EE 371 Autumn 2016 - Labs 4 & 5

William Li, Dawn Liang, Jun Park

December 7, 2016

Signatures We certify that the work in this report is our own, and that any work that is not ours is cited.

William Li	Dawn Liang	Jun Park
_____	_____	_____
Signature	*Signature*	*Signature*
_____	_____	_____
Date	*Date*	*Date*

Contributions Jun helped design and write the hardware and software modules as well as wrote some of the lab report. William wrote most of the C programs and the lab report. Dawn wrote the hardware modules and some of the software for Labs 4 & 5, and compiled the lab report.

Contents

1 ABSTRACT

In lab 4, we developed a NIOS II microprocessor using Qsys in Quartus. We then wrote simple C programs in Eclipse to run on our microprocessor. This allowed us to practice communicating between the processor and the hardware, by generating commands from the microprocessor to our scanner module and sending signals from the scanner module back to the microprocessor.

In lab 5, we continued working with the QSys tools and concepts from lab 4 and created an asynchronous serial network to transfer data from a scanner system to a base station. We then worked with another group to transmit data between the two boards, displaying the ASCII character representation of data we received on our Eclipse console. One of the boards modeled the scanner on the gondola while the other modeled the base station, and they cycled through the roles.

2 INTRODUCTION

Lab 4 We designed and tested a NIOS II microprocessor on the DE1_SoC board. The NIOS II microprocessor was designed within the Qsys environment of Quartus. First, we ran simple C template programs on the microprocessor (Count Binary, Lights and Switches, Hello World Small). Then we modified Hello World Small to accept user inputs in the Eclipse console, to activate the switching behaviour, and to read board information to modify behavior, using a switch to complement the normal behavior. Ultimately, this lab gave us a chance to communicating between the scanner (from the Eclipse console) and the board's inputs and outputs. The lab illustrated how the verilog modules, the NIOS II microprocessor, and the C program all interact with one another.

Lab 5 We designed a serial communication system for our scanner and base station to communicate with another groups scanner and base station. The data is turned into parallel data for storage on each board's scanner systems' data buffers, but is transmitted serially. The data we received is then displayed on the Eclipse console for the user to see. Once again, we are modeling the scanner system on the gondola and the base station on the ground.

3 DISCUSSION

3.1 Design

3.1.1 Design Specification

Lab 4 For our climate data collection system, we must modify our previous scanner system to be able to store 10 pieces of 8-bit data. It must be able to take 8-bits of input data and produce 8-bits of output data when it is transferring. Additionally, the system must interface with Eclipse, and be controlled using textual commands and read information from the hardware.

Lab 5 Our overall scanner/home base must include an asynchronous serial communication interface, a scanner system, and a computer-user interface.

The communication system enables the board must be able to transmit data serially to another board, as well as interface with a processor to receive commands and send data for display. It must take a 10-bit serial input and produce an 11-bit output, including a start bit, 8-bit data, and stop bit (in the case of output, a dummy bit is used to represent the quiescent state, for start bit detection).

The user interface, implemented using C running on our NIOS II processor, must take textual input via Eclipse and generate control signals for the scanner system, as well as receive data from the communication/scanner systems and display it. Namely, it is in charge of issuing the startScanning and transfer commands, and it must retrieve the scanner's readyToTransfer and data signals and the communication system's incoming data.

The scanner system stores the data received in the communication system. It must store pieces of 8-bit data, alternating between two scanners to maintain continuous collection. Additionally, it must transfer the data to the communication system when prompted by the user interface, reading from the scanners' memory; otherwise, it flushes out data when prompted by the other scanner, to maintain the cyclic scanning system and continuous data collection.

3.1.2 Design Procedure

Lab 4 The majority of lab 4 was following the procedures detailed in the provided documents NIOSII_hardware_tutorial.pdf, Introduction_to_Qsys_Tool.pdf, NIOS II_swDeveloperHB.pdf, and DE1-SoC_User_manual.pdf. By studying these examples, we adapted our pre-existing framework to building our final board and processor system, which involved communicating with our scanner system in hardware.

Lab 5 There were three main components to lab 5: the serial-parallel communication interface, the scanner system, and the NIOS II processor and its related software.

For the serial communication interface, we built our system based off the block diagram provided in the lab instructions, further modularised for ease of testing. We subdivided the system into a PISO and SIPO section, each of which included a bitCounter and a shift register. The bitCounter included a bsc and bic, for keeping track of the data frame being received/transmitted. Each module was simulated using iverilog and gtkwave to ensure functionality.

For the scanner system, we reused our old scanner system, slightly modified to accomodate the data I/O and control signals from the communication and user interface systems respectively.

The NIOS II processor was made from our old templates, and we used the same principles to write our software as we did in lab 4. As the main function of the Eclipse program was to take textual user inputs to generate contral signals and read board information to display in the console, the program was relatively simple.

3.1.3 System Description

Lab 4 We built three processors to run various C programs.

- count_binary.c: the most basic processor, meant only to flash LEDs in a binary counting fashion.
- lights_and_switches.c & modified: similar processor, reads the state of the switches and uses it to control the LEDs. Contains a cpu, ram, jtag uart, and PIOs for switches and LEDs. We then modified the program such that SW0 complemented the state of the LEDs. The lights_and_switches software was provided by the tutorial, and we added a switch-reading component for the modified version.
- scannerIO: takes text input from Eclipse and uses it to generate commands for the scanner system, as well as reads the state of the scanner system and displays it on the console. Contains a cpu, ram, jtag uart, and PIOs for control signals. The software was based off our modified lights_and_switches program; it takes textual user inputs to the console to generate control signals, and displays information from the board in the console.

Lab 5 There were three components to lab 5, which should come together to create a scanner/home base system.

- communication system: the interface takes a serial input and converts it to a parallel output, as well as takes a parallel input and turns it into a serial output.
- scanner system: the scanner system is the same as the one used in Lab 4. It begins scanning when it receives the startScanning command, at which point it begins storing the 8-bit data inputs that it receives. It outputs the data stored in memory when given a transfer command, or flushes a scanner and continues storing data continuously.
- NIOS II processor & c program: the c program takes user inputs and issues commands to the other two components, as well as reads information from the board and displays it to the user as an ascii character. The commands generated control the behaviour of the communication and scanner systems.

3.1.4 Software Implementation

Lab 4

- count_binary.c is from the Eclipse template and was used to explore basic NIOS to C interaction.
- lights_and_switches.c is from page 26 of the Introduction_to_Qsys_Tool.pdf. It was used to explore the use of base addresses, and the need to declare variables as (volatile char *) to temporarily disable compiler optimization. As well as having the proper reset condition when loading the program onto the board (its why for the reset, key is prefered over switch).
- hello_world_small.c is from the Eclipse template and was used to explore displaying a message from the NIOS II to the console. The modified version of hello_world_small that accepts a user input, which in turn activates the behavior seen in the lights_and_switches program. Additionally, turning on SW0 complements that behaviour. It is worth noting that the alt form of the stdio had to be used, since the NIOS II we built had limited memory capacity.
- The scanner control program takes texual user inputs and uses them to control the scanner and communications systems. Based on the value of the character entered, different control lines are switched on/off.

Lab 5 We did not manage to complete the software part of Lab 5, but we would have designed the user interface to print prompts for user input and convert textual inputs into commands for the scanner and communication systems. We attempted bit-masking for interaction with signals from the board in order to display them to the console, however we were unsuccessful.

3.1.5 Hardware Implementation

Lab 4 We modified our previous scanner system from Lab 3 to include a data buffer, holding 10 pieces of 8-bit data. Otherwise, the hardware was exactly the same as before.

The NIOS II processors all contain a cpu, ram, jtag uart, and PIOs. The PIOs depended on which board I/Os we were using, e.g. switches or LEDs.

Lab 5 Our entire system consists of the serial-parallel interface, the NIOS II processor, and the scanner system.

The serial-parallel interface was constructed exactly as described in the provided block diagram. The communication is divided into a serial-to-parallel side and a parallel-to-serial side. Each direction includes a bitCounter and a shift register. For the bit counter: data is transmitted at 16-samples per bit, so there is an internal sample/bit counter that keeps track of how much data has been transmitted. The input frame is 10-bits (start, 8-bits of data, stop), and the output frame is 11-bits (quiescent, start, 8-bits of data, stop). For both conversion directions, a shift register is used to convert between serial and parallel, and the bitCounter signals when te end of the frame has been reached and all data has been transmitted.

The NIOS II processor is almost exactly as the ones built in lab 4. We used a NIOS II/e processor, as we did not need the high-level functionality of the NIOS II/f or NIOS II/s, and used 8192 bits of memory. We included PIOs for the 8-bits of data going in/out of the processor to the serial/parallel interface, and control/communication signals for the scanner and communication systems. For the scanner system, these were startScanning and transfer as control outputs, and readyToTransfer as signals read from the system. For the communication system, these were load and transmitEnable as control outputs, and charReceived and charSent as signals read from the system.

The scanner system is exactly the system we used in lab 4, with a few slight modifications so that the data being input/output communicates with the NIOS II processor rather than dummy data used in lab 4.

3.2 Test

3.2.1 Test Plan

Lab 4 Each of the C programs drawn from the tutarial were tested on the microprocessor we built. Building our microprocessor and running count_binary, lights_and_switches, hello_world_small were done entirely from the tutorials. Therefore to test them, double check that all steps outlined in the tutorial were followed correctly and check to see that the outputs to the console match that of those mentioned in the tutorial.

Lab 5 We tested our communication and scanner systems in simulation using iverilog and gtkwave. The communication system had to take serial/parallel inputs and produced parallel/serial outputs. The framing of the inputs/outputs should be correct. The scanner system had been tested in previous labs; it should take 8-bit inputs and store them, then retrieve them upon transferring. It should continuously receive and store data, flushing when necessary, and transfer when prompted. We were unable to get the software part of Lab 5 working; if we had, we would check to ensure that it communicates first with itself, and then another board, properly.

3.2.2 Test Specification

Lab 4 To test the microprocessor, each template C program should function properly.

- For count_binary: check that it is counting up starting from 00 indefinitely until stopped by the user.
- For the hello_world_small the correct behavior is to display Hello World to the Eclipse console.
- For lights_and_switches: the switches on the DE1_SoC board should turn on their corresponding leds.
- For the modified lights_and_switches: test that the lights_and_switches behavior only occurs if the user input is a G, otherwise the program should continue to wait for a G input. Additionally, turning on SW0 should invert the LEDs
- For the scanner system: the HEX displays and leds should demonstrate the correct scanner state and buffer data, ensuring that the scanner system is functioning as before. Additionally, the StartScanning and Transfer signals should be controlled by user inputs in Eclipse, and the the console should receive the readyToTransfer signal from the scanner. The console should print out the readyToTransfer signal when it is received from the scanner.

Lab 5 Each of the subsystems should function properly.

- NIOS II: the processor should communicate both with the user and with the other two systems. Typing a command should generate the corresponding control signal on the board, e.g. 's' for startScanning, 't' for transfer. The signals issued from the other two systems (communication and scanner) should appear on the console at the appropriate times.
- communication system: the communication system should take a serial input and produce a parallel output, or take a parallel input and produce a serial output. The parallel output should pick only the centre 8-bits of the 10-bit shift register, and output a charReceived signal when it reaches the end bit (10th bit). The serial output should pick only the least significant bit, and output a charSent signal when it reaches the end bit (11th bit).
- scanner system: the scanner system should store the data applied to the input, alternating between the two scanners to continuously receive and store data. Since the system is from lab 3, testing need not be as rigorous.
- overall, the system should successfully pass an 8-bit ascii character between itself as well as with another board. We should ensure that slight delays in transmission and bit-counting do not corrupt the data. It should also display the value to the console of the user interface and on the HEX displays of the board, and the control signals to the LEDs.

3.2.3 Test Cases

Lab 4

countBinary - reset case

- Input: Reset set to 1
- Measure: Nothing
- Pass condition: LEDs output the value 0, then 1, then 2, and so on in binary. The Eclipse console should display the numbers counting in increasing order.

countBinary - pass case

- Input: Reset set to 1 then 0
- Measure: Nothing
- Pass Condition: LEDs continuously counting upwards in binary.The Eclipse console should display the numbers counting in increasing order.

countBinary - fail case

- Input: Reset set to 1, then to 0
- Measure: Nothing
- Pass condition: LEDs do not display the numbers in binary, the Eclipse console does not display any numbers

lights and switches - reset case

- Input: Reset set to 1, switches [9:0] set to 0
- Measure: Nothing
- Pass condition: LEDs should not output anything

lights antd switches - pass case

- Input: Reset set to 1 then 0, switches [9:0] asserted to 1
- Measure: Nothing
- Pass condition: LEDs [9:0] should be all lit

lights and switches - fail case

- Input: Reset set to 1 then 0, switches [9:0] asserted to 1
- Measure: Nothing
- Pass condition: LEDs[9:0] do not light up when the respective switch is asserted to 1

scanner/base station - reset case

- Input: Reset set to 1
- Measure: state and data buffer
- Pass condition: state = lowPower, data_buffer = 0 (lowPower mode and empty data buffer)

scanner/base station - pass cases

- turning on the scanners
  - Input: initialOn set to 1
  - Measure: The states of each scanner
  - Pass condition: One scanner should be in scanning mode, and the other should be in lowPower mode
- transferring
  - Input: startTransfer set to 1
  - measure: state of the scanner
  - Pass condition: If idle, the scanner should go to transferring mode. Otherwise, the scanner should hold its current state
- communication with self
  - Input: input = output while scanning
  - Measure: output
  - Pass condition: the system should pass a value to itself and produce it again
- communication with other board
  - Input: input from other board
  - Measure: output
  - Pass condition: the system should produce the value that it is fed from the other board

scanner/base station - fail cases

- quiescent state
  - Input: initialOn set to 0, startTransfer set to 1
  - measure: state of the scanner
  - Pass condition: Nothing should happen, as the scanners havent been turned on yet
- premature transfer
  - Input: startTransfer set to 1 while not in the idle state
  - measure: State of the scanner
  - Pass condition: Nothing should happen, since the scanner isnt in the idle state
- transfer v. flushing
  - Input: startTransfer set to 1 and Flush = 1
  - Measure: state of the scanner
  - Pass condition: the scanner should prioritise transferring over flushing, so it should go to the transferring state

Lab 5

SPS interface - reset case

- Input: Reset to 1
- Measure: outputs of the interface
- Pass condition: the interface should override inputs for the quiescent state, i.e. all outputs go to 1

SPS interface - pass case

- Input: 8-bit parallel input and serial input
- Measure: serial output and 8-bit parallel output, charReceived and charSent
- Pass condition: the serial output should change with each bit of the parallel input; the parallel output should accumulate the serial inputs. charReceived and charSent should trigger when the frames end

Our scanner test cases were the same as from Lab 3, and from earlier in Lab 4.

We did not manage to get our C Program working, but if we were to test it, we would ensure that it generates the correct commands from the correct inputs ('s' starts scanning, 't' starts transferring and automatically prompts the load and transmitEnable signals) and correctly displays data (data coming in to the SPS interface, data being transmitted via the SPS interface).

4 RESULTS

Lab 4 We successfully completed the requirements as stated in the documentation. We were able to get the countBinary design to successfully run our NIOS II processor; the LEDs on the FPGA were counting up in binary. In addition, we were able to we were able to get Lights and Switches operating correctly as well on the FPGA (the switches were able to turn off their respective LEDs). We finally took a template (Hello World Small) program and modified the code base to be able to run our scanner/base station system; we were thus able to operate the entire system off the Eclipse console. The Eclipse system had displayed the status of the scanner/base station on our console, we were able to interact with the system by typing in key characters to the console.

Lab 5 We successfully created the hardware modules for the system. We thoroughly tested each module in GTKWave and used SignalTap to probe our designs; everything functioned as expected. However, the C code that was to be uploaded onto our system did not work (possibly due to our misunderstanding of how C is written for a NIOS II processor, and other unforeseeable errors that were out of our control). In these cases, we had tried every possible solution to remedy our errors, but our efforts in the end proved to be futile.

4.1 Analysis of Errors

In both labs, we had experienced many problems with uploading our C code (for both labs) up onto the NIOS II processor. Eclipse gave us many errors when we tried to upload the given code onto our NIOS II processor, often to do with the .elf file (broken or was not existent). In addition to this software bug, the documentation given to us to help build the project had significantly differed from the expectations from the professor; many times this had created a lot of confusion when we tried to build our design. To help solve this error, we had to clean and rebuild our project several times; sometimes this method did not work, so we had to build a completely new project and copy over our C program. In the end, we realized that when we uploaded the code onto the NIOS II processor, the reset switch had to be asserted off in order for the code to successfully upload onto the processor.

In addition, we had several issues with communicating between the hardware on the board and the NIOS II processor. We built and thoroughly tested the hardware system in simulation (the NIOS II Processor, communication and scanning systems), but struggled to get the C program running on our system. These were due both to unfamiliarity with the environment and libraries, as well as issues with Eclipse itself. For example, we attempted to read the readyToTransfer signal received from the board, but all approaches failed. Even with the help of other groups and TAs, we could not get it working. Therefore, we were unable to run the smoke test, and were thus unable to talk our partner groups FPGA. We tried rectifying this issue by employing SignalTap, GTKWave, and various debuggers to identify our problems; however none of these tools managed to diagnose the issue, and we do not understand how our hardware and software pieces for Lab 5 could not come together and produce a satisfying result.

5 SUMMARY & CONCLUSION

Summary The focus of this project was to take what we had learned in our previous labs and build a complete scanner/base station data collection system with data networking. The entire system was designed to mimic a real world scenario where we had to be able to engineer a complete digital design with limited specification information from the ground up.

Conclusion Overall, this project gave us the chance to design an advanced digital logic systems in VHDL with a NIOS II microprocessor. We also learned basic networking protocols and applied it to our project; we also learned how to implement a serial based communication network that interacted with another groups scanner/base station system. We were also able to improve our VHDL and debugging skills, and had another chance to practice our skills coding in C.

6 APPENDIX

6.1 Lab 4

6.1.1 Block Diagrams

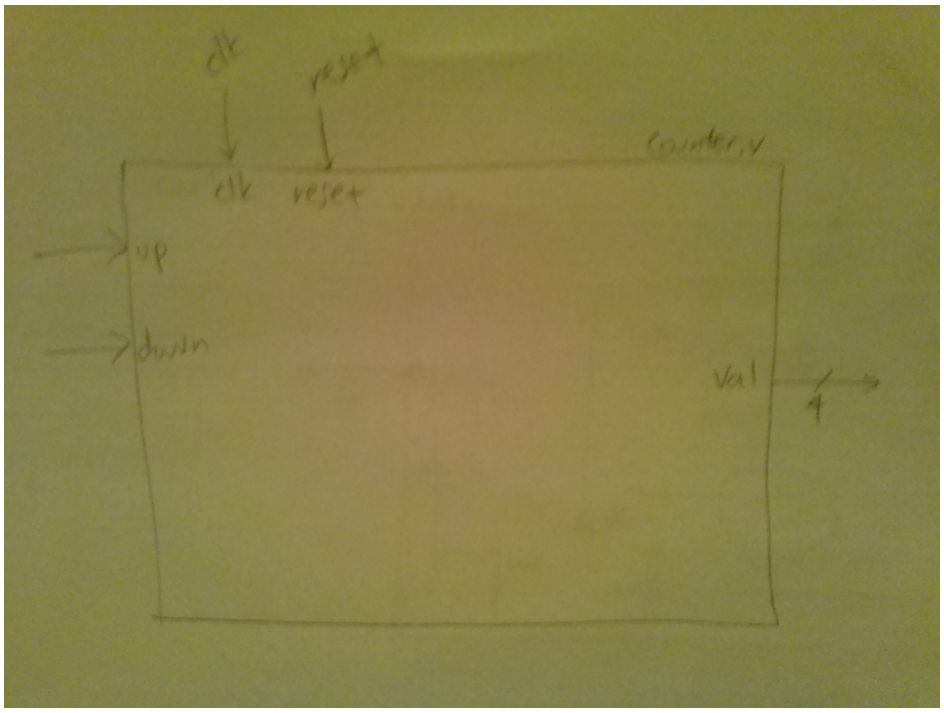

Figure 1: counter module block diagram

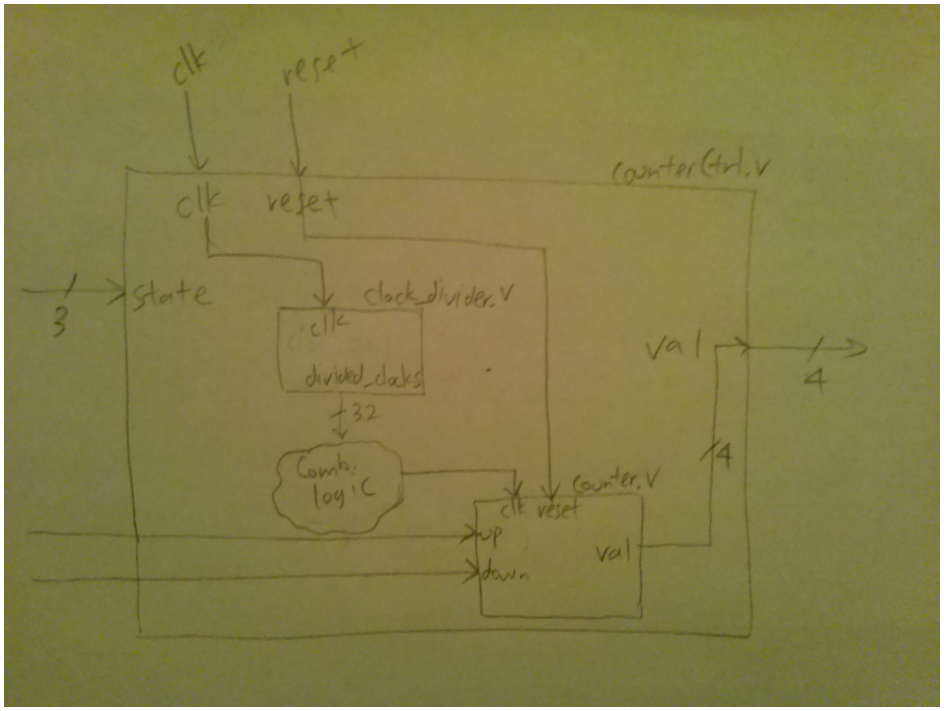

Figure 2: counterCtrl module block diagram

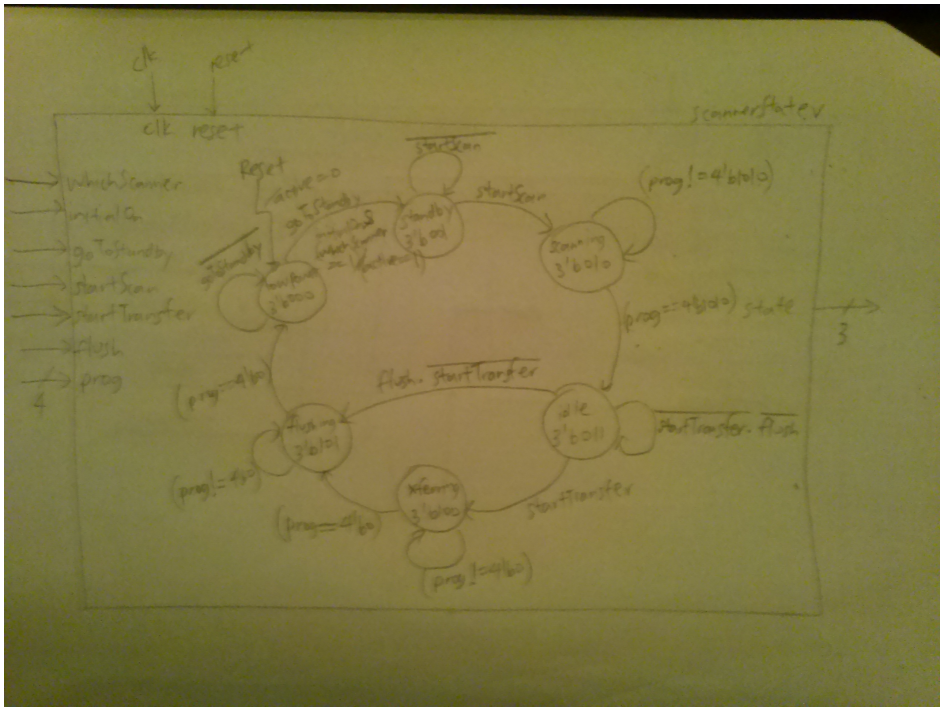

Figure 3: state transition diagram for the scanners

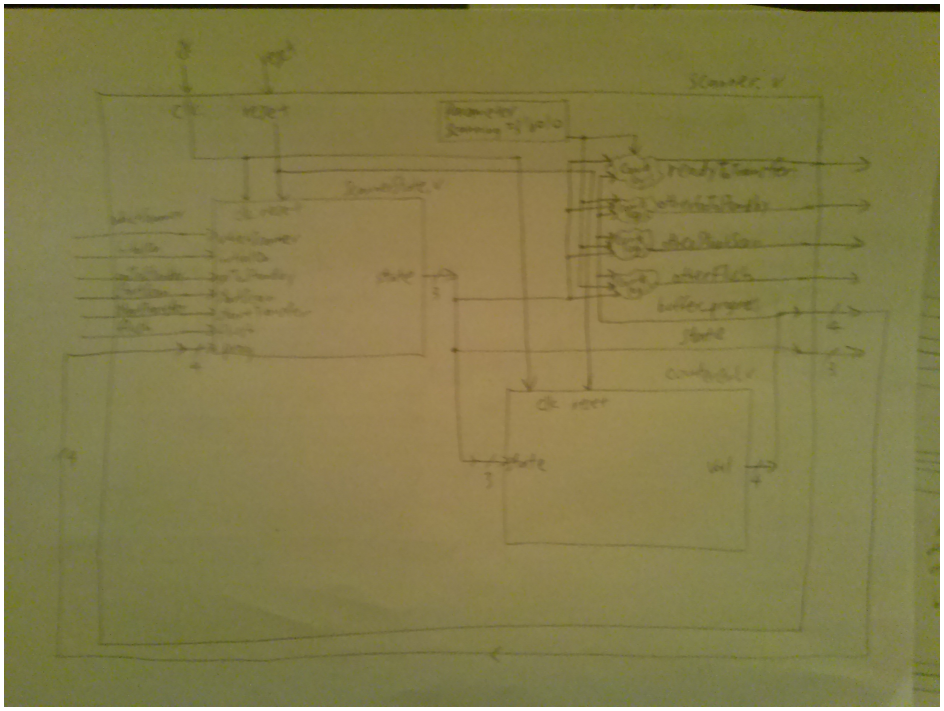

Figure 4: scanner module block diagram

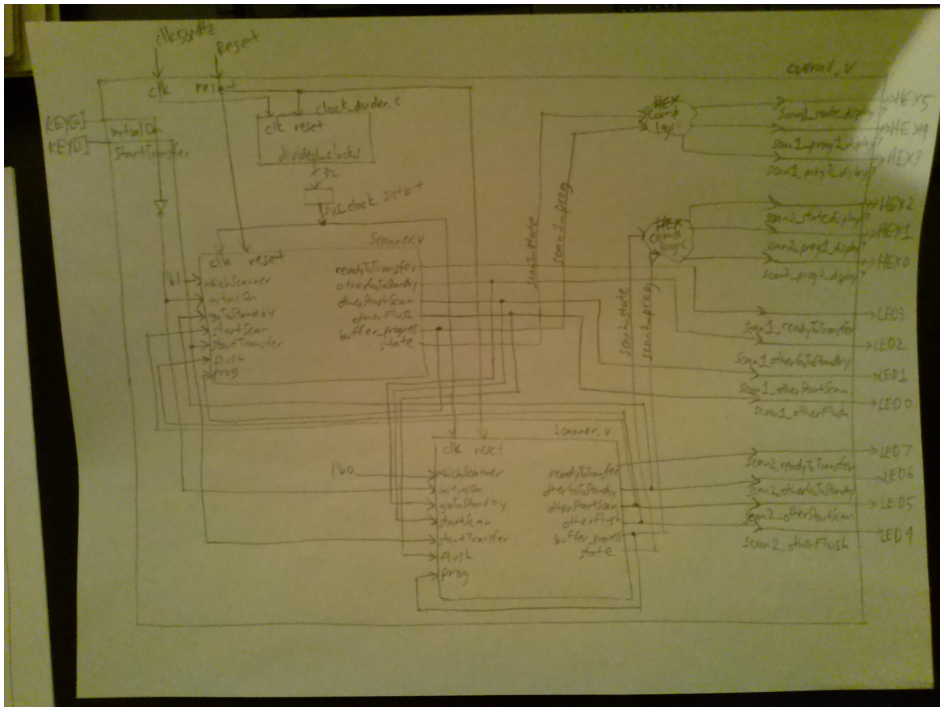

Figure 5: overall module block diagram

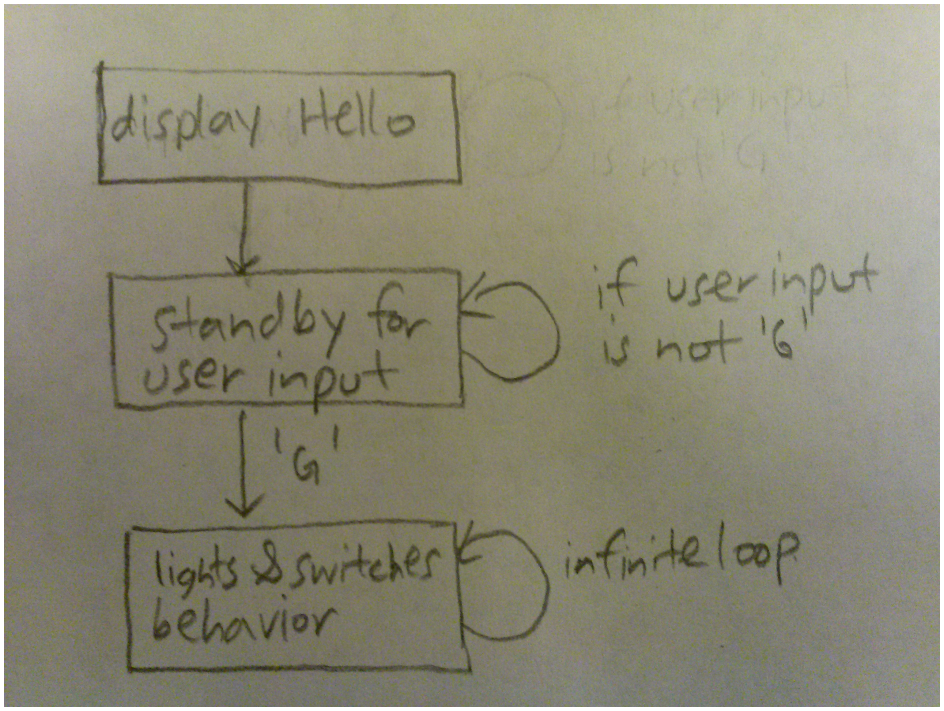

Figure 6: modified lights and switches block diagram

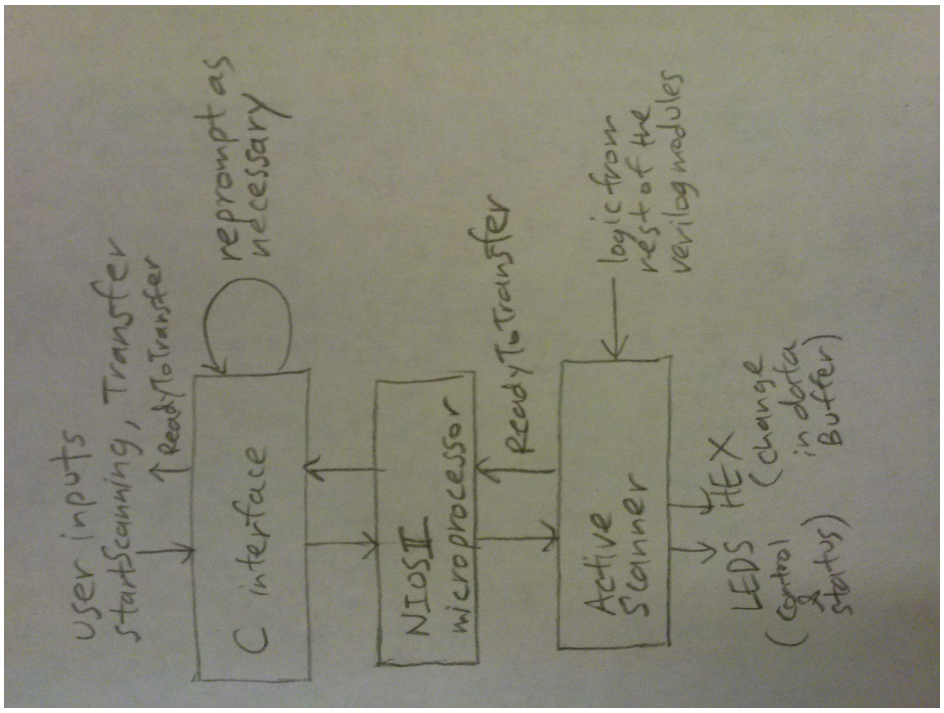

Figure 7: scannerIO user interface block diagram

6.1.2 Verilog

```
// EE371 Lab4 Autumn 2016
```

```
// Authors: Dawn Liang, Jun Park, William Li
// Date: 13 Nov 2016
//
// counter that counts every clock edge up/down to a min/max (0/10)
// resets to 0
module counter(data_out, data_in, percentage, up, down, clk, reset);
  output [3:0] percentage;
  input up, down, clk, reset;
  input [7:0] data_in;
  output [7:0] data_out;

  reg [7:0] mem [9:0]; // 10 x 8-bit memory

  // initialise the memory
  initial begin
    ps = 0;
    mem[0] = 8'b11111111;
    mem[1] = 8'b11111111;
    mem[2] = 8'b11111111;
    mem[3] = 8'b11111111;
    mem[4] = 8'b11111111;
    mem[5] = 8'b11111111;
    mem[6] = 8'b11111111;
    mem[7] = 8'b11111111;
    mem[8] = 8'b11111111;
    mem[9] = 8'b11111111;
  end

  // combinational logic: count every clock edge
  // min/max set at 0/10
  reg [3:0] ps, ns;
  always@(*) begin
    if (up && ~down && (ps < 10)) begin
      ns = ps + 1;
    end else if (down && ~up && (ps > 0)) begin
      ns = ps - 1;
    end else begin
      ns = ps;
    end
  end

  // output logic
  assign percentage = ps;
  assign data_out = mem[ps];

  // sequential logic; reset & next state
  always@(posedge clk) begin
    if (reset) begin     // clear data
      ps <= 0;
      mem[0] <= 8'b11111111;
      mem[1] <= 8'b11111111;
      mem[2] <= 8'b11111111;
      mem[3] <= 8'b11111111;
      mem[4] <= 8'b11111111;
      mem[5] <= 8'b11111111;
      mem[6] <= 8'b11111111;
      mem[7] <= 8'b11111111;
      mem[8] <= 8'b11111111;
      mem[9] <= 8'b11111111;
```

```
    end else begin
      ps <= ns;
      if (down && ~up) begin   // xfer/flushing; erase data
        mem[ps] <= 8'b11111111;
      end else if (ps < 9) begin  // save new value
        mem[ps] <= data_in;
      end else begin      // hold last value (full)
        mem[ps] <= mem[ps];
      end
    end
  end
endmodule
```

```
// EE371 Lab3 Autumn 2016
// Authors: Dawn Liang, William Li, Jun park
// Date: 13 Nov 2016
//
// testbench for the counter module; tests all counting conditions
module counter_testbench();
  reg up, down, clk, reset;
  wire [3:0] percentage;
  wire final_clock;
  wire [7:0] data_out, data_in;

  assign data_in = percentage;

  counter dut (.data_out(data_out), .data_in(data_in), .percentage(percentage),
      .up(up), .down(down), .clk(clk), .reset(reset));

  // set up clock
  parameter CLOCK_PERIOD = 10;
  initial clk = 0;
  always begin
    #(CLOCK_PERIOD/2);
    clk = ~clk;
  end

  // begin simulation
  initial begin
    reset <= 1;            @(posedge clk);
                          @(posedge clk);
    reset <= 0; up <= 0; down <= 0;  @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
          up <= 1;        @(posedge clk); // 0
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk); // 5
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk); // 10
                          @(posedge clk);
                          @(posedge clk);
                down <= 1;  @(posedge clk);
```

```
            up <= 0;      @(posedge clk); // 10
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk); // 5
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk); // 0
                          @(posedge clk);
                down <= 0;  @(posedge clk);
                          @(posedge clk);
    $finish;
  end

  // gtkwave filedump
  initial begin
    $dumpfile("counter.vcd");
    $dumpvars;
  end
endmodule
```

```
// EE371 Lab3 Autumn 2016
// Authors: Dawn Liang, Jun Park, William Li
// Date: 13 Nov 2016
//
// FSM controlling the state of the scanner
module scannerState(state, whichScanner, initialOn, goToStandby, startScan, prog,
    startTransfer, flush, clk, reset);
  output [2:0] state;
  input whichScanner, initialOn, goToStandby, startScan, startTransfer, flush;
  input [3:0] prog;
  input clk, reset;

  // state encodings
  parameter lowPower = 3'b000,
            standby = 3'b001,
            scanning = 3'b010,
            idle = 3'b011,
            xferring = 3'b100,
            flushing = 3'b101;

  // combinational logic: next state
  reg [2:0] ps, ns;
  initial ps = lowPower;
  reg active;
  initial active = 0;
  always@(*) begin
    if (active) begin
      case (ps)
        lowPower: begin
                  if (goToStandby) ns = standby;
                  else ns = ps;
                end
        standby: begin
                  if (startScan) ns = scanning;
```

```
                else ns = ps;
              end
        scanning: begin
                if (prog == 4'b1010) ns = idle;
                else ns = ps;
              end
        idle:  begin
                if (startTransfer) ns = xferring;
                else if (flush) ns = flushing;
                else ns = ps;
              end
        xferring: begin
                if (prog == 4'b0) ns = lowPower;
                else ns = ps;
              end
        flushing: begin
                if (prog == 4'b0) ns = lowPower;
                else ns = ps;
              end
        default: ns = ps;
      endcase
    end else begin
      if (initialOn) begin
        if (whichScanner) ns = scanning;
        else ns = lowPower;
      end else begin
        ns = lowPower;
      end
    end
  end

  // output logic
  assign state = ps;

  // sequential logic: reset, next state
  always@(posedge clk) begin
    if (reset) begin
      ps <= lowPower;
      active <= 0;
    end else begin
      ps <= ns;
      if (initialOn) active <= 1;
      else active <= active;
    end
  end
endmodule
```

```
// EE371 Lab3 Autumn 2016
// Authors: Dawn Liang, William Li, Jun Park
// Date: 13 Nov 2016
//
// testbench for scannerState FSM; tests all state transitions
module scannerState_testbench();
  reg clk, reset;
  reg goToStandby, startScan, startTransfer, flush, initialOn;
  reg [3:0] prog;
  wire [2:0] state;
```

```
  // set up clock
  parameter CLOCK_PERIOD = 10;
  initial clk = 0;
  always begin
    #(CLOCK_PERIOD/2);
    clk = ~clk;
  end

  scannerState dut (.state(state), .initialOn(initialOn), .prog(prog),
      .goToStandby(goToStandby), .startScan(startScan),
    .startTransfer(startTransfer), .flush(flush), .clk(clk), .reset(reset));

  initial begin
    reset <= 1;                                                    @(posedge clk);
                                                                  @(posedge clk);
    reset <= 0; goToStandby <= 0; startScan <= 0; startTransfer <= 0; flush <= 0;
        initialOn <= 0; prog <= 4'b0; @(posedge clk);
                                              initialOn <= 1;        @(posedge
                                                  clk);
          goToStandby <= 1;                     initialOn <= 0;        @(posedge
              clk);
          goToStandby <= 0;                                            @(posedge clk);
                      startScan <= 1;                                    @(posedge
                          clk);
                      startScan <= 0;                          prog <=
                          4'b1010;@(posedge clk);
                                                                       @(posedge clk);
                               startTransfer <= 1;                        @(posedge
                                   clk);
                               startTransfer <= 0;             prog <= 4'b0;
                                   @(posedge clk);
          goToStandby <= 1;                                            @(posedge clk);
          goToStandby <= 0; startScan <= 1;                               @(posedge
              clk);
                      startScan <= 0;                          prog <=
                          4'b1010;@(posedge clk);
                                        flush <= 1;                 @(posedge clk);
                                        flush <= 0;       prog <= 4'b0; @(posedge
                                            clk);
                                                                       @(posedge clk);
    $finish;
  end

  // gtkwave filedump
  initial begin
    $dumpfile("scannerState.vcd");
    $dumpvars;
  end
endmodule
```

```
// EE371 Lab4 Autumn 2016
// Authors: Jun Park William Li Dawn Liang
// Date: 06 Dec 2016
//
// controls a counter that counts up/down at different speeds depending on
// the state input
module dataBuffer(data_out, data_in, percentage, state, clk, reset);
  input [2:0] state;
```

```
  input [7:0] data_in;
  input clk, reset;

  output [7:0] data_out;
  output [3:0] percentage;

  // state encodings
  parameter lowPower = 3'b000,
         standby = 3'b001,
         scanning = 3'b010,
         idle = 3'b011,
         xferring = 3'b100,
         flushing = 3'b101;

  // clock division
  wire [31:0] divided_clocks;
  clock_divider div (.divided_clocks(divided_clocks), .clk(clk));
  reg final_clock;

  // control signals
  reg up, down;
  always@(*) begin
    case (state)
      lowPower: begin
                final_clock = divided_clocks[0];
                up = 0; down = 0;
              end
      standby: begin
                final_clock = divided_clocks[0];
                up = 0; down = 0;
              end
      scanning: begin                    // count up at 1x speed
                final_clock = divided_clocks[3];
                up = 1; down = 0;
              end
      idle:  begin
                final_clock = divided_clocks[0];
                up = 0; down = 0;
              end
      xferring: begin                    // count down at 4x speed
                final_clock = divided_clocks[1];
                up = 0; down = 1;
              end
      flushing: begin                    // count down at 8x speed
                final_clock = divided_clocks[0];
                up = 0; down = 1;
              end
      default: begin
                final_clock = divided_clocks[0];
                up = 0; down = 0;
              end
    endcase
  end

  // counter
  counter prog (.data_out(data_out), .data_in(data_in), .percentage(percentage),
      .up(up), .down(down), .clk(final_clock), .reset(reset));
endmodule
```

```
// EE371 Lab4 Autumn 2016
// Authors: Dawn Liang, Jun Park, William Li
// Date:
//
// testbench for dataBuffer module; tests all possible state inputs
module dataBuffer_testbench();
  reg [2:0] state;
  reg clk, reset;
  wire [3:0] percentage;
  wire [7:0] data_in, data_out;
  assign data_in = percentage;

  dataBuffer dut (.data_out(data_out), .data_in(data_in), .percentage(percentage),
      .state(state), .clk(clk), .reset(reset));

  // set up clock
  parameter CLOCK_PERIOD = 10;
  initial clk = 0;
  always begin
    #(CLOCK_PERIOD/2);
    clk = ~clk;
  end

  initial begin
    reset <= 1;          @(posedge clk);
                        @(posedge clk);
    reset <= 0; state <= 3'b000; @(posedge clk); // lowPower
                        @(posedge clk);
                        @(posedge clk);
          state <= 3'b001; @(posedge clk); // standby
                        @(posedge clk);
                        @(posedge clk);
          state <= 3'b010; @(posedge clk); // scanning
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
                        @(posedge clk);
```

```
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
          state <= 3'b011; @(posedge clk); // idle
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
          state <= 3'b100; @(posedge clk); // xfer
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
          state <= 3'b101; @(posedge clk); // flush
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
    $finish;
  end

  // gtkwave filedump
  initial begin
    $dumpfile("dataBuffer.vcd");
    $dumpvars;
  end
endmodule
```

```
// EE371 Lab4 Autumn 2016
// Authors: Dawn Liang, William Li, Jun Park
// Date: 06 Dec 2016
//
// scanner module including FSM and counter for control and memory systems
// outputs current state & buffer progress, and control signals for other
// scanner
module scanner(data_out, buffer_progress, state, readyToTransfer, otherGoToStandby,
    otherStartScan, otherFlush,
          data_in, whichScanner, initialOn, goToStandby, startScan, startTransfer,
              flush, clk, reset);
  input clk, reset;
```

```
  input initialOn, goToStandby, startScan, startTransfer, flush, whichScanner;
  input [7:0] data_in;

  output readyToTransfer, otherGoToStandby, otherStartScan, otherFlush;
  output [3:0] buffer_progress;
  output [2:0] state;
  output [7:0] data_out;

  parameter scanning = 3'b010;

  scannerState ctrl (.state(state), .whichScanner(whichScanner),
      .initialOn(initialOn), .goToStandby(goToStandby), .startScan(startScan),
     .prog(buffer_progress), .startTransfer(startTransfer), .flush(flush), .clk(clk),
        .reset(reset));
  dataBuffer mem (.data_out(data_out), .data_in(data_in),
      .percentage(buffer_progress), .state(state), .clk(clk), .reset(reset));

  assign readyToTransfer = (buffer_progress == 5 && state == scanning);
  assign otherGoToStandby = (buffer_progress == 8 && state == scanning);
  assign otherStartScan = (buffer_progress == 9 && state == scanning);
  assign otherFlush = (buffer_progress == 5 && state == scanning);
endmodule
```

```
// EE371 Lab3 Autumn 2016
// Authors: Dawn Liang, William Li, Jun Park
// Date: 13 Nov 2016
//
// testbench for scanner module: tests normal operationg conditions
module scanner_testbench();
  reg clk, reset;
  reg initialOn, goToStandby, startScan, startTransfer, flush;
  wire readyToTransfer, otherGoToStandby, otherStartScan, otherFlush;
  wire [3:0] buffer_progress;
  wire [2:0] state;
  wire [7:0] data_in, data_out;
  assign data_in = buffer_progress;

  // set up clock
  parameter CLOCK_PERIOD = 10;
  initial clk = 0;
  always begin
    #(CLOCK_PERIOD/2);
    clk = ~clk;
  end

  scanner dut (.buffer_progress(buffer_progress), .state(state),
      .readyToTransfer(readyToTransfer),
    .otherGoToStandby(otherGoToStandby), .otherStartScan(otherStartScan),
        .otherFlush(otherFlush),
    .data_out(data_out), .data_in(data_in), .whichScanner(1'b1),
    .initialOn(initialOn), .goToStandby(goToStandby), .startScan(startScan),
        .startTransfer(startTransfer), .flush(flush),
    .clk(clk), .reset(reset));

  initial begin
    reset <= 1;                                                    @(posedge clk);
                                                                 @(posedge clk);
    reset <= 0; initialOn <= 0; goToStandby <= 0; startScan <= 0; startTransfer <= 0;
```

```
flush <= 0;  @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
           goToStandby <= 1;                      @(posedge clk);
           goToStandby <= 0;                      @(posedge clk);
                                                @(posedge clk);
   initialOn <= 1;                                  @(posedge clk);
   initialOn <= 0;                                  @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
```

```
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
startTransfer <= 1;             @(posedge clk);
startTransfer <= 0;             @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
                          @(posedge clk);
```

```
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
goToStandby <= 1;                                 @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
            startScan <= 1;                         @(posedge clk);
goToStandby <= 0;                                 @(posedge clk);
                                                @(posedge clk);
            startScan <= 0;                         @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
```

```
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
```

```
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                flush <= 1;  @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                flush <= 0;  @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
                                                    @(posedge clk);
    $finish;
  end

  // gtkwave filedump
  initial begin
    $dumpfile("scanner.vcd");
    $dumpvars;
  end
endmodule
```

```
module scannerIO(HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LED, SW, clk, rst);
  input clk, rst;
  input [7:0] SW;
  input [3:0] KEY;
  output [9:0] LED;
  output [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;

  wire [31:0] divided_clocks;
  clock_divider cdiv (.clk(clk), .divided_clocks(divided_clocks));
  wire final_clock = divided_clocks[22];

  // state encodings
  parameter LOWPOWER = 3'b000;
  parameter STANDBY = 3'b001;
  parameter SCANNING = 3'b010;
```

```
parameter IDLE = 3'b011;
parameter TRANSFER = 3'b100;
parameter FLUSHING = 3'b101;

// scanner i/o signals
wire [7:0] s1_data_out, s1_data_in, s2_data_out, s2_data_in;
wire [2:0] s1_state, s2_state;
wire [3:0] s1_buffer_progress, s2_buffer_progress;

// scanner/scanner interaction signals
wire s1_readyToTransfer, s2_readyToTransfer,
     s1_otherGoToStandby, s2_otherGoToStandby,
     s1_otherStartScan, s2_otherStartScan,
     s1_otherFlush, s2_otherFlush;

// scanner/processor interaction signals
wire startScanning, transfer, readyToTransfer;
reg [7:0] data_transfer;

assign s1_data_in = s1_buffer_progress;             // input data is just the
    address of the mem slot
assign s2_data_in = s2_buffer_progress;

assign readyToTransfer = s1_readyToTransfer || s2_readyToTransfer; // if either
    scanner is ready to transfer
assign LED[9] = readyToTransfer;

always@(*) begin                                    // transfer data from currently
    transferring scanner
  if (s1_state == TRANSFER) data_transfer = s1_data_out;
  else if (s2_state == TRANSFER) data_transfer = s2_data_out;
  else data_transfer = 7'b0;
end

// scanner subsystem
scanner s1 (.data_out(s1_data_out), .buffer_progress(s1_buffer_progress),
    .state(s1_state), .readyToTransfer(s1_readyToTransfer),
  .otherGoToStandby(s1_otherGoToStandby), .otherStartScan(s1_otherStartScan),
      .otherFlush(s1_otherFlush),
  .data_in(s1_data_in), .whichScanner(1), .initialOn(startScanning),
      .goToStandby(s2_otherGoToStandby),
  .startScan(s2_otherStartScan), .startTransfer(transfer), .flush(s2_otherFlush),
      .clk(final_clock), .reset(~rst));
scanner s2 (.data_out(s2_data_out), .buffer_progress(s2_buffer_progress),
    .state(s2_state), .readyToTransfer(s2_readyToTransfer),
  .otherGoToStandby(s2_otherGoToStandby), .otherStartScan(s2_otherStartScan),
      .otherFlush(s2_otherFlush),
  .data_in(s2_data_in), .whichScanner(0), .initialOn(startScanning),
      .goToStandby(s1_otherGoToStandby),
  .startScan(s1_otherStartScan), .startTransfer(transfer), .flush(s1_otherFlush),
      .clk(final_clock), .reset(~rst));

// takes readyToTransfer from scanner subsystem
// sends startScanning and transfer to scanner subsystem
nios2 processor (.data_in_export(data_transfer),
    .readytotransfer_export(readyToTransfer), .startscanning_export(startScanning),
    .transfer_export(transfer),
     .hex0_export(), .hex1_export(), .hex2_export(), .hex3_export(), .hex4_export(),
        .hex5_export(),
```

```
      .keys_export(), .leds_export(), .switches_export(),
      .clk_clk(clk), .reset_reset_n(rst));

  // display ctrl & status lines to red LEDs
                                // s1
                                // lowPower = no LED
  // assign LED[4] = (s1_state == STANDBY);  // standby = LED4
  // assign LED[3] = (s1_state == SCANNING);  // scanning = LED3
  // assign LED[2] = (s1_state == IDLE);   // idle = LED2
  // assign LED[1] = (s1_state == TRANSFER);  // transferring = LED1
  // assign LED[0] = (s1_state == FLUSHING);  // flushing = LED0

                                // s2
                                // lowPower = no LED
  // assign LED[9] = (s2_state == STANDBY);  // standby = LED4
  // assign LED[8] = (s2_state == SCANNING);  // scanning = LED3
  // assign LED[7] = (s2_state == IDLE);   // idle = LED2
  // assign LED[6] = (s2_state == TRANSFER);  // transferring = LED1
  // assign LED[5] = (s2_state == FLUSHING);  // flushing = LED0

  // display the input data & transfer data onto HEX displays
  displayToHex h0 (.data(s2_data_in[3:0]), .HEX(HEX0));
  displayToHex h1 (.data(s2_data_in[7:4]), .HEX(HEX1));
  displayToHex h2 (.data(s1_data_in[3:0]), .HEX(HEX2));
  displayToHex h3 (.data(s1_data_in[7:4]), .HEX(HEX3));
  displayToHex h4 (.data(data_transfer[3:0]), .HEX(HEX4));
  displayToHex h5 (.data(data_transfer[7:4]), .HEX(HEX5));
endmodule
```

6.1.3 C Program

```
/*****************************************************************************
 * Copyright (c) 2009 Altera Corporation, San Jose, California, USA. *
 * All rights reserved. All use of this software and documentation is *
 * subject to the License Agreement located at the end of this file below.*
 *****************************************************************************/
/*****************************************************************************
 *
 * Description
 * ************
 * A simple program which, using an 8 bit variable, counts from 0 to ff,
 * repeatedly. Output of this variable is displayed on the LEDs, the Seven
 * Segment Display, and the LCD. The four "buttons" (SW0-SW3) are used
 * to control output to these devices in the following manner:
 *  Button1 (SW0) => LED is "counting"
 *  Button2 (SW1) => Seven Segment is "counting"
 *  Button3 (SW2) => LCD is "counting"
 *  Button4 (SW3) => All of the peripherals are "counting".
 *
 * Upon completion of "counting", there is a short waiting period during
 * which button/switch presses will be identified on STDOUT.
 * NOTE: These buttons have not been de-bounced, so one button press may
 *      cause multiple notifications to STDOUT.
 *
 * Requirements
 * *************
 * This program requires the following devices to be configured:
```

```
 *  an LED PIO named 'led_pio',
 *  a Seven Segment Display PIO named 'seven_seg_pio',
 *  an LCD Display named 'lcd_display',
 *  a Button PIO named 'button_pio',
 *  a UART (JTAG or standard serial)
 *
 * Peripherals Exercised by SW
 * ***************************
 * LEDs
 * Seven Segment Display
 * LCD
 * Buttons (SW0-SW3)
 * UART (JTAG or serial)

 * Software Files
 * ***************
 * count_binary.c ==> This file.
 *                main() is contained here, as is the lion's share of the
 *                functionality.
 * count_binary.h ==> Contains some very simple VT100 ESC sequence defines
 *                for formatting text to the LCD Display.
 *
 *
 * Useful Functions
 * ****************
 * count_binary.c (this file) has the following useful functions.
 *  static void sevenseg_set_hex( int hex )
 *    - Defines a hexadecimal display map for the seven segment display.
 *  static void handle_button_interrupts( void* context, alt_u32 id)
 *  static void init_button_pio()
 *    - These are useful functions because they demonstrate how to write
 *      and register an interrupt handler with the system library.
 *
 * count_binary.h
 *  The file defines some useful VT100 escape sequences for use on the LCD
 *  Display.
 */

#include "count_binary.h"

/* A "loop counter" variable. */
static alt_u8 count;
/* A variable to hold the value of the button pio edge capture register. */
volatile int edge_capture;


/* Button pio functions */

/*
 Some simple functions to:
 1. Define an interrupt handler function.
 2. Register this handler in the system.
*/

/*****************************************************************
 * static void handle_button_interrupts( void* context, alt_u32 id)*
 *                                                                 *
 * Handle interrupts from the buttons.                             *
 * This interrupt event is triggered by a button/switch press. *
```

```
 * This handler sets *context to the value read from the button *
 * edge capture register. The button edge capture register *
 * is then cleared and normal program execution resumes. *
 * The value stored in *context is used to control program flow *
 * in the rest of this program's routines.         *
 *                                                 *
 * Provision is made here for systems that might have either the *
 * legacy or enhanced interrupt API active, or for the Nios II IDE *
 * which does not support enhanced interrupts. For systems created *
 * using the Nios II softawre build tools, the enhanced API is *
 * recommended for new designs.                    *
 ***************************************************************/
#ifdef BUTTON_PIO_BASE

#ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
static void handle_button_interrupts(void* context)
#else
static void handle_button_interrupts(void* context, alt_u32 id)
#endif
{
   /* Cast context to edge_capture's type. It is important that this be
    * declared volatile to avoid unwanted compiler optimization.
    */
   volatile int* edge_capture_ptr = (volatile int*) context;
   /* Store the value in the Button's edge capture register in *context. */
   *edge_capture_ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE);
   /* Reset the Button's edge capture register. */
   IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE, 0);

   /*
    * Read the PIO to delay ISR exit. This is done to prevent a spurious
    * interrupt in systems with high processor -> pio latency and fast
    * interrupts.
    */
   IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE);
}

/* Initialize the button_pio. */

static void init_button_pio()
{
   /* Recast the edge_capture pointer to match the alt_irq_register() function
    * prototype. */
   void* edge_capture_ptr = (void*) &edge_capture;
   /* Enable all 4 button interrupts. */
   IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON_PIO_BASE, 0xf);
   /* Reset the edge capture register. */
   IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE, 0x0);
   /* Register the interrupt handler. */
#ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
   alt_ic_isr_register(BUTTON_PIO_IRQ_INTERRUPT_CONTROLLER_ID, BUTTON_PIO_IRQ,
     handle_button_interrupts, edge_capture_ptr, 0x0);
#else
   alt_irq_register( BUTTON_PIO_IRQ, edge_capture_ptr,
     handle_button_interrupts);
#endif
}
#endif /* BUTTON_PIO_BASE */
```

```
/* Seven Segment Display PIO Functions
 * sevenseg_set_hex() -- implements a hex digit map.
 */

#ifdef SEVEN_SEG_PIO_BASE
static void sevenseg_set_hex(int hex)
{
   static alt_u8 segments[16] = {
      0x81, 0xCF, 0x92, 0x86, 0xCC, 0xA4, 0xA0, 0x8F, 0x80, 0x84, /* 0-9 */
      0x88, 0xE0, 0xF2, 0xC2, 0xB0, 0xB8 };        /* a-f */

   unsigned int data = segments[hex & 15] | (segments[(hex >> 4) & 15] << 8);

   IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_PIO_BASE, data);
}
#endif

/* Functions used in main loop
 * lcd_init() -- Writes a simple message to the top line of the LCD.
 * initial_message() -- Writes a message to stdout (usually JTAG_UART).
 * count_<device>() -- Implements the counting on the respective device.
 * handle_button_press() -- Determines what to do when one of the buttons
 * is pressed.
 */
static void lcd_init( FILE *lcd )
{
   /* If the LCD Display exists, write a simple message on the first line. */
   LCD_PRINTF(lcd, "%c%s Counting will be displayed below...", ESC,
            ESC_TOP_LEFT);
}

static void initial_message()
{
   printf("\n\n*************************\n");
   printf("* Hello from Nios II! *\n");
   printf("* Counting from 00 to ff *\n");
   printf("*************************\n");
}

/*******************************************************
 * The following functions write the value of the global*
 * variable 'count' to 3 peripherals, if they exist in *
 * the system. Specifically:                *
 * The LEDs will illuminate, the Seven Segment Display *
 * will count from 00-ff, and the LCD will display the *
 * hex value as the program loops.          *
 * *****************************************************/

/* static void count_led()
 *
 * Illuminate LEDs with the value of 'count', if they
 * exist in the system
 */

static void count_led()
{
#ifdef LED_PIO_BASE
   IOWR_ALTERA_AVALON_PIO_DATA(
      LED_PIO_BASE,
```

```
      count
      );
#endif
}

/* static void count_sevenseg()
 *
 * Display value of 'count' on the Seven Segment Display
 */

static void count_sevenseg()
{
#ifdef SEVEN_SEG_PIO_BASE
   sevenseg_set_hex(count);
#endif
}

/* static void count_lcd()
 *
 * Display the value of 'count' on the LCD Display, if it
 * exists in the system.
 *
 * NOTE: A HAL character device driver is used, so the LCD
 * is treated as an I/O device (i.e.: using fprintf). You
 * can read more about HAL drivers <link/reference here>.
 */

static void count_lcd( void* arg )
{
   FILE* __attribute__ ((unused)) lcd; /* Attribute suppresses "unused variable"
       warning. */
   lcd = (FILE*) arg;
   LCD_PRINTF(lcd, "%c%s 0x%x\n", ESC, ESC_COL2_INDENT5, count);
}

/* count_all merely combines all three peripherals counting */

static void count_all( void* arg )
{
   count_led();
   count_sevenseg();
   count_lcd( arg );
   printf("%02x, ", count);
}


static void handle_button_press(alt_u8 type, FILE *lcd)
{
   /* Button press actions while counting. */
   if (type == 'c')
   {
      switch (edge_capture)
      {
         /* Button 1: Output counting to LED only. */
      case 0x1:
         count_led();
         break;
         /* Button 2: Output counting to SEVEN SEG only. */
      case 0x2:
```

```
            count_sevenseg();
            break;
            /* Button 3: Output counting to D only. */
         case 0x4:
            count_lcd( lcd );
            break;
            /* Button 4: Output counting to LED, SEVEN_SEG, and D. */
         case 0x8:
            count_all( lcd );
            break;
            /* If value ends up being something different (shouldn't) do
               same as 8. */
         default:
            count_all( lcd );
            break;
         }
      }
      /* If 'type' is anything else, assume we're "waiting"...*/
      else
      {
         switch (edge_capture)
         {
         case 0x1:
            printf( "Button 1\n");
            edge_capture = 0;
            break;
         case 0x2:
            printf( "Button 2\n");
            edge_capture = 0;
            break;
         case 0x4:
            printf( "Button 3\n");
            edge_capture = 0;
            break;
         case 0x8:
            printf( "Button 4\n");
            edge_capture = 0;
            break;
         default:
            printf( "Button press UNKNOWN!!\n");
         }
      }
}

/*******************************************************************************
 * int main()                                                                  *
 *                                                                             *
 * Implements a continuous loop counting from 00 to FF. 'count' is the loop    *
 * counter.                                                                    *
 * The value of 'count' will be displayed on one or more of the following 3    *
 * devices, based upon hardware availability: LEDs, Seven Segment Display,     *
 * and the LCD Display.                                                        *
 *                                                                             *
 * During the counting loop, a switch press of SW0-SW3 will affect the         *
 * behavior of the counting in the following way:                              *
 *                                                                             *
 * SW0 - Only the LED will be "counting".                                      *
 * SW1 - Only the Seven Segment Display will be "counting".                    *
 * SW2 - Only the LCD Display will be "counting".                              *
```

```
 * SW3 - All devices "counting".                                *
 *                                                              *
 * There is also a 7 second "wait", following the count loop, *
 * during which button presses are still                        *
 * detected.                                                    *
 *                                                              *
 * The result of the button press is displayed on STDOUT.   *
 *                                                              *
 * NOTE: These buttons are not de-bounced, so you may get multiple *
 * messages for what you thought was a single button press! *
 *                                                              *
 * NOTE: References to Buttons 1-4 correspond to SW0-SW3 on the Development *
 * Board.                                                       *
 ****************************************************************************/

int main(void)
{
   int i;
   int __attribute__ ((unused)) wait_time; /* Attribute suppresses "var set but not
       used" warning. */
   FILE * lcd;

   count = 0;

   /* Initialize the LCD, if there is one.
    */
   lcd = LCD_OPEN();
   if(lcd != NULL) {lcd_init( lcd );}

   /* Initialize the button pio. */

#ifdef BUTTON_PIO_BASE
   init_button_pio();
#endif

/* Initial message to output. */

   initial_message();

/* Continue 0-ff counting loop. */

   while( 1 )
   {
      usleep(100000);
      if (edge_capture != 0)
      {
         /* Handle button presses while counting... */
         handle_button_press('c', lcd);
      }
      /* If no button presses, try to output counting to all. */
      else
      {
         count_all( lcd );
      }
      /*
       * If done counting, wait about 7 seconds...
       * detect button presses while waiting.
       */
      if( count == 0xff )
```

```
      {
         LCD_PRINTF(lcd, "%c%s %c%s %c%s Waiting...\n", ESC, ESC_TOP_LEFT,
                 ESC, ESC_CLEAR, ESC, ESC_COL1_INDENT5);
         printf("\nWaiting...");
         edge_capture = 0; /* Reset to 0 during wait/pause period. */

         /* Clear the 2nd. line of the LCD screen. */
         LCD_PRINTF(lcd, "%c%s, %c%s", ESC, ESC_COL2_INDENT5, ESC,
                 ESC_CLEAR);
         wait_time = 0;
         for (i = 0; i<70; ++i)
         {
            printf(".");
            wait_time = i/10;
            LCD_PRINTF(lcd, "%c%s %ds\n", ESC, ESC_COL2_INDENT5,
               wait_time+1);

            if (edge_capture != 0)
            {
               printf( "\nYou pushed: " );
               handle_button_press('w', lcd);
            }
            usleep(100000); /* Sleep for 0.1s. */
         }
         /* Output the "loop start" messages before looping, again.
          */
         initial_message();
         lcd_init( lcd );
      }
      count++;
   }
   LCD_CLOSE(lcd);
   return 0;
}
/*****************************************************************************
 *                                                                           *
 * License Agreement                                                         *
 *                                                                           *
 * Copyright (c) 2006 Altera Corporation, San Jose, California, USA. *
 * All rights reserved.                                                      *
 *                                                                           *
 * Permission is hereby granted, free of charge, to any person obtaining a *
 * copy of this software and associated documentation files (the "Software"), *
 * to deal in the Software without restriction, including without limitation *
 * the rights to use, copy, modify, merge, publish, distribute, sublicense, *
 * and/or sell copies of the Software, and to permit persons to whom the *
 * Software is furnished to do so, subject to the following conditions: *
 *                                                                           *
 * The above copyright notice and this permission notice shall be included in *
 * all copies or substantial portions of the Software.      *
 *                                                                           *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR *
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, *
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE *
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER *
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING *
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER *
 * DEALINGS IN THE SOFTWARE.                                                 *
 *                                                                           *
```

```
 * This agreement shall be governed in all respects by the laws of the State *
 * of California and by the laws of the United States of America. *
 * Altera does not recommend, suggest or require that this reference design *
 * file be used in conjunction or combination with any other product. *
 ******************************************************************************/
```

```
#include <stdio.h>
#define switches (volatile char *) 0x0005010
#define leds (volatile char *) 0x0005000
int main() {
  while (1)
    *leds = *switches;
  return 0;
}
```

```
/*
 * "Small Hello World" example.
 *
 * This example prints 'Hello from Nios II' to the STDOUT stream. It runs on
 * the Nios II 'standard', 'full_featured', 'fast', and 'low_cost' example
 * designs. It requires a STDOUT device in your system's hardware.
 *
 * The purpose of this example is to demonstrate the smallest possible Hello
 * World application, using the Nios II HAL library. The memory footprint
 * of this hosted application is ~332 bytes by default using the standard
 * reference design. For a more fully featured Hello World application
 * example, see the example titled "Hello World".
 *
 * The memory footprint of this example has been reduced by making the
 * following changes to the normal "Hello World" example.
 * Check in the Nios II Software Developers Manual for a more complete
 * description.
 *
 * In the SW Application project (small_hello_world):
 *
 * - In the C/C++ Build page
 *
 *   - Set the Optimization Level to -Os
 *
 * In System Library project (small_hello_world_syslib):
 * - In the C/C++ Build page
 *
 *   - Set the Optimization Level to -Os
 *
 *   - Define the preprocessor option ALT_NO_INSTRUCTION_EMULATION
 *     This removes software exception handling, which means that you cannot
 *     run code compiled for Nios II cpu with a hardware multiplier on a core
 *     without a the multiply unit. Check the Nios II Software Developers
 *     Manual for more details.
 *
 * - In the System Library page:
 *   - Set Periodic system timer and Timestamp timer to none
 *     This prevents the automatic inclusion of the timer driver.
 *
 *   - Set Max file descriptors to 4
 *     This reduces the size of the file handle pool.
 *
```

```
 *   - Check Main function does not exit
 *   - Uncheck Clean exit (flush buffers)
 *     This removes the unneeded call to exit when main returns, since it
 *     won't.
 *
 *   - Check Don't use C++
 *     This builds without the C++ support code.
 *
 *   - Check Small C library
 *     This uses a reduced functionality C library, which lacks
 *     support for buffering, file IO, floating point and getch(), etc.
 *     Check the Nios II Software Developers Manual for a complete list.
 *
 *   - Check Reduced device drivers
 *     This uses reduced functionality drivers if they're available. For the
 *     standard design this means you get polled UART and JTAG UART drivers,
 *     no support for the LCD driver and you lose the ability to program
 *     CFI compliant flash devices.
 *
 *   - Check Access device drivers directly
 *     This bypasses the device file system to access device drivers directly.
 *     This eliminates the space required for the device file system services.
 *     It also provides a HAL version of libc services that access the drivers
 *     directly, further reducing space. Only a limited number of libc
 *     functions are available in this configuration.
 *
 *   - Use ALT versions of stdio routines:
 *
 *        Function             Description
 *      =============== ====================================
 *      alt_printf   Only supports %s, %x, and %c ( < 1 Kbyte)
 *      alt_putstr   Smaller overhead than puts with direct drivers
 *                   Note this function doesn't add a newline.
 *      alt_putchar  Smaller overhead than putchar with direct drivers
 *      alt_getchar  Smaller overhead than getchar with direct drivers
 *
 */

#include "sys/alt_stdio.h"
#define switches (volatile char *) 0x0005010
#define leds (volatile char *) 0x0005000

int main()
{
  alt_putstr("Hello from Nios II!\n");
  char input = 'a';
  while(input != 'G') input = alt_getchar();

  while (1) {
    *leds = *switches;
  }

  return 0;
}
```

```
#include "sys/alt_stdio.h"
#include "sys/unistd.h"
#define switches (volatile char *) 0x00050c0
```

```
#define leds (volatile char *) 0x00050b0
#define keys (volatile char *) 0x00050a0
#define hex0 (volatile char *) 0x0005090
#define hex1 (volatile char *) 0x0005080
#define hex2 (volatile char *) 0x0005070
#define hex3 (volatile char *) 0x0005060
#define hex4 (volatile char *) 0x0005050
#define hex5 (volatile char *) 0x0005040
#define readyToTransfer (volatile char *) 0x0005010
#define xfer (volatile char *) 0x0005020
#define startScanning (volatile char *) 0x0005030
#define data_in (volatile char *) 0x0005000

int main()
{
  alt_putstr("SCANNER CTRL BASE I/O\n");
  char input = 'a';

  /* Event loop never exits. */
  while (1) {
    *startScanning = 0;
    *xfer = 0;

    if((*readyToTransfer &= 0x1) == 0x1) {
      alt_putstr("\n ready to transfer!\n");
    }

    input = alt_getchar();
    if(input == 's') {
      *startScanning = 1;
      alt_putstr("\n start scanning...\n");
      usleep(1000000);
    }
    if(input == 't') {
      *xfer = 1;
      alt_putstr("\n transferring...\n");
      usleep(1000000);
    }
  }
  return 0;
}
```

6.1.4 iverilog & gtkwave

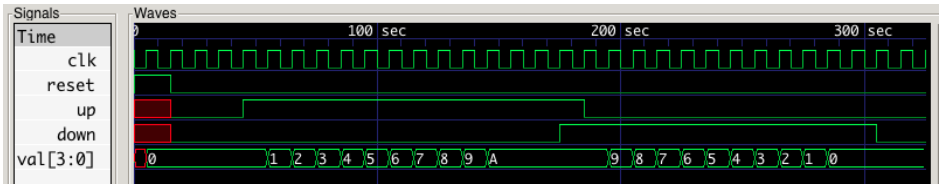

Figure 8: counter module waveform

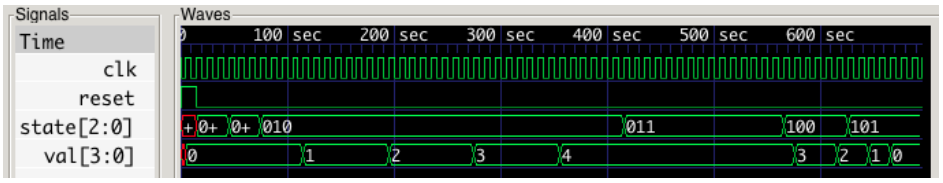

Figure 9: counterCtrl module waveform

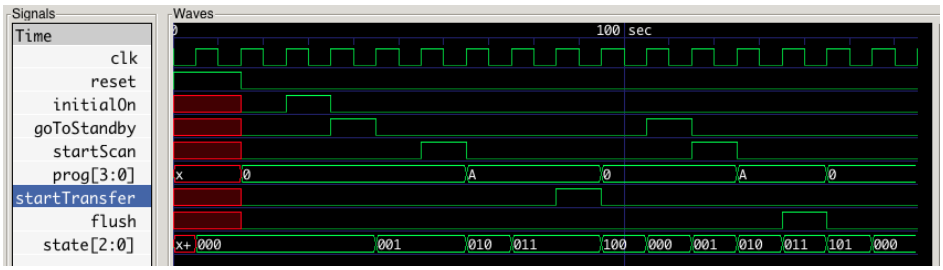

Figure 10: scannerState module waveform

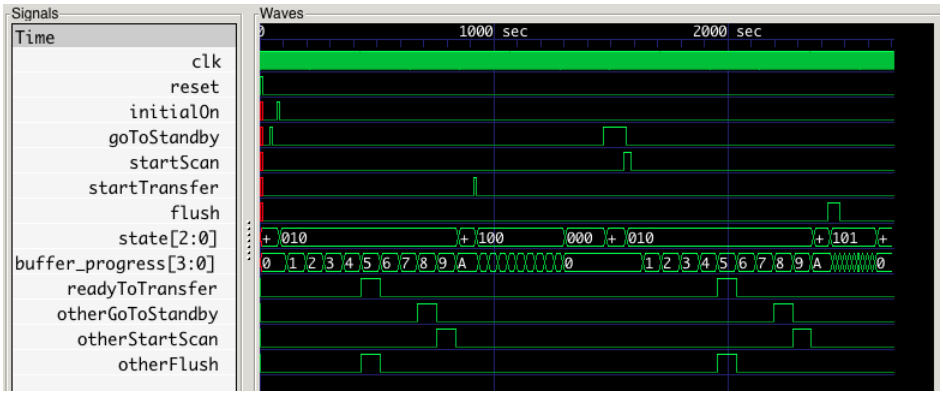

Figure 11: scanner module waveform

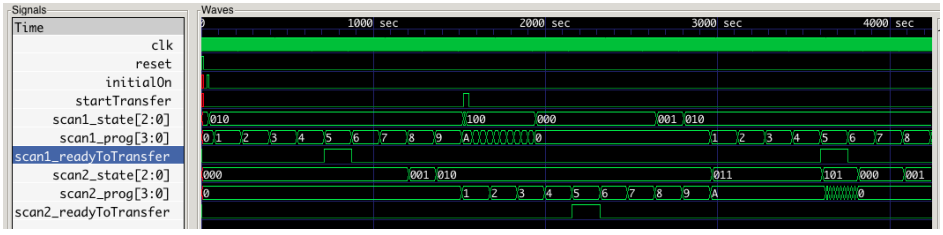

Figure 12: overall module waveform

6.1.5 Miscellaneous

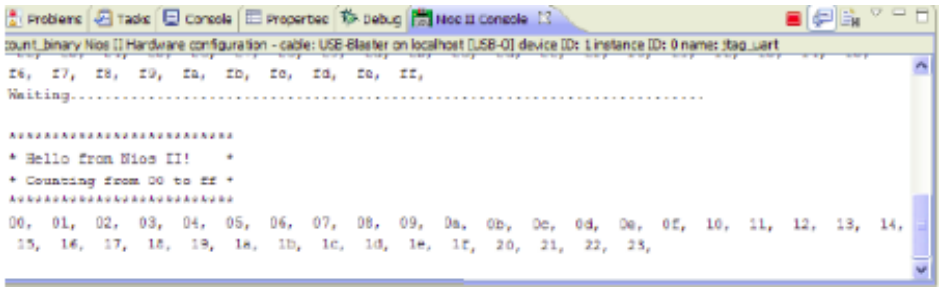

Figure 13: countbinary output

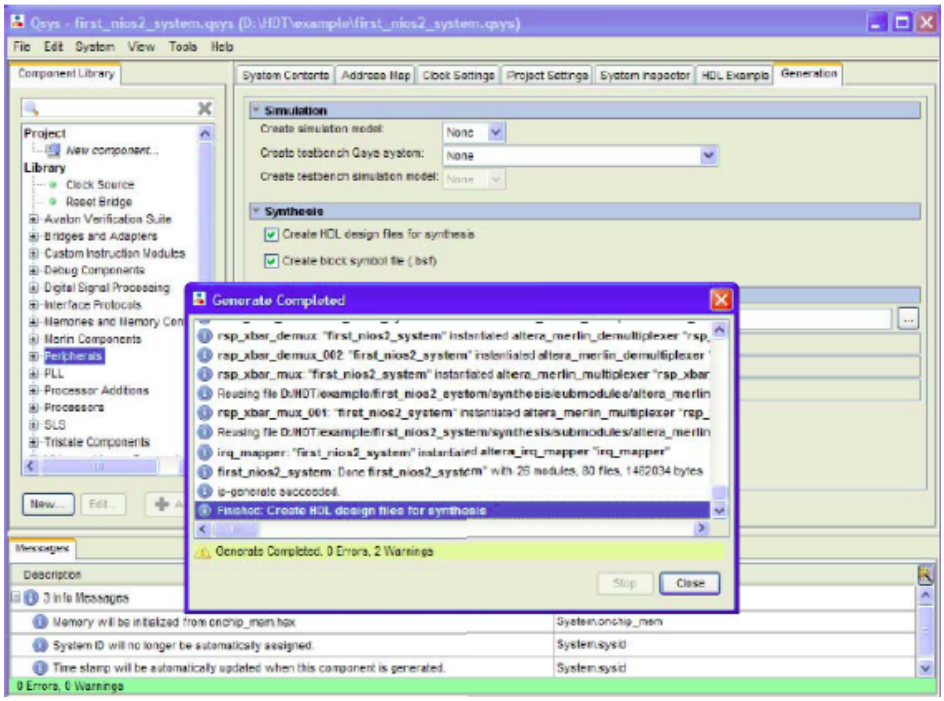

Figure 14: using qsys to generate a NIOS II processor

6.2 Lab 5

6.2.1 Block Diagrams

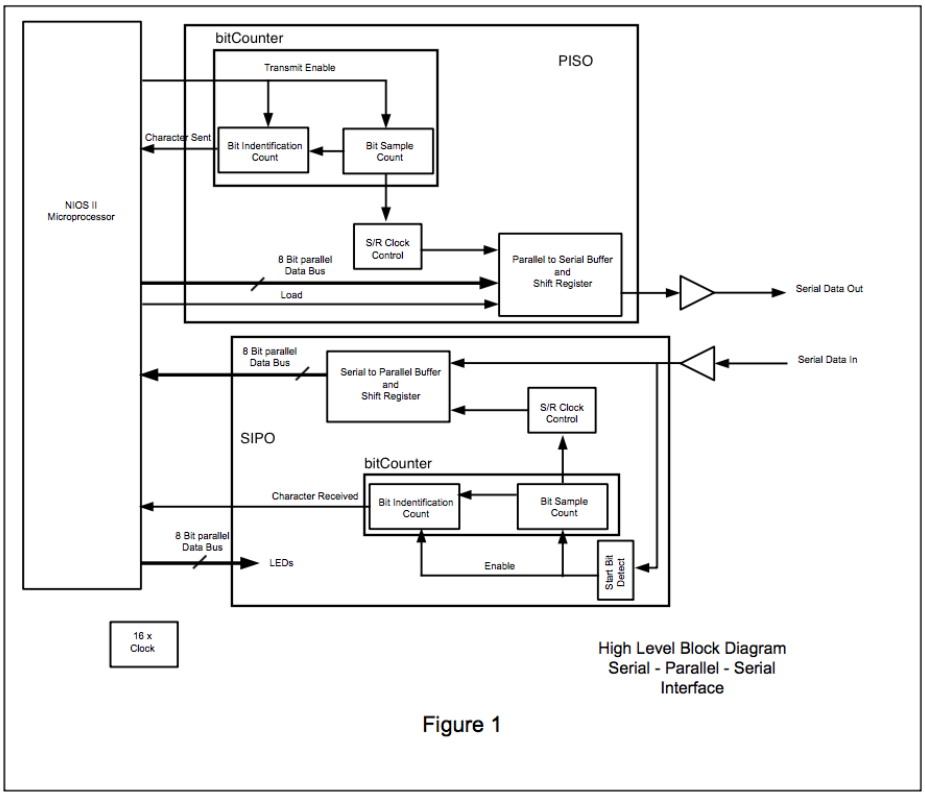

Figure 15: SPS_interface module block diagram

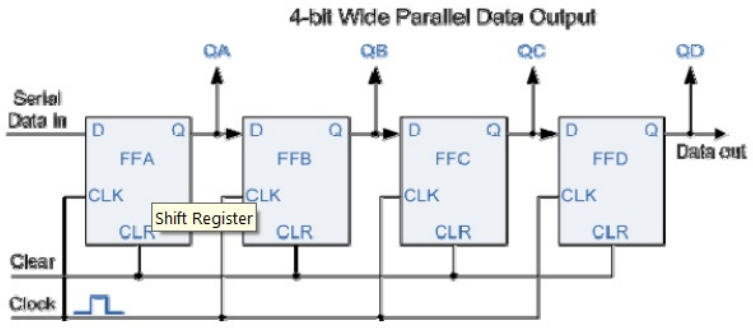

Figure: Serial-in and Parallel-out shift register

Figure 16: spShiftReg module block diagram

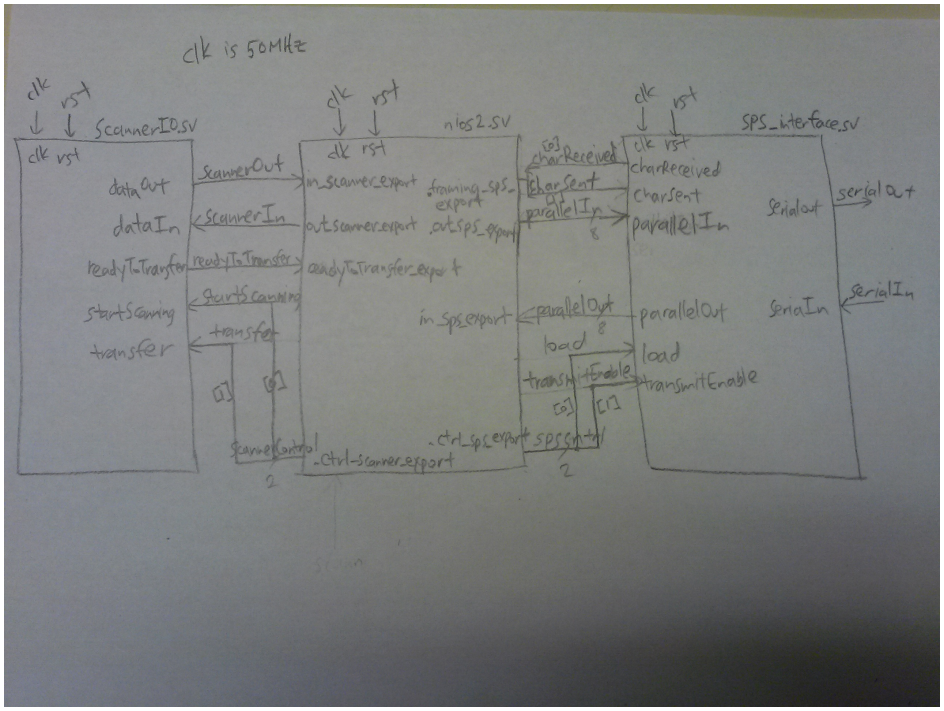

Figure 17: overall scanner/home base module block diagram

6.2.2 Verilog

```
// bit sample count
// counts samples (0-15) and outputs a posedge when hitting sample 7
module bsc(bitProgress, bit_clk, enable, clk, rst);
  input clk, rst, // 16xdata rate clk
       enable;
  output  bit_clk;

  output reg [3:0] bitProgress;
  initial bitProgress = 0;

  always@(posedge clk) begin
    if (rst) bitProgress <= 0;
    else begin
      if (enable) bitProgress <= bitProgress + 1'b1;
      else bitProgress <= 0;
    end
  end

  assign bit_clk = (bitProgress == 7);
endmodule
```

```
module bsc_testbench();
  reg    clk, rst,
       enable;
  wire bit_clk;
  wire [3:0] bitProgress;

  // set up clock
```

```
  parameter PERIOD = 10;
  initial clk = 0;
  always begin
    #(PERIOD/2);
    clk = ~clk;
  end

  bsc dut (.bitProgress(bitProgress), .bit_clk(bit_clk), .enable(enable), .clk(clk),
      .rst(rst));

  // begin simulation
  initial begin
    rst <= 1;          @(posedge clk);
    rst <= 0; enable <= 0; @(posedge clk); // 0
           enable <= 1; @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk); // 5
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk); // 10
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk); // 15
                    @(posedge clk); // 0
           enable <= 0; @(posedge clk);
                    @(posedge clk);

    $finish;
  end

  // gtkwave dump
  initial begin
    $dumpfile("bsc.vcd");
    $dumpvars;
  end
endmodule
```

```
// bit identification count
// counts bits received and signals when frame has reached end
module bic(frameProgress, endFrame, enable, clk, rst);
  input clk, rst, // bit_clk
       enable;
  output endFrame;

  parameter FRAME_WIDTH = 10;

  output reg [3:0] frameProgress;
  initial frameProgress = 0;

  always@(posedge clk) begin
    if (rst) frameProgress <= 0;
```

```
    else begin
      if (enable) frameProgress <= frameProgress + 1'b1;
      else frameProgress <= 0;
    end
  end

  assign endFrame = (frameProgress == FRAME_WIDTH - 1);
endmodule
```

```
module bic_testbench();
  reg    clk, rst,
         enable;
  wire endFrame;
  wire [3:0] frameProgress;

  // set up clock
  parameter PERIOD = 10;
  initial clk = 0;
  always begin
    #(PERIOD/2);
    clk = ~clk;
  end

  bic #(11) dut (.frameProgress(frameProgress), .endFrame(endFrame), .enable(enable),
      .clk(clk), .rst(rst));

  // begin simulation
  initial begin
    rst <= 1;          @(posedge clk);
    rst <= 0; enable <= 0; @(posedge clk);
           enable <= 1; @(posedge clk); // 0
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk); // 5
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk); // 10
                    @(posedge clk);
           enable <= 0; @(posedge clk);
                    @(posedge clk);
    $finish;
  end

  // gtkwave filedump
  initial begin
    $dumpfile("bic.vcd");
    $dumpvars;
  end
endmodule
```

```
// bitCounter module
// counts samples and outputs a bit-clock and a signal for the end of the frame
module bitCounter(frameProgress, endFrame, bit_clk, enable, clk, rst);
```

```
  input clk, rst,   // 16xdata rate clk
      enable;
  output bit_clk, endFrame;

  parameter FRAME_WIDTH = 10;

  output [3:0] frameProgress;

  bsc bitSample (.bit_clk(bit_clk), .enable(enable), .clk(clk), .rst(rst));
  bic #(FRAME_WIDTH) bitIdent (.frameProgress(frameProgress), .endFrame(endFrame),
      .enable(enable), .clk(bit_clk), .rst(rst));
endmodule
```

```
module bitCounter_testbench();
  reg    clk, rst,
      enable;
  wire endFrame, bit_clk;
  wire [3:0] frameProgress;

  // set up clock
  parameter PERIOD = 10;
  initial clk = 0;
  always begin
    #(PERIOD/2);
    clk = ~clk;
  end

  bitCounter #(11) dut (.frameProgress(frameProgress), .endFrame(endFrame),
      .bit_clk(bit_clk), .enable(enable), .clk(clk), .rst(rst));

  // begin simulation
  initial begin
    rst <= 1;          @(posedge clk);
    rst <= 0; enable <= 0; @(posedge clk);
           enable <= 1; @(posedge clk); // bsc = 0, bic = 0
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk); // bsc = 5
                    @(posedge clk);
                    @(posedge clk); // bic = 1
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk); // bsc = 10
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk); // bsc = 15
                    @(posedge clk); // bsc = 0
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk); // bsc = 5
                    @(posedge clk);
                    @(posedge clk); // bic = 2
```

```
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 10
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 15
@(posedge clk); // bsc = 0
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 5
@(posedge clk);
@(posedge clk); // bic = 3
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 10
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 15
@(posedge clk); // bsc = 0
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 5
@(posedge clk);
@(posedge clk); // bic = 4
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 10
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 15
@(posedge clk); // bsc = 0
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 5
@(posedge clk);
@(posedge clk); // bic = 5
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 10
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 15
@(posedge clk); // bsc = 0
@(posedge clk);
@(posedge clk);
```

```
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 5
@(posedge clk);
@(posedge clk); // bic = 6
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 10
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 15
@(posedge clk); // bsc = 0
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 5
@(posedge clk);
@(posedge clk); // bic = 7
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 10
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 15
@(posedge clk); // bsc = 0
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 5
@(posedge clk);
@(posedge clk); // bic = 8
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 10
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 15
@(posedge clk); // bsc = 0
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 5
@(posedge clk);
@(posedge clk); // bic = 9
@(posedge clk);
@(posedge clk);
@(posedge clk); // bsc = 10
@(posedge clk);
@(posedge clk);
@(posedge clk);
```

```
                    @(posedge clk);
                    @(posedge clk); // bsc = 15
                    @(posedge clk); // bsc = 0
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk); // bsc = 5
                    @(posedge clk);
                    @(posedge clk); // bic = 10, 0 depending on param
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk); // bsc = 10
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk); // bsc = 15
                    @(posedge clk); // bsc = 0
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk); // bsc = 5
                    @(posedge clk);
                    @(posedge clk); // bic = 0, 1 depending on param
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk); // bsc = 10
                    @(posedge clk);
          enable <= 0; @(posedge clk);
                    @(posedge clk);
    $finish;
  end

  // gtkwave filedump
  initial begin
    $dumpfile("bitCounter.vcd");
    $dumpvars;
  end
endmodule
```

```
module psShiftReg(data, serialOut, parallelIn, load, bit_clk, clk, rst);
  input    clk, rst,  // 16xdata rate clk
           bit_clk, load; // bit_clk
  input [7:0]  parallelIn;
  output     serialOut;

  output reg [10:0] data;
  initial data = 11'b11111111111;

  reg [10:0] temp;
  initial temp = 11'b11111111111;

  always@(*) begin
    if (load) begin
      temp[0] = 1'b1;
      temp[1] = 1'b0;
```

```
      temp[9:2] = parallelIn;
      temp[10] = 1'b1;
    end

    if (bit_clk) temp <= {1'b1, temp[10:1]};
    else temp <= temp;
  end

  always@(posedge clk) begin
    if (rst) data <= 11'b11111111111;
    else data <= temp;
  end

  assign serialOut = data[0];
endmodule
```

```
module psShiftReg_testbench();
  reg    clk, rst,
         load, bit_clk;
  reg [7:0] parallelIn;
  wire   serialOut;
  wire [10:0] data;

  // set up clock
  parameter PERIOD = 10;
  initial clk = 0;
  always begin
    #(PERIOD/2);
    clk = ~clk;
  end

  psShiftReg dut (.data(data), .serialOut(serialOut), .parallelIn(parallelIn),
      .load(load), .bit_clk(bit_clk), .clk(clk), .rst(rst));

  // begin simulation
  initial begin
    rst <= 1; parallelIn <= 8'b10101010; @(posedge clk);
    rst <= 0; load <= 0; bit_clk <= 0; @(posedge clk);
          load <= 1;        @(posedge clk);
          load <= 0;        @(posedge clk);
                bit_clk <= 1; @(posedge clk);
                bit_clk <= 0; @(posedge clk);
                        @(posedge clk);
                bit_clk <= 1; @(posedge clk);
                bit_clk <= 0; @(posedge clk);
                        @(posedge clk);
                bit_clk <= 1; @(posedge clk);
                bit_clk <= 0; @(posedge clk);
                        @(posedge clk);
                bit_clk <= 1; @(posedge clk);
                bit_clk <= 0; @(posedge clk);
                        @(posedge clk);
                bit_clk <= 1; @(posedge clk);
                bit_clk <= 0; @(posedge clk);
                        @(posedge clk);
                bit_clk <= 1; @(posedge clk);
                bit_clk <= 0; @(posedge clk);
                        @(posedge clk);
```

```
                    bit_clk <= 1; @(posedge clk);
                    bit_clk <= 0; @(posedge clk);
                              @(posedge clk);
                    bit_clk <= 1; @(posedge clk);
                    bit_clk <= 0; @(posedge clk);
                              @(posedge clk);
                    bit_clk <= 1; @(posedge clk);
                    bit_clk <= 0; @(posedge clk);
                              @(posedge clk);
                    bit_clk <= 1; @(posedge clk);
                    bit_clk <= 0; @(posedge clk);
                              @(posedge clk);
                    bit_clk <= 1; @(posedge clk);
                    bit_clk <= 0; @(posedge clk);
                              @(posedge clk);
    $finish;
  end

  // gtkwave filedump
  initial begin
    $dumpfile("psShiftReg.vcd");
    $dumpvars;
  end
endmodule
```

```
module spShiftReg(data, parallelOut, serialIn, clk, rst);
  input    clk, rst,  // bit_clk
           serialIn;
  output [7:0] parallelOut;

  output reg [9:0] data;
  initial data = 10'b1111111111;

  always@(posedge clk) begin
    if (rst) data <= 10'b1111111111;
    else data <= {data[8:0], serialIn};
  end

  assign parallelOut = data[8:1];
endmodule
```

```
module spShiftReg_testbench();
  reg    clk, rst,
         serialIn;
  wire [7:0] parallelOut;
  wire [9:0] data;

  // set up clock
  parameter PERIOD = 10;
  initial clk = 0;
  always begin
    #(PERIOD/2);
    clk = ~clk;
  end

  spShiftReg dut (.data(data), .parallelOut(parallelOut), .serialIn(serialIn),
      .clk(clk), .rst(rst));
```

```
  // begin simulation
  initial begin
    rst <= 1;            @(posedge clk);
    rst <= 0; serialIn <= 1'b1; @(posedge clk);
           serialIn <= 1'b0; @(posedge clk);
           serialIn <= 1'b1; @(posedge clk);
           serialIn <= 1'b0; @(posedge clk);
           serialIn <= 1'b1; @(posedge clk);
           serialIn <= 1'b0; @(posedge clk);
           serialIn <= 1'b1; @(posedge clk);
           serialIn <= 1'b0; @(posedge clk);
           serialIn <= 1'b1; @(posedge clk);
           serialIn <= 1'b0; @(posedge clk);
           serialIn <= 1'b1; @(posedge clk);
           serialIn <= 1'b0; @(posedge clk);
           serialIn <= 1'b1; @(posedge clk);
                      @(posedge clk);
    $finish;
  end

  // gtkwave filedump
  initial begin
    $dumpfile("spShiftReg.vcd");
    $dumpvars;
  end
endmodule
```

```
// PISO
module PISO(data, charSent, serialOut, parallelIn, load, enable, clk, rst);
  input    clk, rst,       // 16xdata rate clk
           load, enable;
  input [7:0]  parallelIn;
  output     serialOut, charSent;
  output [10:0] data;

  wire bit_clk;
  bitCounter #(10) counter (.frameProgress(), .endFrame(charSent), .bit_clk(bit_clk),
      .enable(enable), .clk(clk), .rst(rst));
  psShiftReg piso (.data(data), .serialOut(serialOut), .parallelIn(parallelIn),
      .load(load), .bit_clk(bit_clk), .clk(clk), .rst(rst));
endmodule
```

```
module PISO_testbench();
  reg      clk, rst,
         load, enable;
  reg [7:0] parallelIn;
  wire   charSent, serialOut;
  wire [10:0] data;

  // set up clock
  parameter PERIOD = 10;
  initial clk = 0;
  always begin
    #(PERIOD/2);
    clk = ~clk;
  end
```

```
PISO dut (.data(data), .charSent(charSent), .serialOut(serialOut),
    .parallelIn(parallelIn), .load(load), .enable(enable), .clk(clk), .rst(rst));

// begin simulation
initial begin
  rst <= 1; parallelIn <= 8'b10101010; @(posedge clk);
  rst <= 0; load <= 0; enable <= 0; @(posedge clk);
         load <= 1;        @(posedge clk);
         load <= 0; enable <= 1; @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
                         @(posedge clk);
```

```
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
```

```
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                           @(posedge clk);
                  enable <= 0; @(posedge clk);
                           @(posedge clk);
    $finish;
  end

  // gtkwave filedump
  initial begin
    $dumpfile("PISO.vcd");
    $dumpvars;
  end
endmodule
```

```
module SIPO(data, enable, charReceived, parallelOut, serialIn, clk, rst);
  input      clk, rst,   // 16xdata rate clk
             serialIn;
  output       charReceived;
  output [7:0] parallelOut;
  output [9:0] data;

  output reg enable;
  // start bit detection
  always@(*) begin
```

```
    case(enable)
      0: begin
        if (serialIn == 0) enable = 1; // at start bit, turn on bit counting
        else enable = 0;
      end
      1: begin
        if (charReceived) enable = 0; // at stop bit, turn off bit counting
        else enable = 1;
      end
      default: enable = 0;
    endcase
  end

  wire bit_clk;
  bitCounter #(11) counter (.frameProgress(), .endFrame(charReceived),
      .bit_clk(bit_clk), .enable(enable), .clk(clk), .rst(rst));
  spShiftReg SIPO (.data(data), .parallelOut(parallelOut), .serialIn(serialIn),
      .clk(bit_clk), .rst(rst));
endmodule
```

```
module SIPO_testbench();
  reg    clk, rst,
       serialIn;
  wire charReceived, enable;
  wire [7:0] parallelOut;
  wire [9:0] data;

  // set up clock
  parameter PERIOD = 10;
  initial clk = 0;
  always begin
    #(PERIOD/2);
    clk = ~clk;
  end

  SIPO dut (.data(data), .enable(enable), .charReceived(charReceived),
      .parallelOut(parallelOut), .serialIn(serialIn), .clk(clk), .rst(rst));

  // begin simulation
  initial begin
    rst <= 1;          @(posedge clk);
    rst <= 0; serialIn <= 1; @(posedge clk);
          serialIn <= 0; @(posedge clk);
                   @(posedge clk);
                   @(posedge clk);
                   @(posedge clk);
                   @(posedge clk);
                   @(posedge clk);
                   @(posedge clk);
                   @(posedge clk);
          serialIn <= 1; @(posedge clk);
                   @(posedge clk);
                   @(posedge clk);
                   @(posedge clk);
                   @(posedge clk);
                   @(posedge clk);
                   @(posedge clk);
                   @(posedge clk);
```

```
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
serialIn <= 0; @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
serialIn <= 1; @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
serialIn <= 0; @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
serialIn <= 1; @(posedge clk);
        @(posedge clk);
        @(posedge clk);
```

```
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
    serialIn <= 0; @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
    serialIn <= 1; @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
    serialIn <= 0; @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
        @(posedge clk);
```

```
                    @(posedge clk);
                    @(posedge clk);
            serialIn <= 1; @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
            serialIn <= 0; @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
                    @(posedge clk);
    $finish;
  end

  // gtkwave fiedump
  initial begin
    $dumpfile("SIPO.vcd");
    $dumpvars;
  end
endmodule
```

```
module SPS_interface(charReceived, charSent, parallelOut, serialOut, parallelIn,
    serialIn, load, transmitEnable, clk, rst);
  input    clk, rst,   // 16xdata rate clk
           load, transmitEnable,
           serialIn;
  input [7:0]  parallelIn;
  output     charReceived, charSent,
           serialOut;
  output [7:0] parallelOut;

  // clock division
  wire [31:0] divided_clocks;
  clock_divider cdiv (.divided_clocks(divided_clocks), .clk(clk));
```

```
  wire final_clock = divided_clocks[8];

  // I/O buffers
  wire sDataIn, sDataOut;
  buf in (sDataIn, serialIn);
  buf out (serialOut, sDataOut);

  SIPO sipo (.charReceived(charReceived), .parallelOut(parallelOut),
      .serialIn(sDataIn), .clk(final_clock), .rst(rst));
  PISO piso (.charSent(charSent), .serialOut(sDataOut), .parallelIn(parallelIn),
      .load(load), .enable(transmitEnable), .clk(final_clock), .rst(rst));
endmodule
```

```
module SPS_interface_testbench();
  reg      clk, rst,
           load, transmitEnable,
           serialIn;
  reg [7:0] parallelIn;
  wire   charReceived, charSent,
           serialOut;
  wire [7:0] parallelOut;

  // set up clock
  parameter PERIOD = 10;
  initial clk = 0;
  always begin
    #(PERIOD/2);
    clk = ~clk;
  end

  SPS_interface dut (.charReceived(charReceived), .charSent(charSent),
      .parallelOut(parallelOut), .serialOut(serialOut), .parallelIn(parallelIn),
      .serialIn(serialIn), .load(load), .transmitEnable(transmitEnable), .clk(clk),
      .rst(rst));

  // begin simulation
  initial begin
    rst <= 1; parallelIn <= 8'b10101010; serialIn <= 1; @(posedge clk);
    rst <= 0; load <= 0; transmitEnable <= 0;   @(posedge clk);
           load <= 1;                  @(posedge clk);
           load <= 0; transmitEnable <= 1;   @(posedge clk);
                  serialIn <= 0;       @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
                                     @(posedge clk);
```

```
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
@(posedge clk);
```

```
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
                                            @(posedge clk);
    $finish;
  end

  // gtkwave filedump
  initial begin
    $dumpfile("SPS_interface.vcd");
    $dumpvars;
  end
endmodule
```

```
module overall(HEX0, HEX1, HEX2, HEX3, LED, serialOut, serialIn, clk, rst);
  input  clk, rst, // 50 MHz clk
         serialIn;
  output serialOut;
  output [7:0] LED;
  output [6:0] HEX0, HEX1, HEX2, HEX3;

  wire charReceived, charSent;
  wire load, transmitEnable;
  wire serialOut, serialIn;
  wire [7:0] parallelIn, parallelOut,
         scanner_in, scanner_out;
  wire readyToTransfer, startScanning, transfer;
  SPS_interface sps (.charReceived(charReceived), .charSent(charSent),
     .parallelOut(parallelOut), .serialOut(serialOut), .parallelIn(parallelIn),
     .serialIn(serialIn), .load(load), .transmitEnable(transmitEnable), .clk(clk),
     .rst(rst));
  scannerIO scanners (.data_out(scanner_out), .data_in(scanner_in),
```

```
      .readyToTransfer(readyToTransfer), .startScanning(startScanning),
      .transfer(transfer), .clk(clk), .rst(rst));

  wire [1:0] scannerCtrl, SPSCtrl, framingSPS;
  assign startScanning = scannerCtrl[0];
  assign transfer = scannerCtrl[1];
  assign load = SPSCtrl[0];
  assign transmitEnable = SPSCtrl[1];
  assign framingSPS = {charSent, charReceived};

  nios2 nios (.in_scanner_export(scanner_out), .out_scanner_export(scanner_in),
      .ctrl_scanner_export(scannerCtrl), .readytotransfer_export(readyToTransfer),
           .in_sps_export(parallelOut), .out_sps_export(parallelIn),
               .ctrl_sps_export(SPSCtrl), .framing_sps_export(framingSPS),
           .clk_clk(clk), .reset_reset_n(rst));

  // display received data on HEX
  displayToHex hex01 (.fourBitData(parallelIn[3:0]), .HEX(HEX0));
  displayToHex hex02 (.fourBitData(parallelIn[7:4]), .HEX(HEX1));
  displayToHex hex11 (.fourBitData(parallelOut[3:0]), .HEX(HEX2));
  displayToHex hex12 (.fourBitData(parallelOut[7:4]), .HEX(HEX3));

  // display scanner signals on LED[2:0]
  assign LED[0] = readyToTransfer;
  assign LED[1] = startScanning;
  assign LED[2] = transfer;

  // display SPS signals on LED[7:5]
  assign LED[7] = charReceived;
  assign LED[6] = charSent;
  assign LED[5] = transmitEnable;
endmodule
```

```
module displayToHex (fourBitData, HEX);
  input [3:0] fourBitData;
  output reg [6:0] HEX;

  // looking at the hex display, start from 12o'clock and count up by going clockwise
      (from 0 to 5).
  // The segment at the center is the 6th segment (out of the seven segments labeled
      0 to 6).
  // binary value 1'b0 turns on a segment, whereas binary value 1'b1 turns off a
      segment
  //       6543210
  // HEX0 = 7'b0000000;//All segments are turned on
  // HEX0 = 7'b1111111;//All segments are turned off
  // data is sent most significant bit first says peckol
  always@(*) begin
    case (fourBitData)
      4'b0000: HEX = 7'b1000000; // 0
      4'b0001: HEX = 7'b1111001; // 1
      4'b0010: HEX = 7'b0100100; // 2
      4'b0011: HEX = 7'b0110000; // 3
      4'b0100: HEX = 7'b0011001; // 4
      4'b0101: HEX = 7'b0010010; // 5
      4'b0110: HEX = 7'b0000010; // 6
      4'b0111: HEX = 7'b1111000; // 7
      4'b1000: HEX = 7'b0000000; // 8
```

```
      4'b1001: HEX = 7'b0010000; // 9
      4'b1010: HEX = 7'b0001000; //A
      4'b1011: HEX = 7'b0000011; //b
      4'b1100: HEX = 7'b1000110; //C
      4'b1101: HEX = 7'b0100001; //d
      4'b1110: HEX = 7'b0000110; //E
      4'b1111: HEX = 7'b0001110; //F
      default: HEX = 7'b1111111;
    endcase
  end
endmodule
```

```
module displayToHex_testbench;
  reg [3:0] fourBitData;
  wire [6:0] HEX;

  displayToHex dut(.fourBitData(fourBitData), .HEX(HEX));

  parameter delay = 10;
  // begin simulation
  initial begin
    fourBitData = 4'b0000; #delay;
    fourBitData = 4'b0001; #delay;
    fourBitData = 4'b0010; #delay;
    fourBitData = 4'b0011; #delay;
    fourBitData = 4'b0100; #delay;
    fourBitData = 4'b0101; #delay;
    fourBitData = 4'b0110; #delay;
    fourBitData = 4'b0111; #delay;
    fourBitData = 4'b1000; #delay;
    fourBitData = 4'b1001; #delay;
    fourBitData = 4'b1010; #delay;
    fourBitData = 4'b1011; #delay;
    fourBitData = 4'b1100; #delay;
    fourBitData = 4'b1101; #delay;
    fourBitData = 4'b1110; #delay;
    fourBitData = 4'b1111; #delay;
    fourBitData = 4'b1010; #delay;
    fourBitData = 4'b0000; #delay;
    $finish;
  end

  // gtkwave filedump
  initial begin
    $dumpfile("displayToHex.vcd");
    $dumpvars;
  end
endmodule
```

6.2.3 C Program

```
/*
 * "Small Hello World" example.
 *
 * This example prints 'Hello from Nios II' to the STDOUT stream. It runs on
 * the Nios II 'standard', 'full_featured', 'fast', and 'low_cost' example
```

```
* designs. It requires a STDOUT device in your system's hardware.
*
* The purpose of this example is to demonstrate the smallest possible Hello
* World application, using the Nios II HAL library. The memory footprint
* of this hosted application is ~332 bytes by default using the standard
* reference design. For a more fully featured Hello World application
* example, see the example titled "Hello World".
*
* The memory footprint of this example has been reduced by making the
* following changes to the normal "Hello World" example.
* Check in the Nios II Software Developers Manual for a more complete
* description.
*
* In the SW Application project (small_hello_world):
*
* - In the C/C++ Build page
*
*   - Set the Optimization Level to -Os
*
* In System Library project (small_hello_world_syslib):
* - In the C/C++ Build page
*
*   - Set the Optimization Level to -Os
*
*   - Define the preprocessor option ALT_NO_INSTRUCTION_EMULATION
*     This removes software exception handling, which means that you cannot
*     run code compiled for Nios II cpu with a hardware multiplier on a core
*     without a the multiply unit. Check the Nios II Software Developers
*     Manual for more details.
*
* - In the System Library page:
*   - Set Periodic system timer and Timestamp timer to none
*     This prevents the automatic inclusion of the timer driver.
*
*   - Set Max file descriptors to 4
*     This reduces the size of the file handle pool.
*
*   - Check Main function does not exit
*   - Uncheck Clean exit (flush buffers)
*     This removes the unneeded call to exit when main returns, since it
*     won't.
*
*   - Check Don't use C++
*     This builds without the C++ support code.
*
*   - Check Small C library
*     This uses a reduced functionality C library, which lacks
*     support for buffering, file IO, floating point and getch(), etc.
*     Check the Nios II Software Developers Manual for a complete list.
*
*   - Check Reduced device drivers
*     This uses reduced functionality drivers if they're available. For the
*     standard design this means you get polled UART and JTAG UART drivers,
*     no support for the LCD driver and you lose the ability to program
*     CFI compliant flash devices.
*
*   - Check Access device drivers directly
*     This bypasses the device file system to access device drivers directly.
*     This eliminates the space required for the device file system services.
```

```
 *    It also provides a HAL version of libc services that access the drivers
 *    directly, further reducing space. Only a limited number of libc
 *    functions are available in this configuration.
 *
 *   - Use ALT versions of stdio routines:
 *
 *        Function           Description
 *      =============== ====================================
 *      alt_printf   Only supports %s, %x, and %c ( < 1 Kbyte)
 *      alt_putstr   Smaller overhead than puts with direct drivers
 *                   Note this function doesn't add a newline.
 *      alt_putchar  Smaller overhead than putchar with direct drivers
 *      alt_getchar  Smaller overhead than getchar with direct drivers
 *
 */

#include "altera_avalon_pio_regs.h"
#include "sys/alt_stdio.h"
#include "sys/unistd.h"

#define pDataOut (volatile char *) 0x000050a0
#define pDataIn (volatile char *) 0x00005090
#define transmitEnable (volatile char *) 0x00005080
#define load (volatile char *) 0x00005070
#define charReceived (volatile char *) 0x00005060
#define charSent (volatile char *) 0x00005050
#define inToScanner (volatile char *) 0x00005040
#define outOfScanner (volatile char *) 0x00005030
#define startScanning (volatile char *) 0x00005020
#define transfer (volatile char *) 0x00005010
#define readyToTransfer (volatile char *) 0x00005000

int main()
{
  alt_putstr("\nSCANNER I/O CONTROL BASE\n");
  char input = 'a';

  *pDataIn = 0x1; // IOWR_ALTERA_AVALON_PIO_DATA(pDataIn, 0x1);
  alt_putstr(*pDataIn);

  /* Event loop never exits. */
  while (1) {
    *startScanning = 0;
    *transfer = 0;

    alt_putstr(*pDataIn);
    IOWR_ALTERA_AVALON_PIO_DATA(pDataOut, *pDataIn); // read in send out
    usleep(1000000);
    IOWR_ALTERA_AVALON_PIO_DATA(load, *charReceived);


    if(input == 's') {
      *startScanning = 1;
      alt_putstr("start scanning...\n");
      usleep(1000000);
    }

    input = alt_getchar();
    if(input == 't') {
```

```
      *transfer = 1;
      IOWR_ALTERA_AVALON_PIO_DATA(transmitEnable, 1);
      alt_putstr("\n transmitting data...\n");
      usleep(1000000);
    }
  }

  return 0;
}
```

```
// Smoke Test

#include <stdio.h>
#include "altera_avalon_pio_regs.h"
#include "sys/alt_stdio.h"
#include "sys/unistd.h"

#define IN_SPS (volatile char *) 0x5070 //parallel input to the processor from sps
#define OUT_SPS (volatile char *) 0x5060 //parallel output from the processor to sps
#define IN_SCANNER (volatile char *) 0x5040 //parallel input to the processor from
    scanner
#define OUT_SCANNER (volatile char *) 0x5050 //parallel output from the processor to
    scanner

#define readyToTransfer (volatile char *) 0x5020
#define ctrl_scanner (volatile char *) 0x5030
#define ctrl_sps (volatile char *) 0x5010
#define framing_sps (volatile char *) 0x5000


int main() {
  alt_putstr("SMOKE TEST\n");
  char input = 'a';
  *OUT_SPS = 0x01;

  /* Event loop never exits. */
  while (1) {
    input = alt_getchar();
   /* //ctrl_scanner[0] = startScanning;
    *
     (*ctrl_scanner & 0x1) ==

     //ctrl_scanner[1] = transfer;
     (*ctrl_scanner & 0x2) ==

     //framing_sps[0] = charReceived;
    (*framing_sps & 0x1) ==

     //framing_sps[1] = characterSent;
    (*framing_sps & 0x2) ==

     //ctrl_sps[0] = load;
    (*ctrl_sps & 0x1) ==

     //ctrl_sps[1] = transmitEnable;
    (*ctrl_sps & 0x2) == */

    // if (charSent = 1) transmitEnable = 0
```

```
   // else if(input == 'e') transmitEnable = 1
   // else transmitEnable


//  *startScanning = 0;
//  *readyToTransfer = 0;

    if ((*framing_sps &= 0x02) == 0b01) {
      *(*ctrl_sps &= 0b01) = 0; // transmitEnable set to 0
      usleep(1000000);
    }
    if (input == 'e') {
      *(*ctrl_sps &= 0xb01) = 1; // transmitEnable set to 1
      usleep(1000000);
    } else {
      *(*ctrl_sps &= 0b01) = 0; // transmitEnable set to 0
      usleep(1000000);
    }

   /* input = alt_getchar();
    if(input == 's') {
      *startScanning = 1;
      alt_putstr("\n start scanning...\n");
      usleep(1000000);
    }
    if(input == 't') {
      *readyToTransfer = 1;
      alt_putstr("\n transferring...\n");
      usleep(1000000);

    }
  }*/
  }
  return 0;
}
```

6.2.4 iverilog & gtkwave

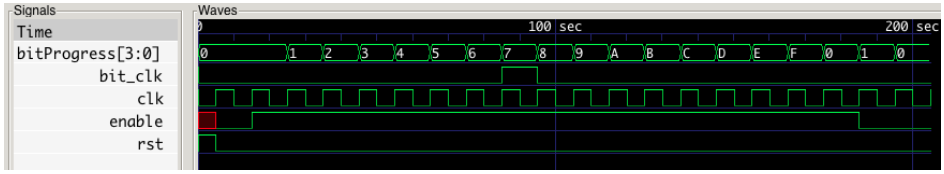

Figure 18: bsc module waveform

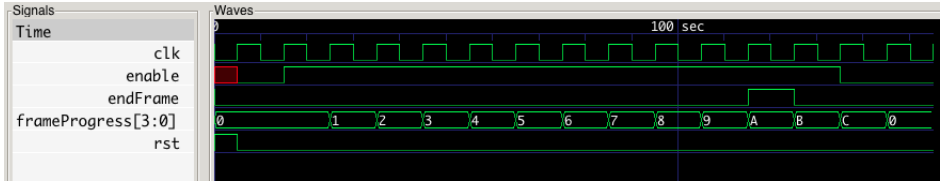

Figure 19: bic module waveform

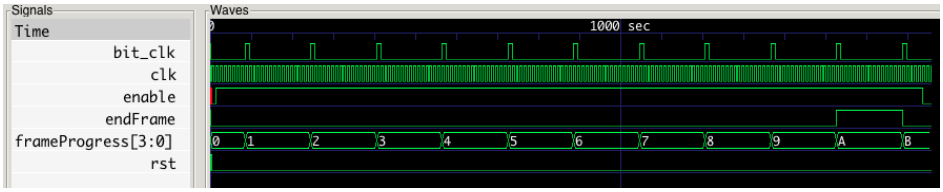

Figure 20: bitCounter module waveform

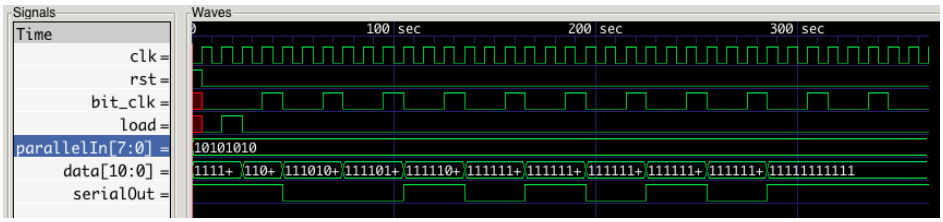

Figure 21: psShiftReg module waveform

Figure 22: spShiftReg module waveform

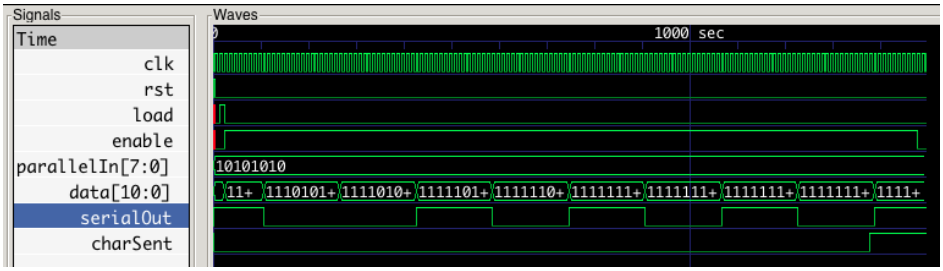

Figure 23: PISO module waveform

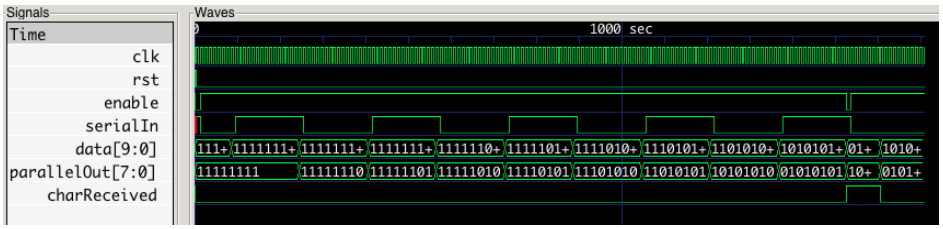

Figure 24: SIPO module waveform

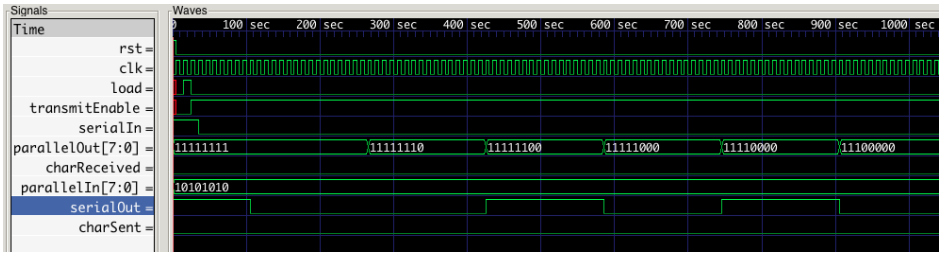

Figure 25: SPS_interface module waveform