FPGA Specification Document

CPEN 430 Lab Report #5
Ethernet Packet Communication
6 December 2018

Damon George
Robert Brajcich

IT WORKS!!

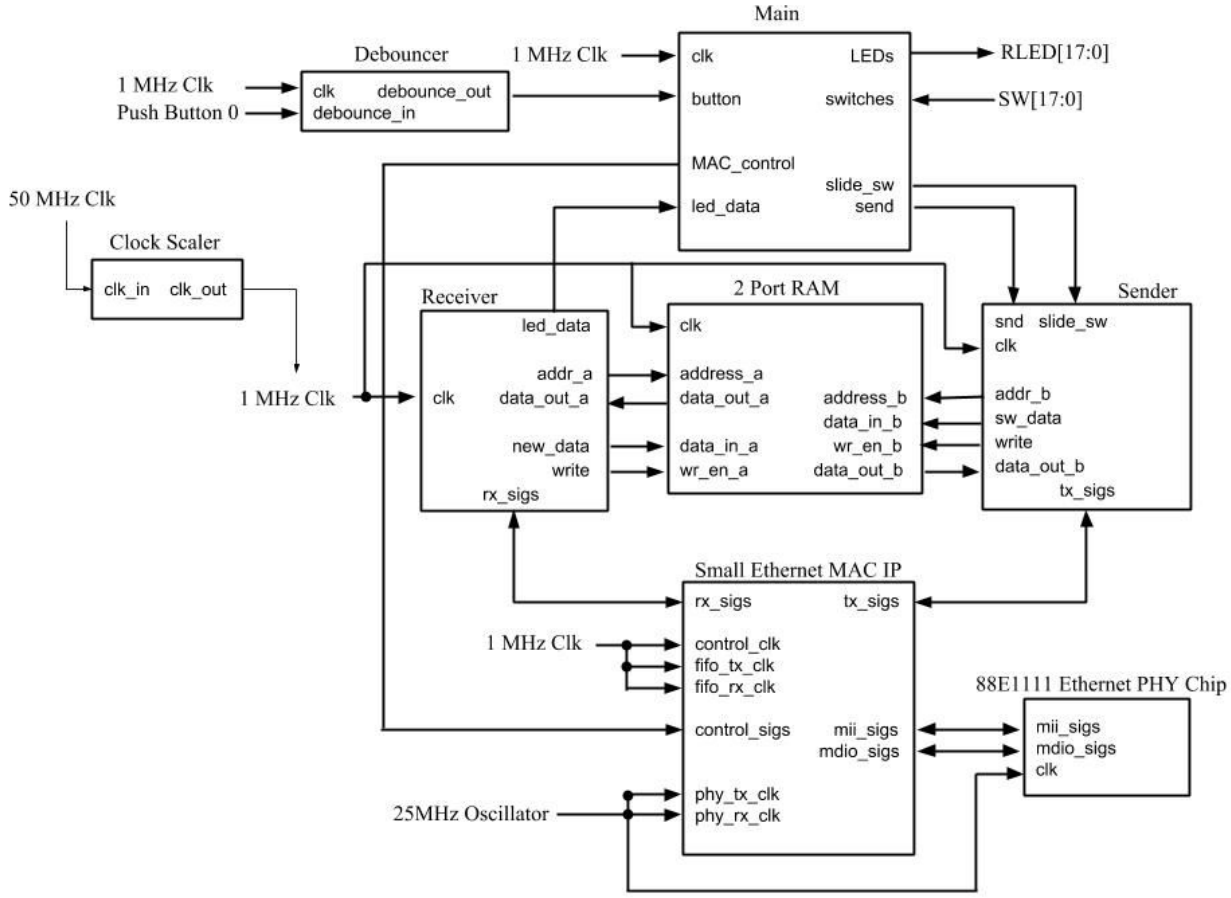
TABLE OF CONTENTS

Functional Description

General Overview

This lab is primarily intended to demonstrate the communication between an ethernet interface on each of two DE2-115 development boards, with a very basic interface to test the packet-sending functionality. The primary inputs for the system are the slide switches sw17 to sw0 and the pushbutton key3. The primary outputs are the 18 onboard red LEDs. The user can slide the 18 switches up or down in order to specify a pattern of LEDs, which is stored in on-chip memory. By then pressing key3, the pattern is read from memory, encapsulated in an ethernet packet and sent to the other board, where it gets displayed on that board's red LEDs. Once a pattern has been sent to the other board, it will continue to show it until another pattern is sent to it. This implementation is loaded onto both boards such that a two-way communication allows each board to control the LEDs on the other one. We were able to make this lab function fully as intended.

Top Level Block Diagram



The *Clock Scaler* block converts the onboard 50 MHz clock to a 1 MHz clock.

The *Debouncer* block is responsible for debouncing the signal of key0 in order to eliminate edge jitter.

The *2PortRam* block was generated using the on-chip memory IP in Quartus. The ethernet packet data is stored in memory and updated each time the slide switches are changed. The memory is then read when key0 is pressed and the ethernet packet is written. Incoming LED pattern data is also written to memory before being output to the LED pins.

The *Receiver* block handles incoming ethernet packets. It reads from the small ethernet MAC IP block when a ready signal informs it of new data. The new LED data is then written to the RAM block.

The *Sender* block handles outgoing ethernet packets. When signalled by the main block (prompted by key0), it reads the packet data from memory and sends it to the small ethernet MAC IP block to be output.

The *Small Ethernet MAC IP* block is generated by Quartus to interface with the onboard ethernet physical layer chip.

The *Ethernet PHY Chip* is the physical chip on the DE2-115 board that handles the layer 1 ethernet protocol elements. We only have to provide it the clock it needs and use the MAC IP block to interact with it. There is also some initialization done on it through the *Small Ethernet MAC* block.

The *Main* block is responsible for handling the inputs and outputs of the system as well as maintaining its logic flow. It begins by performing the necessary initialization for the *Small MAC* and PHY chip. It provides the slide switch signals to the sender block and informs it to output a packet when key0 is pressed. It also accepts the most up-to-date incoming LED data from the receiver block and outputs it to the physical pins to turn them on.

Lower Level Block Descriptions

Clock Scaler

The clock scaler module is made up of a simple counter that counts (on edges of clk_in) to a scale value, and toggles clk_out to generate a slower clock signal.

Debouncer

The debouncer module waits for a change in the debounce_in signal, and begins counting clk cycles until it either detects another change, in which case the counter resets, or until it hits 20 milliseconds of a consistent input, in which case the input value propagates to debounce_out.

2PortRam

The 2PortRam module provides the on-chip memory interface for the design, and is generated using the wizard in Quartus. The RAM holds 16 words, and each is 32 bits long. The first 8 words are used to hold the packet data that will be output to the ethernet block when a send operation is triggered. The following 8 words are used to hold incoming packet data, including the LED data that was sent from the other board. Port A is used by the receiver module, which reads and writes data into the second half of the RAM. Port B is used by the sender module, which writes the slide switch data into the first half of RAM, and transfers the packet data from the first 8 words of RAM to the ethernet module when the send flag is asserted. The RAM is constructed with data in and out registers, a write enable flag, and an address input for each of the two ports. The clk signal is shared between both ports and is synced to the 1 MHz clock that drives all the modules.

Receiver

The Receiver module has two primary responsibilities. It reads and outputs the most up-to-date LED data from RAM back to the main module to be displayed, and it receives incoming packets from the MAC IP module and writes the relevant data to RAM. It accomplishes this using a state machine that keeps track of whether it is receiving data or just idling while outputting the LED data to the main module. It is driven using the 1 MHz clock. The rx_sigs connecting from the receiver to the MAC IP module include a flag that tells the receiver when a message is ready, and the data bus and corresponding signals that allow the receiver to read the data from the ethernet module's incoming FIFO including a write enable flag and rx_ready flag. When a message is ready from the ethernet module, it reads the whole packet into memory. It then waits to see if the rx_err flag is set at the end of the message. If there is no error, it copies the LED data word from the packet to another location in memory, and returns to its idle state in which it outputs the updated LED data from memory to main.

Sender

The sender module also has two responsibilities. The first takes place during its idle state. It simply connects the incoming slide switch data from main straight to the RAM data input port, and holds the RAM write enable high to constantly write the most up-to-date switch data. This location in RAM is surrounded by the rest of the packet header information that must be sent as well. The other responsibility is to send the switch data to the ethernet module, which begins when the send flag from main is asserted (which occurs when key3 is pressed). The sender enters a wait state that does nothing until the ethernet fifo is ready for data, while placing the first word of the packet from memory on the tx data bus. When ready, the module moves to the send state, which asserts the write signal for the ethernet module and the sop or eop flags if necessary. The state machine then returns to the wait state, and continues back and forth between these states until every word of the packet has been written to the ethernet module. It then returns to the idle state to continue reading the slide switch data into memory.

Small Ethernet MAC IP

This module was generated using Intel's IP included in Quartus. We selected the Small MAC 10/100 Mbps variant because of the simplicity of configuring it for our needs. We supplied it with a 1 MHz clock for the control and FIFO tx/rx ports. It contains various flags and data ports for sending and receiving from the FIFO. It accomplishes some of the layer 2 tasks required for ethernet, including CRC calculation and verification. It also implements the MII interface required to send the data and receive it through the PHY chip.

Main

The main module is responsible for the initialization of the ethernet module and PHY chip, and then to send the switch data to the sender module and signal that module to send out the data when key3 is pressed. Main first starts by performing a hardware reset on the PHY chip, which includes delays to make sure the chip is allowed sufficient time to power on. Next, the MAC is reset as well, also with a delay. Next, the main configuration register of the MAC is read in order to enable the tx and rx interfaces. Then the Main state machine waits until the MAC is properly configured. Next, Main writes to the PHY chip's registers. This is accomplished by writing to a portion of the MAC's register address space which corresponds to the address space of the PHY chip. In turn, the MAC module performs the register reads and writes over the MDIO connection to the PHY chip. Using this method, this module writes configurations to registers 20, 16, and 0 of the PHY chip. This is done in order to turn on the physical transceiver, enable automatic crossover for the ethernet cables, and set the speed to 100 Mbps. Then this module performs a software reset on the PHY chip, waits, and then begins the running state. In the running state, this module just outputs the LED data from the receiver module, sends the switch data to the sender, and signals the sender when key3 is pushed. Furthermore, key1 is used to reset this state machine to the beginning, as well as to reset the state machine of the sender and receiver as well.

Clock and Reset Methodology

Our design uses the onboard 50 MHz clock signal with a clock scaler to generate a 1 MHz clock that is used to synchronize the logic in all our modules. This includes the control clock for the ethernet module as well as the FIFO tx and rx clocks. The only other clock used is from an onboard 25MHz oscillator, and is for the MII interface between the Ethernet MAC module and the PHY chip.

Our design uses a synchronous reset on key1 to set all our modules to their initialization or starting states. This allows us to ensure that the ethernet MAC module and PHY chip are properly initialized before we attempt to use them.

I/O Summary

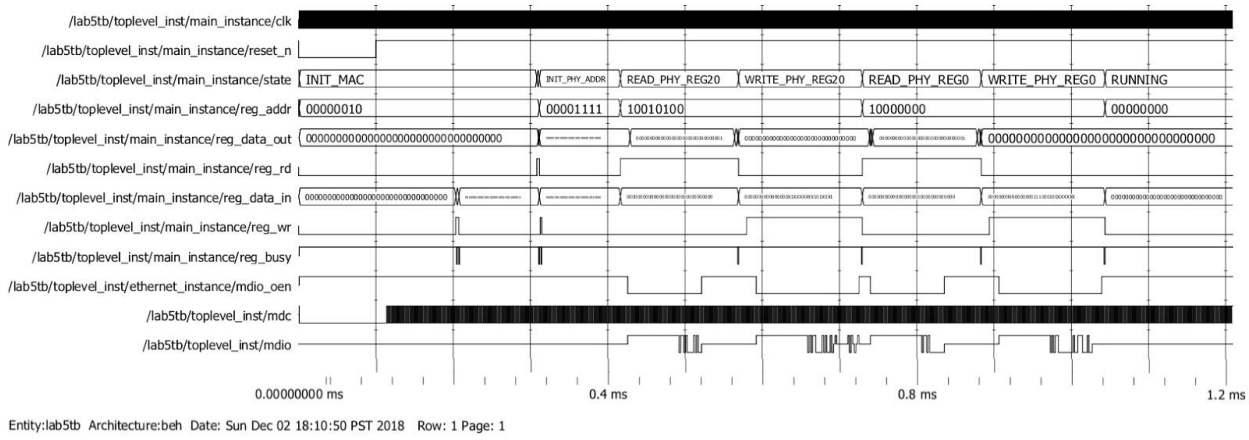
I/O	Location
clk	PIN_Y2
button	PIN_R24
reset_button	PIN_M21
red_leds[17]	PIN_H15
red_leds[16]	PIN_G16
red_leds[15]	PIN_G15
red_leds[14]	PIN_F15
red_leds[13]	PIN_H17
red_leds[12]	PIN_J16
red_leds[11]	PIN_H16
red_leds[10]	PIN_J15
red_leds[9]	PIN_G17
red_leds[8]	PIN_J17
red_leds[7]	PIN_H19
red_leds[6]	PIN_J19
red_leds[5]	PIN_E18
red_leds[4]	PIN_F18
red_leds[3]	PIN_F21
red_leds[2]	PIN_E19
red_leds[1]	PIN_F19
red_leds[0]	PIN_G19

I/O	Location
switches[17]	PIN_Y23
switches[16]	PIN_Y24
switches[15]	PIN_AA22
switches[14]	PIN_AA23
switches[13]	PIN_AA24
switches[12]	PIN_AB23
switches[11]	PIN_AB24
switches[10]	PIN_AC24
switches[9]	PIN_AB25
switches[8]	PIN_AC25
switches[7]	PIN_AB26
switches[6]	PIN_AD26
switches[5]	PIN_AC26
switches[4]	PIN_AB27
switches[3]	PIN_AD27
switches[2]	PIN_AC27
switches[1]	PIN_AC28
switches[0]	PIN_AB28

I/O	Location
rx_clk	PIN_A15
rx_data[3]	PIN_C15
rx_data[2]	PIN_D17
rx_data[1]	PIN_D16
rx_data[0]	PIN_C16
rx_en	PIN_C17
rx_err	PIN_D18
tx_clk	PIN_B17
tx_data[3]	PIN_B19
tx_data[2]	PIN_A19
tx_data[1]	PIN_D19
tx_data[0]	PIN_C18
tx_en	PIN_A18
tx_err	PIN_B18
phy_reset	PIN_C19
mdc	PIN_C20
mdio	PIN_B21

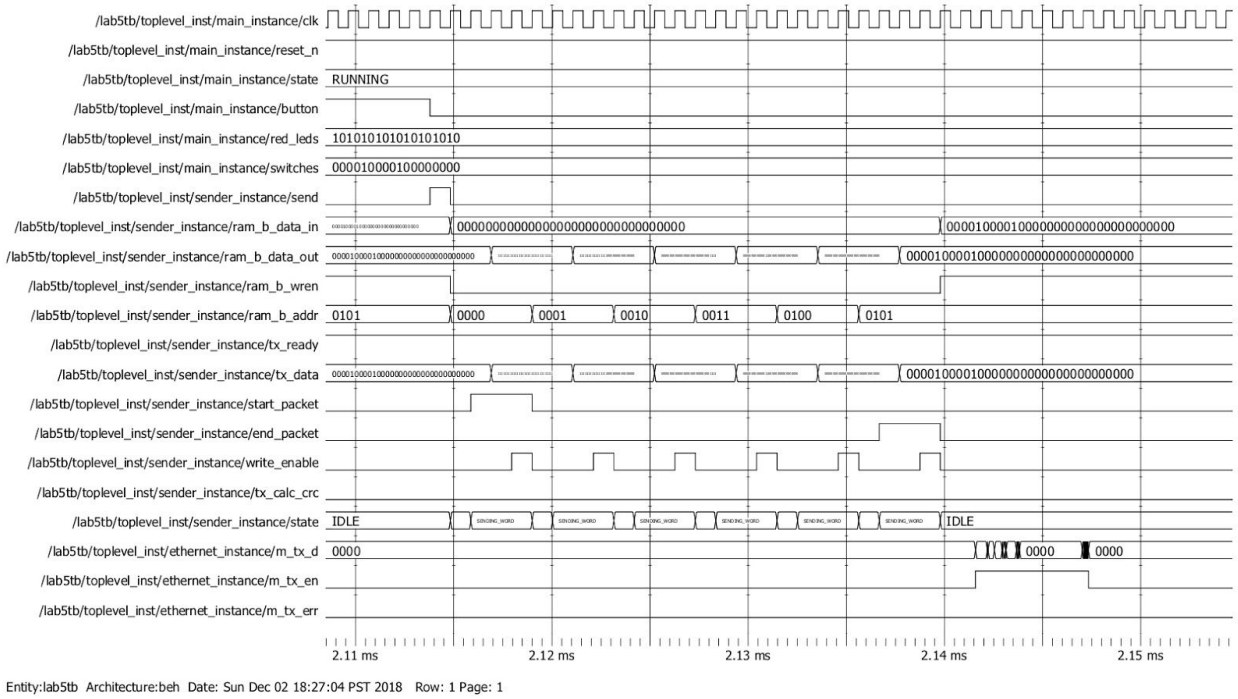
Testbench Design

Initialization



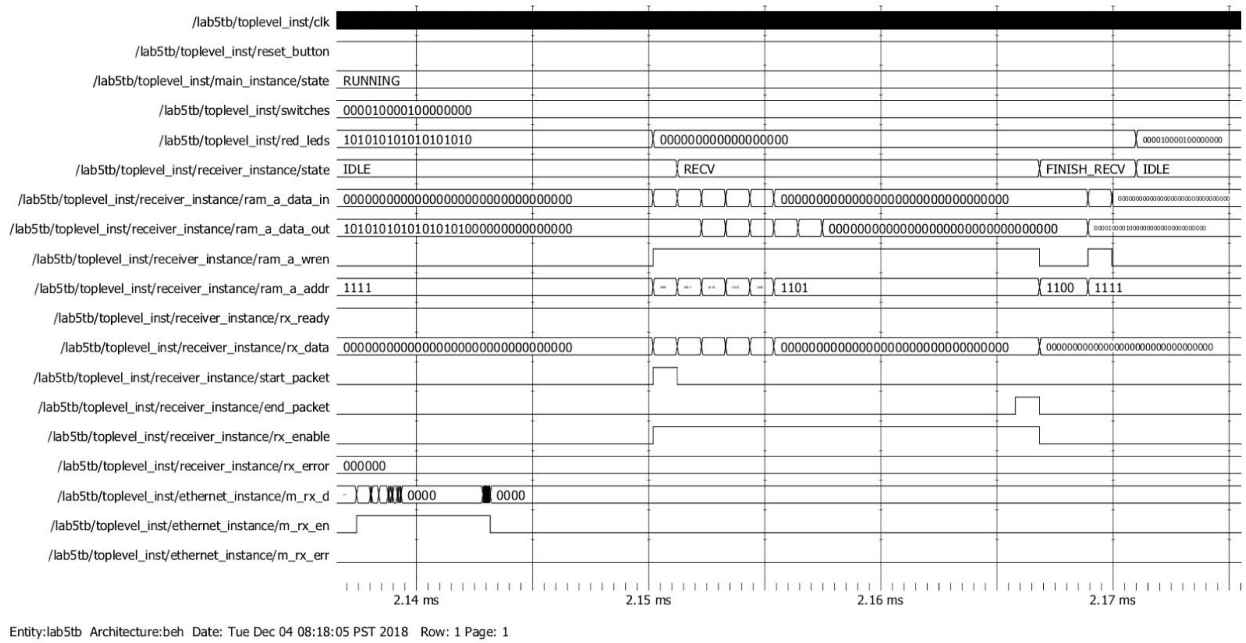
The simulated waveform above shows an overview of the initialization sequence in the Main module's state machine. In the simulation, the initial RESET_PHY state is skipped because it includes a long delay. However, the following MAC initialization states are clearly visible, as are the reads and writes to PHY registers, which include the corresponding MDIO output. The pulses where reg_busy drops low indicate the end of a register operation, which explains why each of those configuration states moves to the next state when the reg_busy flag goes low for a cycle. The bottom three signals show the MAC module successfully interacting with the PHY chip through the MDIO interface to pass on the configuration we are sending to it.

TX



The above waveform shows the send operation of the Sender and Small MAC modules. The sender module moves out of its IDLE state when the send signal is pulsed high for a clock cycle. Then the sender sets the start_packet signal, reads the first word from memory, and pulses write enable to tell the MAC to read the valid data. Start_packet is then cleared, and the sender increments the RAM address, sending the rest of the packet data to the MAC until it reaches the final word at RAM address 0101, where is pulses the end_packet signal. The sender then returns to the IDLE state while the MAC module clearly sends the data over the MII interface to the PHY chip.

RX



Entity:lab5tb Architecture:beh Date: Tue Dec 04 08:18:05 PST 2018 Row: 1 Page: 1

The above waveform shows the receive operation taking place for data that we sent from the send operation and configured to loopback in the ethernet module. When the sop and rx_enable signals go high, our module begins writing the incoming packet data into RAM in incrementing address locations, as shown above.Then, at the end of the receive operation (signaled by the eop signal going high), there is no pulse on the rx_err signal, so the receive module copies the LED data from the packet just put in memory into memory location 1111, which is where the actual LED data is stored that is output to the lights. Then the module returns to its IDLE state in which it exposes this LED data to main and waits for the next receive.

Synthesis Analysis

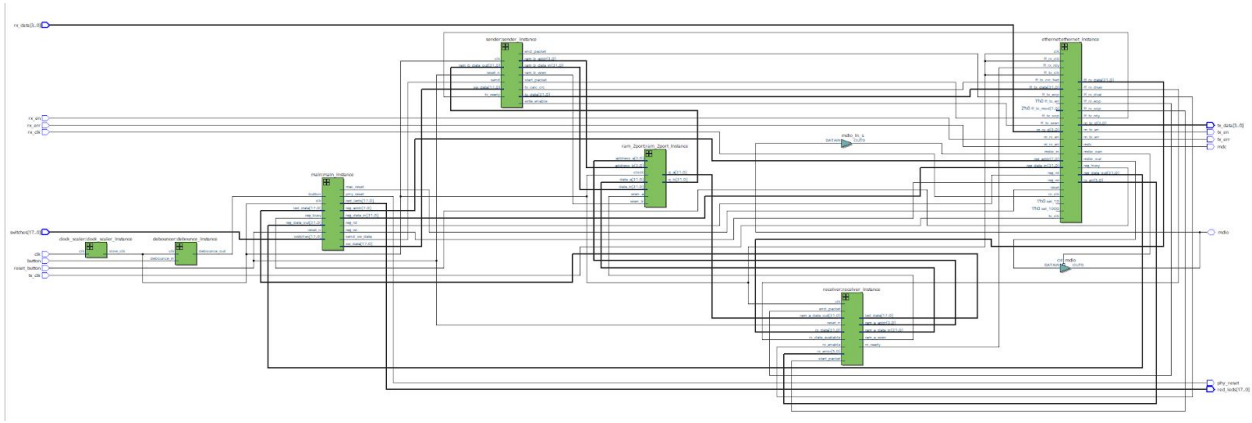
Resource Utilization

Flow Status	Successful - Mon Dec 03 11:01:53 2018
Quartus Prime Version	18.0.0 Build 614 04/24/2018 SJ Standard Edition
Revision Name	toplevel
Top-level Entity Name	toplevel
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	2,176 / 114,480 (2 %)
Total registers	1662
Total pins	56 / 529 (11 %)
Total virtual pins	0
Total memory bits	5,280 / 3,981,312 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

This design used more of the logic elements on the board than any of our previous designs. The majority of this logic was a part of the Ethernet Core IP module that we included, and not from the code we wrote. There were many pins involved, including 18 LED pins, 18 slide switch pins, multiple push button pins, a clock, and 17 pins necessary for the communication between the FPGA and the ethernet PHY chip.

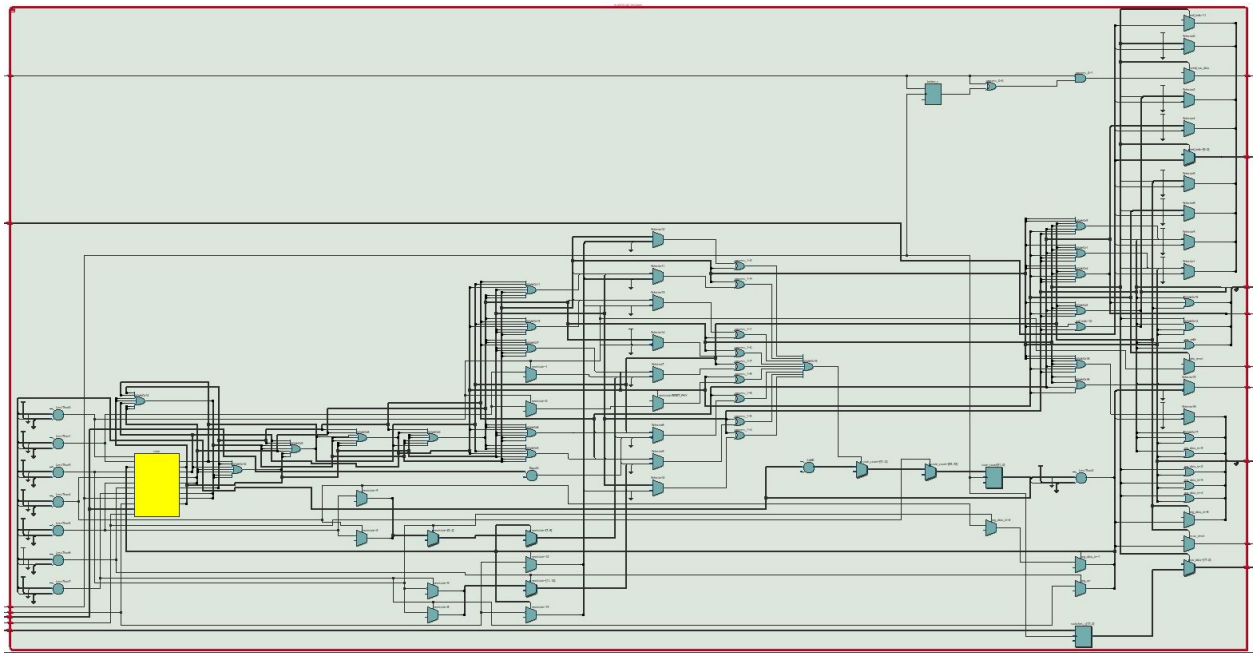
Synthesized Circuit

Top Level RTL



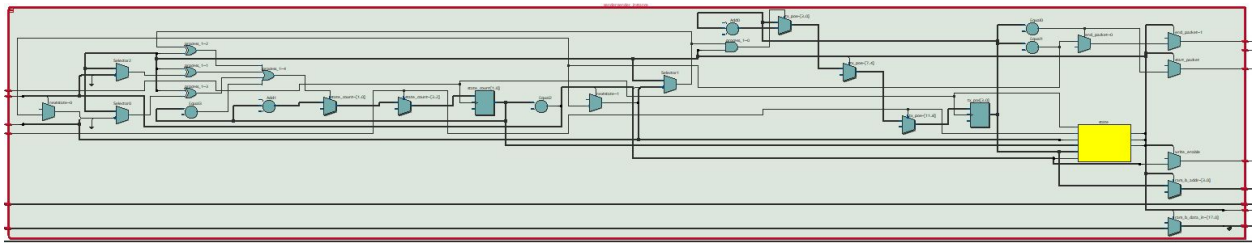
This is the top level RTL view for our design. The two small blocks on the left are the clock scaler and debouncer modules. The far right module is our ethernet module. The middlemost block is the ram2port module, and the remaining three contain the majority of our implementation: main, sender, and receiver. We will not detail the circuitry of the clock scaler or debouncer since they are recycled from past labs, or the ram2port and ethernet module since they were generated using Intel IP.

Main RTL



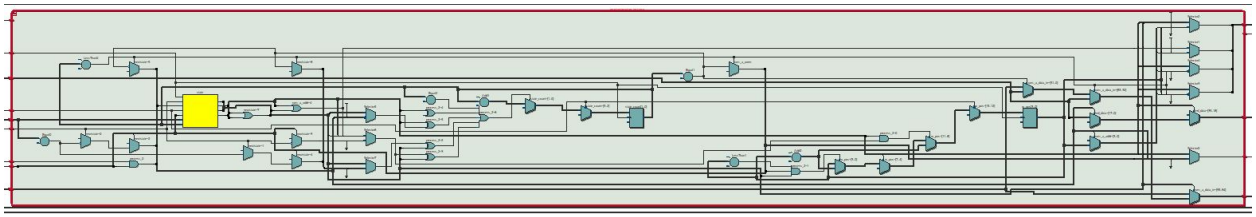
The above figure shows the RTL view of our main block. The large yellow box on the left represents the state machine with its various initialization steps. To the left, various comparators can be seen that help move the machine through its states at the correct times. You can then see a series of OR gates and muxes that control the logic for cases that should occur when transitioning between given states, and that also end up resetting the state count. There are three registers a little further to the right that hold the values of the push button, the state count and the slide switches. The column of muxes in the top right is used to drive the LED output logic. It is complicated because our design lights up a unique LED sequence during each initialization step for debugging purposes. The bottom right contains combinational logic that primarily drives the appropriate bits to the ethernet module's input data bus.

Sender RTL



This is the sender RTL view for our design. On the left are a couple of muxes with XOR gates that implement logic for what should happen when our state machine transitions from one specific state to another given state. Past this are a couple of circles that implement an equalto comparator and an addition operator for the state_count variable. The remainder of the logic is primarily to perform certain actions based on the state. Finally, the yellow box on the right implements the state machine for this module.

Receiver RTL



The figure above is the receiver module's RTL. The state machine is clearly visible on the left, with the combinational logic to the left of it that controls moving between the states. To the right of the state machine is the logic used to reset the state_count, which is held in the register in the middle of the figure. To the right of that is the logic used to control the ram, such as the register used to hold the current write address to the ram. Finally, there are some output MUXs to control the outputs to the port A of the RAM as well as the LED outputs.

Timing Analysis

After declaring the 50MHz, 1 MHz, and 25 MHz clocks, our design met timing. A few warnings were generated by the Ethernet Core IP, but they did not seem significant enough to render the design nonfunctional, and the design worked fine on the board even with these warnings.

Appendix

Code is attached next.