Final Project

EE365

Zander Poole

**Overview**

This report will go through the process of creating the system that was outlined in the problem statement. This problem statement asked the student to replicate a system design for displaying a message on an LCD, 7-segment display, and on Putty through UART. This design implementation should used the Trenz ZynqBerry board and Vivado 2019.1.

1. **Design Problem Statement**

The system that is shown and outlined in the problem statement is used to display a message of a sequence of 10 16-bit HEX strings on an LCD, 7-segment display, as well as through UART in Putty. The project was centered around using IPs in Vivado to create a design that could be controlled through C code in Xilinx’s SDK. According to the specifications of this design this C code is where the sequence of HEX strings should be stored. This means that this is also where the data should be sent to the components of the design that send the data to the LCD and 7-segment display. Another requirement for this design is that it needs to use an AXI-timer with a 1-second interrupt, meaning the data will be sent every one second.

Prior to diving into the design of the system two tutorials for creating a custom IP using an integrator and creating a custom IP block. The first of the two, *Creating a custom IP in Vivado using the IP Integrator*, takes the user through creating an IP Integrator that creates a PWM controller to control the light intensity of the 4 PMOD LEDs. This tutorial started by creating a new project and adding a new IP to edit. To start, a constant *PWM\_MAX\_COUNTER* is added as a constant integer, this variable controls the maximum pulse width. Four output ports were then added, *PWM0*, *PWM1*, *PWM2*, and *PWM3*. These outputs are for the communication of the signal to the four LEDs respectively. A counter was then implemented to increment the PWM signal, and then comparator statements to transmit the PWM signal to the registers was added following the counter. The max counter and four outputs are then added to the top level of the IP in the generic and port respectively. These variables were then also added to the generic map and port map, and assigned to themselves. The IP was then packaged and added to a block design along with the Zynq board and other required modules. The ports were then added to a constraint file to map them to pins on the board and then generate bitstream was run. Once the bitstream was generated and the hardware was exported the work needed to move to SDK. In SDK a new application project was created and the given C code was added. This C code basically just changes the PWM signal to gradually increase the intensity then reset. Once the board is programmed the LEDs acted as pictured in Figure 1 below.



Figure 1: LED Behavior with PWM

For the second tutorial, it guided the reader through creating a multiplier IP block with a design source implementation. First, a new IP was created and the given VHDL source file was added. Unlike the first tutorial the new variables, and in this case component of multiplier.vhd, are instantiated and assigned to one of the IPs slv registers and one of the output registers. The IP was then packaged and added to the block design as in the first tutorial, and subsequently bitstream was generated and hardware was exported. In SDK, a new application project was created and this time using the helloworld example C code. The code was then edited, as per the tutorial, to send input data to the multipliers IP location and then printing the outputs. The design/output can be seen in Figure 2 below.

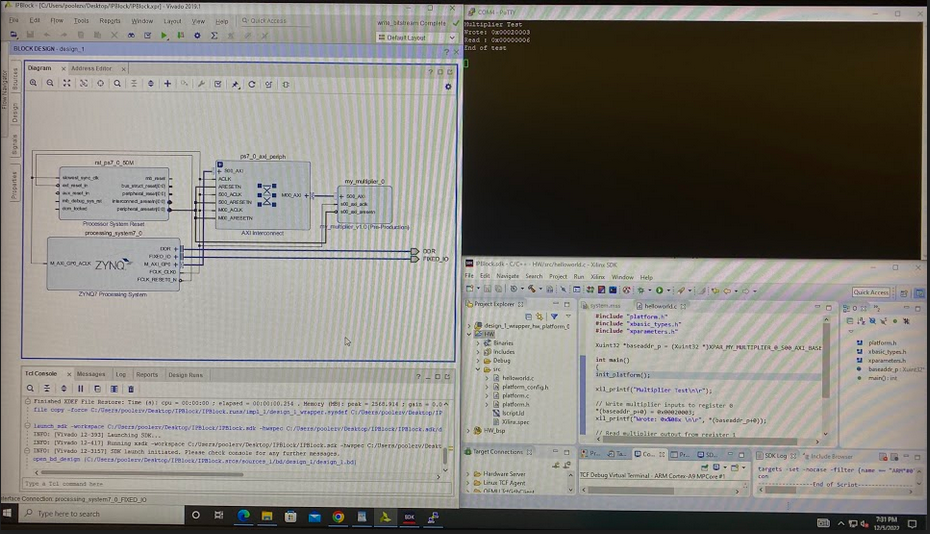


Figure 2: Design and Output of my\_multiplier

1. **Detailed Design**

The first thing that was done to start the design of this system was the created of a block diagram that detailed what the system would look like, this can be seen in Figure 3 below.

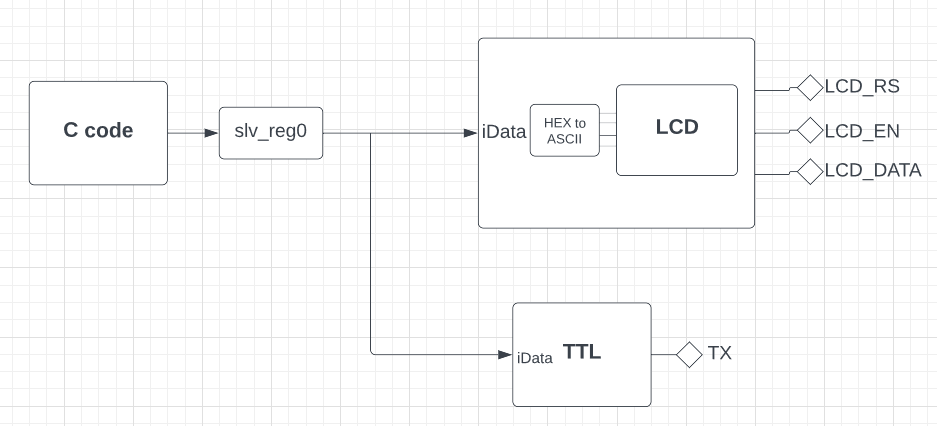


Figure 3: Block Diagram

Now that the layout of the design has been established, a new project was created and a block design is created where the Zynq board is added as well as the AXI timer IP block. The first custom IP was then created, *myTTL*, this IP would handle the communication with the 7-segment display through a single output signal. To create this IP the two files from a previous implementation of TTL serial were added, *TTL\_user\_logic.vhd* and *TTL\_master.vhd*. In the AXI instantiation for the IP a component of *TTL\_user\_logic* was added, and then was instantiated assigning the *iData* signal to the lowest 16 bits of the register slv\_reg0 and the output signal *TX* to the output *TX* which was added to the port of the entity. The *TX* output was then added to the port of the top level of the IP and assigned to itself in the instantiation of the AXI level components port map. With the input data being assigned to the slv register, it allows the input of the data to be sent through the C code by calling the IP address. Next, the IP block for LCD communication was created, *myLCD*. This IP was created in the same way as the TTL IP block, using another set of previously created code for LCD communication, *LCD\_user\_logic.vhd* and *LCD\_master.vhd*. For this application iData was set to the same register for *myLCD’s* AXI instantiation, and the output variables were created in the port of both files and connected in the same way. The only real difference being, the LCD communication implementation uses three outputs, *LCD\_RS*, *LCD\_EN*, and *LCD\_DATA*, where *LCD\_DATA* is a 8 bit vector. One addition to this part of the system is the creation of a HEX to ASCII converter, which is necessary because the LCD only understands input data of its ASCII value, not HEX letters. This is then instantiated four times in *LCD\_user\_logic* and creates four ASCII values that represent the four HEX letters.

Once the two IPs were packaged, they were added to the block diagram and routed automatically. The only addition is these two blocks need output ports which were added and connected to the ports on the IPs. The one thing to note about the block design is that the interrupt port of the AXI timer to the Zynq board IP, this will allow for the use of the interrupt. Now that the block design is complete, output products were generated and a design wrapper was created. Once this is done, generate bitstream can be run and the hardware can be exported.

1. **Module and Specification Testing**

Now in SDK, a new board support package can be created using the proper processor and from here the options are opened where an example code, *xtmrctr\_intr\_example.c*, can be found. This is the code that will be used and modified to meet the needs of this design implementation. The first modification, aside from fixing the variables that were incorrectly written, is creating the volatile int *Data\_Sel*, which is the way of indexing the array of 10 HEX strings, which is created and added next. Following the creation of the two main variables, a new implementation of the TimerCounterHandler function is written. In this implementation, in the if statement, the HEX value of the current index is sent to the addresses of the two IPs. This is sent to the slv register which was previously set to the input data ports of the LCD and TTL IPs. A line is then printed through UART that states the current data being written. *Data\_Sel* is then increased by 1, meaning the next time the function is called the next element will be sent and printed. Finally there is an if statement that checks if *Data\_Sel* is equal to 10, its max value, and if it is it resets to zero. Another edit to the example is the comment out of the body of the while loop in TmrCtrIntrExample function, which causes the system to run indefinitely. One of the most important edits is setting the reset value, *RESET\_VALUE*, to 0xFD050F7F. This value controls the length of the interrupt, in this case a one second interrupt is needed. It is known that since the system clock is 50 MHz one second is equal to 50,000,000, so 50,000,000 can be subtracted from 0xFFFFFFFF to get the reset value of 0xFD050F7F. Now that the C code is written the board is ready to be programmed and the code can be run.

1. **Results and Analysis**

Once the board was programmed and the code was run, the output sequence was seen on the LCD, 7-segment display, and in Putty. These outputs can be seen in Figure 4 below.



Figure 4: Output of System

The system was replicated and meets all specifications given in the problem statement. This did not come without some trouble as there were multiple miscues such as miscalculated maximum counter values, as well as coding errors in the C code. Overall, the system performs exactly as expected and follows all requirements necessary.

1. **References**

Khondker, Abul. “Moodle Course: Adv Digital Circuit Design.” *Course: Adv Digital Circuit Design*, https://moodle.clarkson.edu/course/view.php?id=27556.

Trenz Electronic. “Trenz Electronic TE0726 .” *Trenz Electronic TE0726 Reference Manual*, https://shop.trenz-electronic.de/trenzdownloads/Trenz\_Electronic/Modules\_and\_ Module\_Carriers/special/TE0726/REV03/Documents/TRM-TE0726-03.pdf.

Johnson, Jeff. “Creating a Custom IP Block in Vivado.” *FPGA Developer*, 4 Aug. 2014, https://www.fpgadeveloper.com/2014/08/creating-a-custom-ip-block-in-vivado.html/.

Diligent. “Creating a Custom IP Core Using the IP Integrator.” *Creating a Custom IP Core Using the IP Integrator - Digilent Reference*.

1. **Appendix**

[**myLCD\_v1\_0.vhd**](https://github.com/poolezv/EE363_Final/blob/main/myLCD_v1_0.vhd)



[**myLCD\_v1\_0\_S00\_AXI.vhd**](https://github.com/poolezv/EE363_Final/blob/main/myLCD_v1_0_S00_AXI.vhd)

****

[**myTTL\_v0\_0.vhd**](https://github.com/poolezv/EE363_Final/blob/main/myTTL_v1_0.vhd)

****

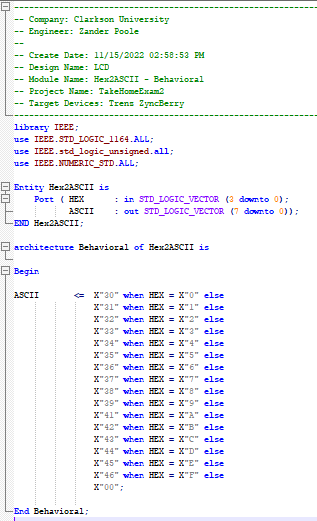
[**myTTL\_v1\_0\_S00\_AXI.vhd**](https://github.com/poolezv/EE363_Final/blob/main/myTTL_v1_0_S00_AXI.vhd)

****

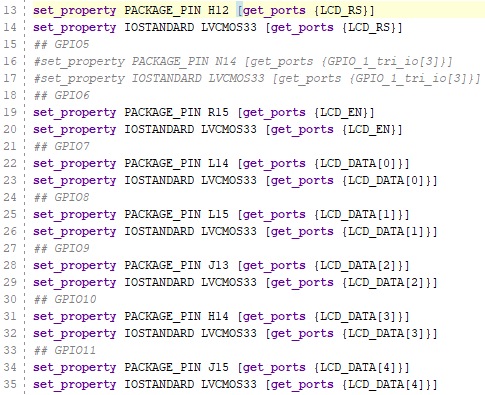
[**xtmrctr\_intr\_example.c**](https://github.com/poolezv/EE363_Final/blob/main/xtmrctr_intr_example.c)

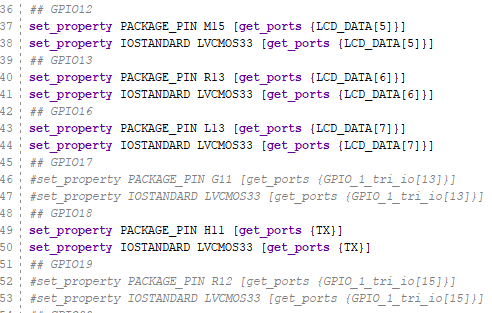
****

[**Hex2ASCII.vhd**](https://github.com/poolezv/EE363_Final/blob/main/Hex2ASCII.vhd)

****

**constraints.xdc**

****

****

[**ALL CODE HERE**](https://github.com/poolezv/EE363_Final.git)