Lab 3

A Logic Processor

In this experiment, you will design and build a bit-serial logic operation processor. The design will utilize two 4-bit shift registers, several multiplexers, and some type of counter. The circuit will be capable of performing seventeen different functions.

Assignment

- Read the Experiment #3 (updated 9/6/17 for minor notation improvements), build your circuit, and complete the Lab 3 Pre-Lab before the lab section. You will need the information from the Data Sheets to correctly wire up your circuit.
- Take a look at the TTL wiring guide for helpful hints on building your circuit.
- Check out this animation for help in visualizing the operation of the circuit.
- Work on Lab 3 report (outline/checklist here) and Lab 4 Pre-Lab after the lab section.

Demo

Everything should be clocked fast (1kHz+) except for the last demo point.

- Show correct loading of the A and B registers. (1 point)
- Show that the computation cycle is the right length. We will hit execute and hold it high. The shift registers should shift a total of 4 times. No more, no less. (1 point).
- Demonstrate the four routing operations. (1 point)
- Demonstrate the 8 functional operations. (1 point)
- Show that the computation cycle completes even if execute returns to 0 mid-computation. This will be clocked slowly (1 Hz), and execute will be switched high. As soon as it starts shifting, execute will be switched low. The circuit must complete its computation cycle as normal. (1 point)

FAQ

- **"I don't really understand prelab A, could you clarify it for me?"**
  - Prelab A asks you to describe the simplest circuit that takes in two inputs: a Signal and a Mode. When Mode is one value (0 or 1, doesn't matter), the circuit outputs the original Signal; when Mode is one value (1 or 0, on the other hand), the circuit outputs the inverted Signal (Signal'). You should consider using such a circuit alongside a simpler multiplexer (as compared to 8:1) to reduce the number of combinational functions you have to create.

- **"I'm not sure i understood the meaning of the state signal (Q). how can we use it in the logic design?"**
  - In a state machine, different states represent different situations for your circuit operation. Our particular Lab 3 can be split into two broad situations: circuit running, and circuit not running. Of course, a Mealy machine's state does not solely dictates the current activity of the circuit, but instead deduces the activities through a combination of the state values and the various inputs/signals. Put it in simple sentence: the state signal Q tells you what condition your circuit operation is currently at, and also on the next clock edge what your circuit has to do next.

- **"For both the mealy and moore state machines, one of the required inputs is 'state' but that is not an already defined input. How do we go about making a logic for the state input?"**
  - Think of your state machine as a regular truth table first, with all of its inputs and outputs on each side of the table. The only difference is that for a regular truth table, your inputs and outputs are separate signals (e.g. inputs come from the switches, and outputs go to the LEDs). On the other hand, the 'state' signals in the state machine is both the input and the output on the truth table. That is, given a current state value plus some other inputs, the circuit will determine what the next state value is, and update the current state value with the next state value.

    This can be done either synchronously or asynchronously. If you want your state value to be updated synchronously, you should use flip-flops to store and update the state bits accordingly. If you want your state value to be updated asynchronously, you just have to use latches in place of the flip-flops. However, notice that a latch just matches its output with its input, it is transparent to the circuit, which means you don't even need to use latches in the first place for asynchronous state machines.

    Either way, once your circuit has determined what your next state value is, that signal is essentially your state value, with or without the flip-flops or latches. And since you need the current state value to determine the next state value, you just have to feed those state signals back into your circuit as inputs while forming a loop.

    Note that this lab requires you to build a synchronous state machine. Therefore you should use flip-flops to store the transitioning bits (Q',C1',C0')

- **"Just to clarify, the two inputs Load A and Load B indicate that if the control unit is not in a computation cycle, load the register(s) that has a high input with the values D3-D0, correct? Therefore, Load A and Load B are not asynchronous, yes? "**

- The load operations should be parallel load, and it should be done in the initial state only. There are extra logic you can make to restrain the loading operation to be allowed only in the initial state, but it is not required for the demo purpose.

  If you are in the initial state, then nothing is moving. When you flip up the LoadA / LoadB, the logic unit is going to raise the load signal to the shift register (this is processed immediately, i.e., asynchronously), then the shift register will parallel load D3-D0 on the next clock edge (this is synchronous).

- **"How are you going to check the length of the computation? "**
  - If your computation works correctly, then that automatically implies that your computation cycle is correctly done, and points will be automatically given. However, if your computation doesn't work correctly, you will have to find ways to show us that your state transition works, for example, by showing the state transitions in binary form.

- **" ... for the don't cares of counter. Even if we are at state not executing, wouldn't we want to keep track of the data and make sure that when we execute, we get correct data in the right order? Or we just don't care, as long as we do the operation right?"**
  - If you take a look at the state machine, you will see that the counting bits (C1C0) are only counting in state (Q=1), and they will always stay 0 when (Q=0). Since we will never run into cases where (QC1C0=001, 010, 011), we don't really care what we give for the next state transition. Of course, you might argue that the next state could be forced into ($Q^+C1^+C0^+$=000) for "extra security", but the resulting equations of your state machine will be difficult to be simplified into simple logic.
- **"My shift/counter/flip flop register is not shifting/counting/storing data, am I missing something?"**
  - Make sure you are looking at the correct datasheet (or diagram within the same datasheet) for the chip you have. The suffix (e.g. the trailing "A") may mean that the pin-out is completely different from the other variant without the "A". This is presumably because some variants are pin-compatible with now-discontinued chips.