Xuan Yuan Huang

Alvin Lim

CS M152A Lab 2

Team #11

Lab 1 Report

**Exercise 1**

1. Design: Adder/multiplier sequencer given in Lab 1.pdf
2. Implementation: Converting design to source code, which was given in a zip file and then loaded into a new project in Xilinx ISE project
3. Simulation: Prepared and launched ISIM from ISE
4. Logic Synthesis: Back in the ISE, load constraint file (.ucf) then synthesize using ISE’s XST
5. Technology Mapping: Skipped
6. Cell Placement: Combined with step 7
7. Route: Running “Implement Design” (PAR) in ISE, combining cell placement and routing in “place-and-route”
8. Bitstream Generation: Running “Generate Programming File” (BITGEN) in ISE

**Exercise 2** (Spaces added for readability)

00 00 0100

00 00 0000

00 01 0011

10 00 01 10

01 10 00 11

11 00 0000

11 01 0000

11 10 0000

11 11 0000

Expected Output:

Register 0: 0x0040

Register 1: 0x0003

Register 2: 0x00C0

Register 3: 0x0100

**Exercise 3 - Fibonaccci Program**

00 01 0001 PUSH R1 0x1

11 00 0000 SEND R0 // F1

11 01 0000 SEND R1 // F2

01 00 01 00 ADD R0 R1 R0

11 00 0000 SEND R0 // F3

01 00 01 00 ADD R0 R1 R0

11 00 0000 SEND R0 // F4

01 00 01 01 ADD R0 R1 R1

11 01 0000 SEND R1 // F5

01 00 01 00 ADD R0 R1 R0

11 00 0000 SEND R0 // F6

01 00 01 01 ADD R0 R1 R1

11 01 0000 SEND R1 // F7

01 00 01 00 ADD R0 R1 R0

11 00 0000 SEND R0 // F8

01 00 01 01 ADD R0 R1 R1

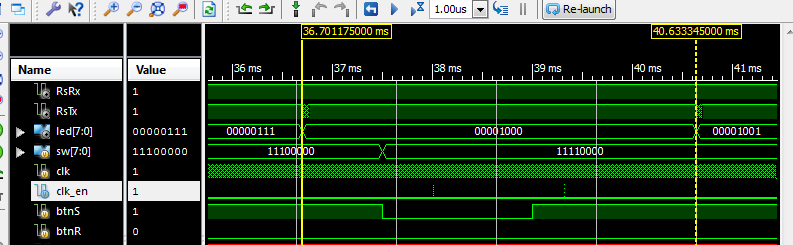
11 01 0000 SEND R1 // F9

01 00 01 00 ADD R0 R1 R0

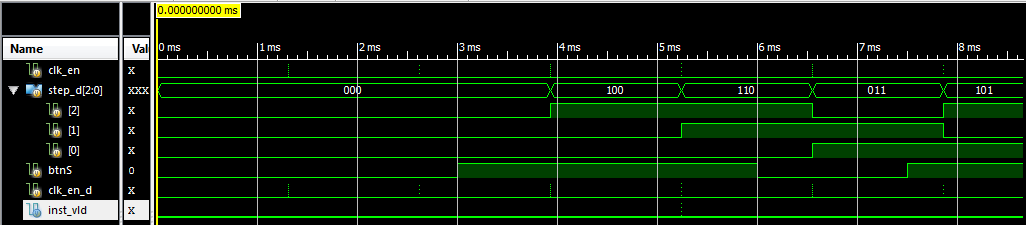
11 00 0000 SEND R0 // F10

**Exercise 4**

***Clock Enable***

1. The register clk\_en seems to be a boolean that controls when the logical clock should tick forward i.e. when the system should process input to output.
2. About 3.93217 ms  
   
3. clk\_dv is 0 whenever clk\_en is 1; that is, clk\_en is high when clk\_dv reaches its maximum value

***Instruction Valid***

1. N/A
2. Inst\_vld = 1 for the first time at 5,243.915 us
3. The timing signal is meant to be 763 Hz, so we need to delay inst\_vld from clk\_en by 3 clock cycles. This is done by blocking clk\_en from the clk, clk\_en\_d from clk\_en, and inst\_vld from clk\_en\_d
4. As we can see here, inst\_vld = 1 when ~step\_d[0] && step\_d[1] && clk\_en\_d 

***Register File***

1. ­N/A

else if (i\_wstb)

rf[i\_wsel] <= i\_wdata;

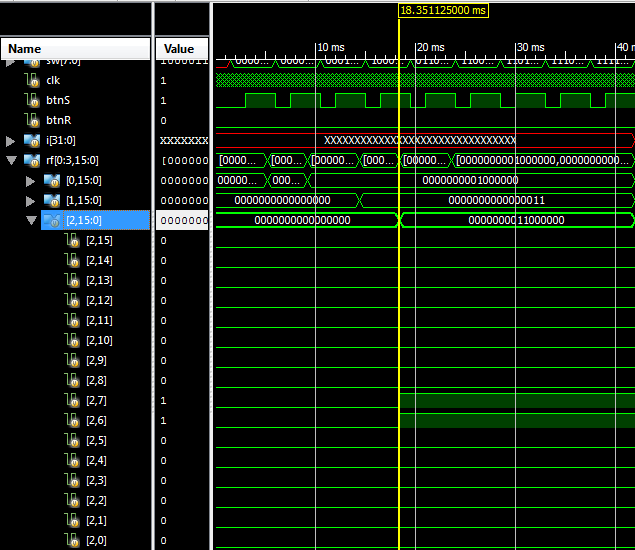
This line is of code is sequential logic; rf[i\_wsel] needs to wait a clock cycle before it is updated with the value that is in i\_wdata.



assign o\_data\_a = rf[i\_sel\_a];

assign o\_data\_b = rf[i\_sel\_b];

This code is an example of combinatorial logic. To implement this readout logic, we simply need a 16-bit wide wire, and connect it to the corresponding rf[i\_sel\_a] (which is 16-bits wide itself).

1. At 18.3511 ms.

**Exercise 5**

***Nicer UART Output***

Implemented a counter i which kept track of how many bytes have been received. Every 5th byte received is supposed to be ‘\r’, so we can just use $display() once every 5 times, resetting the counter i to 0. As for the bytes in between $display() calls, we implemented a reg [7:0] mem[3:0] to hold the every 4 bytes of output.

***An Easier Way to Load Sequencer Program***

The existing sequencer testbench sends a static sequence of instructions to the UUT (Unit Under Test) after the reset. In the last session you have observed this in the waveform viewer how the instructions are sent. Answer the following questions in your lab report

1. Identify the part of the tb.v where the instructions are sent to the UUT.

The instructions are sent to the UUT in the tskRunInst task.

1. Which user tasks are called in this process?

Now we would like to change the static set of instructions. Instead, we will be loading instructions from a text file. The format of the file is the following:

1. The name of the file is “seq.code”
2. The file is up to 1024 lines long.
3. The first line of the file contains a binary number that indicates how many instructions are included in this file.
4. Each of the remaining (n--‐1) lines contains a single instruction in binary.

For example, here’s the file--‐equivalent of the simple sequencer instructions currently in use:

Line 1: 1001

Line 2: 00000100

Line 3: 00000000

Line 4: 0001001

Line 5: 10000110

Line 6: 01100011

Line 7: 11000000

Line 8: 11010000

Line 9: 1110000

Line 10: 11110000

Modify the testbench such that it loads seq.code into an array, and executes every instruction in the file.

Hint: for file I/O, you may use the built--‐in Verilog system task $readmemb (google Verilog quick reference), or the c--‐like $fopen and $fscanf tasks.

Fibonacci Numbers

Now that you have an easier way to program the sequencer from simulation, design a sequence of instructions such that the first 10 numbers of the Fibonacci series is printed from the UART