Simulation | Lab 10 | UT.6.01x Courseware | edX https://courses.edx.org/courses/UTAustinX/UT...
edX (https://www.edx.org)
UTAustinX: UT.6.01x Embedded Systems - Shape the World
KarenWest (/dashboard) ▼
Courseware (/courses/UTAustinX/UT.6.01x/1T2014/courseware) Course Info (/courses/UTAustinX/UT.6.01x/1T2014/info)
Discussion (/courses/UTAustinX/UT.6.01x/1T2014/discussion/forum) Progress (/courses/UTAustinX/UT.6.01x/1T2014/progress)
Questions (/courses/UTAustinX/UT.6.01x/1T2014/a3da417940af4ec49a9c02b3eae3460b/)
Syllabus (/courses/UTAustinX/UT.6.01x/1T2014/a827a8b3cc204927b6efaa49580170d1/)

Help

PROCEDURE

R9, R10, R25, and R29 are zero-ohm resistors on the LaunchPad. Notice in Figure 10.3 that R9 shorts PD0 to PB6, R10 shorts PD1 to PB7, R25 connects PB0 to the USB device header, and R29 connects PB1 to +5V. If you use PB1 or PB0, make sure R26 and R29 are removed. If you use both PD0 and PB6, remove R9. If you use both PD1 and PB7, remove R10. R25 and R29 are not on the older LM4F120 LaunchPads, just the new TM4C123 LaunchPads. The TM4C123 LaunchPads I bought over summer 2013 did not have R25 and R29 soldered on, so I just had to worry about R9 and R10.

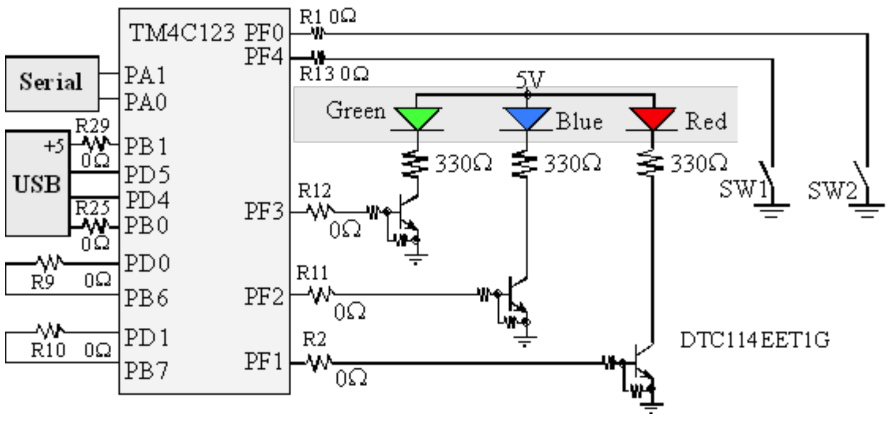

Figure 10.3. Switch and LED interfaces on the Tiva LaunchPad Evaluation Board. The zero ohm resistors can be removed so the corresponding pin can be used for its regular purpose.

We posted a version 1.0.0.6 which is little more reasonable about what it passes and what it fails. As with all graders, it begins by checking initialization registers. During the I/O portion of the grading, we get a notification whenever you write to either of the output ports. The grader checks for valid output pattern sequences. We have defined 9valid output patterns, listed below. For each valid output pattern, there are a only finite number of valid output patterns that could be next.

Pattern 1) **All lights are red.** Once the output is at this pattern, the valid next patterns are {1,2,4,6,7,8,9}
Pattern 2) **West is green**, south is red, don't walk is red**.** Once the output is at this pattern, the valid next patterns are {2,3}
Pattern 3) **West is yellow**, south is red, don't walk is red**.** Once the output is at this pattern, the valid next patterns are {1,3,4,6}
Pattern 4) **South is green**, west is red, don't walk is red**.** Once the output is at this pattern, the valid next patterns are {4,5}
Pattern 5) **South is yellow**, west is red, don't walk is red**.** Once the output is at this pattern, the valid next patterns are

{1,2,5,6}
Pattern 6) **Walk is green**, south is red, west is red**.** Once the output is at this pattern, the valid next patterns are {1,6,7}
Pattern 7) **Don't Walk is off**, south is red, west is red**.** Once the output is at this pattern, the valid next patterns are {1,2,4,6,7,8,9}
Pattern 8) **Don't Walk is off**, west is red, **south is green.** Once the output is at this pattern, the valid next patterns are {4,5}
Pattern 9) **Don't Walk is off**, **west is green**, south is red**.** Once the output is at this pattern, the valid next patterns are {2,3}

There are a couple of consequences of this grading algorithm:
1) You should output to the road lights first and then to the walk lights,
2) You should output to the walk and don't walk lights at the same time
3) For simulation, we do not check for timing, so make the delays short during simulation testing.

Most microcontrollers have a rich set of timer functions. For this lab, you will the PLL and SysTick timer to wait a prescribed amount of time.

The basic approach to this lab will be to first develop and debug your system using the simulator. After the software is debugged, you will interface actual lights and switches to the LaunchPad and run your software on the real microcontroller. As you have experienced, the simulator requires different amount of actual time as compared to simulated time. On the other hand, the correct simulation time is maintained in the SysTick timer, which is decremented every cycle of simulation time. The simulator speed depends on the amount of information it needs to update into the windows. Because we do not want to wait the minutes required for an actual intersection, the cars in this traffic intersection travel much faster than "real" cars. In other words, you are encouraged to adjust the time delays so that the operation of your machine is convenient for you to debug and for the grading engine to observe during demonstration.

Part a) Decide which port pins you will use for the inputs and outputs. Avoid the pins with hardware already connected. Run the starter code in the simulator to see which ports are available for the lights and switches; these choices are listed in Tables 10.1 and 10.2. The "don't walk" and "walk" lights must be PF1 and PF3 respectively, but where to attach the others have some flexibility. In particular, Table 10.1 shows you three possibilities for how you can connect the six LEDs that form the traffic lights.Table 10.2 shows you three possibilities for how you can connect the three positive logic switches that constitute the input sensors. Obviously, you will not connect both inputs and outputs to the same pin.

Red east/west	PA7	PB5	PE5
Yellow east/west	PA6	PB4	PE4
Green east/west	PA5	PB3	PE3
Red north/south	PA4	PB2	PE2
Yellow north/south	PA3	PB1	PE1
Green north/south	PA2	PB0	PE0

Table 10.1. Possible ports to interface the traffic lights (yellow is suggested or default).

Walk sensor	PA4	PB2	PE2
North/south sensor	PA3	PB1	PE1
East/west sensor	PA2	PB0	PE0

Table 10.2. Possible ports to interface the sensors (yellow is suggested or default).

Part b) Design a finite state machine that implements a good traffic light system. Draw a graphical picture of your finite state machine showing the various states, inputs, outputs, wait times and transitions.

Part c) Write and debug the C code that implements the traffic light control system. During the debugging phase with the simulator, use the logic analyzer to visualize the input/output behavior of your system. Another debugging technique is to dump {the state index, the input, the output} each time through the FSM controller to create a log of the operation

During the simulation grading I will automatically set the inputs and check your outputs. **Hint: I recommend reducing the wait times for all states to be less than a second, so the simulation grading runs faster.** Once you get a 100 in simulation, you can increase the wait times for the real board to be more reasonable.

GRADING IN SIMULATION

DR. JONATHAN VALVANO: Let's show you how to get a grade for the Lab 10

in Simulation mode.

We begin in edX and copy this four-digit number.

We go over to Keil.

We make sure we are in Simulation mode, because we're

going to get the grade for Simulation mode.

We got to build it, which is to compile.

We've got to debug.

Now we're debugging in Simulation.

In order to interact with the traffic light,

we're going to need a few of the peripherals.

We're going to need Port F so we can see the walk light,

and we're going to need the traffic signal here

so we can see the walk buttons.

While I grade, it's nice to see the command window,

because I can see the tests that are being performed.

So I open up the command window-- make it big so you can see what's going on.

I'm going to paste a number from edX right here, the four-digit number.

Help