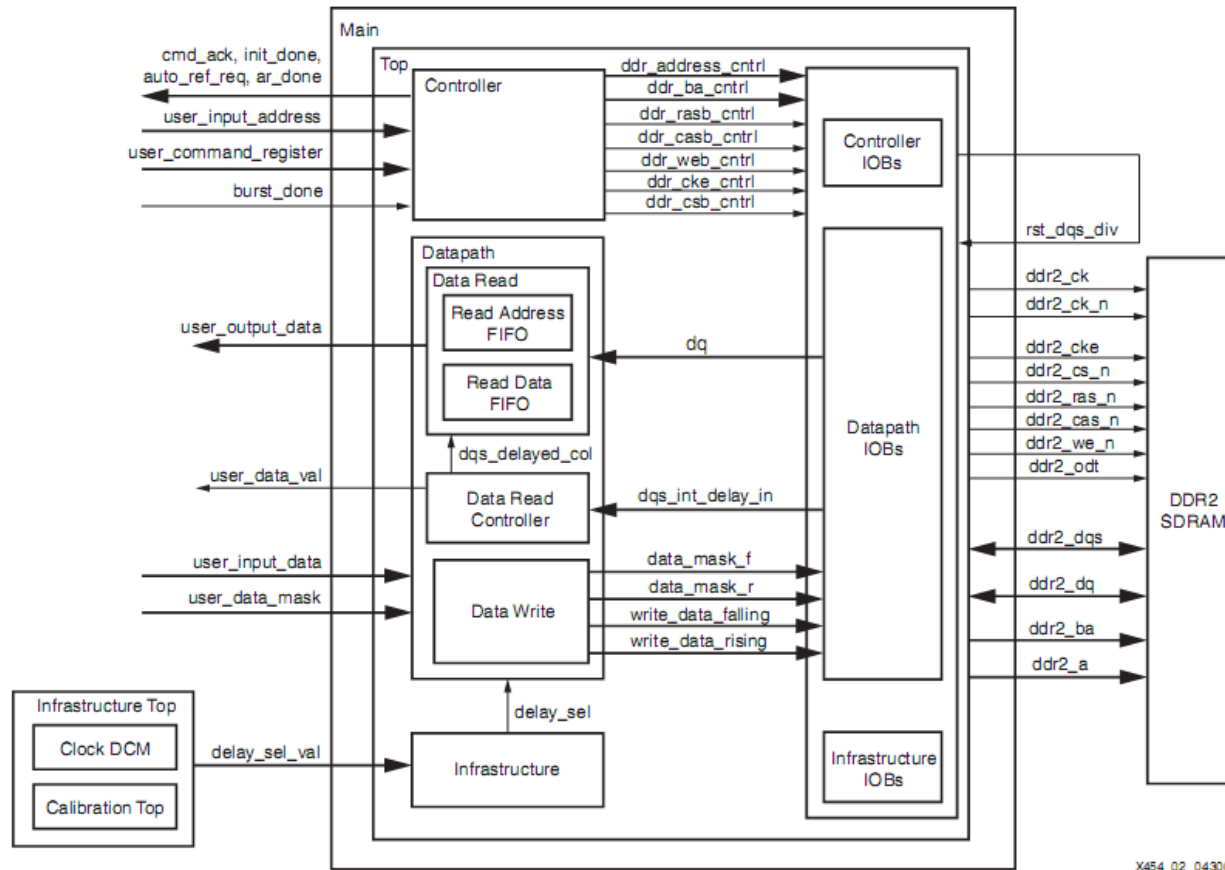


Figure 2: DDR2 SDRAM Interface Modules

X454_02_043008

Double Data Rate to Single Data Rate

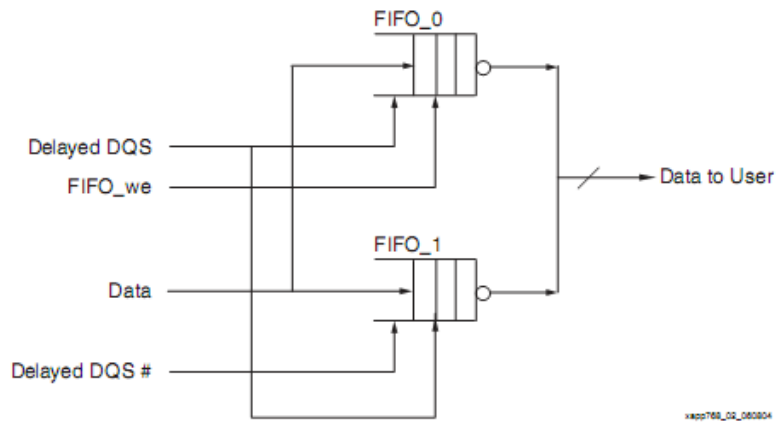
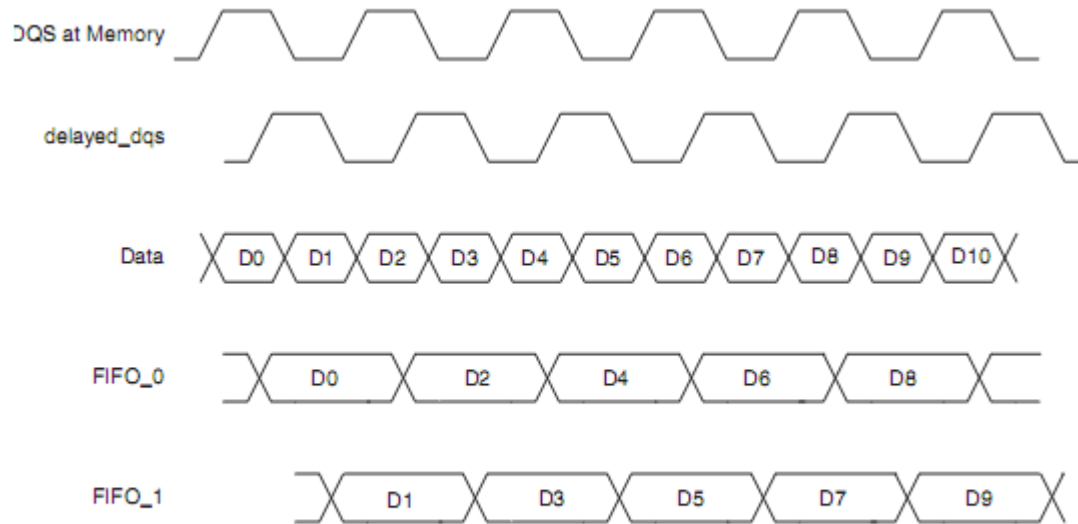


Figure 2: FIFO Block Diagram



Figure 5: FIFO Write Enable Timing Diagram

Read Timing Diagram



app768_08_081404

Figure 6: Read Data Timing Diagram

DDR Timing Difficulties

- Source synchronous: data and clock from data source
- Clock at 166 MHz:
 - Period: 6 ns
 - Half Period: 3 ns
- Routing from IOB to destination delays can easily vary from $< .5$ ns to $> 2+$ ns – a significant portion of our half-period!
 - IOB placement?
 - Destination placement?
 - More than 15 pages defining Input Timing alone in Spartan 3A Data Sheet

DDR Timing Solution

- Xilinx provides the “Memory Interface Generator”
 - Software tool that takes as inputs...:
 - Memory definition (speed, CAS, data width, banks, etc.)
 - FPGA part
 - Preferred input locations (sides, etc.)
 - ...and produces:
 - RTL PHY
 - Pin locations
 - UCF file to define pins and place RTL PHY in specific LUTs, etc., with full set of constraints
- PHY auto-calibrates:
 - Calibrates delay lines to keep delayed DQS correctly aligned across temperature/voltage variations

RST_DQS_DIV

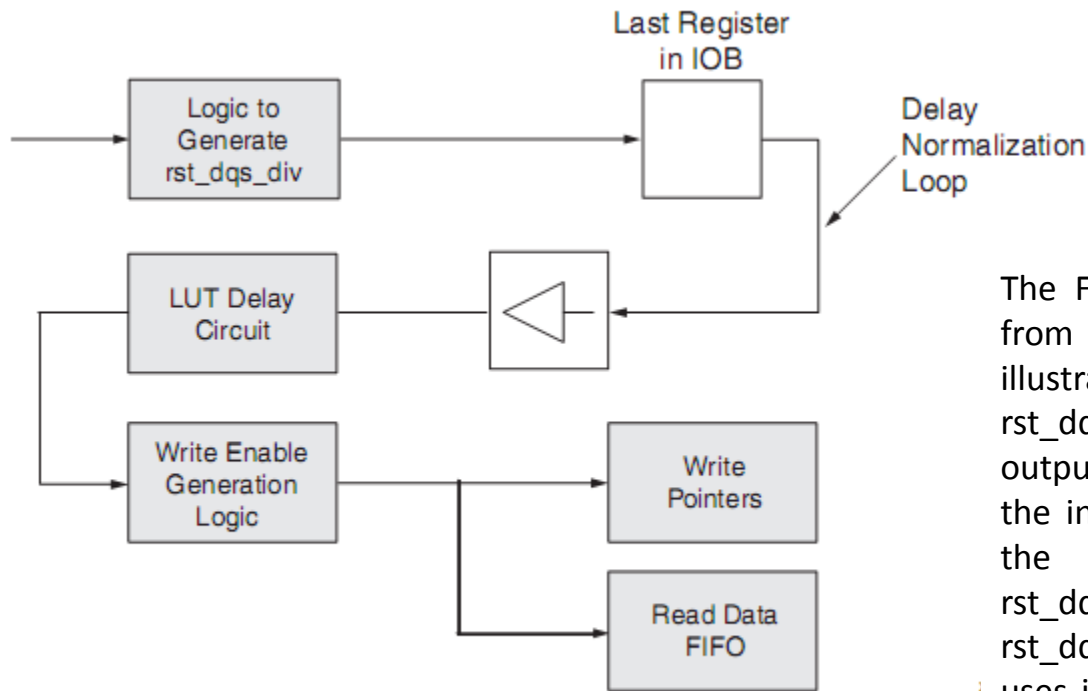


Figure 4: Circuit Illustrating `rst_dqs_div` Signal

The FIFO write enable signal is generated from a signal named `rst_dqs_div`. Figure 4 illustrates the idea behind `rst_dqs_div`. The `rst_dqs_div` signal is driven to an IOB as an output and is then taken as an input through the input buffer. This technique normalizes the IOB and trace delays between `rst_dqs_div` and the DQS clock signals. The `rst_dqs_div` from the input pad of the FPGA uses identical routing resources as the DQS before it enters the LUT delay circuit. The trace delay of the loop should be the sum of the trace delays of the clock forwarded to the memory and the DQS.

Oops

The screenshot shows the Xilinx FPGA Editor interface. The main window, titled "Array1", displays a routing grid with a red path and a yellow circle highlighting a red 'X' symbol, indicating a routing error. The "List1" window shows a table of nets:

All Nets	
Name Filter	
dqqs_div rst	
Apply	
Name	
1	Inst_processor/DDR_SDRAM_MT46V16M16_5B/DDR_SDRAM_MT46V16M16_5B/mpmc_core_0/gen_s3_ddr_phy.mpmc_phy_if_0/dqs_div rst

The "World1" window shows a zoomed-in view of the routing area, highlighting a specific routing path with a white box. The status bar at the bottom of the editor displays the net name: "net "Inst_processor/DDR_SDRAM_MT46V16M16_5B_apmc_clk_s"". The bottom right corner of the status bar shows the identifier "xc3sd1800a-5cs484" and the message "No Logic Changes".

UCF/Timing Report

```
#####  
## Constraint from rst_dqs_div_in PAD to input of LUT delay element.  
#####  
NET "*/DDR_SDRAM_MT46V16M16_5B/mpmc_core_0/gen_??_ddr_phy.mpmc_phy_if_0/dqs_div_rst" MAXDELAY = 468 ps;
```

=====

Timing constraint: NET

"Inst_processor/DDR_SDRAM_MT46V16M16_5B/DDR_SDRAM_MT46V16M16_5B/mpmc_core_0/
gen_s3_ddr_phy.mpmc_phy_if_0/dqs_div_rst" MAXDELAY = 0.468 ns;

1 net analyzed, 1 failing net detected.

1 timing error detected.

Maximum net delay is 2.549ns.

Slack: -2.081ns

 Inst_processor/DDR_SDRAM_MT46V16M16_5B/DDR_SDRAM_MT46V16M16_5B/mpmc_core_0/gen_s3_ddr_phy.mpmc_phy_if_0/dq
 s_div_rst

Error: 2.549ns delay exceeds 0.468ns timing constraint by 2.081ns

From	To	Delay(ns)
AB5.I	SLICE_X0Y70.G3	2.499
AB5.I	SLICE_X1Y70.F3	2.448
AB5.I	SLICE_X1Y70.G4	2.481
AB5.I	SLICE_X1Y71.G2	2.549

Adding Memory (Roughly)

- Add a MPMC
 - During this process, add a memory that matches your part as closely as possible; update to match exactly
 - This adds a memory to the MB, as well as the memory controller PHY, but without constraints
- Create a new Memory PHY in the ISE project
 - During this process, add a memory that matches your part as closely as possible; update to match exactly
 - Choose the banks where you would like the memory to go
 - Generate
 - Go back and edit to match your pins exactly; check with memory generator
- Take the UCF from the Memory PHY, and map to the MPMC PHY
 - xilperl
C:\Xilinx\10.1\EDK\hw\XilinxProcessorIPLib\pcores\mpmc_v4_03_a\data\convert_ucf.pl --mhs ...\processor.mhs
...\mem_phy\user_design\par\mem_phy.ucf ...\new.ucf
- Incorporate into your UCF

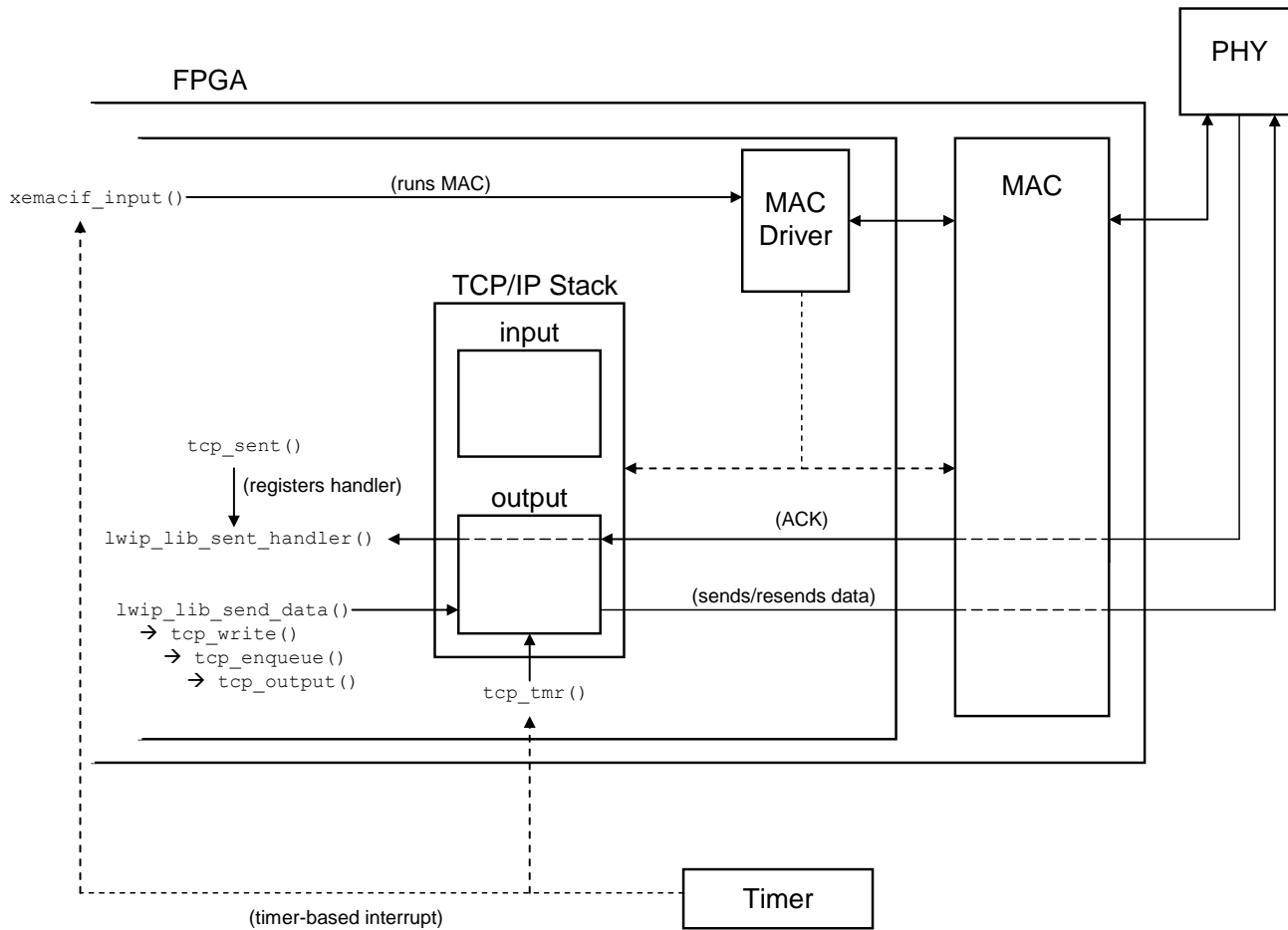
IP, TCP

- IP
 - Delivers packets to destination based on logical address
 - (Ethernet delivered based on physical address)
- TCP
 - Connection-oriented
 - Data exchanged once a connection is made
 - Reliable: Guaranteed data delivery
 - Packets are ack'ed; retransmitted if necessary
 - Packets can arrive out of order
 - Connection protocol used for higher level protocols, like HTTP

LWIP

- “lwIP is a small independent implementation of the TCP/IP protocol suite that has been developed by Adam Dunkels at the Computer and Networks Architectures (CNA) lab at the Swedish Institute of Computer Science (SICS).”
 - IP, ICMP, UDP, TCP, DHCP, ARP, ...
- Xilinx includes as part of their libraries for MB
- Wrote their own “netif” to interface with TEMAC and ethernet_lite

How To Send with LWIP



How To Send with LWIP

- An acknowledgement handler must be registered using `tcp_sent()`. In `lwip_lib`, this function is `lwip_lib_sent_handler()`, which determines which of the sent buffers has been acknowledged when ACK packets are received.
- The application sends data using `lwip_lib_send_data()`. This function, in turn, calls `tcp_write()`, which enqueues the data to be sent but does not actually send the data to the MAC. A call to `tcp_output()` forces the data to be sent, rather than waiting for more data to be enqueued.
- An interrupt is generated by the external timer at regular intervals (every 1 ms, in the case of `lwip_lib`). Within the interrupt handler, `xemacif_input()` is called (which is part of the Virtex 2-based port of the lwIP netif structure) to check the MAC for incoming data. The interrupt handler also calls `tcp_tmr()`, which is a requirement of lwIP to allow it to check for packets that need to be resent. Either `xemacif_input()` or `tcp_tmr()` may force a call to `tcp_output()` to send or resend data as appropriate.
- When an acknowledgement is received, `xemacif_input()` causes the `lwip_lib_sent_handler()` to be called with a parameter indicating the amount of data acknowledged by the other side of the TCP connection. As mentioned before, `lwip_lib_sent_handler()` then determines if any of the sent buffers have been completely acknowledged and can be marked as free for reuse.

LWIP_LIB

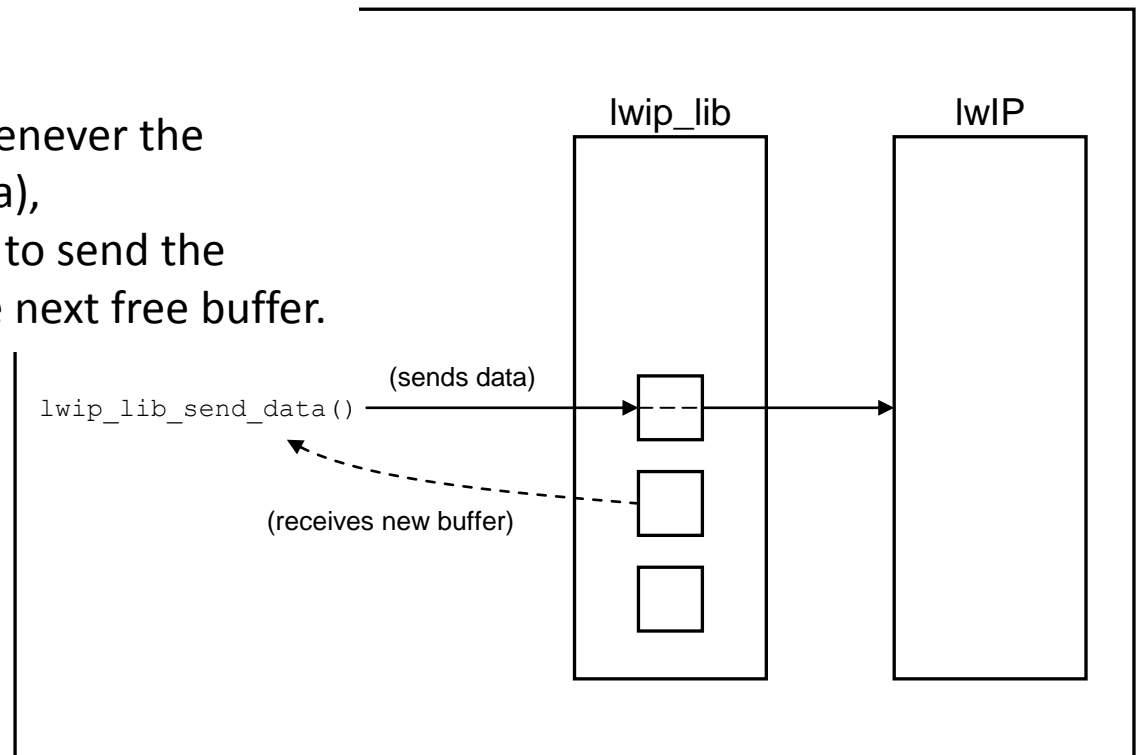
- `lwip_lib` is a relatively simple encapsulation of the lwIP Raw API for use in single-threaded MicroBlaze programs.
- The library simulates a separate lwIP “thread”:
 - timer interrupt handles asynchronous data transfer
- Exposes only a few straightforward functions to the user application to handle lwIP initialization and sending and receiving data
- Always the server

- Written by EP student Andy Lehman!

How to Send with LWIP_LIB

Before generating data to be sent, `lwip_lib_configure_buffers()` (or, equivalently, `lwip_lib_full_startup()`) must be called to setup the send buffers. The return value will be the first buffer that can be used to store application data to be sent.

When the buffer is full (or whenever the application wants to send data), `lwip_lib_send_data()` is called to send the current buffer and receive the next free buffer.



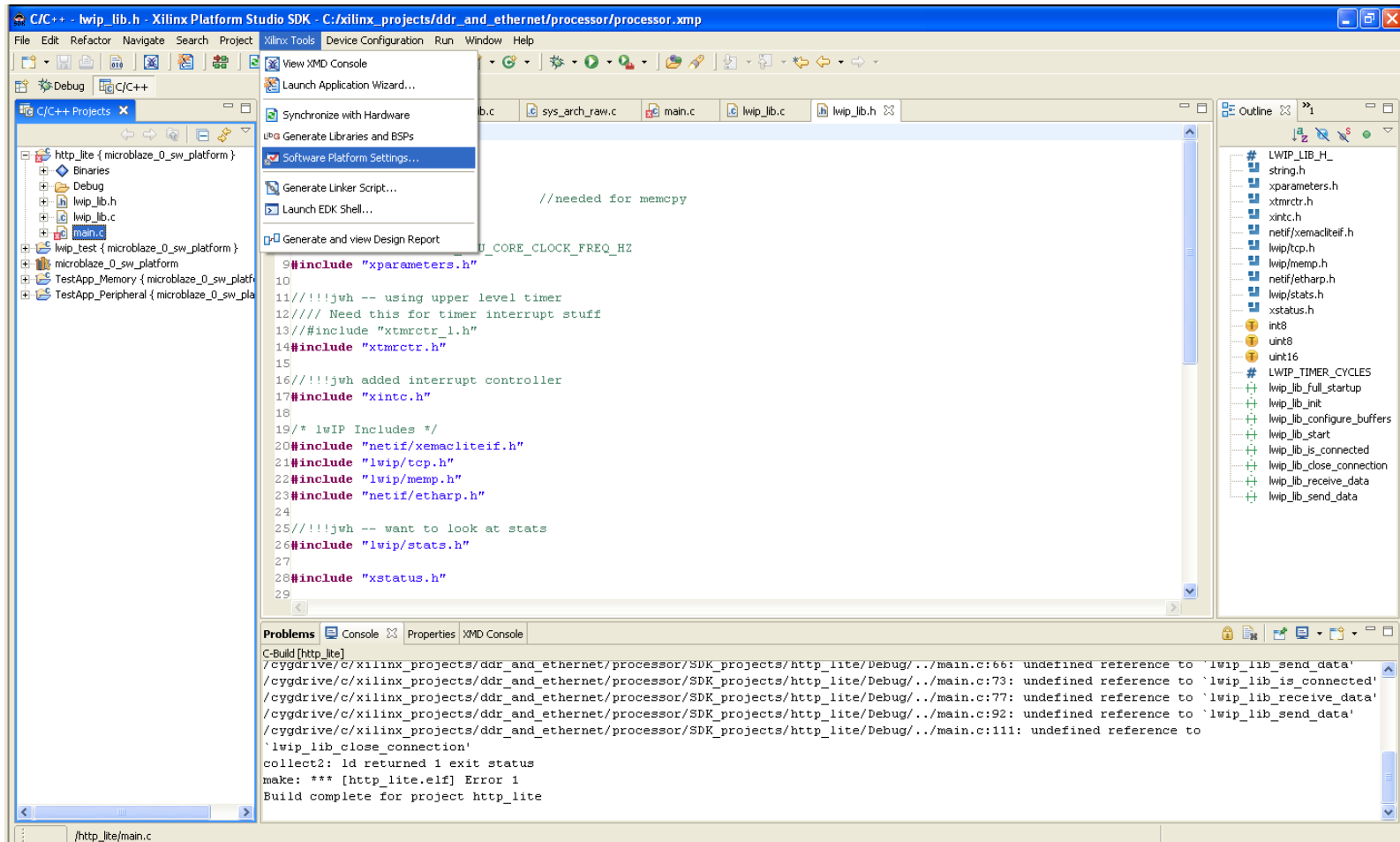
“Canonical” Names

- lwip lib will interact with the hardware using the canonical definitions in xparameters.h:
 - XPAR_INTC_0_DEVICE_ID
 - XPAR_INTC_0_EMACLITE_0_VEC_ID
 - XPAR_INTC_0_TMRCTR_0_VEC_ID
 - XPAR_TMRCTR_0_DEVICE_ID
 - XPAR_EMACLITE_0_DEVICE_ID
- Canonical: “In simplest, or standard form”

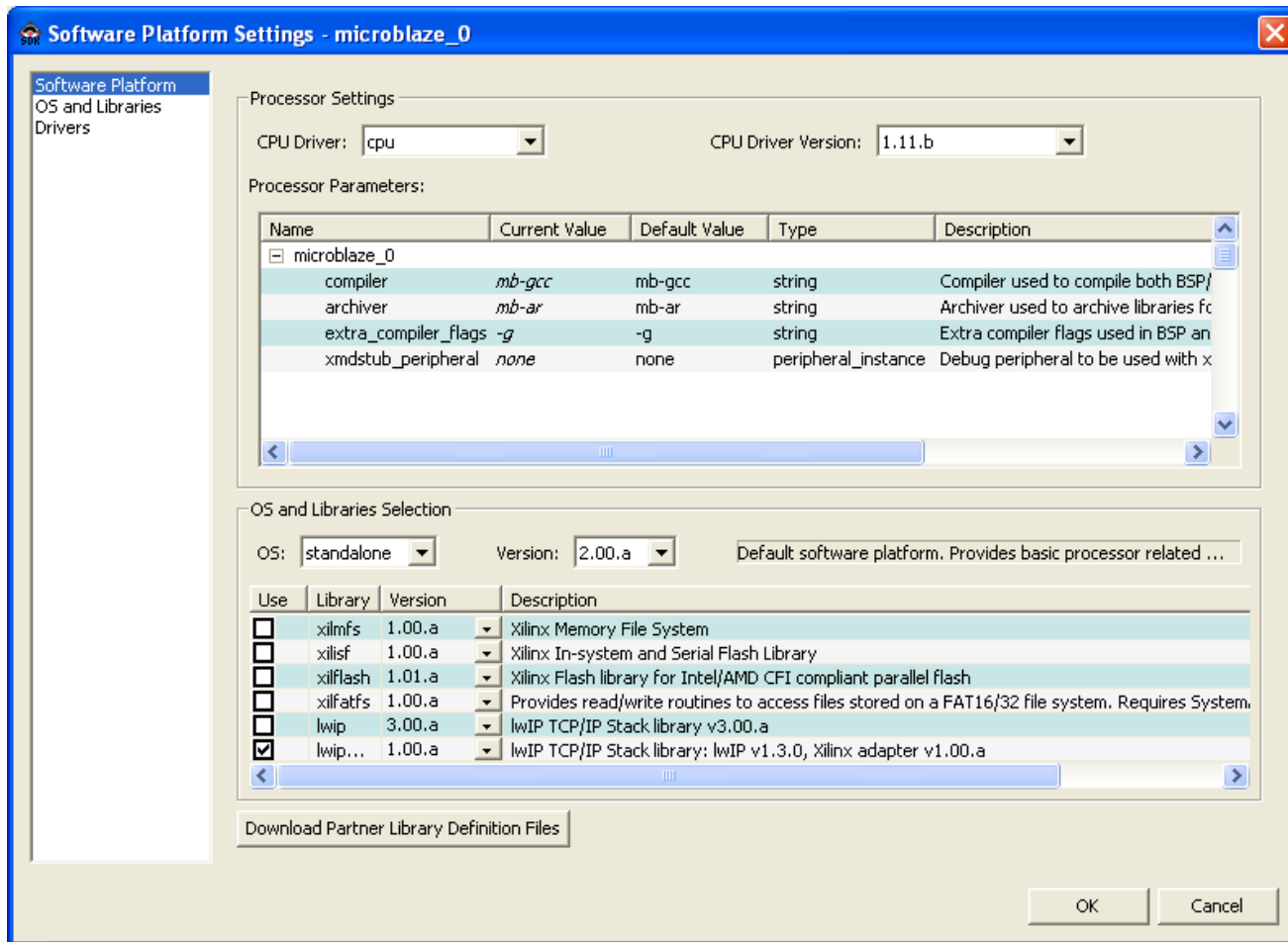
Add lwip and lwip_lib to your project

- Lwip
 - Enable library
 - Adjust settings if interested
 - Update linker
 - Move code to external memory
 - Create a heap
- Lwip_lib
 - Copy c files into project

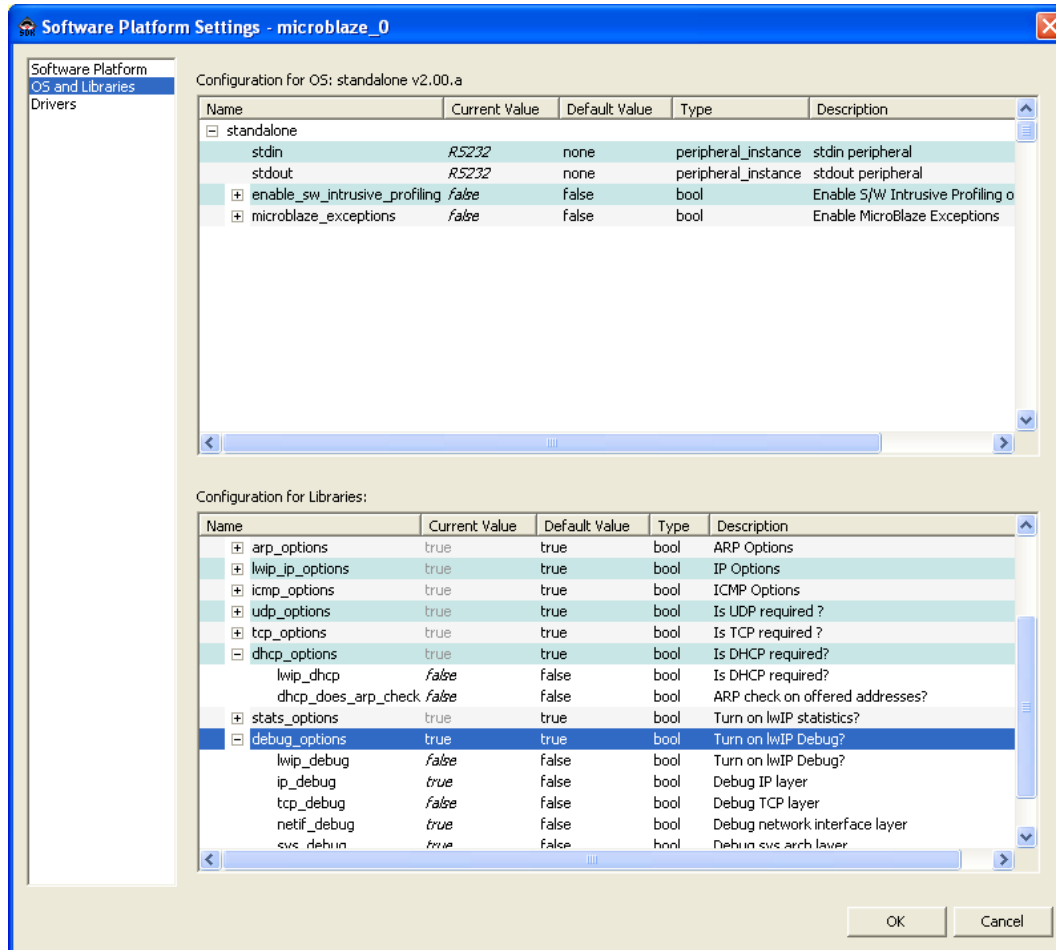
Enable LWIP for your Project



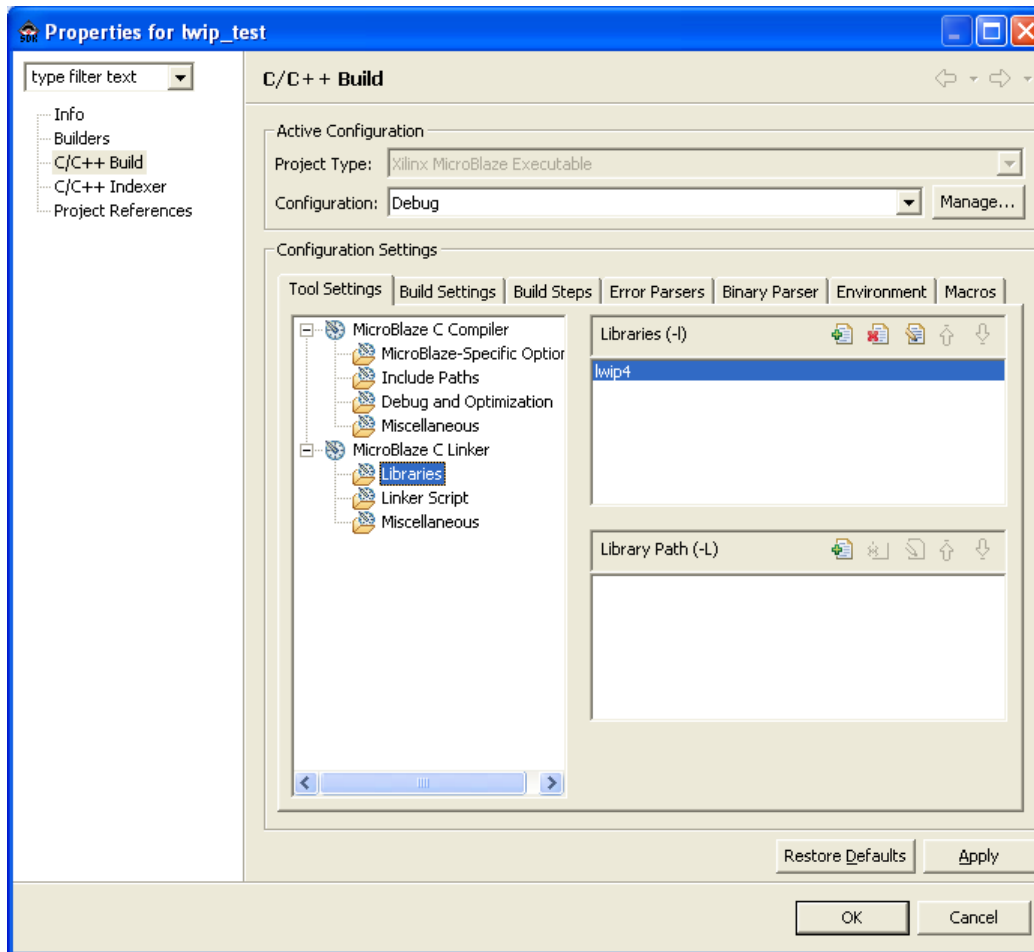
Enable LWIP for your Project



LWIP Options



Add library to the linker configuration



Generate a Linker File

Linker Script Generator

Application project name:

ELF file used to populate section info:

Code Sections

Assign all Code Sections to:

Section	Size (bytes)	Memory
.text	0x0001E6C4	DDR_SDRAM_MT46V16M16_5B_C_MPMC_BASEADDR

Data Sections

Assign all Data Sections to:

Section	Size (bytes)	Memory
.rodata	0x000014C8	ilmb_cntrl_dlmb_cntrl
.sbss2	0x00000000	ilmb_cntrl_dlmb_cntrl
.data	0x00000598	ilmb_cntrl_dlmb_cntrl
.sbss	0x00000000	ilmb_cntrl_dlmb_cntrl
.bss	0x00071180	DDR_SDRAM_MT46V16M16_5B_C_MPMC_BASEADDR

Heap and Stack:

Section	Size (bytes)	Memory
Heap	0x8000	ilmb_cntrl_dlmb_cntrl
Stack	0x1000	ilmb_cntrl_dlmb_cntrl

Reference Views (read-only)

Memories:

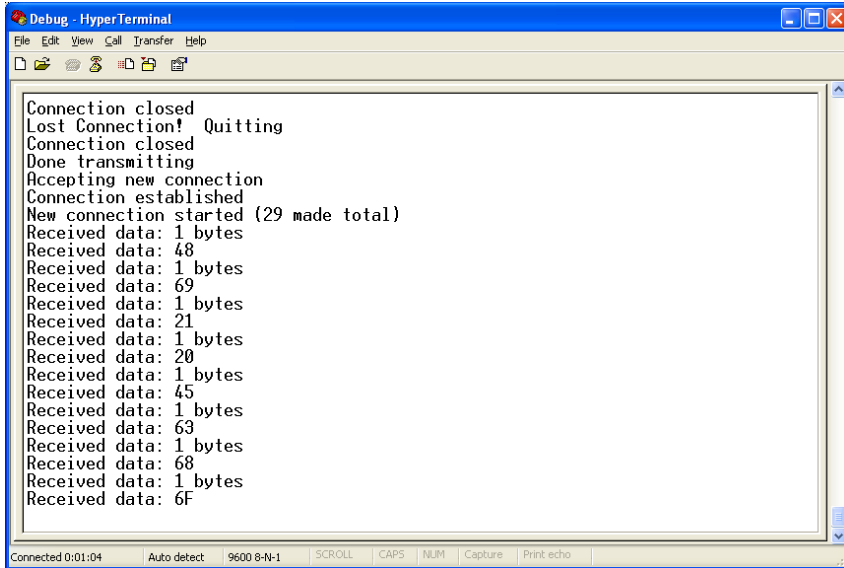
Memory	Address	Size
ilmb_cntrl_dlmb_cntrl	0x00000000	64K
DDR_SDRAM_MT46V16M16_5B_C_MPMC_BASEADDR	0x86000000	32768K

Boot and Vector Sections:

Section	Address	Memory
.vectors.reset	0x00000000	ilmb_cntrl_dlmb_cntrl
.vectors.sw_exception	0x00000008	ilmb_cntrl_dlmb_cntrl
.vectors.interrupt	0x00000010	ilmb_cntrl_dlmb_cntrl
.vectors.hw_exception	0x00000020	ilmb_cntrl_dlmb_cntrl

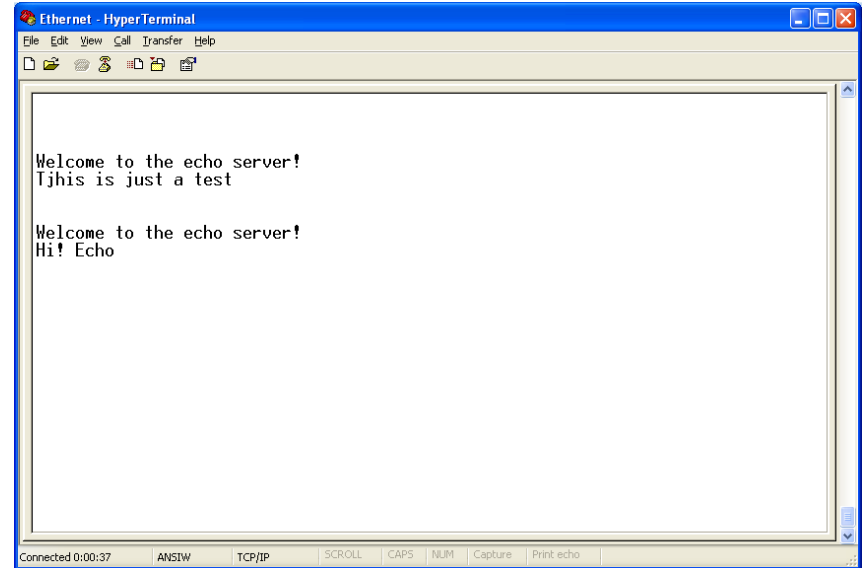
Output Linker Script:

Echo Server Example



```
Connection closed
Lost Connection! Quitting
Connection closed
Done transmitting
Accepting new connection
Connection established
New connection started (29 made total)
Received data: 1 bytes
Received data: 48
Received data: 1 bytes
Received data: 69
Received data: 1 bytes
Received data: 21
Received data: 1 bytes
Received data: 20
Received data: 1 bytes
Received data: 45
Received data: 1 bytes
Received data: 63
Received data: 1 bytes
Received data: 68
Received data: 1 bytes
Received data: 6F
```

Connected 0:01:04 Auto detect 9600 8-N-1 SCROLL CAPS NUM Capture Print echo



```
Welcome to the echo server!
Tjhis is just a test

Welcome to the echo server!
Hi! Echo
```

Connected 0:00:37 ANSW TCP/IP SCROLL CAPS NUM Capture Print echo

HTTP

- HTTP is a application layer request/response protocol:
- Consider: `http://192.168.0.30/index.html`
 - Create TCP connection on port 80 to host 192.168.0.30
 - Request:
 - `GET /index.html http/1.1`
 - Response:
 - `HTTP/1.1 200 OK`
 - `Content-Type: text/html`
 - `Content-length: 162`
 - `Connection: close`
 - `[...HTML file...]`
- Browser does this – the “Connection: close” informs the browser to close
- `xapp1026.zip` has functions to handle this
 - will need to strip `lwip`, replace with `lwip_lib`

Testing with Telnet

- Connect on port 80 (Telnet is usually 23)
- Send a file with “GET /index.html”
- See what the response is!