EE241 Digital Design Lab #9 Sequential Logic: Traffic Light Controller
Demo on May 2 2018

Objective:

Use sequential logic methods to implement a traffic light controller.

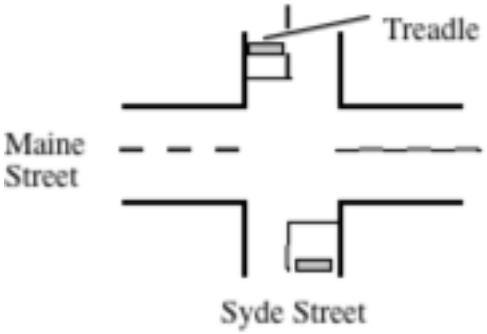
Instructions:

Design and build a traffic light controller. The particular intersection and implementation method you use will be randomly assigned. Write a specification for how the traffic light controller is to work based on the nature of the intersection and any other relevant data. Draw a state diagram, and develop a state table for clocked sequential logic. Design a sequential controller using programmable logic with Quartus Prime, using schematic capture or Verilog for an Altera "Max II" FPGA (supplied). Design a test scenario that will exercise all of the possibilities that can occur at the intersection, and show that your design should work properly. Implement the design, and demonstrate it using the same test scenario. Document it in an informal report. The report should include documentation of the design steps taken, schematic, and observations on the correct (or not) operation in both the simulation and as built. You may gain extra credit by <u>also</u> doing a program for a 16V8 or 22V10 GAL for WinCUPL, or with 74LS74's for discrete logic. (Design, simulate but not build!) If you do that, try to minimize the cost. You may add "extras" for extra credit later, such as an all-way red interval between greens, walk signals, cameras to detect and photograph red light violators, or extra sensors. Start out basic and see that work first though.

Implementation Methods: (Pick "Classical", "One-hot" or "Almost One Hot" for one of #1-#7)

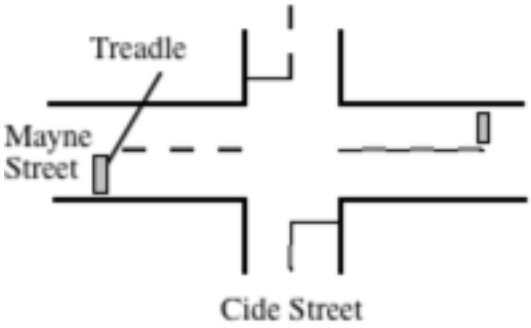
A: Classical state design using D latches.
B: "One Hot" state design using D latches.
C: "Almost One Hot" design using D latches.

Intersection #1:



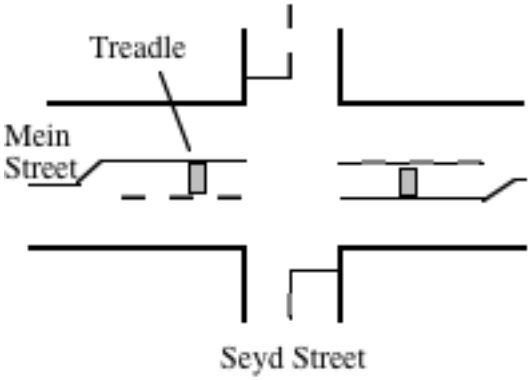
Syde Street is to get a green only when a sync clock (coordinated with other intersections) goes high, and a car is waiting. A Syde street green (and yellow) must be a substantial portion of the clock. The treadle registers a "1" when a vehicle wheel rolls over it, but does not stay "1". This is a very common type of control.

Intersection #2:



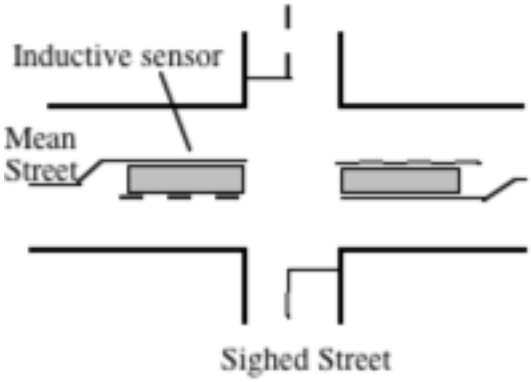
Cide Street is to get a green normally. But when the treadle on Mayne Street registers a "1" when a vehicle wheel rolls over it, the light is to hastily change to a green for Mayne street. It stays green long enough for the car to get throught the intersection, then changes back unless another car has come along. It gives Cide street a minimum length green and Mayne Street a maximum on its green. (There was a light like this in Martinsburg, VA when I was young.)

Intersection #3:



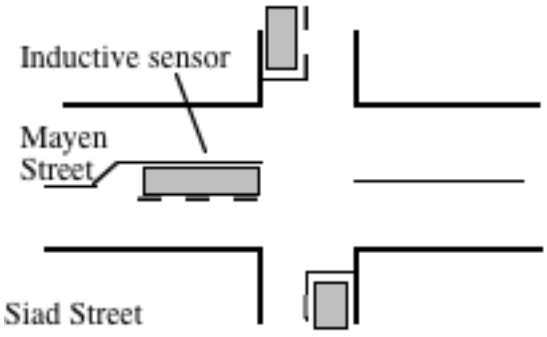
Mein street has a turning lane for each direction. The traffic light normally goes through a fixed cycle alternating between Mein Street and Seyd Street, with no special turning arrow. (Cars can turn on the Green). But after a car hits either treadle, the following cycle gives both turning lanes a green arrow at the beginning of the cycle before Mein Street turns green, and after Seyd Street has gone.

Intersection #4:



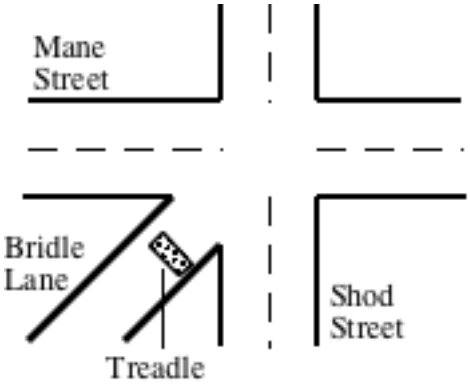
Mean Street has a turning lane for each direction. The traffic light normally goes through a fixed cycle alternating between Mein Street and Sighed Street, with a red for the turning lanes. (Cars cannot turn on the Mean Green). But if a car is detected on either sensor, the following cycle gives that direction a green arrow at the beginning of the cycle before Mein Street from the opposite direction turns green, and after Sighed Street has gone.

Intersection #5:



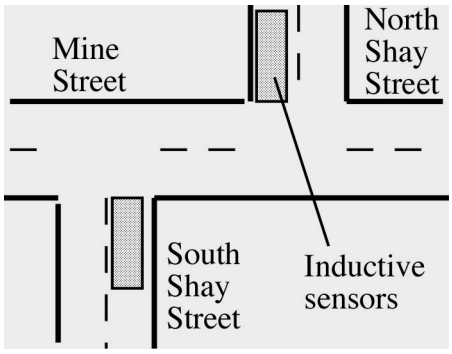
Mayen Street has a turning lane for one direction. The traffic light normally remains Green for Mayen Street. But if a car is detected on either sensor for Siad street, it turns green for that direction,followed by the other direction if there are any cars there. A green arrow is given for the turning lane at the beginning of the cycle before Mayen Street from the opposite direction turns green. If a car needs to turn with no Siad street traffic waiting, it will eventually get a green, with opposite direction traffic blocked.

Intersection #6:



Mane street intersects both Shod street and Bridle lane, which comes in at a 45 degree angle. There is a treadle that is used to indicate when a car is waiting on Bridle lane. The normal cycle includes only Mane and Shod streets. When a car is waiting, Bridle street goes after Shod, but only if the car got there before Shod street goes. (The local mayor's opponent lives on Bridle Lane.) See next page.

Intersection #7:



Mine Street intersects Shay Street, but North Shay Street is not directly opposite South Shay street, which is offset to the West. So there are actually two lights along Mine Street that are coordinated, and each part of Shay Street goes separately. When South Shay street gets a green, the Eastbound East Mine street light needs to be green so the traffic turning right can continue, and likewise for North Shay street. Both North and South Shay Streets have inductive sensors. If either is tripped, both directions get a turn.

Treadles are trickier than inductive sensors. You must remember that a car has rolled over the sensor. It usually won't stop on top of the sensor. In both cases, you do not want to give a waiting car a green if the green will be very short. Always give a full cycle green.

You will need timing methods. It may be possible to count cycles of a really slow clock. You may use a "one shot" timer, but that is more complex, and time and effort is limited. Given a trigger transition (which can be either edge) a “one-shot” gives a pulse of a given width. One device to consider is the 74LS123. It has two one-shots in one package. However, it tends to be sensitive and is easily retriggered if noise is present. Another option is the 555 timer, which can be configured to be a 1-shot. Using 555’s requires more components, but that’s better than the noise issues you get with the 74 series 1-shots, which don’t work all that well on solderless breadboards. Typically, you will, say, turn a light yellow and at the same time, trigger a one-shot. When the one-shot goes back to being a zero, it is time to turn the light red. You could then use the same one shot for the minimum green time for the other direction, if using the same length of time is helpful. The one-shot trigger would be a sequential machine output and the one shot output would be a sequential machine input.

The last few times we did this lab we had some problems with WinCupl. The behavior of the GAL’s were sometimes not consistent with the source program. I was unable to run the problems to ground. So, this year we’re using FPGA’s. Soo other documentation on how to use them. The book has material on Verilog. Look at examples.

Make needed simplifying assumptions. Try to minimize the number of states. If you want to make a simplifying assumption, ask. If you think you might need more than a reasonable number of states, consider simplifying the intersection of how we will handle it; ask. I want to help you keep it simple. You can use an asymmetric clock and use the clock as a Mealy input to get the yellow/green timing, as another way to save states and complexity. I’m not going to get picky about how you make it work. You can make the timing for various directions whatever you’d like, even to the point of using the same interval for all greens and yellows. (You’ll run it faster than a normal traffic light would go, a few seconds for each light, so you can demonstrate it without taking too much time.) The whole point is to take a sequential machine from design to working. In the real world, a microcontroller would typically be used to do this job, since there is no need for high speed processing. (Cars are pretty slow by digital standards. They can carry a lot of bits though.)