Motion Sensing Stopwatch

Stopwatch with Infrared Stop Function

Daniel Koch, Joel Kirschke, Kevin Kiliman, Victor Wszedybyl
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
dkoch@oakland.edu, jakirsch@oakland.edu, kpkiliman@oakland.edu, vcwszedybyl@oakland.edu

***Abstract*—The purpose of this project is to create a motion sensing digital stopwatch and display it on a Nexys 4 FPGA board. This was created by programming the logic in VHDL on the Nexys 4 and communicating with an IR sensor on an Arduino. This project functions as a normal stopwatch and is able to start and stop when it detects movement from the attached IR sensor. The only recommendation would be to make the timer more accurate by counting nanoseconds as well.**

I. Introduction

A stopwatch provides an accurate way to keep track of time and can be used in many different scenarios. Having one that can detect motion allows the user to track times without the need of another person to operate the timer. To achieve the design of a stopwatch, multiple components needed to be incorporated together. The internal design required three 0-9 counters, one 0-6 counter, a 1 ms counter, a 10 ms counter, one 4-to-1 multiplexer (MUX), one 7-segment decoder, a 2-to-4 decoder, and two finite state machines. There is also an algorithm in the Arduino that is used to monitor the analog signal from the IR sensor and send a logic high signal to the Nexys 4 when motion is detected. Our design uses four seven-segment displays that count up to 59.99 seconds, and is accurate to the hundredth of a second.

When the IR sensor detects motion, the current time will stop and be displayed on the seven-segment display until the user starts it again or the reset button is pressed. Whenever nothing is detected from the sensor, the real time is displayed.

This project can be useful for any application that requires counting time and more specifically in situations where only one person is available. Running, swimming, or biking laps are all good examples that could utilize this project.

II. Methodology

There are three buttons on the Nexys 4 that are for starting the stopwatch, resetting it, and stopping it. The infrared sensor is set to detect if something is up to 80 cm in front of it, which will communicate to the Arduino and the Arduino will then send a digital signal to the Nexys board. All functional coding is done in VHDL on the Nexys 4. The Arduino will just send the Nexys 4 a logic high or low signal for the sensor. Normal operation of our project would be starting the stopwatch and having an object or person trigger the sensor, then displaying that recorded time. The Nexys 4 board will start in a reset = '0' state which will prevent the stopwatch from counting and will be locked at 00:00 until the user pushes the start button or triggers the IR sensor. After the user starts the stopwatch they can either push the stop button, or pass in front of the IR sensor to stop the timer. Once the time is stopped, the display shows this time until the user presses the reset button, which will return the board to the reset state and the display will show 00:00 again.

A. Finite State Machine (FSM)

A finite state machine is the first component used in the stopwatch circuit. The FSM is used to control when the stopwatch starts and stops, which is represented in Figure 1.

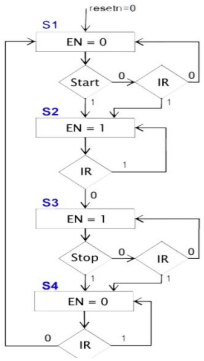

Figure 1 - Initial Finite State Machine Logic [1]

The FSM has inputs from the clock pulse, reset button, IR sensor, and also a start and stop signal. The signal from the IR sensor is read as analog value to the Arduino. The algorithm in the Arduino then determines if the IR sensor detected motion and sends a logic high or low voltage which is then read as a logic high or logic low signal to the FPGA. The signal sent from the Arduino is 5 V, so this voltage must be brought down to 3 V in order to comply with the the 3.3 V logic levels used by the Nexys 4 and, more importantly, to avoid damage to the microprocessor [2]. The voltage divider shown in Figure 2 was connected between the Arduino and the Nexys 4 in order to accomplish this.

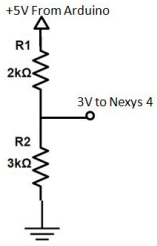

Figure 2 - Voltage Divider Circuit

Two buttons are also used in our design to send a start and stop signal to the FSM. This FSM will decide whether or not the stopwatch will be enabled and send out an output called CE which will enable or disable the first counter which can be seen in Figure 3.

B. Counters

Five counters were used to make this stopwatch work, as shown in Figure 3. The board has a native clock signal with a period of 10 ns, but the stopwatch counts in increments of 10 ms. This was done by using a counter that used a clock divider and took the input of 10 ns and changed the increment to 10 ms. This output signal will go to two places. The first will be the hundredths place for the seconds. This counter, along with the next three counters, has three inputs and two outputs. An enable, reset, and clock are the inputs. The outputs are one 4-bit signal and one 1-bit signal. The enable tells the counter when to increment by one, the 4-bit output produces a signal that reflects the current count, and the 1-bit output acts as a signal that tells the next counter to increment by one. When the count is at 9 and it resets to 0, a signal is also produced that goes to the next counter's enable at the next clock signal. This type of cascading counting continues down to the last counter, where the process will begin again. The last counter displays the second's tens place so the count goes up to 5 before sending out a signal to inform the next counter to increment.

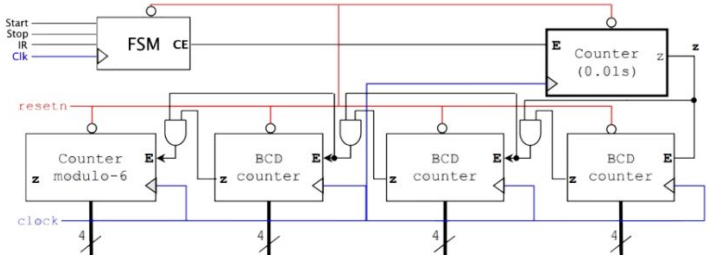

Figure 3 - Counter Diagram [1]

C. 4-to-1 Multiplexer

The data from each BCD counter will be sent to the 4-to-1 MUX. The output of this MUX will be sent as an input to the 7-segment decoder, but first there is a decision of which input to select. This will come from a FSM that is essentially a counter that counts up to 3 and then resets to 0, which is shown in Figure 4. The enable of the FSM is connected to the "z" signal of the 1 ms counter. This allows one number on the 7-segment display to be shown for 1 ms before the next number is displayed for 1 ms.

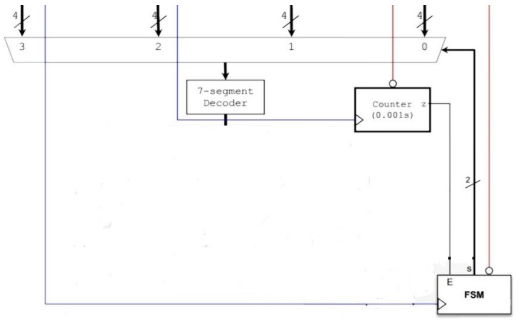

Figure 4 - 4-to-1 Multiplexer Diagram [1]

D. 7-Segment Decoder

The signals from the MUX are passed into the 7-segment decoder, shown in Figure 5. This is a simple component that translates the binary count from the counters into the form needed for the Nexys board to properly display the numbers on the 7-segment displays located on the board. A select statement is used in this component to give the proper output, and the output is then sent directly to the board to display the numbers. The state diagram that enables each 7-segment display is shown in Figure 6 and the complete circuit of the stopwatch is shown in Figure 7.

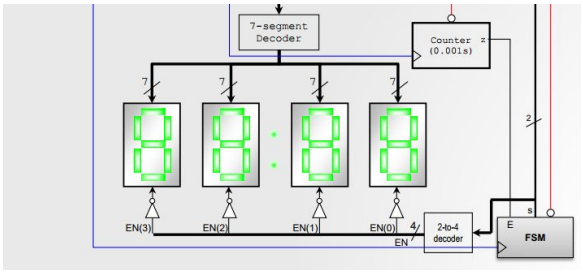

Figure 5 - 7-Segment Decoder Diagram [1]

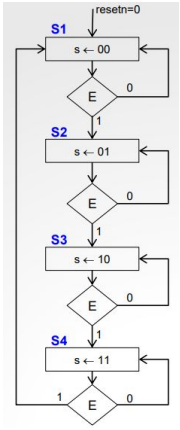

Figure 6 - State Diagram for Enabling 7-Segment Displays [1]

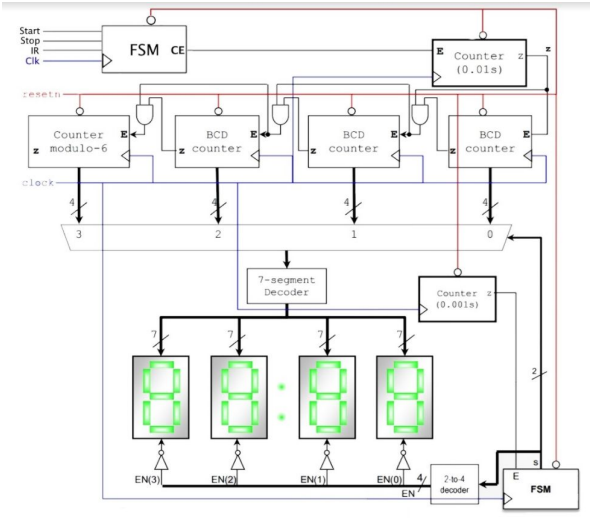

Figure 7 - Complete Circuit Diagram for Stopwatch [1]

III. Experimental Setup

The setup for the project uses the Nexys 4 DDR board used in class, along with an Arduino Uno microcontroller in order to interface with the IR sensor. In order to write and verify the VHDL code used in the project, we used the software Vivado 2016.2. Using Vivado, we were also able to simulate the functionality of the code using a testbench which simply provides start and stop signals that would signal the stopwatch to act accordingly. The expected results were to be able to start and stop the

counters, and to have each of the individual signals for the 7 segment displays to increment correctly. The simulation is shown in Figure 8. This ended up working correctly in the simulation, and we proceeded to load the program into the Nexys 4 where it worked correctly as well.

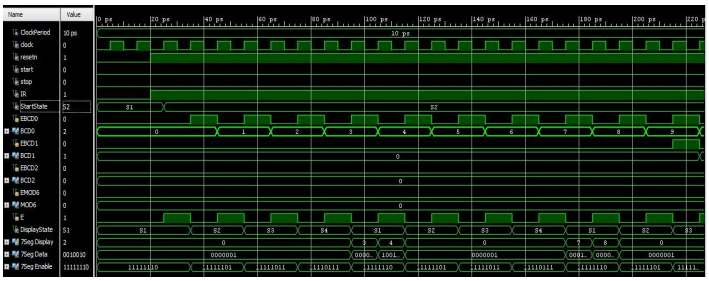

Figure 8 - Stopwatch Vivado Behavioral Simulation

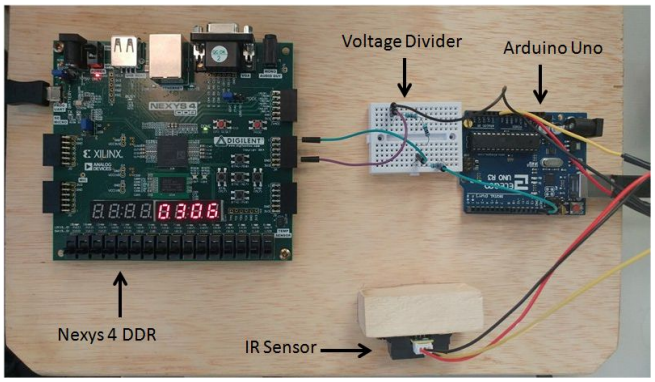

Figure 9 - Stopwatch Project Setup

IV. Results

The results found in our project aligned with our expectations. The time counts accurately down to the hundredth of a second. The IR sensor works successfully as it should whenever it detects an object in front of it. The buttons we used also worked effectively at starting and stopping the stopwatch. The reset button on the FPGA also functions as a reset for the stopwatch and resets all the active 7-segment displays to 0.

Conclusion

This project gave the group a better understanding of coding FPGAs with VHDL code as well as implementing an IR sensor and having it communicate with an Arduino and FPGA board. Creating this stopwatch project required the knowledge of Finite State Machines, Counters, Multiplexers, 7-Segment displays, and Arduino code. We learned how to implement all of these components together in order to create a functioning stopwatch. There are no known issues with the project but one thing we could have done is made it count up to 59 minutes instead of stopping after 59 seconds or even count microseconds.

References

[1] Llamocca, Daniel. "DIGITAL SYSTEM DESIGN VHDL Coding for FPGAs Unit 7." *RECRLab*, Electrical and Computer Engineering Department, Oakland University, www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html.

[2] "Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics." *Xilinx.com*, Xilinx, Inc., 13 Apr. 2017, www.xilinx.com/support/documentation/data_sheets/ds181_Artix_7_Data_Sheet.pdf