COE 405
Design & Modeling of Digital Systems

Course Project#1 – Term 131
Traffic Light Controller[i]
Due on: Sunday Dec. 8, 2013

In this project, you will implement a traffic light controller that operates main street, side street and walk lamps. The traffic light controller is for an intersection between a Main Street and a Side Street. Both streets have a red, yellow, and green signal light. Pedestrians have the option of pressing a walk button to turn all the traffic lights red and cause a single walk light to illuminate. Lastly, there is a sensor on the Side Street which tells the controller if there are cars still on the Side Street. This is summarized in Figure 1.

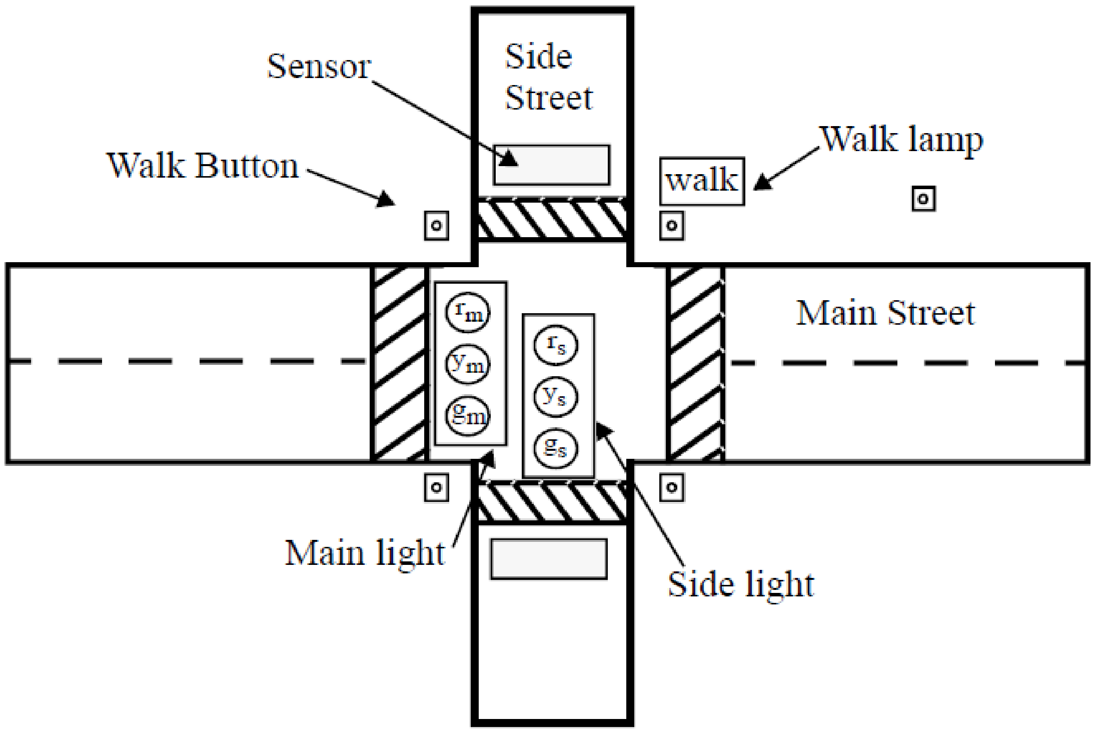

Figure 1: Diagram for intersection with corresponding lights.

You may assume that the 4 walk buttons placed at each street corner are hooked into the traffic light controller using a wired-OR. For this reason, you may assume that the controller only needs a single input called *Walk-Request*.

The side street sensor is placed near the intersection to tell the controller when there are cars passing over the sensor. You may assume the sensor remains constantly high if several cars pass over the sensor, rather than quick pulses, provided the cars are close enough together. You do not need to implement this specific functionality. This input is named *Sensor*.

The traffic lights are timed on three parameters (in seconds), the base interval (t_{BASE}), the extended interval (t_{EXT}), and the yellow light interval (t_{YEL}). The default values listed in Table 1 are to be loaded into registers in the FPGA on reset, and may be reprogrammed on demand using switches and buttons on your kit with the *Time_Parameter_Selector*, *Time_Value*, and *Reprogram* signals. *Time_Parameter_Selector* uses the Parameter Number code to select the interval during programming. *Time_Value* is a 4-bit value representing the value to be programmed; therefore, it has a duration of seconds between 0 and 15. The *Reprogram* button tells the system to set the currently selected interval to *Time_Value*.

Table 1: Default Timing Parameters.

Interval Name	Symbol	Parameter Number	Default Time (sec)	Time Value
Base Interval	t_{BASE}	00	6	0110
Extended Interval	t_{EXT}	01	3	0011
Yellow Interval	t_{YEL}	10	2	0010

The operating sequence of this intersection begins with the Main Street having a green light for 2 lengths of t_{BASE} seconds. Next, the Main lights turn to yellow for t_{YEL}, and switches to the Side Street green light. The Side street is green for t_{BASE}, and its yellow is held for t_{YEL}. Whenever a stoplight is green or yellow, the other street's stoplight is red. Under normal circumstances, this cycle repeats continuously.

There are two ways the controller can deviate from the typical loop. First, a walk button allows pedestrians to submit a walk request. The internal Walk Register should be set on a button press and the controller should service the request after the Main street yellow light by turning all lights to red, and the walk light to on. After a walk of t_{EXT} seconds, the traffic lights should return to its usual routine by turning the Side Street green. The walk button should be ignored during the walk service. The second deviation is the traffic sensor. If the traffic sensor is high at the end of the first t_{BASE} length of the Main street green, the light should remain green only for an additional t_{EXT} seconds, rather than the full t_{BASE}. Additionally, if the traffic sensor is high during the end of the Side Street green, it should remain green for an additional t_{EXT} seconds.

When the machine is reset, display on the LCD screen "**Traffic Light Controller**". When the Walk signal is asserted display "**You can walk now**".

Block Descriptions/Implementation

The block diagram for the Traffic Light Controller implementation is shown in Figure 2.

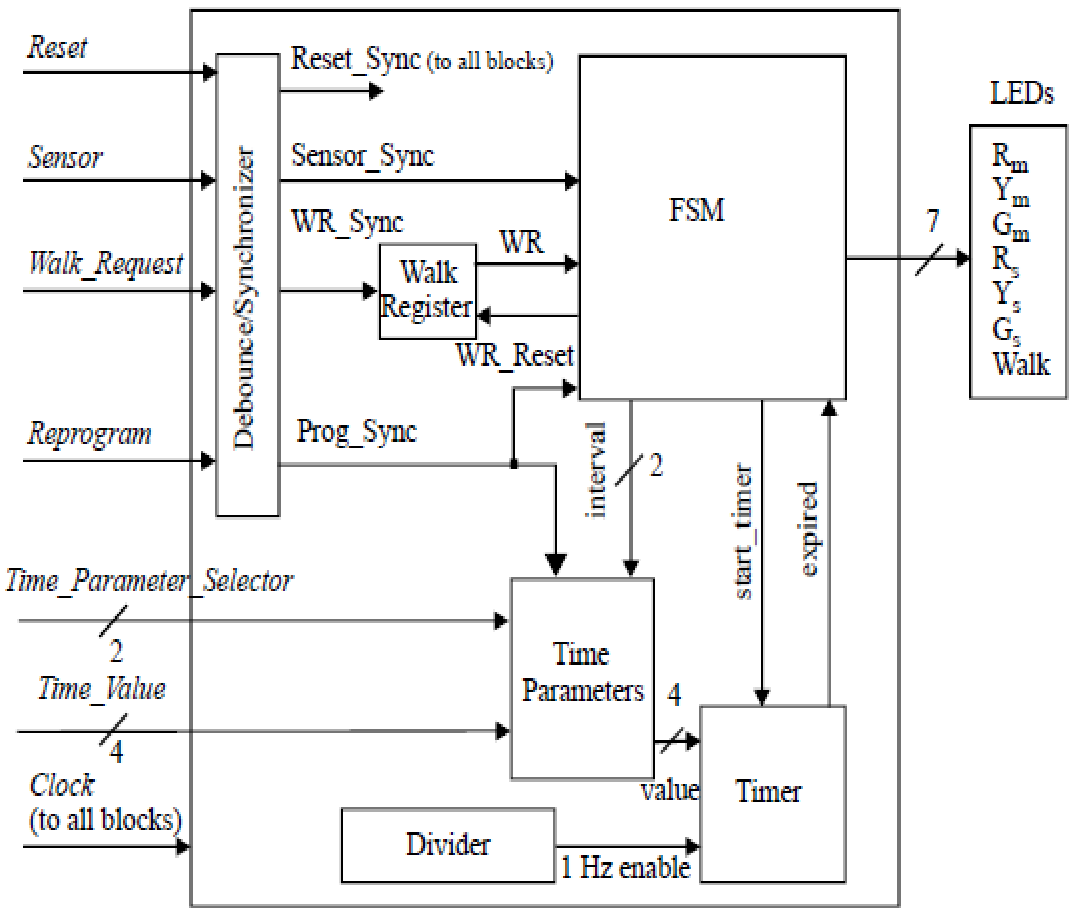

Figure 2: Diagram for intersection with corresponding lights.

Debouncer/ Synchronizer

Your clocked state machine is controlled by several asynchronous inputs that might be changed by the user at any time, potentially creating a problem with metastability in the state registers if one of the inputs changes too near a rising clock edge. In general asynchronous inputs need to be synchronized to the internal clock before they can be used by the internal logic. You should feed all asynchronous inputs through an instance of the synchronize module and use the output signal in the design of your system.

A second problem arises from the mechanical “bounce” inherent in switches and buttons: as a metal contact opens and closes it may bounce a couple of times, creating a sequence of on/off transitions in rapid succession. So you need to use debouncing circuitry to filter out these unwanted transitions. debounce.v is a Verilog implementation of a digital retriggerable one-shot that requires that an input transition be stable for 0.01sec before reporting a transition on its output. debounce.v wil be given to you. This module happens to produce a synchronous output, so a separate synchronizer is not required. You should use an instance of the debounce module to debounce any switch inputs you use in your design.

Walk Register

The Walk Register allows pedestrians to set a walk request at any time except for the walk service duration. There is also a signal controlled by the finite state machine that will be able to reset the Register during the actual walk service.

Time Parameters

The time parameters module stores the three different time parameter values, namely t_{BASE}, t_{EXT}, and t_{YEL} on the FPGA. The module acts like a (small) memory from the FSM and Timer blocks, where the FSM addresses the three parameters and the timer reads the data. From the user's perspective, the three time parameter values can be modified.

On a reset, the three parameters should be respectively set to 6, 3, and 2 seconds. However, at any time, the user may modify any of the values by manipulating *Time_Parameter_Selector*, *Time_Value*, and *Reprogram*. Each of these values are 4 bits, and is selected using a 2 bit address. Whenever a parameter is reprogrammed, the FSM should be reset to its starting state.

Divider

The divider is necessary for the timer to properly time the number of seconds for any particular traffic light state. Using only the clock (50Mhz) as input, it generates a 1 Hz enable, which is sent to the timer. The signal generated is a pulse that is high for one clock cycle every 1sec.

Timer

The timer is responsible for taking the start_timer, 1Hz enable, and Time Parameters *value* to properly time the traffic light controller. When done counting a particular state, the expired signal will go high to signal to the FSM that it should change states.

Finite State Machine

The finite state machine controls the ordering for the traffic light. As previously described, it changes states based on the Walk Register and sensor signals, and with the expired signal. Describe your FSM using an ASMD chart.

This project is to be conducted by a team of two students. All team members are responsible about all tasks of the project and should collaborate to achieve a successful project. You are required to understand all the work done in the project, the part you do and the part done by the other team members.

Clearly state your assumptions and have your design well documented. Write a professional report indicating all design stages, modeling and testing of each component and the final design. Include both a hard copy and a soft copy of your report and all Verilog files. All team members are required to make a demo of their project in the project due date. The grading policy for the project is shown below:

Grading Criteria	**Mark**
Demonstration of correct functionality of components by simulation	40
Demonstration of correct functionality of project on FPGA	40
Project Documentation and Report Organization	20
Total	100

---

[i] Courtesy of MIT