**ECE 385**

Fall 2021

Experiment #2

**Experiment 2**

Steven Dimov

Section AB6 N/A

TA: Abigail Wezelis

**Purpose:**

The purpose of this lab was to design a bit-serial logic operation processor capable of eight different functions and four different routing paths by using a finite state machine. Specifically, to physically create the design by utilizing the hardware that comes with the lab kit, such as two 4-bit shift registers, several multiplexers, and the FPGA for its counter and virtual switchbox.

**High Level Block Diagram**

The purpose of the circuit is to perform logical operations (AND, OR, XOR, 1, and their complements) on four bits held in two shift registers and load the results of those operations back into the registers. It does this through combinational logic that sets the leftmost bit inputs of the shift registers equal to the function’s output based on the rightmost bits of the registers. When the registers are shifted four times, the outputs of the function occupy the shift registers based on the positions from which they came. The function can be routed in four ways: either maintaining the previous 4 bits in each register, switching them, or making the first or second shift register hold the output of one of the logical operations based on the previous bits.

The modular design of this project allows for more ease in debugging and testing since each module has its own expected output and truth table, so feeding in inputs to individual modules can assert functionality of that individual component as opposed to testing everything all at once, which provides no insight on what went wrong.

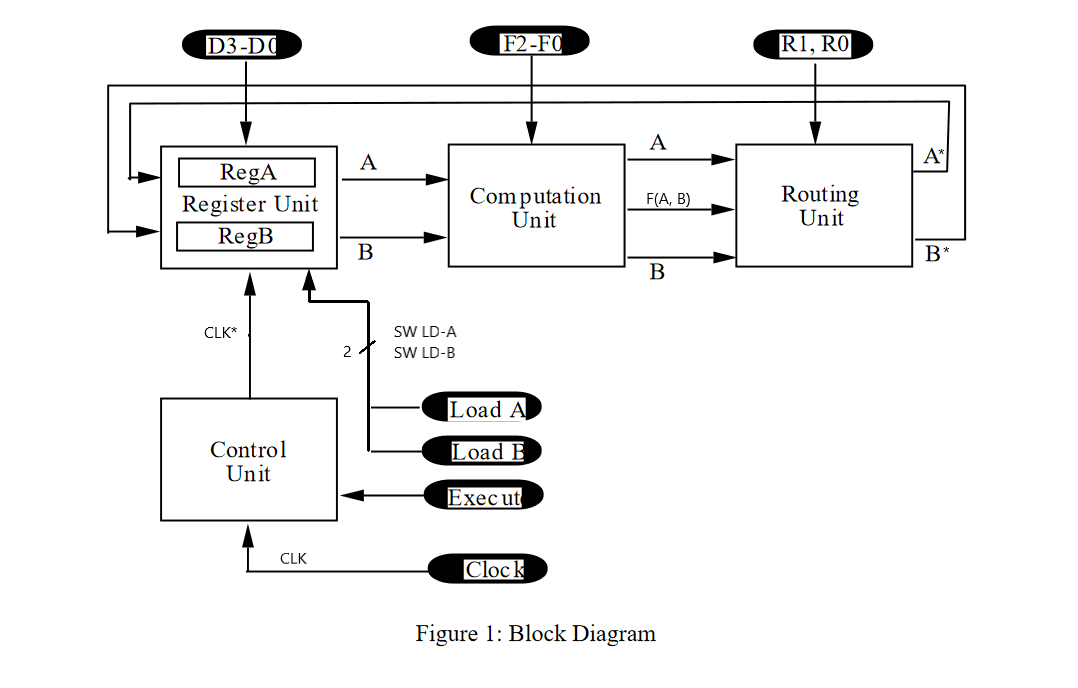


Figure 1: High-level block diagram of TTL circuit.

**Written Description of Circuit**

The eight different logic operations implemented in the experiment are as the lab instructions provides:

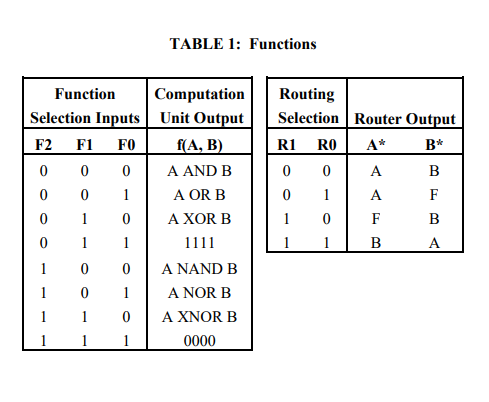


Table 1: Truth tables for the ALU (computation unit) and routing unit. These are controlled by F2-F0 and R1-R0 respectively.

The design of the ALU logic functions is divided in half, since half of them are simply a negated variant of the other. This allows the design to comprise of simply computing the entire bottom half of the table and then using a 4:1 multiplexer to select through the desired output using two inputs F0 and F1. Then using another input, F2 will control whether or not the selected output of the 4:1 mux will be inverted, reducing the number of gates in the design and expanding the capability of four functions to the full eight function table. Specifically, this is done by inverting the third selecting input F2 and XORing it with the output of the 4:1 mux to match the table. This control of the function table selection is shown through the schematic below: (The IC being used for the multiplexer in this case is SN74HC153N)

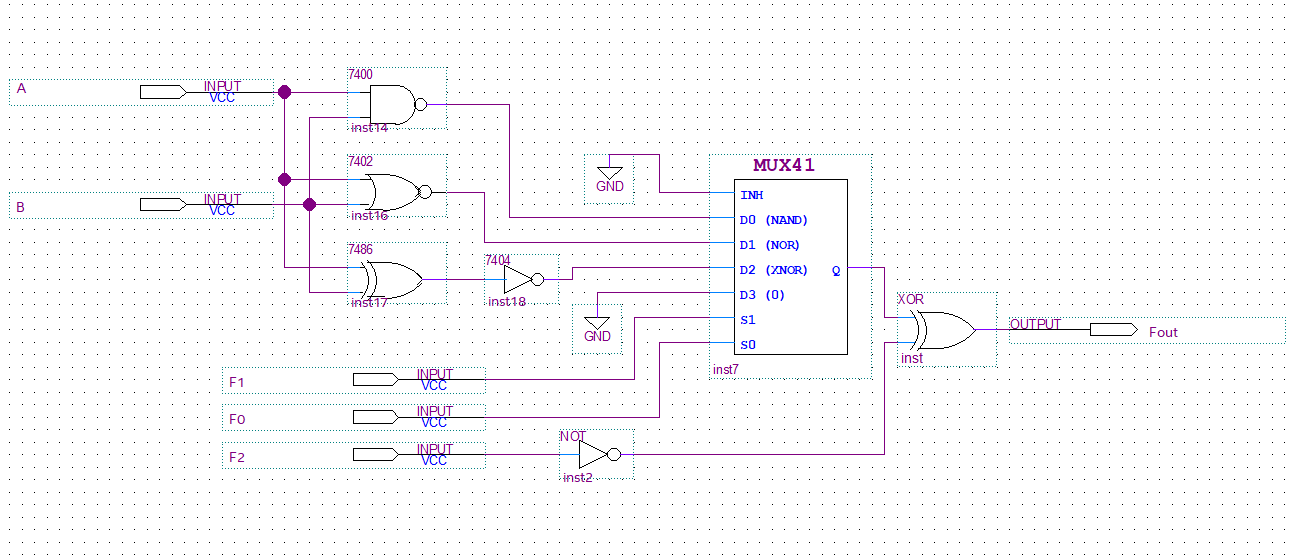


Figure 2: Gate-level schematic of the computation unit.

The output F computed by the circuit above along with the original unaltered input signals A & B are then fed into the routing unit. Then, using two 4:1 multiplexers, A, B, and F are routed to the corresponding selection values that match the expected routing truth table for outputs A\* and B\*.

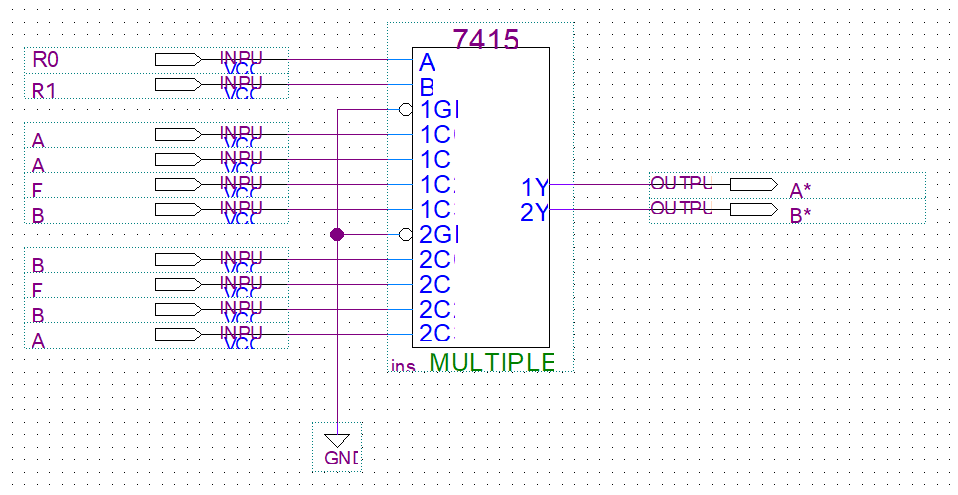


Figure 3: Gate-level schematic of the routing unit.

Using combinational logic, the equations for the states were expressed in NAND gates. The signal a was ANDed with the clock signal, the result of which was used to control shifting.

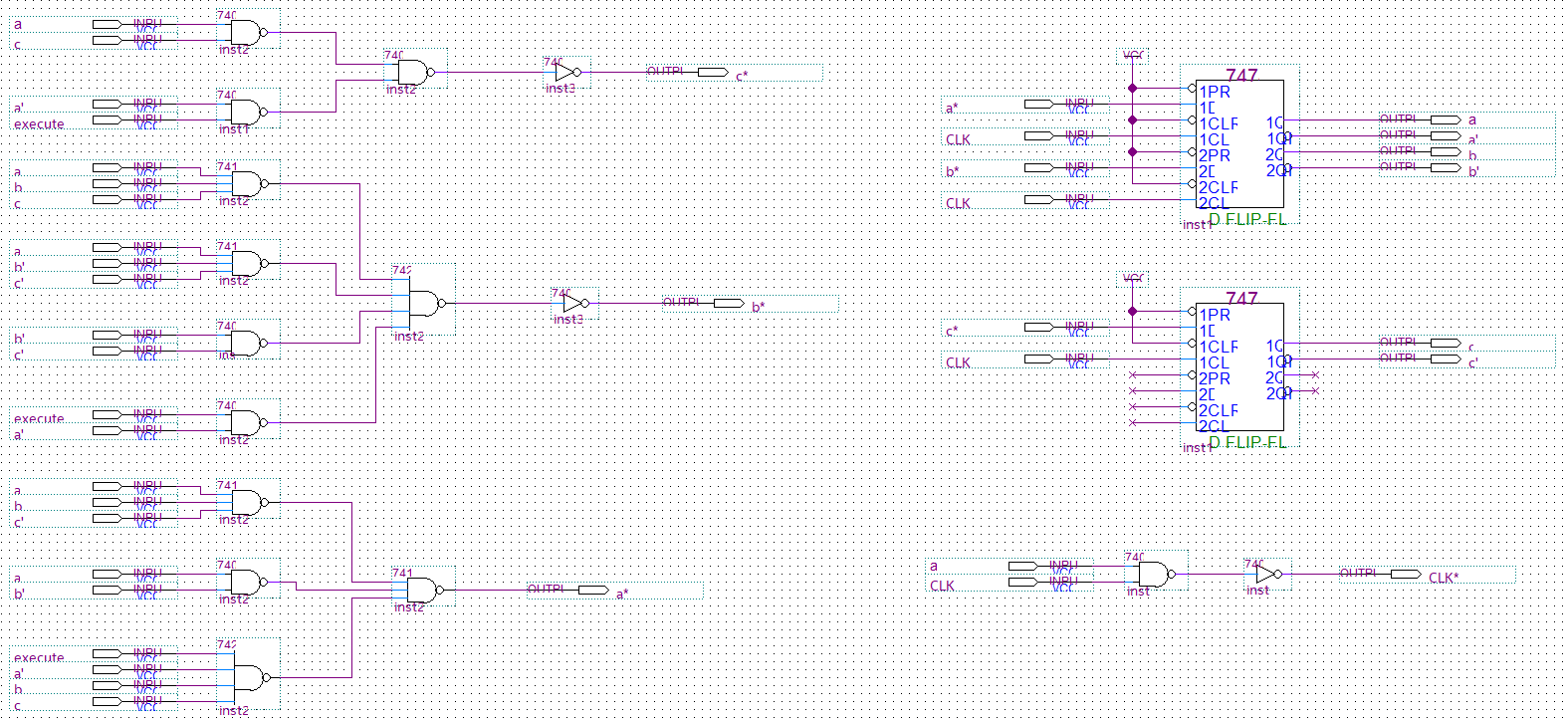


Figure 4: Gate-level schematic of the control unit.

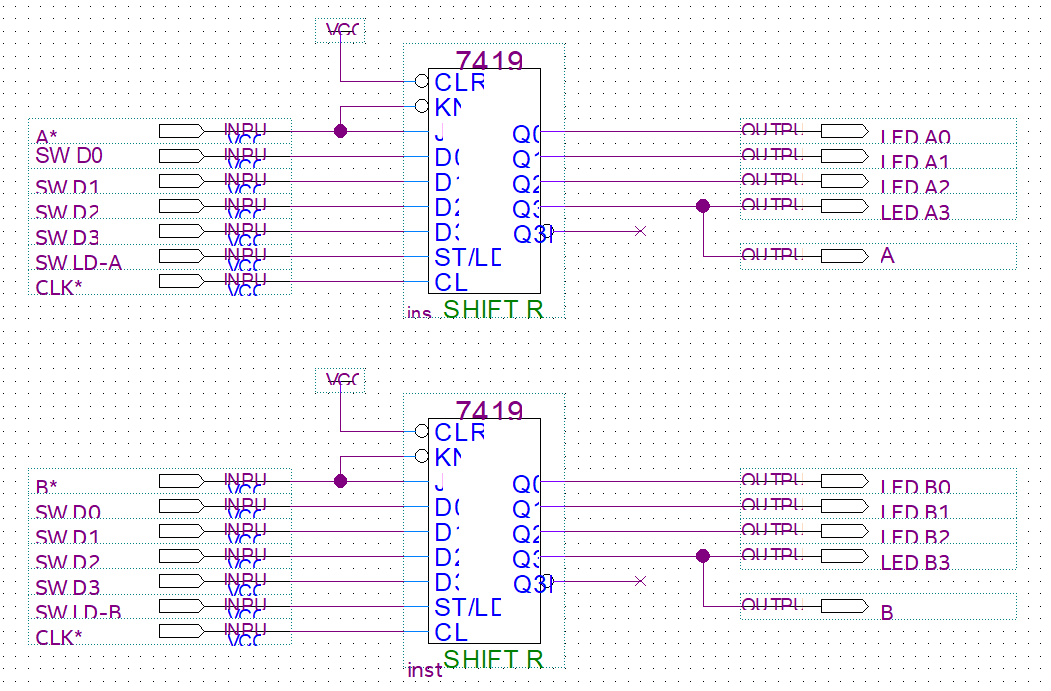
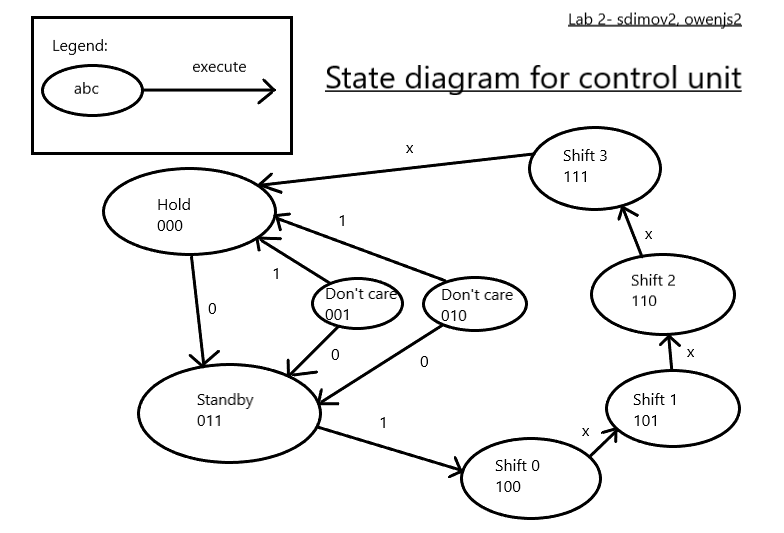


Figure 5: Gate-level schematic of the register unit.

For the first part of this lab, a moore machine with six relevant states was used. These states were represented with three bits, and the two leftover states were configured to only move to the hold and standby states. The most significant bit of the state was the signal to shift the registers, and it occurred in four consecutive states that cycled regardless of execute input. The last of these shift states led to a hold state, which would not return to the standby state unless the machine received an execute input of 0. In the standby state, the machine does not change state unless it receives an execute input of 1. When it does, it would move to the first of four shift states.



With these three bit values for states in the moore machine, K-maps were used to calculate the combinatory logic for each bit. For the least significant bit, c, the equation c\* = (a’ + c’)(a + execute’) was determined. For the middle bit b, the equation b\* = (a’ + b’ + c’)(a’ + b + c)(b + c)(execute’ + a) was determined. For the most significant bit a, the equation a\* = (a\*b\*c’) + (a\*b’) + (execute\*a’\*b\*c) was determined. Though the equations for c and b were POS and the equation for a was SOP, all were expressed with NAND gates using DeMorgan’s laws.

Both a moore machine and a mealy machine were considered for the control unit, but a moore machine was chosen for testing reasons. Though a mealy machine requires less states and (in the case of the suggested circuit with a counter) less components, the greater number of states in a moore machine allowed for more interpretation in testing. Being able to observe the changes between distinct states based on inputs made

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| c\* |  |  |  |  | abc |  |  |  |  |
|  |  | 000 | 001 | 011 | 010 | 110 | 111 | 101 | 100 |
| execute | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
|  | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |

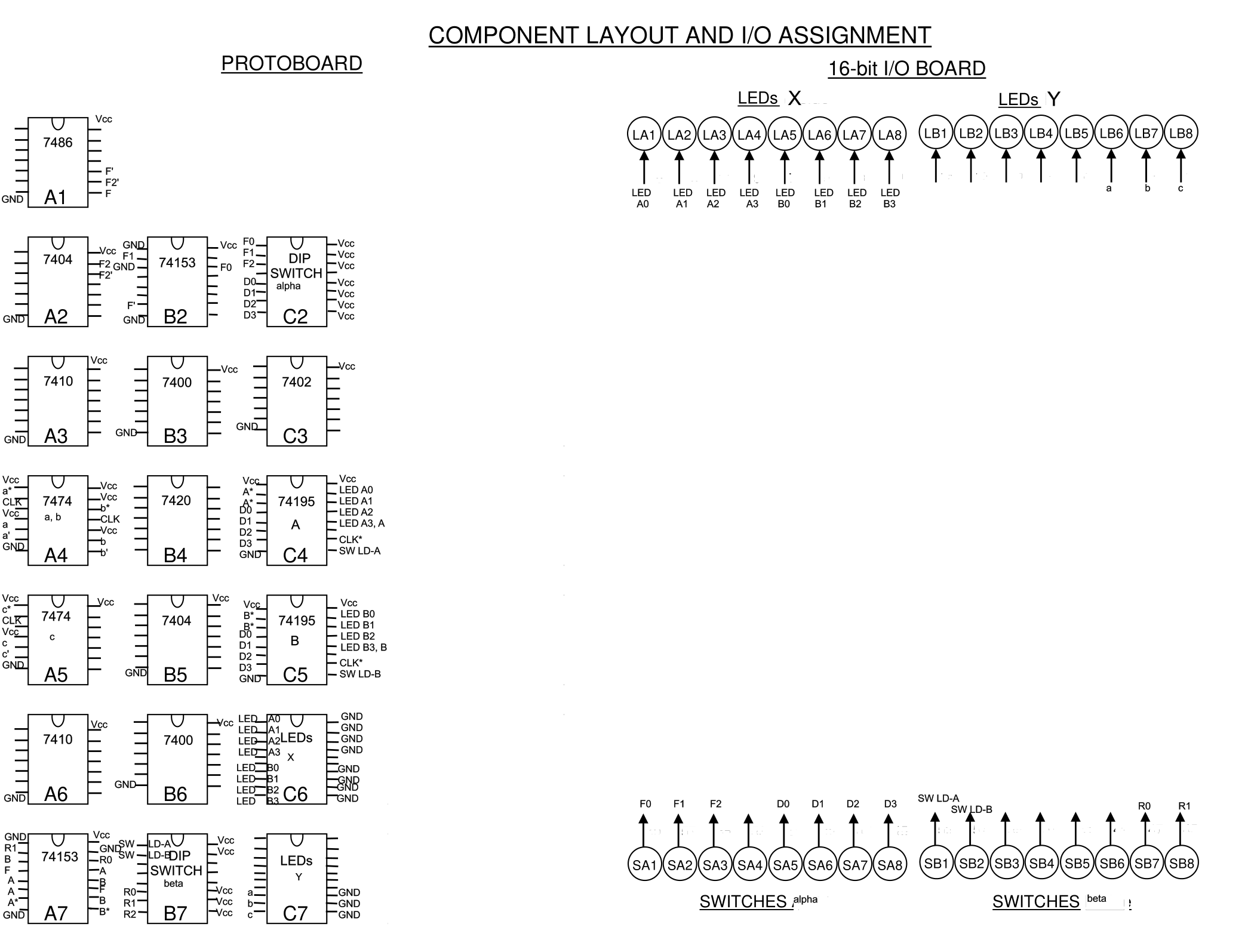
Table 2: The K-map for the least significant bit of the control unit state. This was simplified to the POS equation c\* = (a’ + c’)(a + execute’).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| b\* |  |  |  |  | abc |  |  |  |  |
|  |  | 000 | 001 | 011 | 010 | 110 | 111 | 101 | 100 |
| execute | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
|  | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

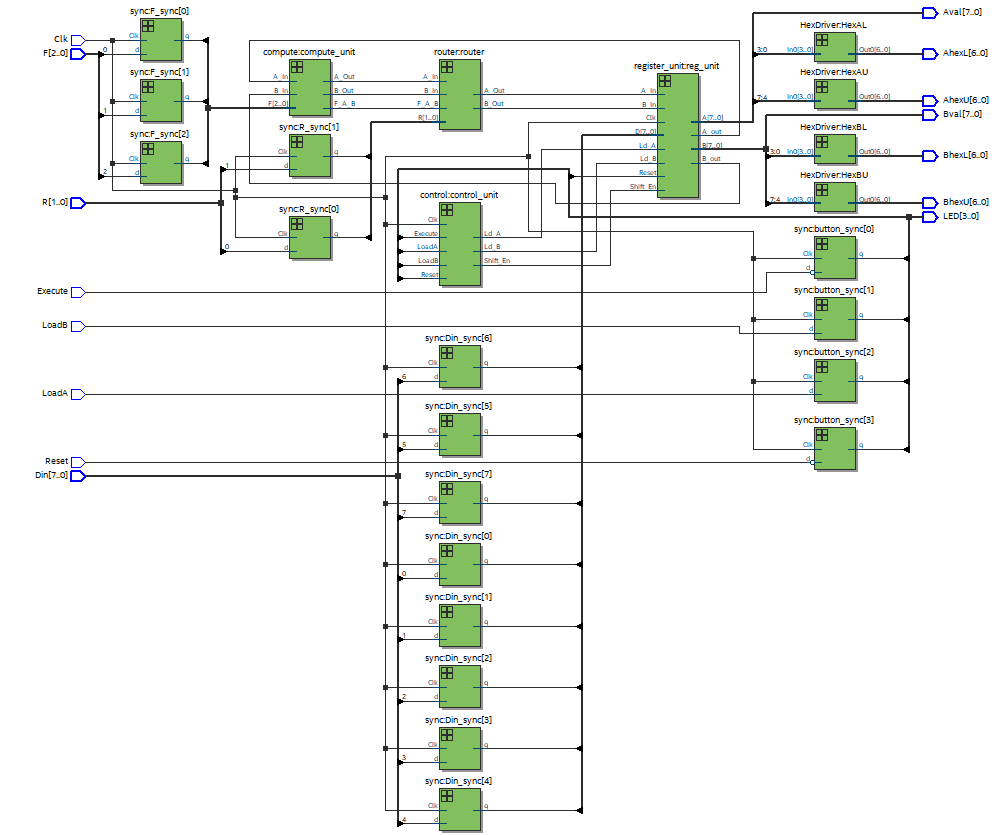
Table 3: The K-map for the middle bit of the control unit state. This was simplified to the POS equation b\* = (a’ + b’ + c’)(a’ + b + c)(b + c)(execute’ + a).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a\* |  |  |  |  | abc |  |  |  |  |
|  |  | 000 | 001 | 011 | 010 | 110 | 111 | 101 | 100 |
| execute | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
|  | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |

Table 4: The K-map for the most significant bit of the control unit state. This was simplified to the SOP equation a\* = (a\*b\*c’) + (a\*b’) + (execute\*a’\*b\*c).



GD0-D3 are used to parallel load the data to the registers, using SW LD-A and SW LD-B to select whether or not A or B respectively will receive the set data in D0-D3. There are dipswitches on the breadboard that attach to the function selection input F0-F2 as well as the routing selection inputs R0 and R1. The FPGA buttons are used for the execute and reset operations. The sequence of a use of the machine would be to set the switches D0-D3 to the desired input for either A or B, select whichever input that would be by turning either SW LD-A and SW LD-B and turning the other off (unless they are the same), then repeat the process for the other load input if necessary. Next, the dipswitch for the function select is set to the binary values that correspond to the desired function. The same process is repeated but instead for the dipswitch for the routing selection. Finally, the execute button is pressed to yield the output. The process can then repeated for the next desired computation.



The two simulation/testing tools used in this lab, ModelSim and SignalTap, differ by how they test the code. ModelSim simulates the code in the computer that was used to code it, and can use a test bench file to alter signals. This is advantageous because it may be faster to compile code in the machine rather than the FPGA and the testbench files allow a good amount of control. SignalTap uploads the compiled sof file to the FPGA and observes the signals for a short duration after a trigger signal. This offers the advantage of not being a simulation, but an actual test. This way, the user can know that the results reflect the real behavior of the hardware more accurately.

This experiment has provided insight in methods of design such as modularity, reducing components, and testing and debugging experience by using SignalTap and ModelSim to test out hardware and software simulations. In out experiment, we used a Moore state machine, but after this lab we have realized that a Mealy state machine could have had less states, saving time on design and possibly performance or debugging.