ECE 385

Fall 2022

Lab 2

A Logic Processor

Zhuoyuan Li, Shihua Cheng

Tianhao Yu

Introduction

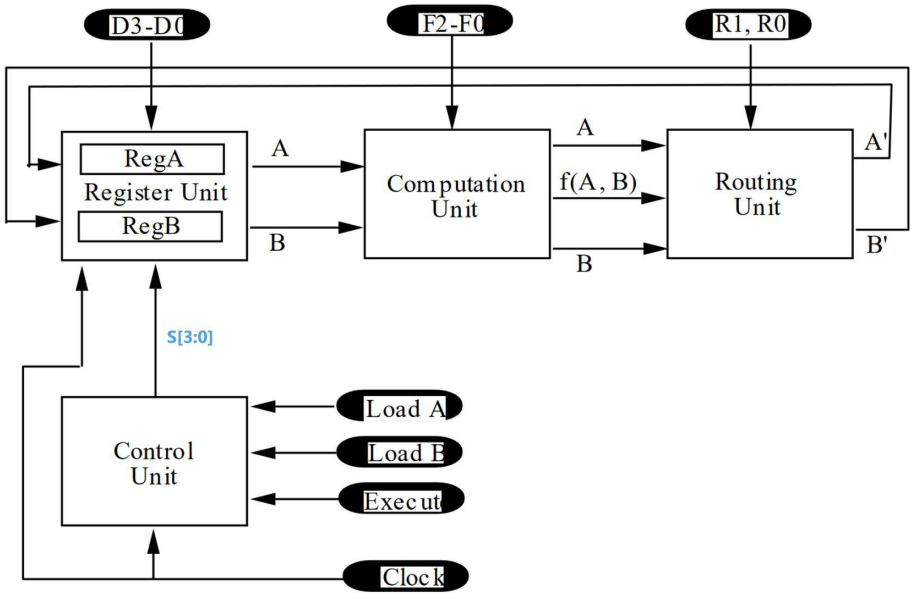
Lab 2.1 aims to use TTL design to implement a 4-bits bit-serial logic processor which can perform 8 functions including AND, OR XOR, NAND, NOR, XNOR, 1111 and 0000. Such operation is carried out at a frequency of 1 bit per cycle, and 4 cycles per execution. The result of operation will route back to the 4-bit shift register in 4 different ways according to the signal input on the routing unit. It consisted of four units: the register unit, the operation unit, the routing unit and the control unit. The lab 2.2 extends such design to 8 bits and aims to implement such a logic processor using an FPGA board.

Operation of the logic processors

In order to load data into the registers, it's necessary to flip the 4-bits input switches to the input the user wants, and then flip on the load A switch. Then, the user may switch off load A, change the 4-bits input switches, and then flip on load B.

In order to initiate a computation and routing, the user may adjust switches F[2:0] and switches R[1:0] to the desired function and routing method, and then switch on the execute switch to start the operation.

Written description, block diagram and state machine diagram of logic processor



Graph 1. high-level block diagram

The 4-bit register unit is used to pass bits, store bits and load new bits. It consists of 8 LEDs, two 4-bit bidirectional universal shift registers, 2 switches for load A, load B, and 4 switches for input bit. Registers are arranged in parallel. LEDs are used to demonstrate the current bit

stored within both registers. 4-bit bidirectional universal shift register is a suitable shift register with 4-bit parallel input and shift right serial input. Several resistors are used for LEDs and switches for protection and float prevention respectively.

The computational unit is used to 1-bit conduct operation on the bit passed by shift register unit according to the switch input F [2:0]:

Computation Unit			
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

Table 1. Function of computational unit

In order to conduct these operations, we utilize a NAND gate, a NOR, and an XOR gate to get the result of AND, OR, and XOR. Then we use the 5V V_{cc} as ”1”. After realizing that the result of all operation with F2 = 1 is the negation of that of all operation with F2 = 0, we use a 4-to-1 mux with S1 = F1, S0 = F0, input 00 as result of AND, 01 as result of OR, 10 as result of XOR, and 11 as “1”. These operations have F2 =0. Then, we XOR the output with F2, so that the result will be negated if F2 = 1.

The routing unit is used to design the way results route back to the register unit using the following rule:

Routing Unit			
R1	R0	A'	B'
0	0	A	B
0	1	A	F
1	0	F	B
1	1	B	A

Table 2. rules of routing unit

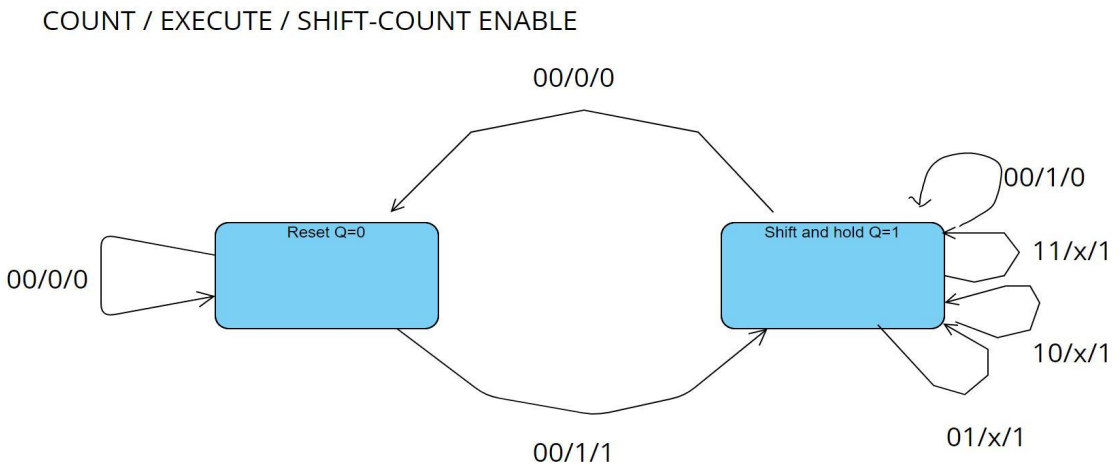
It consists of 2 4-to-1 mux, one of which gets A’ while the other mux gets B’. For each mux, R1 = S1, R0 = S0, and the result of the operational unit together with the original value of A and B are inputted according to the rule above.

The control unit is a Mealy state machine with two states that controls the shifting of the two registers. It has four state machine components (S-shift count enable, Q-state, C1-counter, C2-counter) that are determined by four inputs(E-execute, Q-state, C1-counter, C2-counter).

S, or shift-count enable, will be the output that controls the shift register, Q is the one of two states the Mealy state machine is in, and C1, C2 are served as counters that count the number

of bits that have been operated already. Since Q, C1, and C2's value are necessary to determine the next state, 3 flip-flops are connected to store its value in the last state.

The output S and the input load signal are used to determine the S1 and S0 input for both registers. If S = 1, there will be a right shift in both registers(S1S0=01) regardless of the value of load. If S = 0 and load = 0, then the registers will remain still(S1S0=00), but if S = 0, load = 1, both registers will perform a parallel load(S1S0=11).



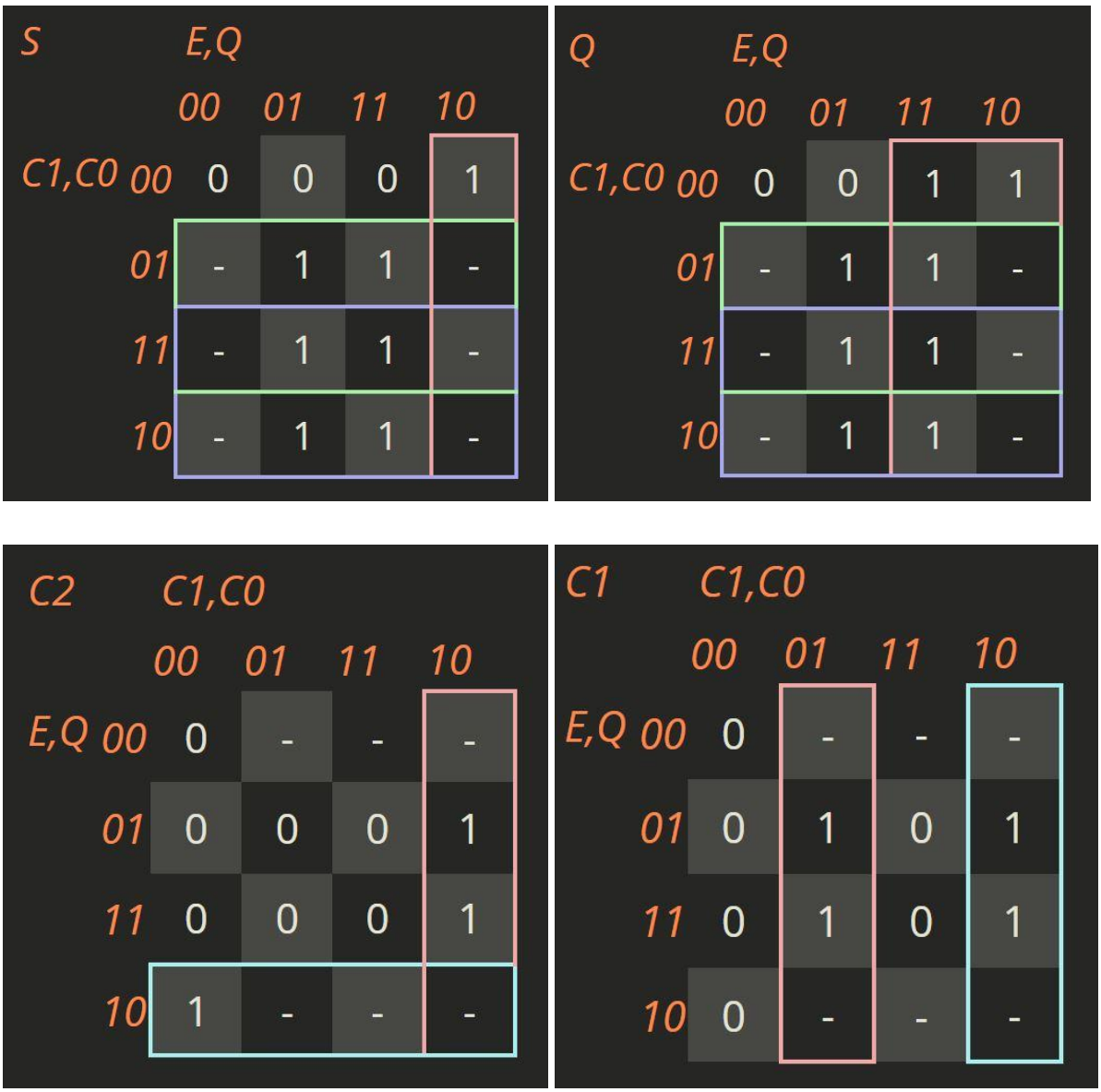
Graph 2. State machine diagram

Design steps taken and detailed circuit schematic diagram

K-maps and truth tables are both used in designing the Mealy state machine in the control unit.

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

Table 3. Truth table for mealy state machine



Graph 3. K-maps for S, Q, C1, C2

S(C1, C0, E, Q) = EQ' + C0 + C1

Q(C1, C0, E, Q) = E + C0 + C1

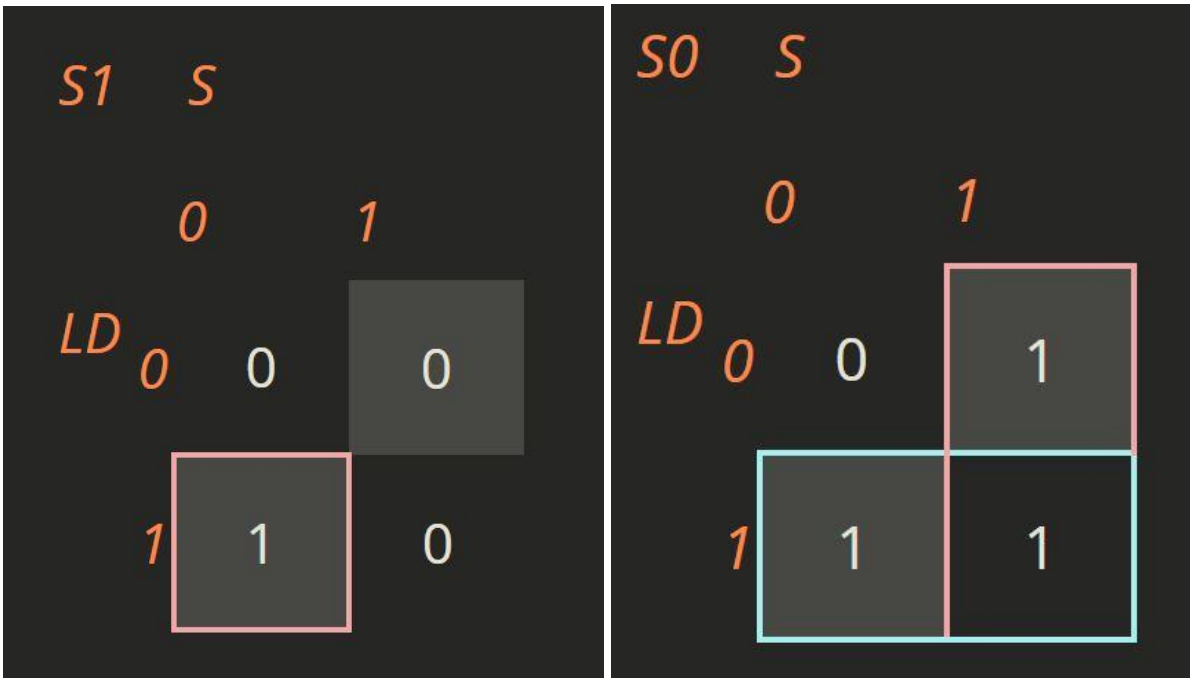
C1(E, Q, C1, C2) = C1'C0 + C1C0'

C0(E, Q, C1, C2) = EQ' + C1C0

In order to determine S1 and S0 input for the register, another truth table and k-map are used.

	S	LD	S1	S2
hold	0	0	0	0
load	0	1	1	1
right shift	1	X	0	1

Table 4. truth table for register input S1 S2.

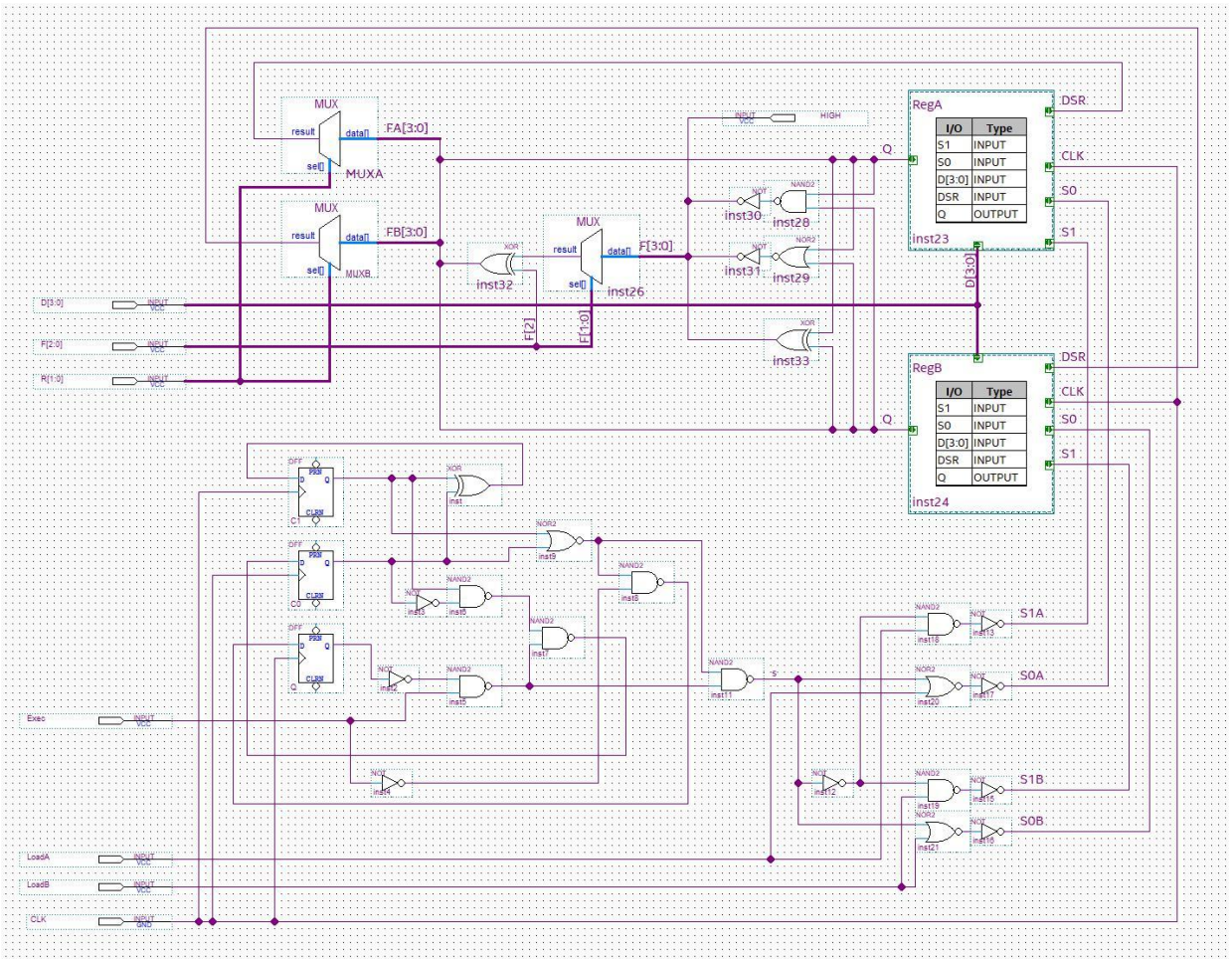


Graph 4. k-maps for S1 and S0

We considered some other designs when implementing the circuit. First, the C1 and C2 in the mealy state machine can be replaced by a counter chip, but this could lead to more chips used, since the current method enables us to reuse many of the idle XOR gates left in the operational unit. There aren't many differences between these two methods.

We also considered the usage of 4-bit parallel-access shift registers(195) as shift registers at the very beginning of our design. Yet, its inputs are parallel inputs, which is not very suitable to the aim of the logic operator which carries out 1 bit operation per clock cycle.

The detailed circuit schematic is shown on the next page. We have used two blocks to represent Register A and Register B. The "OUTPUT" labeled bit Q is the rightmost bit stored in the register. The correspondence between the input bus and the selection bus of the MUXes is not specified in the schematic for conciseness, which has been shown clearly in Table 1 and Table 2.

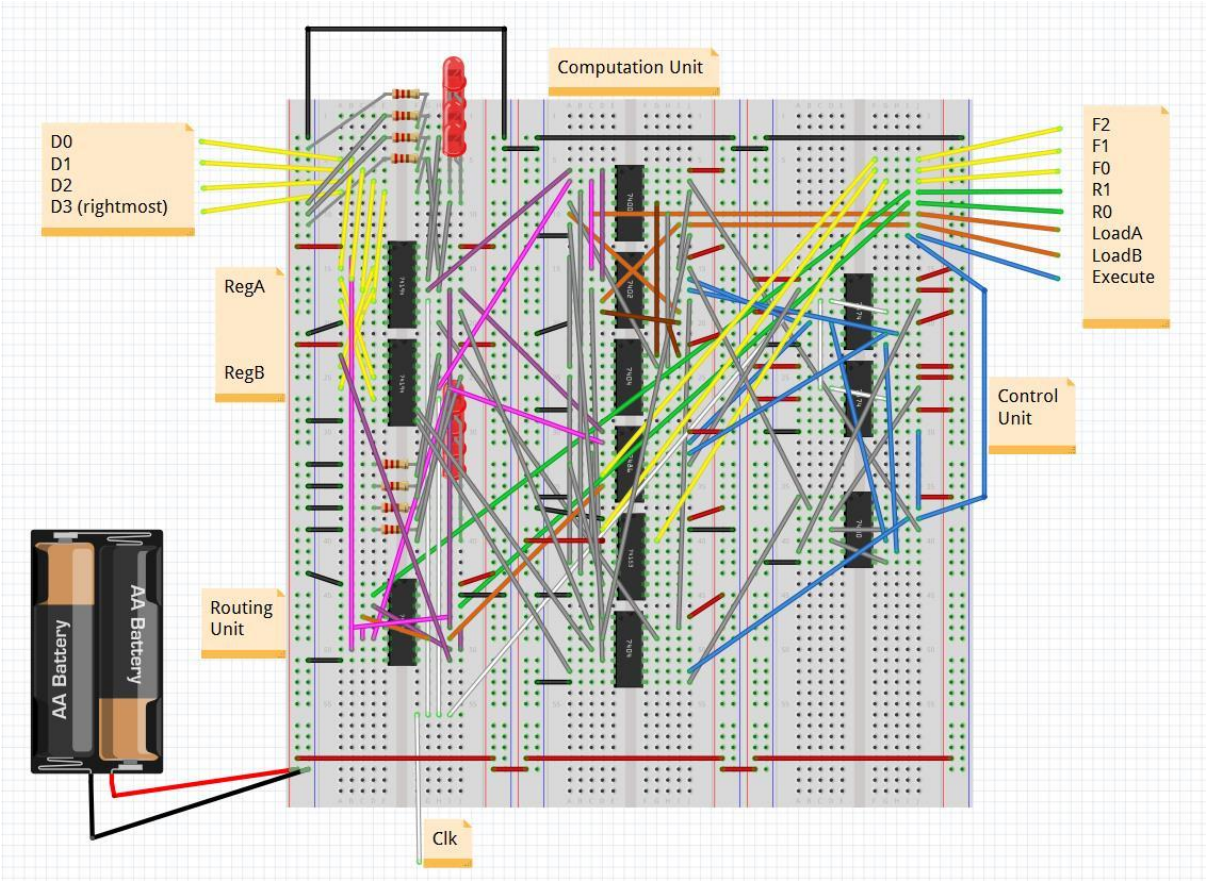


Graph 5. Detailed Circuit Schematic

Breadboard view / Layout sheet

The breadboard view of the circuit is shown on the next page. Inputs have already been labeled. Each output bit of both Register A and B is connected to an LED and a resistor to visualize the contents in both registers. Following is the explanation of the color scheme:

- Red: connected to power supply.
- Black: connected to ground.
- Gray: logic output from 74xx chips.
- Pink: rightmost bit of Register A.
- Purple: rightmost bit of Register B.
- Blue: control unit bits.



Graph 6. Fritzing Breadboard View

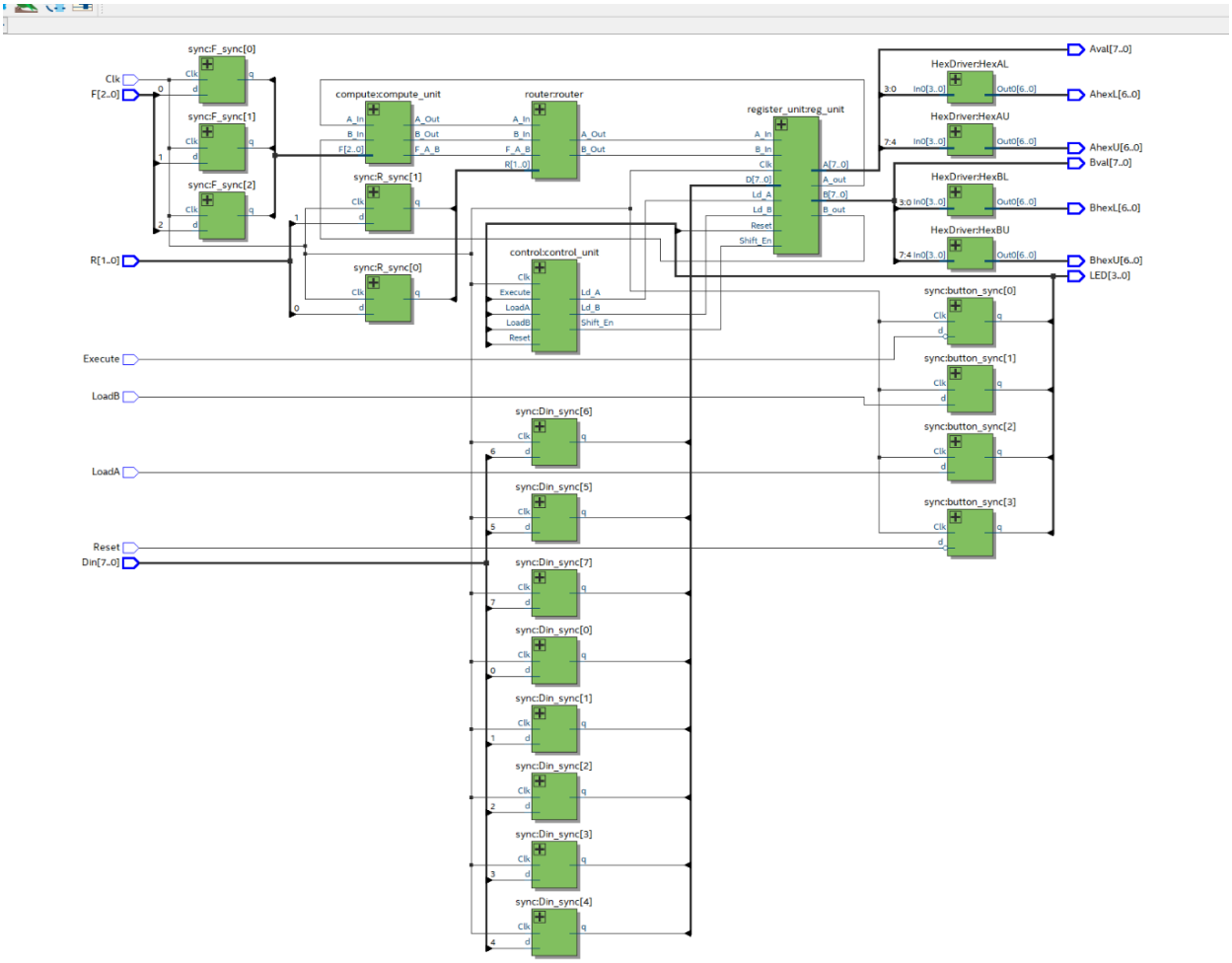
8-bit logic processor on FPGA

The original 4-bit processor has the following modules: reg_4, register_unit, router, compute, synchronizers, hexdriver, control, and processor. Reg_4 is a 4-bit shift register, register_unit is the unit of two reg_4 instances, router directs the output of the computational unit to the registers specified by R1 and R0, compute is the computational unit, synchronizer is used on asynchronous inputs from the switches so that they are changed on rising edge of the clock, hexdriver passes values to the LEDs on the FPGA board so they light up in readable format, control is the FSM control unit, and processor is the top-level entity that creates instances of all necessary modules and integrates them. There is also a testbench module that helped us debug.

We have left Router.sv, compute.sv, synchronizers.sv and hexdriver.sv unchanged. This is because Router and compute are 1-bit units based on serialization, synchronizers work for any number of bits, and hexdriver supports only up to 4 bits. We first switched the reg_4 module to a reg_8 module, changing input and output data logic from [3:0] to [7:0]. Shifting and reset logics are also changed, reset now writes 8 zero bits and shift now concatenates the new bit with the leftmost 7 bits. Then the register_unit module is modified to fit two reg_8

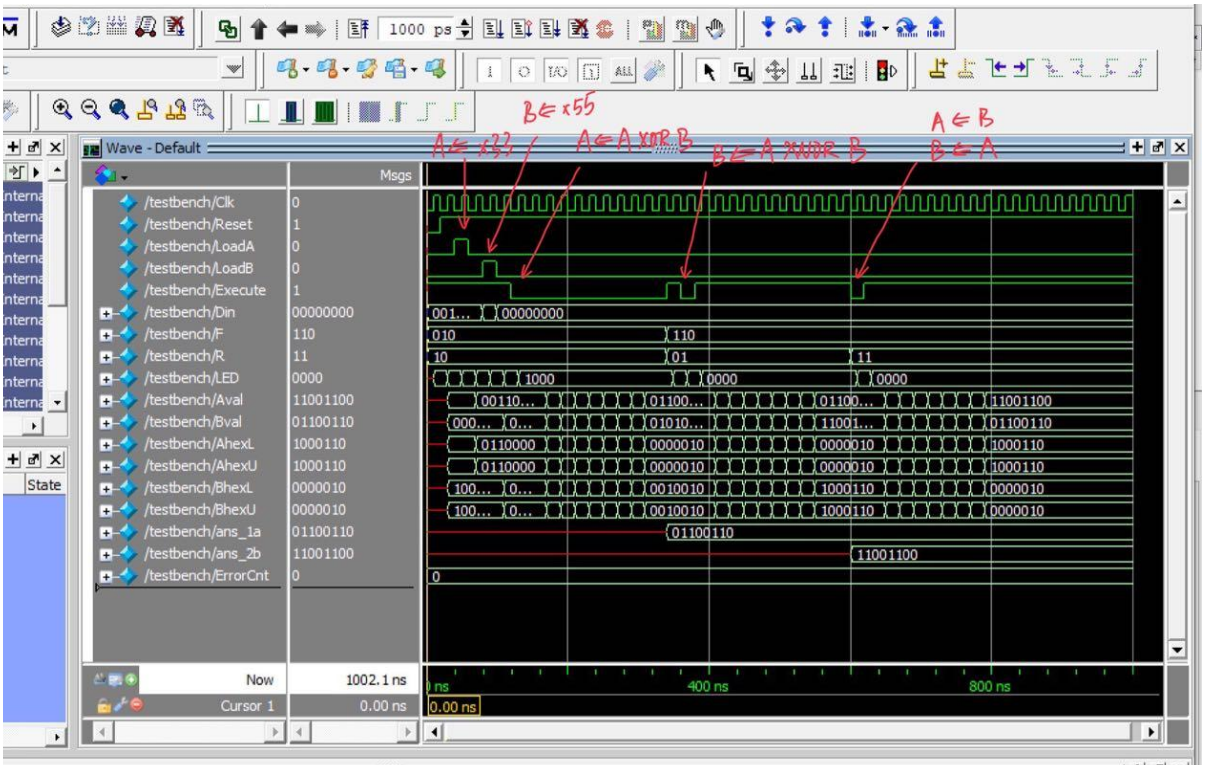
instances. After that, we changed the control module. Originally, there were only 6 states since 4 shift states were needed. Now 10 states are needed because there is one initial state, 8 shift states, and one final state. Therefore, we extended the enum “curr_state” and “next_state” by one bit to fit up to 16 states, and added 4 extra states as shift states. In the “unique case” part, we changed the order of the states so that 8 shift states are gone through before the final state. Finally, we changed everything related to registers in Processor.sv from [3:0] to [7:0], and we added two hexdrivers to observe A[7:4] and B[7:4].

The RTL block diagram of the top level is shown below.



Graph 7. RTL block diagram

On the next page, a ModelSim simulation of the processor using the 8-bit testbench provided is shown



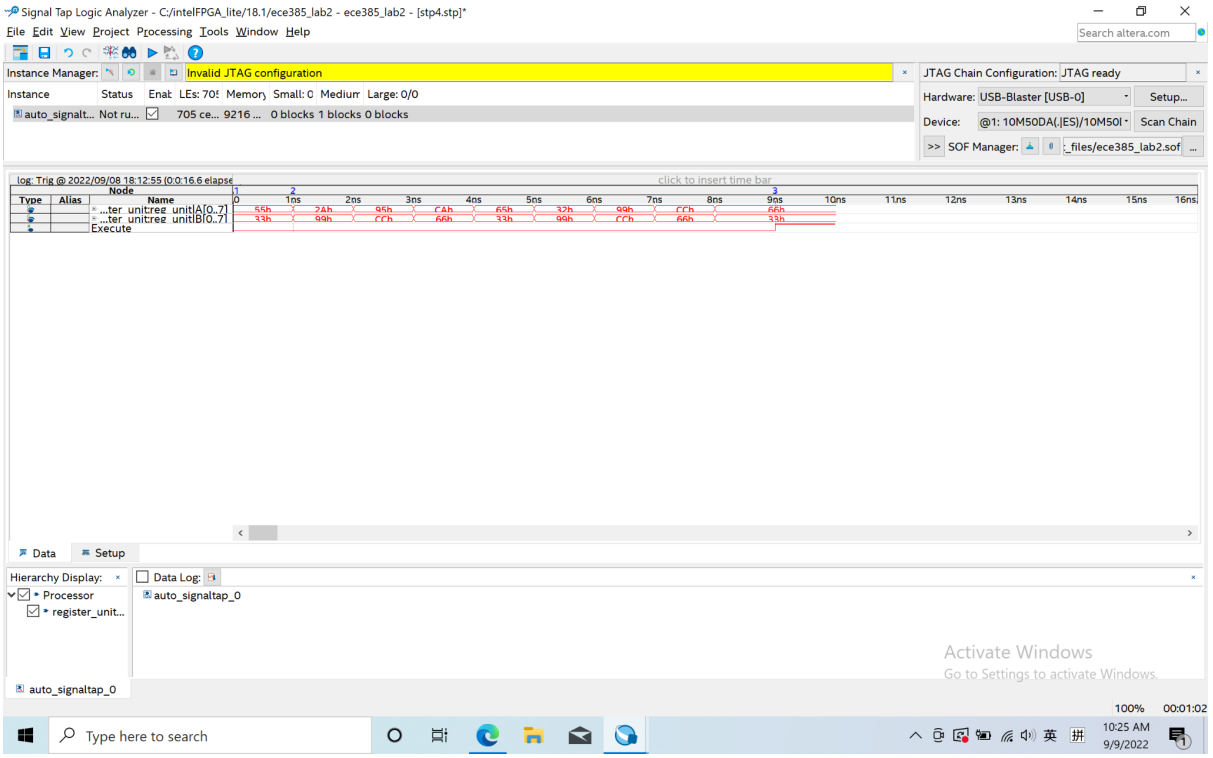
Graph 8. ModelSim Simulation

In this simulation, each instruction is executed when Execute is 0. This is because Execute is inverted in the synchronizer for FPGA board keys. First, Din is changed to x33 and Reg A is loaded at the positive edge of LoadA. Din is then changed to x55 and loaded to Reg B. In the first execution, F=010 and R=10, which means XOR and load result f(A, B) to Reg A. Then the control unit begins shifting states. After shifting, the value of Reg B becomes x66. F and R are switched to F=110 and R=01, which performs B <= A XNOR B. Finally, F is unchanged and R is switched to R=11, which performs a swap of A and B. All the answers are directly calculated in the testbench and the simulated values are compared to the answers to check if the output values are all correct.

To generate SignalTap ILA trace, the steps needed were:

1. Insert three signals: Execute, register_unit:reg_unit A[7:0], B[7:0]
2. Set the Trigger Enable on signal Execute.
3. Set the Trigger Conditions of signal Execute to “either side”.
4. Switch to “Data” in the lower left corner.
5. Load 8’h55 into Reg A and 8’h33 into Reg B on the FPGA board.
6. Click “Run Analysis” in the upper left corner.
7. Click the key representing “Execute” on the FPGA board.
8. Click “Stop Analysis” in SignalTap, the results will show on the screen.

The generated waveform is shown on the next page.



Graph 9. SignalTap Trace

The values of Register A and B are recorded and shown in the trace. The initial values of Reg A and B are 8'h55 and 8'h33. The final values of Reg A and B are 8'h66 and 8'h33, which are as expected because A <= A XOR B is performed. The middle values are useful for debugging if any unexpected final value appears.

Description of all bugs encountered, and corrective measures taken

During the design of the computational unit, we used switches to control inputs. Yet, we connected the switches directly to the chips and encountered a floating value. The illuminance of LEDs that monitor outputs could be influenced by any object beside it. We eventually realized that this was due to the incorrect connection of switches. The charges will build up when the switch is open, causing the wire to perform as an antenna. In order to fix this, we reordered the circuit so that the power source is linked to the chips through a resistor, and the switch is connected between the GND and the wire entering the chips. When the chip is open, the input to the chips is "1", and when the switch is closed the charges flow to the GND, and the input to the chips is "0". Using this method, the charges will no longer build up since they will flow to the ground when we want to input a "0" to the chip.

Conclusion

This lab aims to use TTL design to implement a 4-bits bit-serial logic processor which can perform 8 functions and route back the result in 4 different ways. Its design is split to 4 units, each of which has a different function. This design was further extended to 8-bit using an FPGA board and Systemverilog.

Answer the post lab questions:

By connecting the signal A and signal B to an XOR gate. We can use A to control whether to invert B. If A is 1, signal B will be inverted, and if A is 0, signal B will remain still.

A modular design enables the designer to separately debug each module, which contains fewer components, making it much easier than debugging the circuit as a whole. Thus, the easier debugging can save a tremendous amount of developing time. The testability is also improved, since the designer can test the functioning of each module before assembling them together.

A mealy machine is faster, since its output changes right after an input change occurs. The mealy machine also has fewer states than the moore machine. In this case specifically, the moore machine needs 3 states, which need two bits to represent, while the mealy machine only needs 2 states, which can be represented by one bit, “Q”. Yet, the moore machine is rather safe to use, since it only transfers to the next state at the rising edge of the clock. The Moore machine is also easier to build and design, since the next state only depends on the current state.

ModelSim is a simulation tool. It uses a user defined testbench written in SystemVerilog to generate a simulated waveform. SignalTap, on the other hand, collects the actual signal that is present on the FPGA board after the compiled SystemVerilog program is loaded onto the board. The advantage of ModelSim is that the user can debug more easily and has more accurate control over the input signals. Input signals can be switched with accuracy in nanoseconds. It can also store longer signals than SignalTap, since the on-chip memory is limited. The advantage of SignalTap is that it collects actual signals so it can make sure the program works as expected on a real FPGA board instead of simulation. Also, in many situations, writing a full testbench is more laborious than loading the program onto the board and providing inputs using switches on the board. The output can be viewed on the on-board LEDs easily, and the internal signals can be captured in SignalTap.