ECE 385

Fall 2022

Experiment #2

Lab2

Haocheng Yang (hy38)

Yicheng Zhou (yz69)

Introduction

This circuit performs AND, NAND, OR, NOR, XOR, XNOR, all-high, and all-low to two register units. It can operate on 4-bits or 8-bits. It uses modular design so it is very easy to implement and debug in the design process.

Operation of logic processor

For data to be loaded in the registers, simply flip Load A and flip the expected data on the input [0:3] D, and then open Load A again. For register B the same sequence of input applies.

For a computation to be selected, the following logic applies:

Function Selection Inputs			Computation Unit Output
F2	F1	F0	f(A, B)
0	0	0	A AND B
0	0	1	A OR B
0	1	0	A XOR B
0	1	1	1111
1	0	0	A NAND B
1	0	1	A NOR B
1	1	0	A XNOR B
1	1	1	0000

For a routing to be selected, the following logic applies:

Routing Selection		Router Output	
R1	R0	A*	B*
0	0	A	B
0	1	A	F
1	0	F	B
1	1	B	A

After selecting computational function and routing according to the above figures, it is time to flip the execute switch, and then a computation and routing operation will be initiated.

Written Description

The circuit is comprised of 4 main unit and their associated logic : Register Unit, Computation Unit, Routing Unit, Control Unit.

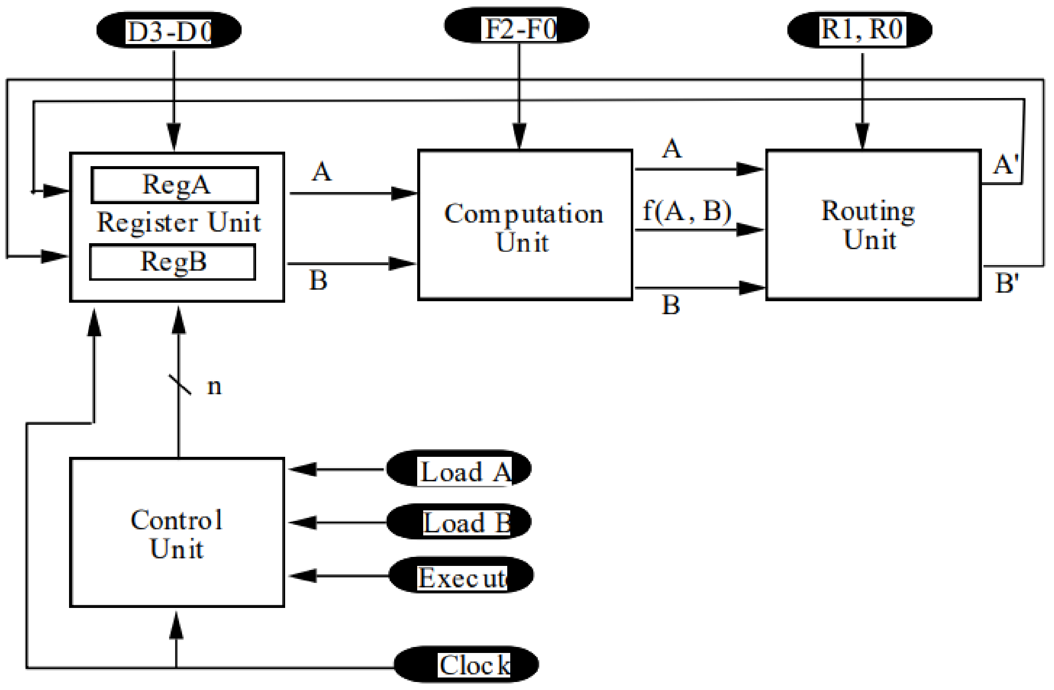
The Routing Unit is a complementary unit of register unit, it takes 3 inputs (A,B,F) and a 2 bit route command from the user. The command logic is as shown above.

The Register unit contains two 4-bit registers (Register A and Register B), it will take mode select from user and either a sequential 4-bit input from routing unit or a parallel 4-bit input from user. The mode select includes “Load A”, “Load B”, “Execute”.

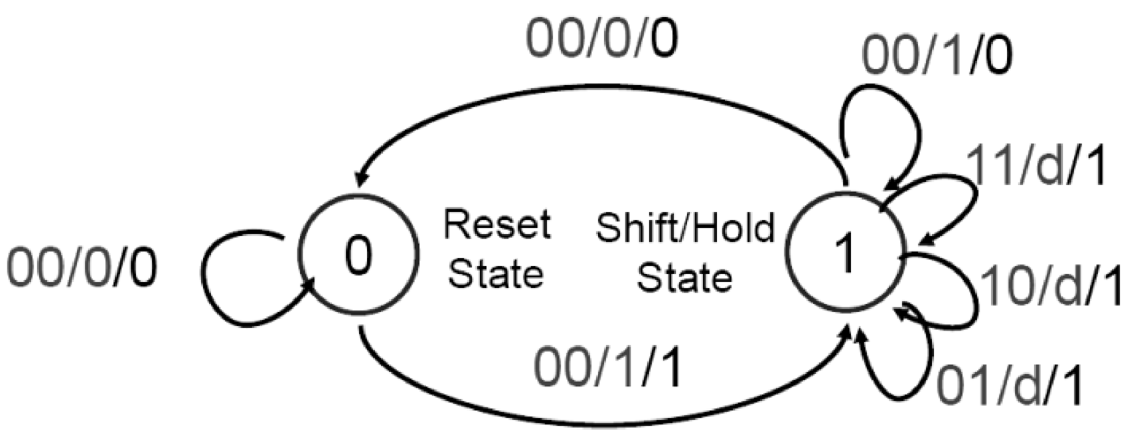
The Computation Unit will take two 4-bit sequential inputs from both register and a 3-bit logic select command form the user. The 3-bit control table is shown above.

The control unit is the center of the whole project. With an onboard counter, it controls the states of the mealy machine and the interpreted command to the registers. Along with user input “E”, this unit controls the on and off of every part of the system.

The high level block diagram is shown below:



Our implementation utilized a mealy machine, whose diagram is shown below:



Design Steps Taken

K-maps and truth table:

We did not utilize any K_maps nor truth tables when designing the units other than the control unit, the truth table and K_mpas associated with control unit is shown below:

TABLE 1: Control unit state transition table using the Mealy state machine

Exec. Switch (‘E’)	Q	C1	C0	Reg. Shift (‘S’)	Q^+	$C1^+$	$C0^+$
0	0	0	0	0	0	0	0
0	0	0	1	d	d	d	D
0	0	1	0	d	d	d	D
0	0	1	1	d	d	d	D
0	1	0	0	0	0	0	0
0	1	0	1	1	1	1	0
0	1	1	0	1	1	1	1
0	1	1	1	1	1	0	0
1	0	0	0	1	1	0	1
1	0	0	1	d	d	d	D
1	0	1	0	d	d	d	D
1	0	1	1	d	d	d	D
1	1	0	0	0	1	0	0
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	1	1	0	0

note that S is inter-unit signal and Q is internal signal of the unit.

S	C1C0	"00"	"01"	"11"	"10"
EQ	"00"	0	D	D	D
	"01"	0	1	1	1
	"11"	0	1	1	1
	"10"	1	D	D	D

Kmap for S

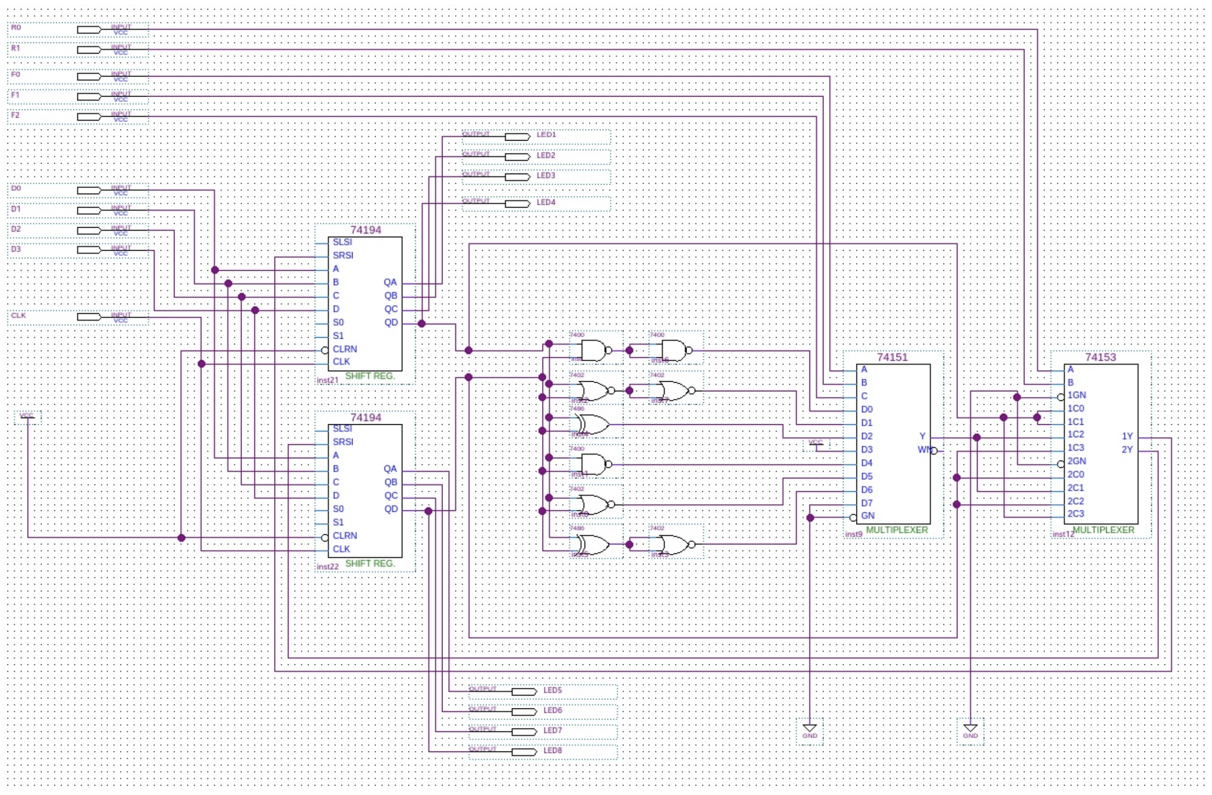
Q+	C1C0	"00"	"01"	"11"	"10"
EQ	"00"	0	D	D	D
	"01"	0	1	1	1
	"11"	1	1	1	1
	"10"	1	D	D	D

Kmap for Q+

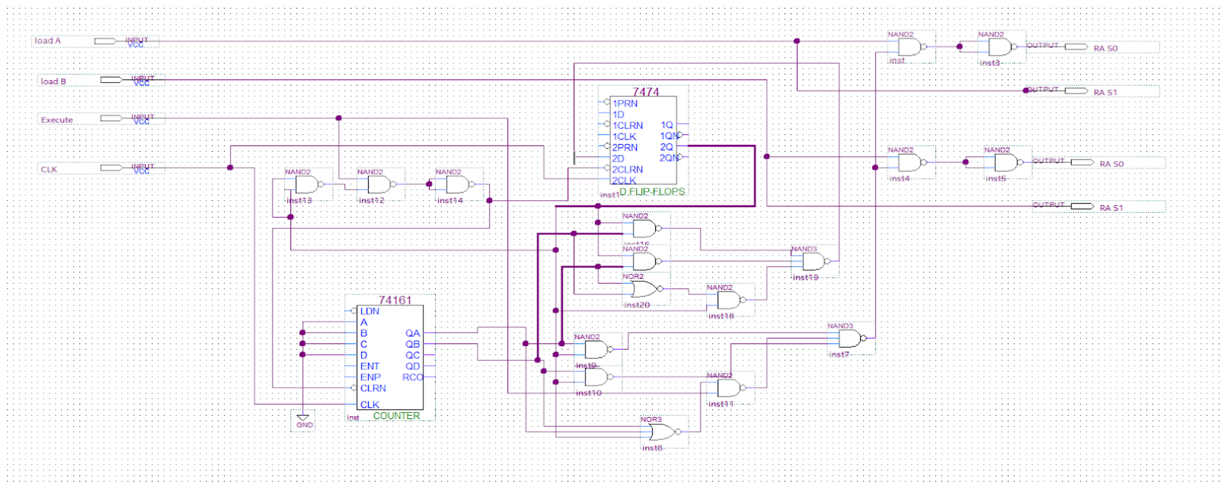
The only decision of anysort we have to make is between mealy machine or moore machine. We chose mealy machine because it has less states and promises a simpler implementation however this proved problematic later.

Detailed Schematic:

Note that due to the footprint of the control unit being too large, the schematic is separated with inter-unit inputs and outputs.



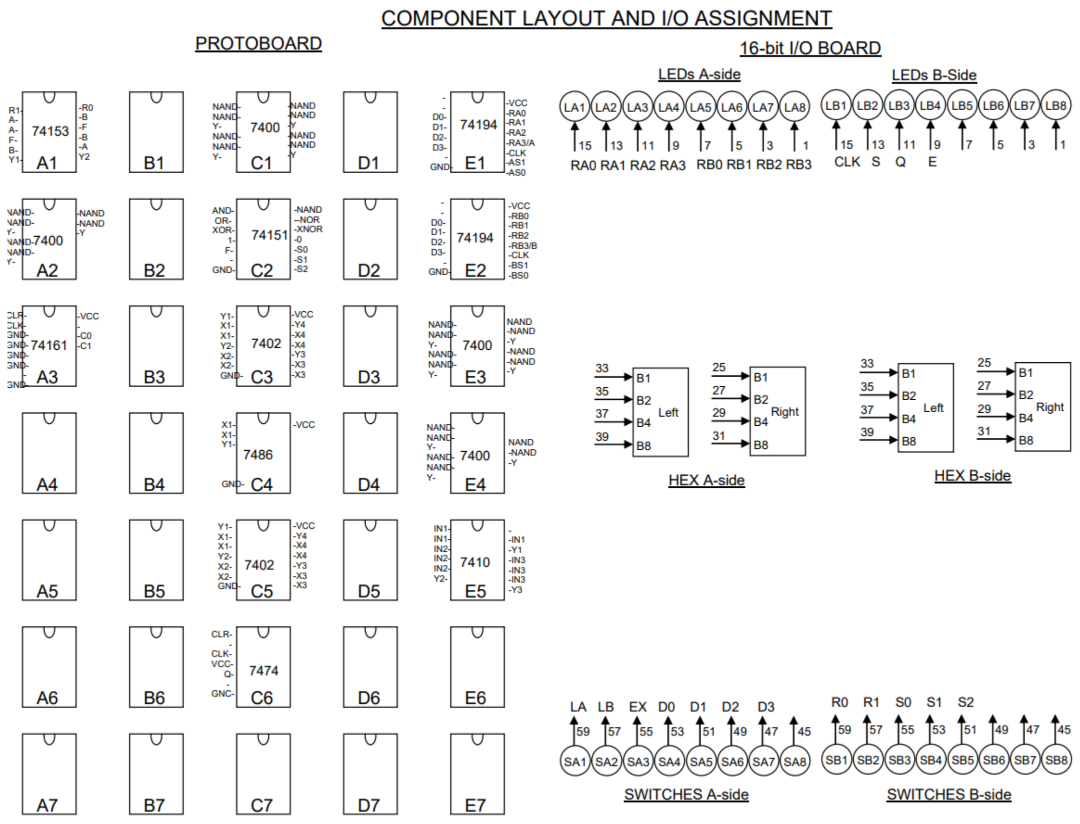
SCHEMATIC FOR COMPUTATION UNIT/ROUTING UNIT/REGISTER UNIT



SCHEMATIC FOR CONTROL UNIT.

Breadboard View / Layout

The diagram below shows the pin layout of the board, it is composed using GG layout.



Note that on the actual board give, there is only 3 column of chips available not 5, thus I choose column A,C,E for clarity and it is not directly reflecting actual hardware

8-bit logic processor on FPGA

Summary of all .SV modules and the changes you made to extend processor to 8-bits.

Synchronizers.sv: No change. Router.sv: No change. There will be always 1-bit only in one logical calculation, even if the system becomes 8-bits. In fact, no matter how many bits the system is changed into, since the ALU only processes one bit at a time, it will always remain unchanged. Register_unit.sv: Change 4-bit inputs and outputs to 8-bit. In this case, it is to change D,A, and B, according to the following code block.

```
input logic [7:0] D // 8 bit version, was [3:0]
output logic [7:0] A // Same logic
output logic [7:0] B // Same logic
```

And consequently for output variables A and B, which are both 4-bit wide previously and now should be 8-bit instead.

Reg_4.sv: Change 4-bit inputs, intermediate values, and outputs to 8-bit. The values which are affected are D, Data_out, and a hard-coded clear value for Data_out.

```
input  logic [7:0] D // 8 bit version, was [3:0]
output logic [7:0] Data_Out // Same logic
Data_Out <= 8'h0; // 8 bit version, was 4’h0
```

Processor.sv: Change 4-bit inputs, intermediate values, and outputs to 8-bit. The values which are impacted are Din, Aval, Bval, A, B, Din_S. The changes we need to do are mostly doing the changes already repeated in the code blocks above, so there is no need to repetitively present such operation. However, another action to do here is to add more hex drivers for the correct output on the FPGA board, where 4 bits occupy one hex display, thus 2 more hex drivers is needed for the upper 4-bit of the new configuration.

```
HexDriver        HexAU (.In0(A[7:4]),
.Out0(AhexU) ); // selects upper 4 bits
HexDriver        HexBU (.In0(B[7:4]),
.Out0(BhexU) ); // selects upper 4 bits
```

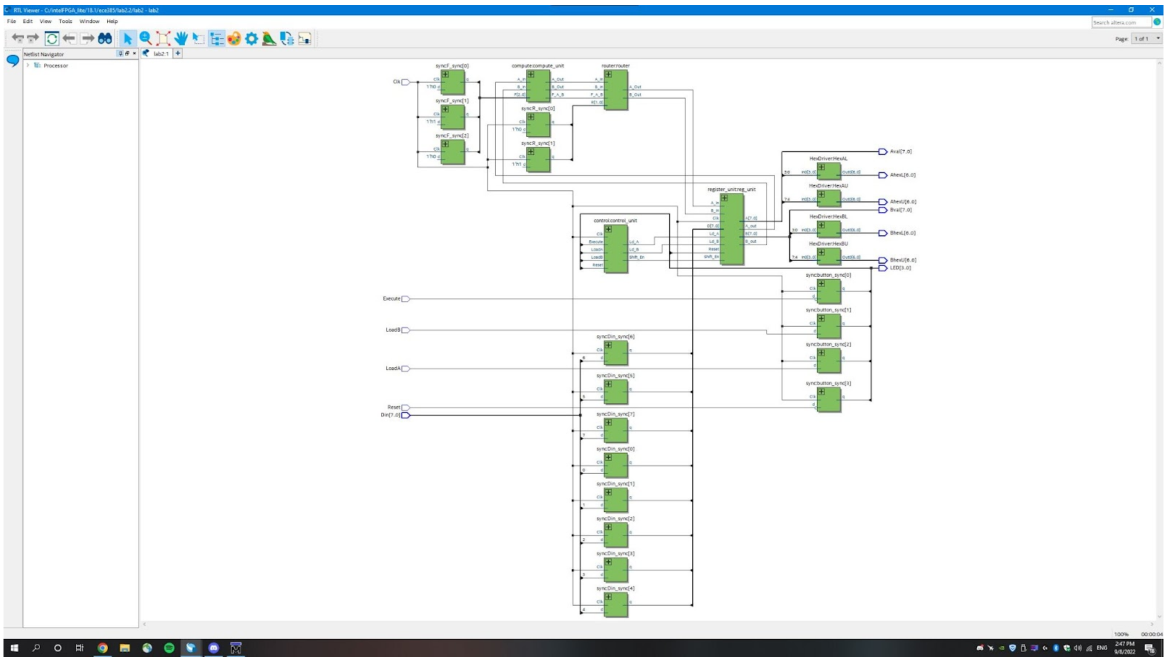
HexDriver.sv: No change.One 1-bit hex number is able to display any 4-bit binary number, thus there is no need to change the module, only addition of module is necessary. The addition procedure is presented in Processor.sv.

Control.sv: Before, the control unit only consists of 6 states: hold, compute 1st bit, compute 2nd bit, compute 3rd bit, compute 4th bit, and reset. Now, there needs to be 8 computing states, 10 in total, thus the following modifications are needed.

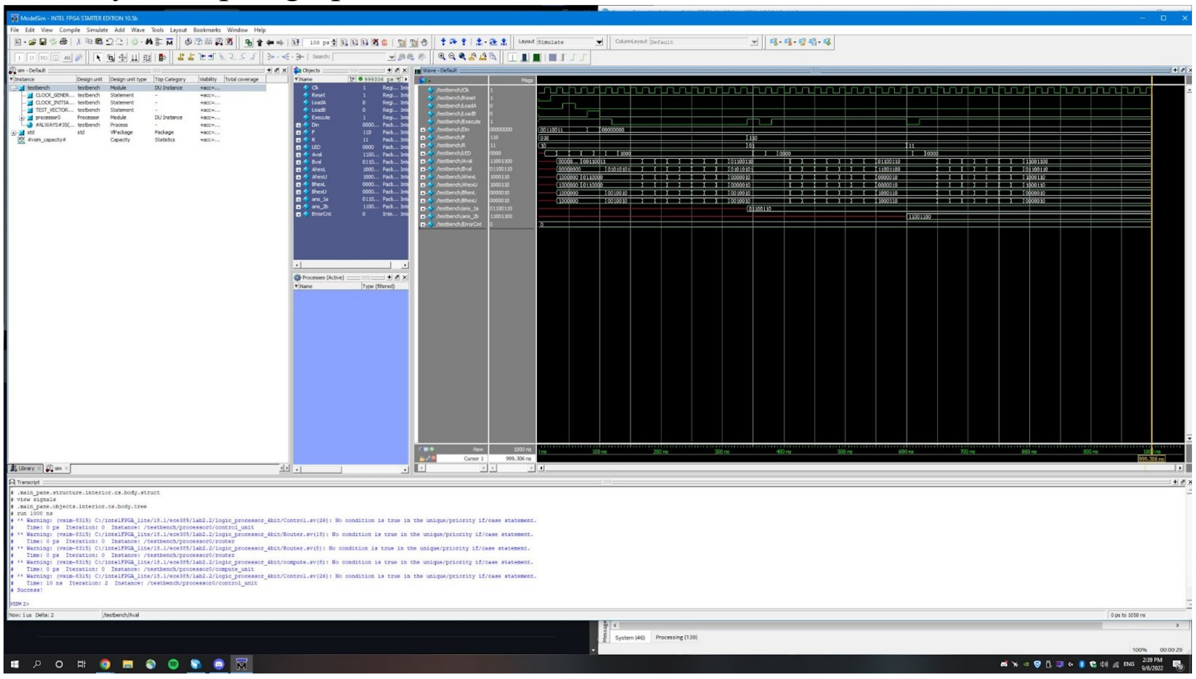
```
enum logic [3:0] {A, B, C, D, E, F, G, H, I, J}
curr_state, next_state; // add states, also bit
A : if (Execute) // change state logic
                   next_state = B;
            B :    next_state = C;
            C :    next_state = D;
            D :    next_state = E;
            E :    next_state = F;
            F :    next_state = G;
            G :    next_state = H;
            H :    next_state = I;
            I :    next_state = J;
J : if (~Execute)
                   next_state = A;
```

Compute.sv: No Change. Same as router, there only will be 1 bit processed per calculation, thus there is no need to change.

Here is the RTL block diagram generated using Quartus. F and R values have been hardwired as for Signal Tap purposes. However, this does not affect the overall diagram by noticeable margins.



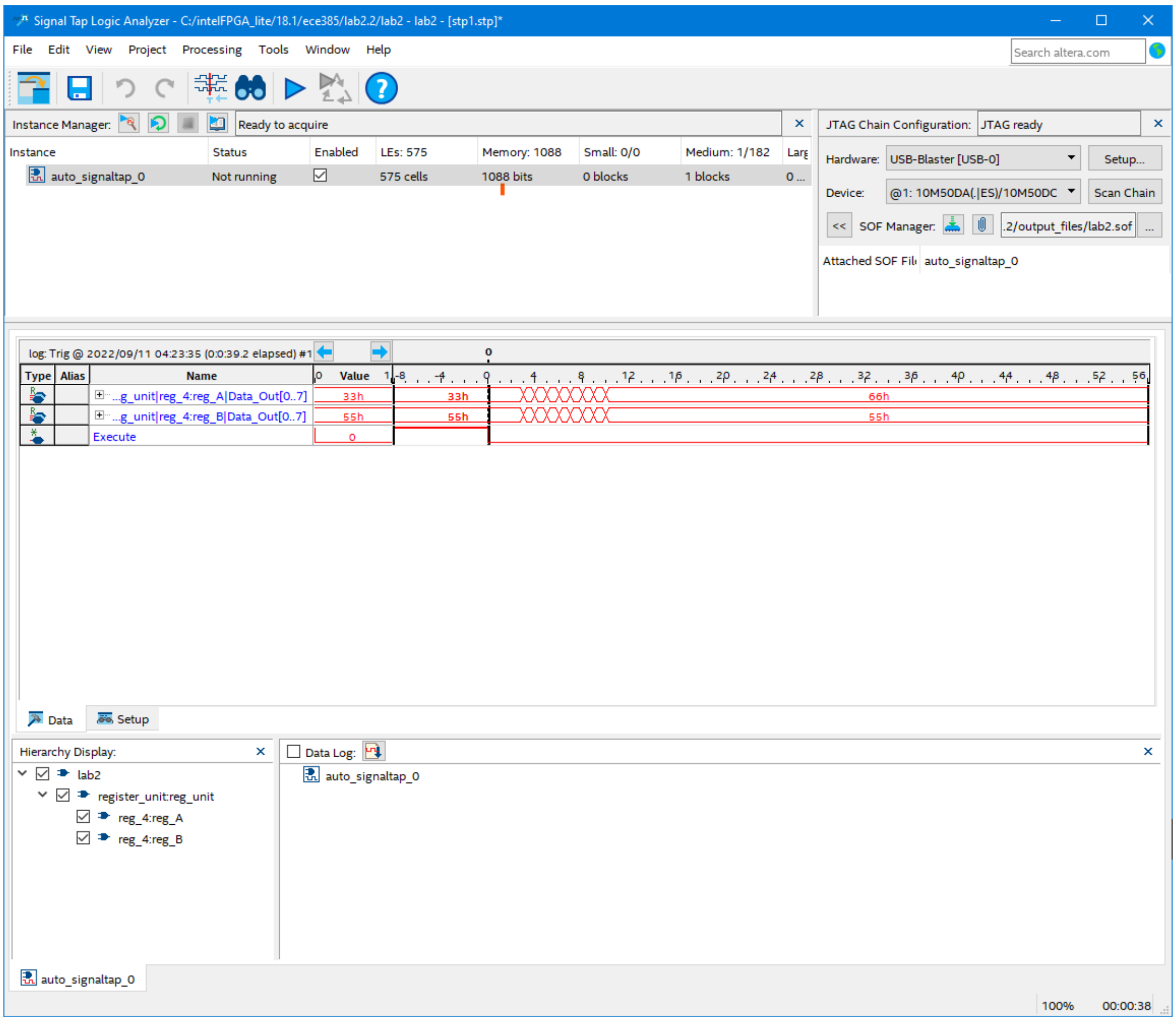
The operations are being performed in this sequence, the test bench first loads A and B with 0x33 and 0x55. Then, the test bench assigns F and R to b010 and b10, respectively, calling for A to be assigned the result of A XOR B. F and R are then changed to b110 and b001, now calling for B to be assigned the result of A XNOR B. Then, routing is changed to be b11, where A, B values are expected to change. The aforementioned operations are what is in the test bench in a very brief paragraph.



```
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run 1000 ns
# ** Warning: (vsim-8315) C:/intelFPGA_lite/18.1/ece385/lab2.2/logic_processor_4bit/Control.sv(26): No condition is true in the unique/priority if/case statement.
#    Time: 0 ps  Iteration: 0  Instance: /testbench/processor0/control_unit
# ** Warning: (vsim-8315) C:/intelFPGA_lite/18.1/ece385/lab2.2/logic_processor_4bit/Router.sv(18): No condition is true in the unique/priority if/case statement.
#    Time: 0 ps  Iteration: 0  Instance: /testbench/processor0/router
# ** Warning: (vsim-8315) C:/intelFPGA_lite/18.1/ece385/lab2.2/logic_processor_4bit/Router.sv(8): No condition is true in the unique/priority if/case statement.
#    Time: 0 ps  Iteration: 0  Instance: /testbench/processor0/router
# ** Warning: (vsim-8315) C:/intelFPGA_lite/18.1/ece385/lab2.2/logic_processor_4bit/compute.sv(8): No condition is true in the unique/priority if/case statement.
#    Time: 0 ps  Iteration: 0  Instance: /testbench/processor0/compute_unit
# ** Warning: (vsim-8315) C:/intelFPGA_lite/18.1/ece385/lab2.2/logic_processor_4bit/Control.sv(26): No condition is true in the unique/priority if/case statement.
#    Time: 10 ns  Iteration: 2  Instance: /testbench/processor0/control_unit
# Success!

VSIM 2>
```

The process to generate this Signal Tap result is as the following: Step 1, add and group the output signals of register unit A and B as two 8-bit bundles. Step 2, add execute signal to the Signal Tap output. Step 3, set Execute trigger to either edge. Step 4, program FPGA and start collection. Step 5, operate as should on FPGA board, and result will be captured as the following screenshot.


Signal Tap Logic Analyzer - C:/intelFPGA_lite/18.1/ece385/lab2.2/lab2 - lab2 - [stp1.stp]*
File Edit View Project Processing Tools Window Help
Search altera.com
Data
Setup

Instance Manager: Ready to acquire

Instance	Status	Enabled	LEs: 575	Memory: 1088	Small: 0/0	Medium: 1/182	Larg
auto_signaltap_0	Not running	✓	575 cells	1088 bits	0 blocks	1 blocks	0 ...

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0]

Device: @1: 10M50DA(.|ES)/10M50DC

SOF Manager: .2/output_files/lab2.sof

Attached SOF File: auto_signaltap_0

log: Trig @ 2022/09/11 04:23:35 (0:0:39.2 elapsed) #1

Type	Alias	Name	Value	-8 … 0	0 … 56
		...g_unit|reg_4:reg_A|Data_Out[0..7]	33h	33h	66h
		...g_unit|reg_4:reg_B|Data_Out[0..7]	55h	55h	55h
		Execute	0		

Hierarchy Display:
- ☑ lab2
  - ☑ register_unit:reg_unit
    - ☑ reg_4:reg_A
    - ☑ reg_4:reg_B

Data Log: auto_signaltap_0

auto_signaltap_0

100% 00:00:38

Description of all bugs encountered, and corrective measures taken.

In Week 1, there were many bugs encountered. First of which was about the counter and its clear signal. In our original design, clear signal is only sent when user initiate an execute command, with the assumption that the lowest 2 bits of the counter would cycle between desired outputs which it did. However, due to the rest of our design relies on a 00 output from counter to “hold” the execution, the clear logic has to be redesign to be on whenever an 00 output is expected giving us more engineering overhead. Second of which was the flipflop logic and its corresponding S command. Initially we expected S to be on full time since we designed a “sub-control” unit in register unit that S and E has to be on at the same time for shift action to be performed. However under desired clock speed it is unlikely that E could be turned off after 1 cycle thus S was redesigned and rebuilt as well. Third of which is what caused our failure at the demo, after post demo inspection we found out that Q logic was wrong and required redesign as well. Those along with some chip connection error caused the catastrophic failure of our board.

In Week 2, there was a part where we have mistaken Reg_4 as a module that we do not need to change, and clearly we were misled by the file name, and it should change accordingly. Also in Week 2, we had a lot of trouble of getting Signal Tap to work, however, after watching Friday's Q&A session, we found out the exact way of generating the Signal Tap diagram.

Post-Lab Questions

1. *Describe the simplest (two-input one-output) circuit that can optionally invert a signal (i.e., one input determines if the output is equal to the other input or equal to the other input inverted).*

The simplest 2 input 1 output circuit that can do a selected inverting of another signal is a XOR gate, where one input is connected to the circuit to be inverted (or not), and the other input is used to select, where 0 is no-invert and 1 is invert.

A	B	Y	Invert? (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

2. *Explain why this is useful for the construction of this lab.*

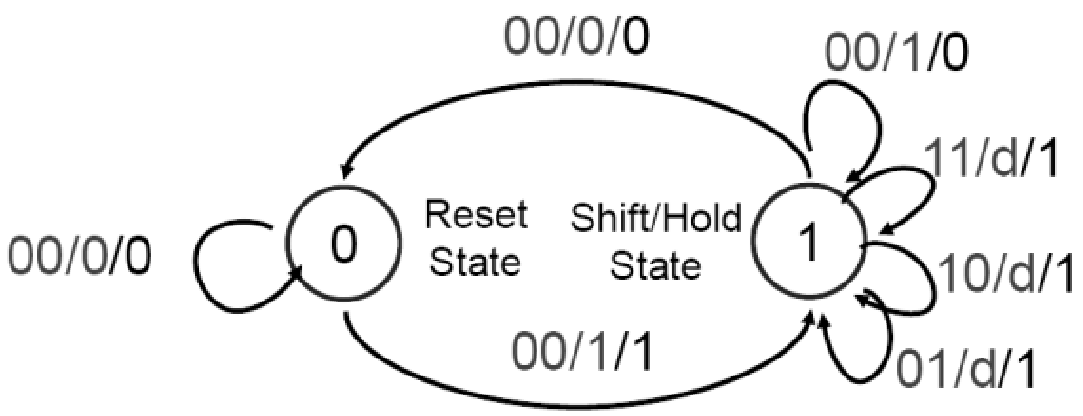
This is useful because there are often times where we need inverted inputs but are not so sure about the design in the debugging process. hard-wiring without an imputed invert select takes too much time and effort to debug, and thus this is very useful by saving debug time.

3. *Explain how a modular design such as that presented above improves testability and cuts down development time.*

A modular design with compatible TTL chips throughout the system will substantially cut down design and testing time because we can debug according to submodule and module behavior instead of debugging to overall circuit behavior. For example, then the control unit is not working before demo, simply unlink it and preserve the function of the rest of the circuit, and revise design and test again. By building a small part of the circuit each time, this makes debug and testing a lot easier.

4. *Discuss the design process of your state machine, what are the tradeoffs of a Mealy machine vs a Moore machine?*

We used a mealy machine as advised on the lab handout, Due to a promised simplicity of hardware implementation. Compared to a moore machine, a mealy machine usually has less states thus less logic. However in our case it required an on board counter which the Moore machine does not, introducing complexity that probably ended up more complex than the “heavier” but more primitive design of Moore machine.



5. *What are the differences between ModelSim and SignalTap?*

   ModelSim only works with a given testbench that is written out. SignalTap works under any condition, as long as there is an FPGA board to be programmed with and to collect data from.

6. *Although both systems generate waveforms, what situations might ModelSim be preferred and where might SignalTap be more appropriate?*

   ModelSim is preferred when a testbench is given, or when a testbench is very easy to write, such as when the logic is very simple, as in the case of A XOR B or other basic binary operations. SignalTap is preferred when actual circuit performance is the concern, and is more likely to be used to replace the oscilloscope–instead of probing the circuit, we can see the diagram with a lot less effort.

Conclusion

In this lab we have learned how to use System Verilog to design and to verify design of fairly complicated computing digital circuits. We have also learned how to use Signal Tap and test bench to verify and debug designs of circuits.

Also a valuable lesson is that before a design is verified by physical implementation, over optimizing the design might cause a lot of trouble for debugging. One of the reasons our board is so difficult to debut is that we are fixated on fitting the whole circuit on one board and that a lot of units are sharing chips, which resulted in our failure to properly isolate any fault of the system eventually causing us to fail.