Lab 4: Design of a MicroBlaze MCS based System

ECE 3829

Nicholas Lanotte & Matthew DiPlacido

October 11th, 2017

Table Of Contents

Introduction

Embedded processing is an integral part of many advanced FPGA implementations. Bridging the world between hardware and software is very important for computer engineers and should not be overlooked. The greatest feature of an FPGA is that it is programmable, meaning it can be altered using software to define different sets logic gates to complete a task/operation. Microcontrollers are similar to FPGA's in that they too are programmable however a Microcontroller completes a set of instructions unlike an FPGA which is a logic circuit. This lab investigation will go into detail about implementing an embedded processor (the Xilinx MicroBlaze) on an Artix-7 FPGA and combining hardware logic and C code, to demonstrate the operation of an embedded processor working in unison with a logic circuit.

Problem

This first part of this experiment was to create a simple logic circuit that drives the seven segment display. The leftmost digit of the display was to be driven by a 2Hz counter that counts from 0-F and then starts back at 0. Once this was completed a new IP defining the Microblaze on the FPGA could be implemented.

Once the Microblaze was created the Xilinx Software Development Kit (XSDK) was used to program the microcontroller. The embedded processor has two input ports (the 16 switches and the 5 push buttons) and two outputs (the seven segment displays lower 3 digits and the 16 LED's). When one of the buttons is pressed the serial port should display the name of this lab and the class number, when another button is pressed the serial port should display the names of the students completing this lab, when a third button is pressed the serial port should prompt the user to enter a three digit number which will then be entered on the seven segment display. Lastly when a fourth button is pressed the 16 LED's should fill up each time the button is pressed, meaning that each press turns on 1 LED which remains on until all of the LED's are turned on.

After these design features were implemented additional features could be added to improve upon the design.

Overview of Design Approach

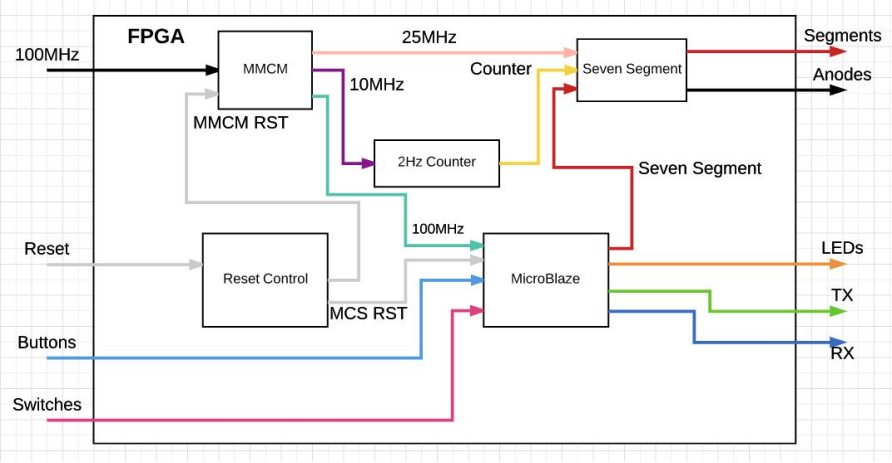

Figure 1: Design Block Diagram

To implement the design described above, many modules were needed to make up the top module. These modules are the MicroBlaze microcontroller, the MMCM, a 2Hz counter, the seven segment display module, and, as an extra feature, a delayed reset module was added so allow the MMCM and Microblaze to be reset with a single button press. The MicroBlaze microcontroller and MMCM are cores generated by Vivado. The MMCM had 3 output clocks, one at 100MHz, one at 25MHz and one at 10 MHz. The MicroBlaze had 2 input channels, one for the 4 buttons and one for the 16 switches. It also has 2 output channels, one for the seven segment display and one for the 16 LEDs. The 2Hz counter is a simple module that takes the 10MHz clock signal and reduces it to 2Hz internally. This signal is used to increment a 4 bit counter from 0-F. This counter is directly connected to the rightmost seven segment display where the value of that counter is displayed. The delayed reset module is also a simple counter that sends a single pulse to the reset port of the MicroBlaze after 100 clock cycles from the Crystal Oscillator have passed. This was implemented because the MicroBlaze can only reset itself when receiving an input clock signal, however when the reset button is pressed the MMCM, which provides the clock signal to the MicroBlaze, is stopped until the module resets. This prevents the MicroBlaze from detecting the reset. By delaying the reset signal by 100 clock cycles, this gives the MMCM plenty of time to reset first, so the MicroBlaze can also be reset afterward. The seven segment display module was also used and has remained unchanged from Lab 2.

The largest part of the lab was created the C code for the MicroBlaze controller. This code utilises a switch statement to control the various functions asked in the lab description. The 4 buttons on the FPGA are treated as a single data bus and used to determine which case to switch to in the switch statement. If no buttons are pressed, this value is 0, which triggers a message saying “Please Press a Button” to be printed to the serial monitor. If the rightmost button is pressed, the button value is equal to 2 and the serial monitor will a display a message saying “ Created by Nicholas Lanotte and Matthew Diplacido”. If the leftmost button is pressed,

the button value will equal 4 and the serial monitor will display “ECE 3829 Lab4: Design of a MicroBlaze MCS based System”. If the lower button is pressed then the button value is equal to 8 and one of the LEDs will light up on the FPGA. With each additional button press, another LED to the right of the previous LED will turn on. All LEDs that are turned on, stay on until the MicroBlaze is reset. Lastly if the upper button is pressed, the button value is equal to 1 and the serial monitor will prompt the user to enter a 3 digit number. After a 3 digit number is entered, it is displayed on the 3 rightmost seven segment displays.

Additional logic was added to ensure that a 3 digit number was implemented over serial. Through testing it was discovered that when enter is pressed on the connected computer and a serial message is sent to the FPGA, the message also ends with a byte that reads 13 in binary. Using this, the serial input logic will keep retrieving data from the serial buffer until this last byte is received. As it receives data, each byte of data is analyzed to ensure it is a valid number character and the total number of characters received is also counted. Once the message is completely received, the code checks these counts to ensure no invalid characters were entered and that there were exactly 3. If so, then that 3 digit number is displayed to the seven segment display. If the serial input is anything but a 3 digit number, the serial monitor will display appropriate error messages describing what was done wrong. It will tell the user if they have not entered enough characters and the number of characters entered. It will also tell the user if too many characters were entered and display the exact number of characters entered. If any invalid characters are entered it will also display how many were entered. It does not update the seven segment display if any of these errors are displayed. These error messages are shown in Figure 1 below.

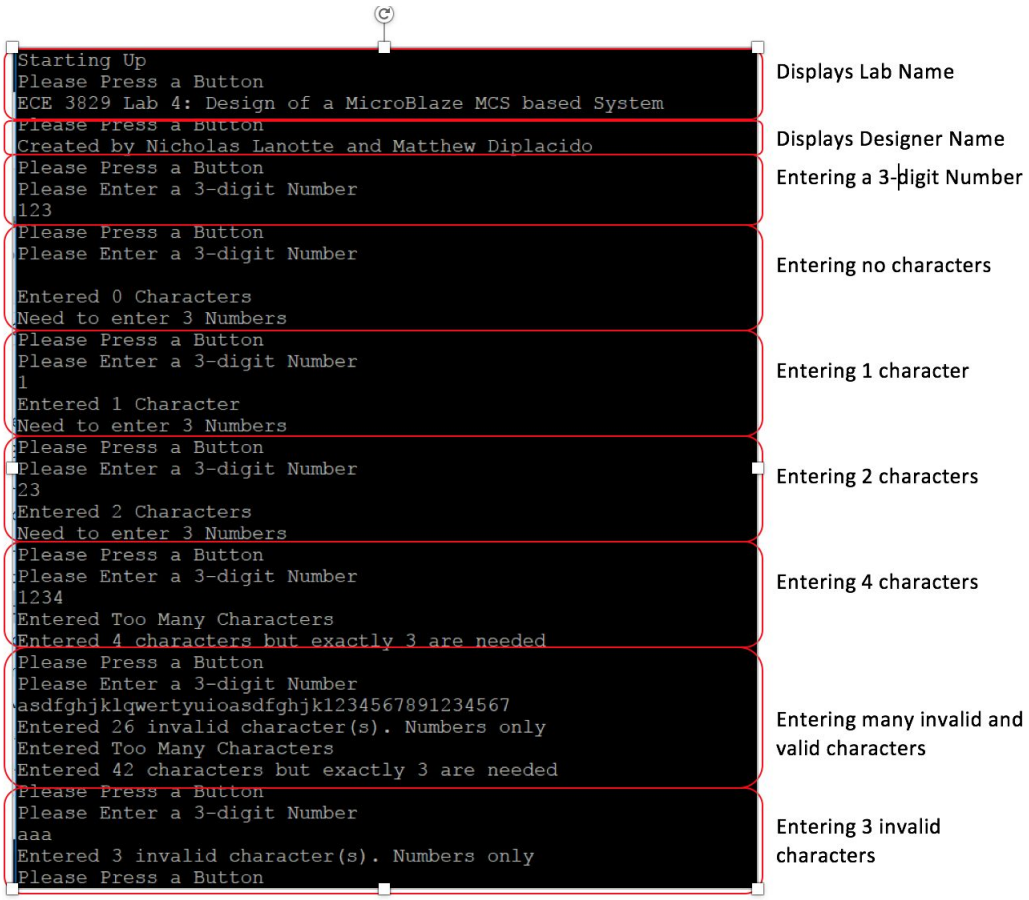

Figure 2: Serial Output Monitor

In Figure 1, all of the serial outputs described are shown. It can be seen that the serial monitor prompts the user to press a button after each action from the previous button is completed. The serial monitor displays the names of the designers and the lab name when the appropriate button is pressed. This also shows the error messages displayed when 0,1, 2, and 4 characters are entered. It also shows that the FPGA is capable of receiving a large string of numbers and characters, counting the total number of characters and the amount of invalid characters. Lastly, this shows that when 3 characters are entered but they are invalid, it displays the invalid character message and thus does will not update the screen.

Lastly, another feature was added to have the 16 LEDs flash on for a fraction of a second to look as if a green dot is moving back and forth on the board. When a switch is flipped on the FPGA, control of the LEDs changes from the button logic described in case 8, to this logic written outside the switch statement. There is a 16 bit variable that begins with value 0000_0000_0000_0001b. This value is shifted left and right and written to the LED channel to make the light “move” back and forth. The LED would move too quickly for the human eye to see if the LED was updated every loop. A simple timer was created in the loop that counts 9999 loops and updates the LEDs on the 10,000th. This timer does not use a while or for loop so it does not block the rest of the code from being executed. This allows buttons to be pressed while the LED is moving back and forth without a delay. A delay in the movement of the LEDs can be seen while the serial monitor is being written to though. Serial takes a while to

communicate over especially running at 9600 baud so the loops where serial is being written or read from are much longer than other loops.

Final Solution

The final implementation of our design used 451 FF's and 752 look up tables (LUT). Compared to previous labs this a much larger utilization of the board and that is due to the fact that an embedded processor is implemented alongside the hardware logic. Table 1 below shows the post implementation utilization of the Basys 3 board for our design.

Resource	Utilization	Available	Utilization...
LUT	752	20800	3.62
LUTRAM	176	9600	1.83
FF	451	41600	1.08
BRAM	8	50	16.00
IO	51	106	48.11
BUFG	4	32	12.50
MMCM	1	5	20.00

Table 1: Basys 3 Utilization

The 2Hz counter logic used to display on the leftmost digit of the seven segment display used the 10MHz clock from the MMCM divided by 5,000,000. In order to divide the 10MHz clock by 5,000,000 requires a 23-bit counter, which when it reaches 5,000,000 will trigger a flag which will indicate that it is time to increment the counter, and thus producing a 2Hz counter. That means that 23 FF's are used for the counter incrementing at 10MHz and 4 FF's are used for the counter incrementing at 2Hz (27 total for the 2HZ counter module).

The reset delay module used 8 FF's, 7 for a counter to delay briefly the reset of Microcontroller slightly and then 1 for a flag to mimic the reset signal from the button. Lastly the seven segment module and the divider for the clock to drive the anodes used 25 for the anodes and the seven segments and then 17 for a counter to divide the input clock so that the anodes are not driven too fast. In total the hardware utilization from just the verilog we wrote was 69 FF's. The remaining FF's are used by the Microblaze embedded processor.

Figure 3: Vivado Warning Messages

Once the implementation was completed for the final design the warning messages remaining in the messages tab could be analyzed. Figure 2 shows the warning messages for the synthesis and for the implementation. Post synthesis, the warnings all pertain to unused sequential elements however these elements are all necessary for the implementation to operate properly and therefore can be ignored.

Conclusions

In this lab the Basys 3 board was used to demonstrate the implementation of an embedded processor with hardware logic on an FPGA. Using the Xilinx software a MicroBlaze processor was created on the FPGA and programmed using the XSDK in C. This processor took inputs and outputs from the board, which were wired using Verilog, and then using the C code modified outputs depending on the inputs. This design used several modules to complete each task which were then combined in a top module where the C code was implemented with the hardware to produce a final implementation. The most difficult part in implementation was figuring out how to properly read in data from the serial port. The code provided for this lab was sufficient in reading in a byte at a time but would miss other bytes even though the function was specified to read in 3 bytes. After some tinkering, it was determined that the read in function simply needed to be called a few times to read in each byte. Another problem arose where a 3 character string would have 4 bytes with the last byte being 13 in binary. This byte would wait in the the serial buffer to be received on the next button press and mess up the next serial reading. This turned out to be the byte sent when return is pressed on the keyboard. There was also a problem where if less than 3 bytes were received it would wait for the third to come in before continuing to execute later code. Knowing that 13 signifies the end of a string of data reading in serial data was made into a while loop that keeps reading in bytes until the character associated with return is received. This allows for any number of characters to be sent and the Microblaze will keep reading the serial until it captures the entire string. This was essential in creating logic to count the total number of bytes and the number of invalid bytes. This understanding of serial communications is the most valuable information obtained from the lab as everything else was simple and familiar. More features could always be added to this lab but the largest improvement is definitely the the feature that verifies the incoming string of characters over serial. Without this, any character could be inputted and the seven segment display which could cause problems and simply doesn't make sense since the seven segment module is only designed to display numbers 0-F. Future extensions of this lab should include this feature.