CS 152A / EE M116L

Introductory Digital Lab Design

LAB #5

Traffic Light Controller

May 29, 2014

Grade:

Tianheng Tu

Name1: Nathan Tung

SID1: 004-059-195

Name2: Mark Iskandar

SID2: 704-050-889

INTRODUCTION

The purpose of Lab 5 is to design and construct a traffic light controller using VHDL, Xilinx schematics, the FPGA board, and the ProtoBoard. Our traffic light model involves red, yellow, and green lights for two intersecting streets (a main street and a side street). In the default case, the traffic lights change in a predictable sequence. However, pedestrians can hit a pushbutton (WALK) that can turn both lights red, and the presence of cars on the side street will activate a sensor (SENSOR) that shortens their wait time. At any time, the entire cycle can be reset to the start state at the push of a (RESET) button.

Most of the logic in our traffic light controller is contained within our FSM, or Finite State Machine, which accounts for user input and provides updates on which state we are currently at in order to display all six traffic lights on the FPGA's LEDs. A clock module nearly identical to the one in Lab 4 is used to divide frequencies for debouncing and timing purposes. Finally, a timer and parameter module are used to count the number of seconds designated for each state within the FSM.

DESIGN COMPONENTS

Overall

Like in Lab 4, a divider clock module breaks down the 100 MHz master clock signal into smaller frequencies and uses these for debouncing and aiding the timer in counting seconds. According to the lab specifications, three possible time parameters 00, 01, and 10 translate into a time value of 6, 3, and 2 seconds, respectively. One module maps parameters to values by using simple logic then passes it into the timer. The timer uses the 1 Hz clock to count up to the specified time value before sending out a timeout signal to the FSM. The FSM has at least one state transition for every single possible state. When the timer's timeout signal is high, we change states and update the six traffic light LED outputs, which are wired straight into the FPGA board. It also sends the next state's time-encoded parameter to the timer to continue the process. The additional pushbuttons (RESET, WALK) and switch (SENSOR) are accounted for in the FSM in order to transition to alternate states. Notice that, as usual, all buttons and switches used are debounced to mitigate preventable errors. A diagram of the overall design is shown on the last page of the report.

FSM

The FSM is written in VHDL and converted into a schematic symbol for use in our top-level implementation. Our FSM has eight states (Green, Red, Tbase), (Green, Red, Tbase), (Yellow, Red, Tyel), (Red, Green, Tbase), (Red, Yellow, Tyel), (Green, Red, Text), (Red, Red, Text), (Red, Green, Text) ranging from state 000 to state 111, respectively. When any variable on the sensitivity list is altered – either a RESET or a timeout signal – the FSM triggers and determines the next state. Furthermore, during these transitions, the FSM accounts for the WALK and SENSOR signals and will alter its transition course based on these variables. In terms of VHDL code, the FSM comprises a series of embedded if-statements that trigger on the rising clock edge of a changed sensitivity variable and outputs all LED values. In addition, after the walking state ends, a variable is sent back to the Walk Register to reset WALK to 0.

Divider

The clock module acts as a frequency divider and is necessary in timing the state transitions. Given the 100 MHz master clock signal from the FPGA, we need to produce clock signals with frequency 1 Hz for counting seconds and 2 Hz, 500 Hz, and 500 MHz for enabling the debouncers. To achieve our "divider," or clock module, we build a modulo-2 counter and a modulo-5 counter. The modulo-2 counter uses a single T flip-flop while the modulo-5 counter uses a modulo-5 BCD counter and two T flip-flops. We can string these together in series with a modulo-10 counter to divide 100 MHz into 1 Hz, 2 Hz, 500 Hz, and 500 MHz.

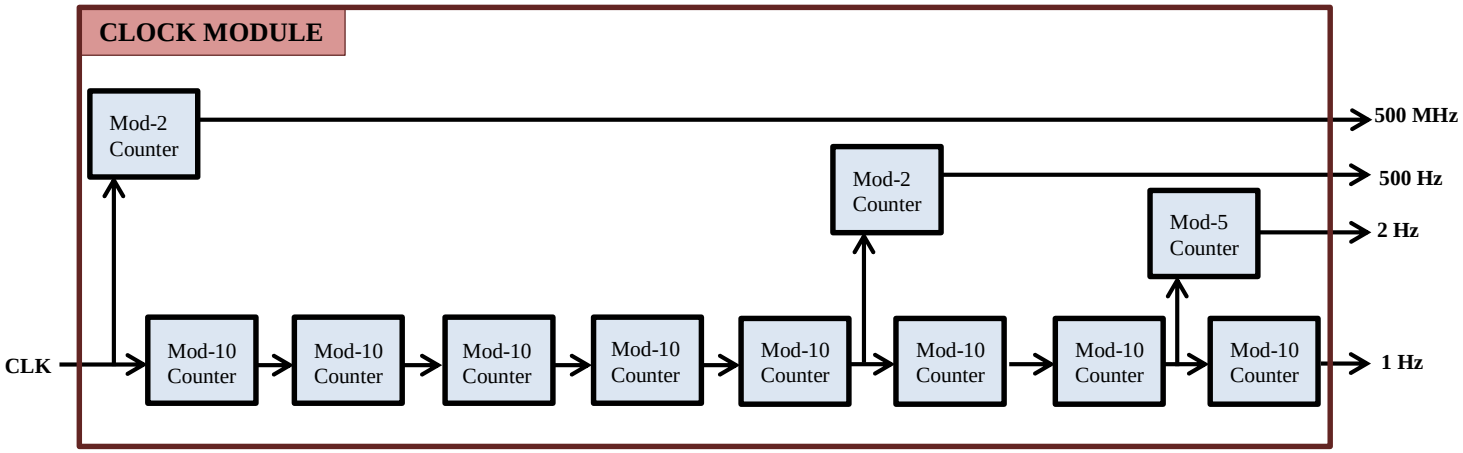

Figure 1: "Divider" clock module for dividing frequency of an input clock signal

ParameterToValue

The ParameterToValue utilizes simple logic with NOT and NOR gates in order to translate an encoded 2-bit parameter into a 4-bit time value (that is, a time value in seconds written in binary). Tbase has the parameter 00 but translates to 0110 for 6 seconds, whereas Text and Tyel have 01 and 10 parameters that translate to 0011 and 0010, respectively.

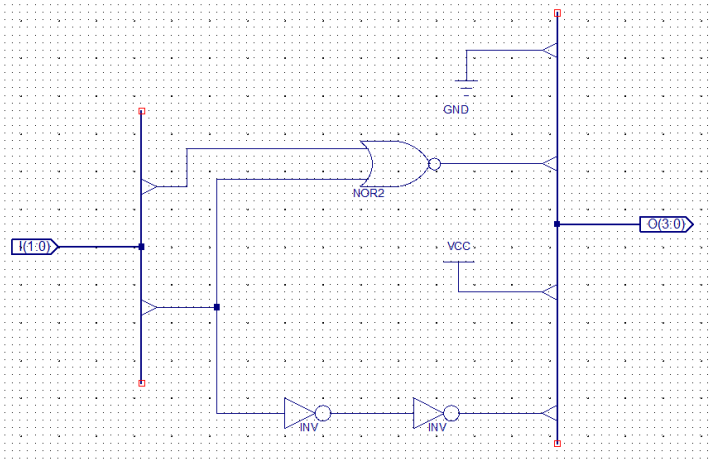

Figure 2: ParameterToValue converts a 2-bit encoding into a 4-bit binary time value

Timer

The timer accepts the 4-bit time value output from ParameterToValue as well as the 1 Hz clock. The clock goes into a 4-bit BCD counter which counts up to the given time value, then resets.

This logic is done by XNORing each of the corresponding bits from the time value and the counter output before ANDing all of them. Therefore, timer has a high output only when the counter's output bits equal the 4-bit time value exactly. The counter then resets to count the next state's time interval, and so on. (For testing purposes, we use a T flip-flop to toggle an FPGA LED on every 1 Hz clock cycle.)

Walk/Walk Register/Speaker

The Walk Register's purpose is to hold the WALK value when the debounced button is pressed until the moment in the FSM where it actually affects the next state. This is done via a D flip-flop, whose input is the debounced WALK button. When the Walk Register's WalkSignal output is finally used as a conditional variable in the FSM and both traffic lights turn red, the FSM sends back a WalkReset value which clears the D flip-flop. At this time, we also output a 1 to an FPGA pin in order to wire up the ProtoBoard speakers to sound as long as both lights are red. We used NAND and NOT gates to AND together the WalkSound pin with the frequency generator before passing the result to the speaker itself. The module is then ready to accept the next pedestrian's walk request. (For testing purposes, when the WalkSignal is high, we also light up an FPGA LED.)

Reset/Sensor

The RESET button is very simple; after passing through the debouncer, it is inputted into the FSM as a sensitivity variable. Whenever RESET is high after the button is pushed, the FSM immediately switches into the starting state 000 and begins running from there. The SENSOR switch is likewise debounced. However, unlike the WALK button, a switch holds its own value until manually flipped on or off, so its value is sent directly to the FSM. The FSM is coded such that the presence of cars on the side street makes the side traffic light turn green faster.

Debouncers

Both buttons and switches need to be debounced in order to prevent unpredictable behavior due to unstable contact in the FPGA board. The buttons are debounced using the pre-designed DebounceSync, while the switches are debounced using a gated SR latch. Our debouncers were recycled from Lab 4.

CONCLUSION

In this lab report, we explained the overall top-level schematic as well as the module-level implementation of the traffic light controller, including the VHDL code found in the FSM. Throughout this lab, we were challenged by unexplained bugs or difficulties in the Xilinx software. For example, our timer could count any specified time value except 0011, or 3 seconds. This could potentially be fixed by adding an extra D flip-flop to hold that value for an extra second before sending a timeout signal to the FSM. Similarly, we learned that the if-statements in the VHDL code had to be written very carefully, as they don't always execute sequentially. Even so, it was beneficial to gain insight in learning how real-life traffic lights might function.

Lab Contribution:

Nathan Tung: 50%

Mark Iskandar: 50%

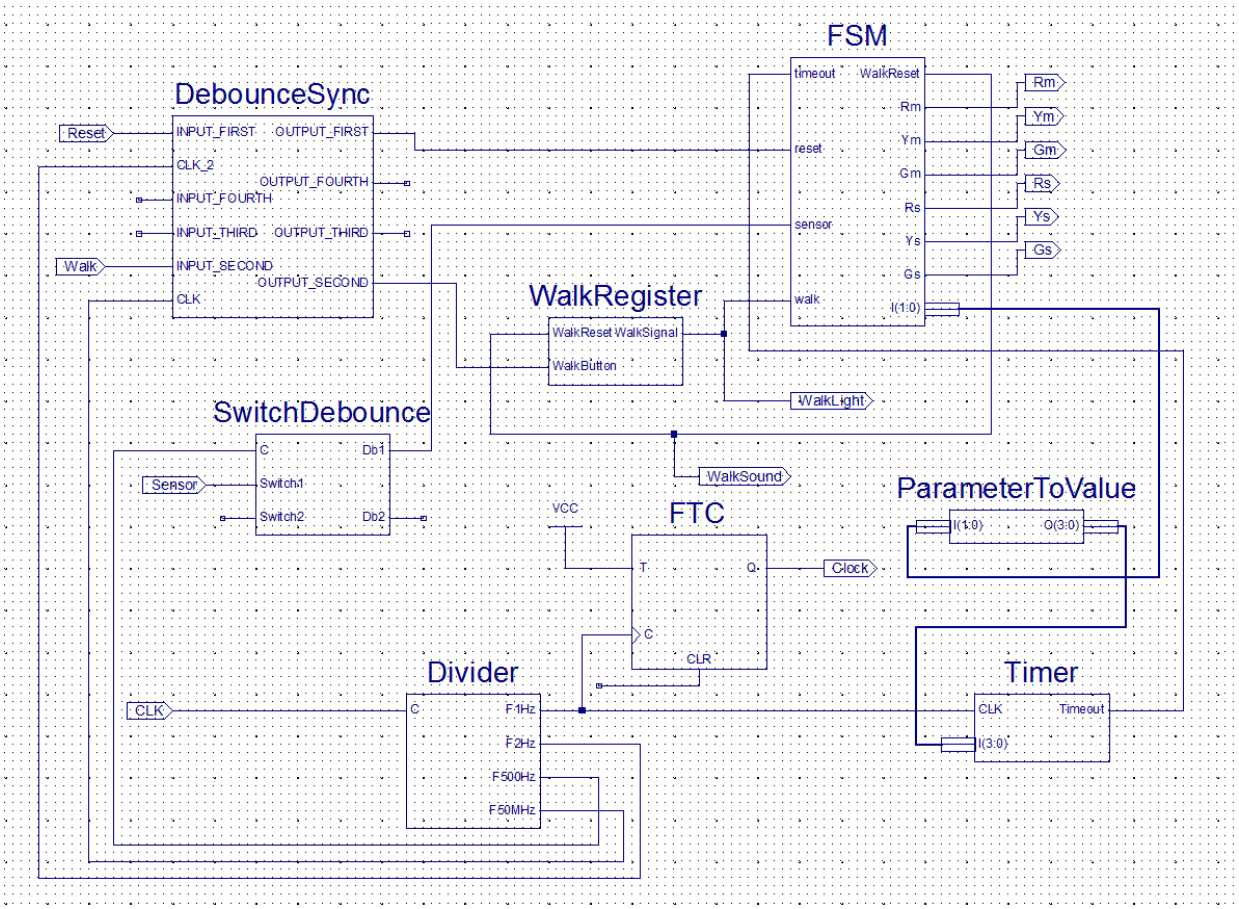

Figure 3: High-level schematic of the traffic light controller