Microprocessor-Based Systems (E155)

Harris **Fall 2004**

Lab 6: Traffic Light Controller Due: Week of Oct 25

Requirement

Design a traffic light sequence simulator using the PIC microcontroller and glue logic on your FPGA board. The traffic lights are to be built with colored LEDs. Light patterns and delay times are to be read from a table, while the output is to be sent to six LEDs by the:

a) Parallel I/O Port D

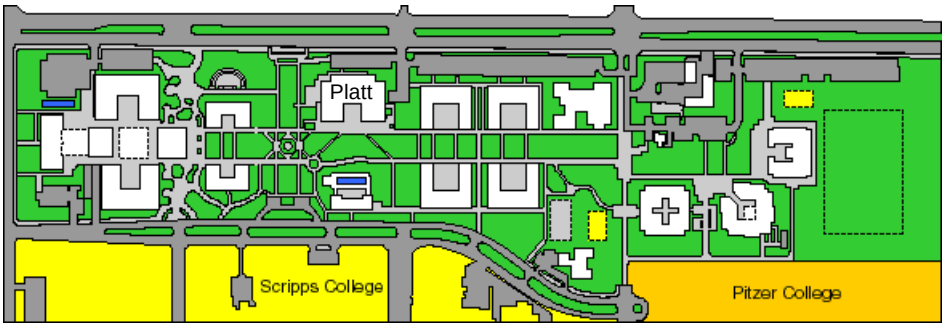
b) Serial Peripheral Interface (SPI)

Discussion

Microcontrollers are often used for logic-timer control. A traffic light controller is a simple example. The light patterns are flashed on for a specified time using the parallel or serial port. The time delays can be generated using either software (delay loops) or hardware (timer facilities). For this particular example, the software method is quite appropriate.

During lunchtime, there has been excessive congestion in front of Platt Dining Hall. Students headed north and south bound in front of Platt have nearly collided with unicyclists headed east or west reading books or juggling swords. No serious injuries have yet occurred, but parents have been calling the President demanding action. He has directed Physical Plant to install a system of traffic lights at the intersection controlling east/west traffic and north/south traffic.[1]

[1]Unfortunately, this system is doomed to fail because nobody in Southern California obeys traffic lights.



Your controller should be built with a sequence descriptor and an interpreter. In this way the control sequence may be changed at any time to better accommodate traffic conditions.

The sequence descriptor is an entry in a table containing the light pattern and duration. In order to have a variable length table, a null entry (pattern of 0 and duration of 0 sec) can be designated as the end of the table, for this will never be part of the sequence.

The interpreter is a program that reads values from the sequence descriptor table. The interpreter sends light patterns for the specified durations. The interpreter should have a delay subroutine to waste exactly 1 second. The delay time read from the table is used to count out the number of times the "1 second delay" subroutine is executed. Your delays should meet a +/- 5% tolerance. When the interpreter reads the null pattern, it should loop back to the beginning of the table.

For the experiment, the following sequence descriptor is suggested:

Pattern		Duration
North-South	East-West	
red	green	15
red	yellow	3
green	red	10
yellow	red	2
null	null	0

You are free to choose exactly how the sequence discriptor table will be implemented in the PIC's memory.

Write your code in assembly language, debug it, and run it on the PIC.

Use good coding practice on your assembly language programs. Comment your code thoroughly. Use EQU statements to define constants and reserve memory using EQU or RES statements.

Control the North-South lights with data coming over Port D and control the East-WEst lights with data coming over the SPI. When the SPI is used, you will need an 8-bit serial-in parallel-out shift register to capture the data. You can build such a shift register on your FPGA. You are free to use either Verilog or schematic entry.

The programming ritual for the SPI is reviewed below. You may find section 15.3 of the PIC datasheet helpful, or you can look at some of the examples in the PIC18F452 Microcontroller book.

1) Configure the data direction register for port C (TRISC/PORTC)

2) Configure the control register (SSPCON1)

3) Configure the status register (SSPSTAT)

4) Write a pattern to the data I/O register (SSPBUF)

5) Repeat step 4 as needed

Hints

- You may use nested delay loops or the built-in timer on the PIC to create your delays. If you use a loop to measure time, refer to the PIC datasheet to see how many cycles each instruction takes. It may be helpful to start by writing a 1-second delay subroutine that you can call multiple times.
- You can debug basic functionality by tracing through your program. However, you'll need to run at full speed to debug timing issues. If your program hangs, it is hard to see where things went wrong. You can gain extra visibility into your program in several ways. You can write different values to a particular memory location at different points in your program so when you stop the program you can see how far it got. You can send values over port A or port E and check the values on the logic analyzer.
- The PIC produces CMOS voltage levels with $V_{ol} = 0.6$, $V_{oh} = 4.3$ that could pick up a fair amount of noise. The FPGA defaults to TTL voltage levels with $V_{il} = 0.8$v, $V_{ih} = 2.0$v. This means there is very little noise margin for any coupled or radiated noise

onto low signals from the PIC to the FPGA. Some people have discovered certain FPGA inputs are too noisy or only work when loaded with a resistor or capacitor that filters the noise. To reduce noise problems, set the FPGA implementation options to use CMOS voltage levels (V_{il} = 1.0v, V_{ih} = 3.5v) rather than TTL levels on the input pins. This may be set by right-clicking on the Generate Programming File flow, selecting Properties, and examining the Config Options tab.

- When you map the SDO and SCK pins as inputs on your FPGA, you need to have a dedicated clock I/O pad as the SCK input. Check the Spartan data sheet; any primary global clock (PGCK) input should work.

What to Turn In

Demonstrate your traffic light controller. Be prepared to change the sequence descriptor and to answer fault-tolerance questions about your software, breadboard, or FPGA designs. Know how to bring up the serial waveforms on the logic analyzer. Your lab notebook should include:

- Your design approach
- A listing of your assembly language code
- Verilog or schematics of your FPGA circuits along with simulation waveforms
- A schematic of your breadboarded circuits
- How many hours did you spend on the lab? This will not count toward your grade.