ECE 385

Fall 2023

Experiment 2

A Logic Processor

Alizain Bandukwala and Akshat Mehrotra

NY 12:45pm Friday

Introduction: The purpose of this circuit was to execute eight functions on two 4-bit numbers which were stored in two registers. The circuit was made up of four main components which were the Control Unit, Computational Unit, Routing Unit, and Register Unit. The second part of this lab was to extend 4-bit numbers into 8 bits using a FPGA board and understand how to use System Verilog.

Description: This circuit was designed to perform eight different functions and use a routing unit to have four different ways to output the function. The inputs were stored into a shift register first loaded as parallel and then after shifted to the right 1 bit to go into the computational unit and perform the desired function one by one on each and then push the output bit back into the shift register serially to display the output to the LED. The computational unit was designed with a 4:1 MUX and 2:1 to determine which of the 8 functions to choose from and the 2:1 was used to either invert the input or not to perform the last four functions in the table. The unit was made up of NAND NOR and XOR gates to perform the desired function one bit at a time received the last output channel from the shift register. After the computations were completed for each bit the routing unit determined where the function would be stored using a 4:1 MUX to determine which register had the function and either the input A or B. All these units were controlled from the control unit which was derived from the truth table given to find S and Q+. S was the output to the Mealy machine given to the register unit and Q+ was used to determine if the unit was in the shift/halt stage or execute. This was made from a flipflop to store the previous output of Q+ to receive Q. The shifting was performed using a 4-state counter which helped control the shifting on the register on every rising edge of the clock which was made from scopy using a square wave and jumped from 0-5V every cycle. This was implemented using a Mealy Diagram to determine which state Q+ would be.

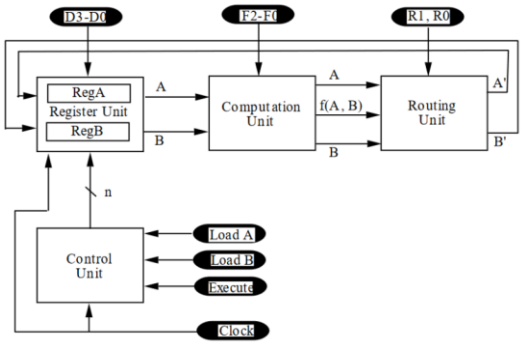

Figure 1: Block Diagram

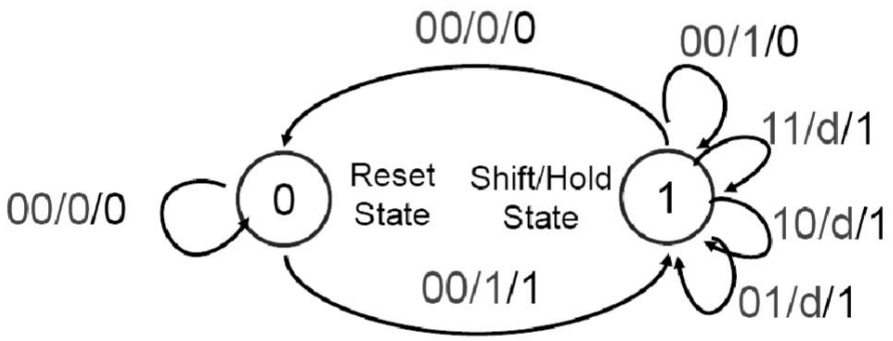

Figure 2: State Diagram

State Diagram:

Design Setup:

	A	B	C	D	Y
0	0	0	0	0	0
1	0	0	0	1	x
2	0	0	1	0	x
3	0	0	1	1	x
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	x
10	1	0	1	0	x
11	1	0	1	1	x
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

Figure 2: Y=S, A=E, B=Q, C=Co, D=C1

	$\overline{C}.\overline{D}$	$\overline{C}.D$	$C.D$	$C.\overline{D}$
$\overline{A}.\overline{B}$	0	x	x	x
$\overline{A}.B$	0	1	1	1
$A.B$	0	1	1	1
$A.\overline{B}$	1	x	x	x

Figure 3: K-Map for (S)

Boolean Equation: S = EQ` + Co + C1

	A	B	C	D	Y
0	0	0	0	0	0
1	0	0	0	1	x
2	0	0	1	0	x
3	0	0	1	1	x
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	x
10	1	0	1	0	x
11	1	0	1	1	x
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

Figure 4: Y=Q+, A=E, B=Q, C=Co, D=C1

	$\overline{C}.\overline{D}$	$\overline{C}.D$	$C.D$	$C.\overline{D}$
$\overline{A}.\overline{B}$	0	x	x	x
$\overline{A}.B$	0	1	1	1
$A.B$	1	1	1	1
$A.\overline{B}$	1	x	x	x

Figure 5: K-Map for (Q+)

Boolean Equation: Q+ = E + Co + C1

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

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

Figure 6: Functions and Routing Label

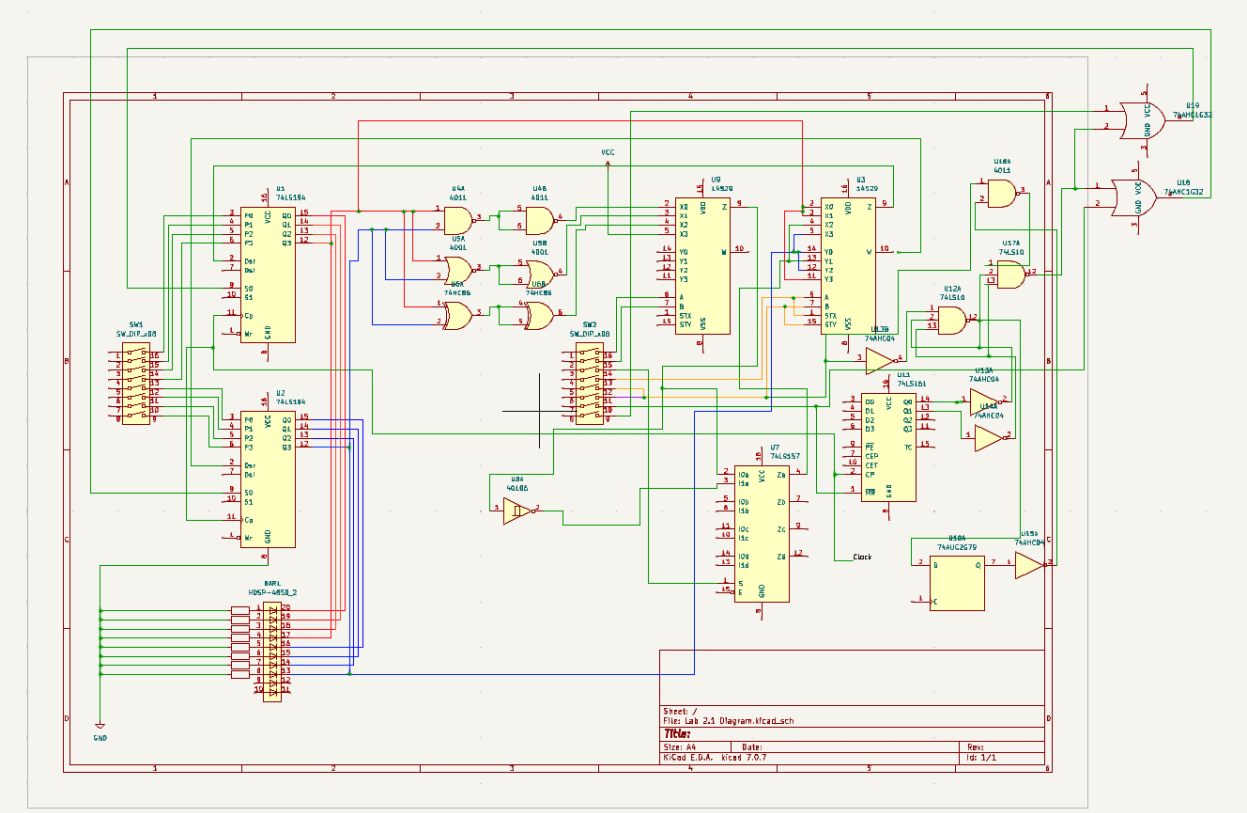

Figure 7: Circuit Schematic

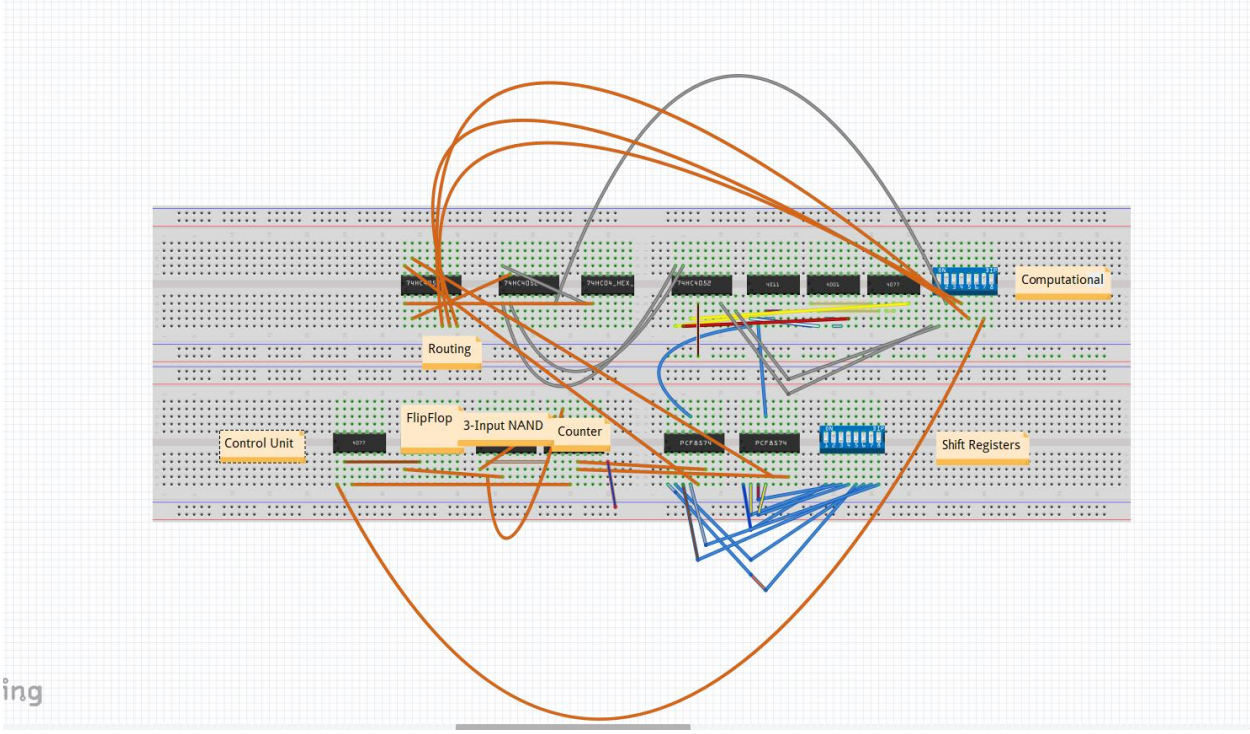

Figure 8: Fritzing Diagram

Computing the first four functions was done by using De Morgan's Law to derive expressions for AND and OR. Using two NANDs creates the function for AND and using two NORs creates the function for an OR because when NAND and NOR are created from inverting AND and OR so to reverse the process just inverting again would get the original function back. These four functions were then cycled through a 4:1 MUX to help choose when to select them and then the output of this MUX was put through a 2:1 MUX to see if the output should be inverted for the last four functions or not. The routing unit was created using a Quad 4:1 MUX for input A and B. This MUX checks if the input goes to A or B and then checks to see if the function output goes into A or B using two selects. Two design implementations were considered which were using an 8:1 MUX to select the functions or using a 4:1 and then a 2:1 MUX to select the inputs and then check for inverting or not. The second option was chosen because it was less wiring on the board and was easier to debug instead of doing operations for each function only four needed to be done in the computational unit and then at the end just invert the output to receive the other four functions.

Post Lab Questions:

1) The simplest two-input one output circuit is a 2:1 Mux because you can either select the first or second input to either invert or not a signal.
2) This improves testability because the circuit is simpler than just having to use more combinational logic with NAND , NOR or NOT gates and just use a single MUX chip too select which input you are choosing and cuts down time because every gate has their set of delay time which will have to taken account for instead of simply going straight to a MUX input and output.
3) The tradeoffs of a Mealy state vs a Morre is that for Mealy changes its output based on the present state input while Moore depends only on the current state. The Moore can be simpler to implement with a synchronous output vs a Mealy with fewer states but an asynchronous output. Also Mealy responds to the same clock cycle vs Moore which responds to one clock cycle later.
4) The difference between vSim and Vivado Debug Core is vSim is a more general term for simulation in digital design versus debug cores are specific for debugging features and circuit components part of the simulation. vSim might be preferred in functional verification and validation and also early state design stages because it will be quicker in testing ideas and concepts. Debug Cores are more used for hardware level debugging and to test stuff on the FPGA in real time because of the visibility for internal signals and behavior of the design. VSim can be used for initial debugging and verification while debug cores can be used for post implementation after synthesis to analyze actual hardware behavior.

8-bit logic processor on FPGA

1.
    a. Control.sv: We had to extend the state machine to accommodate for the additional cycles required for 8 bits. Add states till J and make the curr_state/next_state vars to 5 bit lines. Also had to change the termination condition for the state machine.
    b. Processor.sv: We had to change the interface declaration of the module to accommodate for 8 bits from 4 bits (Din, Aval, Bval). We had to change the internal variables used from 4bits to 8bits(A, B, Din_S). We also had to change the display code for HexDriverA to accommodate 8bit display.
    c. Reg_4.sv: Changed the interface input for reg_4 to have 8bit input and output (D, Data_Out). Changed default reset Data_Out to 8'h0 from 4'h0;
    d. Register_unit.sv: Changed the interface to support 8 bit D, A, and B.
2.

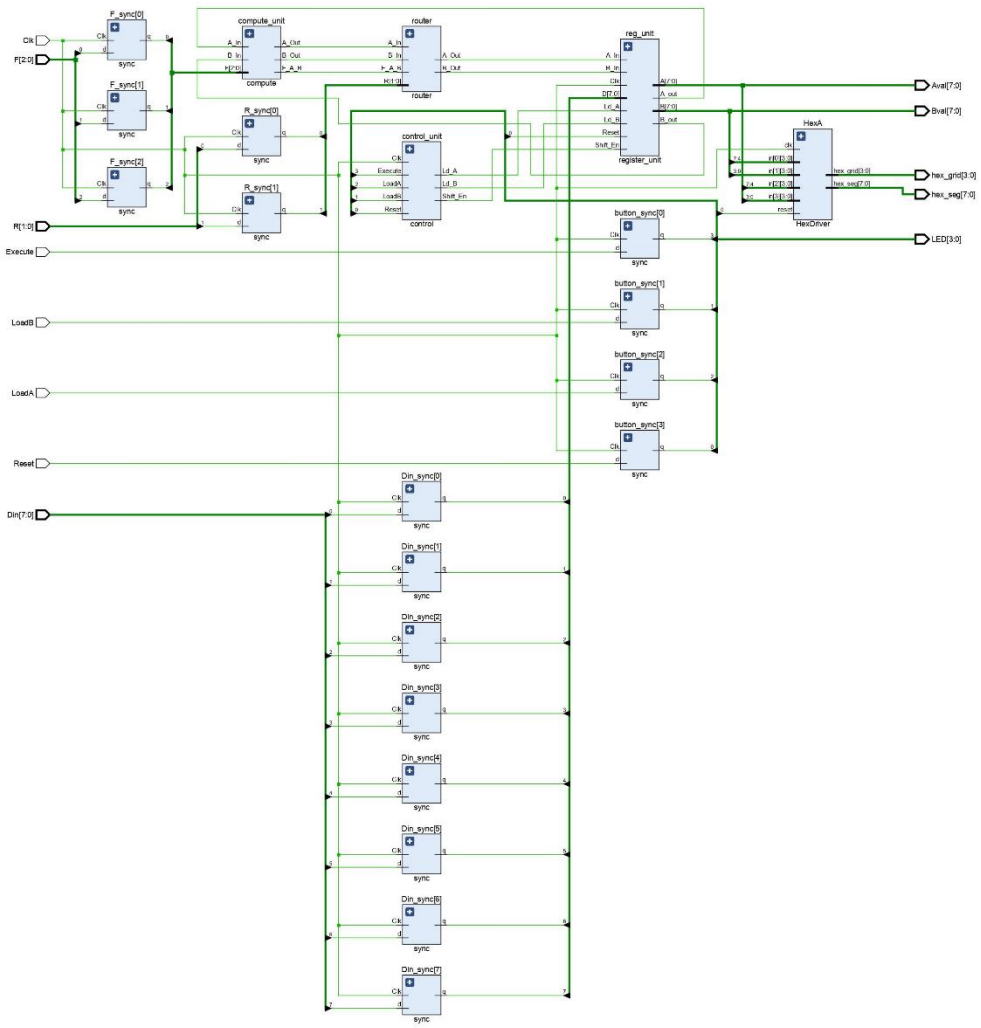

Figure 9: RTL Synthesis

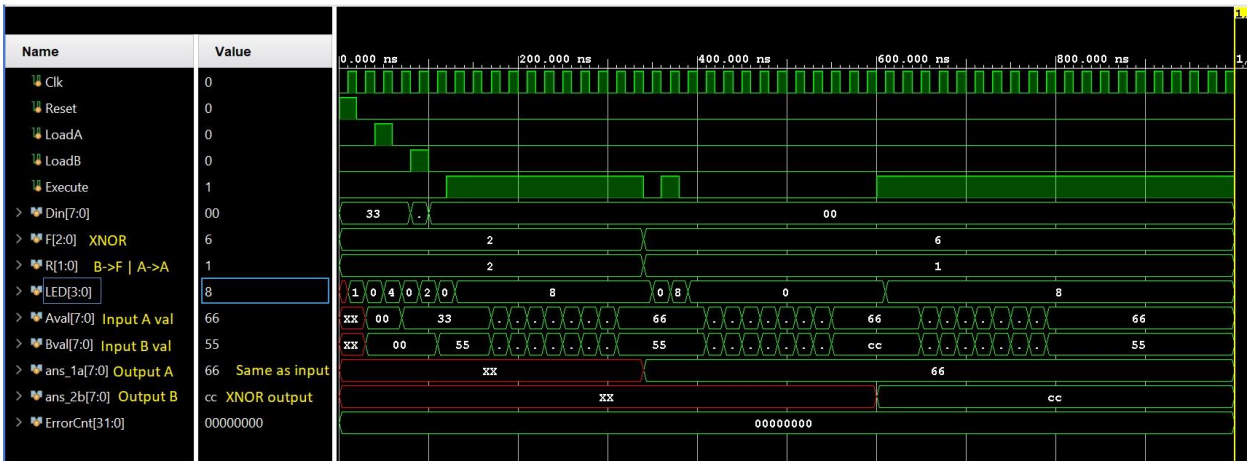

Figure 10: Simulation

4. Under synthetized design and under open synthesized design, click on setup debug cores. Click on the signals we want to process and drag them into the window. Add triggers if needed but otherwise keep clicking next till window disappears. Resynthesize and generate bitstream. In the program to board menu make sure the debug core file is selected in the debug probes file option before writing program to board.

5.

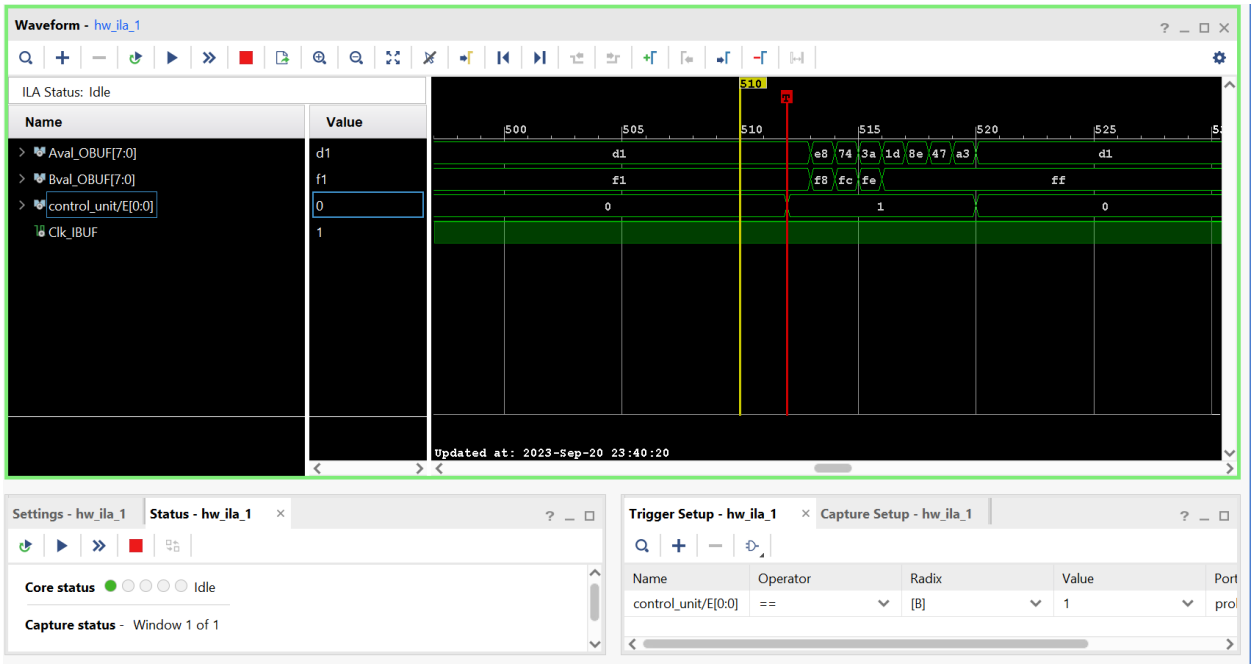

Figure 10: Debug Cores

Here we are tracking execute, Aval, Bval, Clck and also using Execute as a trigger. It is a simple example of debug cores.

Bugs Encountered:

Some bugs encountered with lab 2.1 included not checking for outputs that were being used and leaving them as floating. This problem was not noticed in the beginning because the correct output was being achieved but as the circuit kept growing problems arose and checking for all unused outputs especially on the MUX should be grounded to not cause for random outputs. Another problem occurred was the control unit with the counter the equation developed from the K-map was initially wrong and was making the counter go 4 clock cycles, but S was not giving the correct output for some values and after some debugging, the problem was found in simplifying with DeMorgans Law. Other bugs that occurred were trying to make the circuit easy to debug and cleaning up wires making loops or going to wrong inputs with all the NANDs because most outputs needed to be inverted twice for the desired logic needed for the MUX's. For lab 2.2 the main problem was making sure the HEX drivers were working because the input output bits were just misplaced and an extra state was added, but using the simulation the problem was found easily.

Conclusion:

The purpose of the lab was to create a circuit that can perform 8 functions on two 4-bit numbers and route them to the proper shift registers. This was controlled by a counter and shifted every clock cycle to perform operations on one bit and go back into the register simultaneously. This circuit was then converted in System Verilog to 8-bit numbers which required switching the bits for the inputs and outputs and the states for the clock as the bits need to shift 4 more times. Overall, the lab was used to perform functions on 2 numbers either 4 or 8 bits and using combinational logic to make functions work and MUXs to know what certain inputs should be selected.