RUTGERS

UNIVERSITY

Course Name: EMBEDDED SYSTEMS I / III

Course Number and Section: **14:332:493:03**

Year: Spring 2021

Final Report

Lab Instructor: Philip Southard

Student Name and RUID: Rishabh Chari 177007380

Date Submitted: 5/5/21

GitHub Link: https://github.com/rishabhchari/Embedded-1-Final-Project

Purpose/Objective:

This purpose of this project was to design and implement an embedded system on the Zybo Z7 using at least 2 Pmods. This project was a culmination of the concepts from the course and required knowledge of VHDL, interfacing with Pmods, and the digital and embedded design process. This project involves multiple components that work together to take computer keyboard input and display the characters serially on the OLED Pmod

Embedded System Design:

There are 4 components for this project, the Minicom serial terminal emulator, the USBUART Pmod, a processor on the Zybo Z7, and the OLED controller to drive the OLED display. Each component has a separate function that all comes together in the top-level design.

Minicom Terminal

The first part of the embedded system is the Minicom Terminal. This part does not require any actual VHDL coding, but it serves a very important task of supplying the system its input. Without the minicom terminal there would be no input characters for the UART to receive. The minicom terminal requires two parameters when opening it, the baud rate, and the location of the USBUART pmod connection. The baud rate is the rate at which the characters are sent over the serial communication channel. Therefore, the baud rate parameter for Minicom must be chosen with respect to the UART receiver, in other words if the Minicom baud rate and the UART baud rate do not match the system will not work. The second parameter is the location of the USBUART pmod. Vlab has 3 possible ports for the USBUART connection, ttyUSB0, ttyUSB1, ttyUSB2. If the USB port is incorrectly chosen the system will not work as the input will not be going to the board.

USBUART

The USBUART Pmod is the next component of the system. When the Minicom terminal is configured properly then it will be able to input characters. However, the board will need a way to receive these characters, therefore we use a USBUART Pmod as a receiver. The USBUART has 4 connections, apart from VCC and GND. These connections are CTS, RXD, TXD, and RTS. Each connection serves a specific purpose. CTS and RTS are used for Hardware flow control, which is unneeded for this project. So, CTS and RTS are both tied to ground. The other 2 connections are RXD and TXD. RXD is the connection that allows for the USBUART Pmod to receive data from the host board, which is the Zybo Z7. This project is not sending data from the board for the Pmod to receive. Therefore, this connection is unneeded and will not be in our design. TXD is the connection that links the Minicom terminal input to the host board. It allows for data to be transmitted from the Pmod to the Zybo Z7.

The UART entity has a rx port and a tx port. These two ports act as the receiver and transmitter for the UART entity itself. Therefore, TXD is connected to the rx of the UART and the tx of the UART is left open as the system wants to receive the characters that the Pmod is sending to the board, but it does not want to send characters to the Pmod. The other ports for the UART are clk, en, send, rst , charSend for the inputs. The outputs are ready, newChar, charRec. Again, the project is not sending data to the Pmod so send and charSend are both tied to ground. Clk is mapped to the system clock and enable is mapped to a clock divider output that divides the system clock to the 115200 Hz baud rate. Rst is tied to a debounced reset button which has a debounce time of 20ms. The last 3 ports ready, newChar, and charRec are used in the processor entity.

Finally, to explain the theory of operation of the UART receiver, the receiver first waits for the rx line to go from high to low, because when rx is high then no data will be transferred. Once the rx line is low then the data transmission starts. The data from each character being received are stored into a shift register until the required number of data bits. Once all the data from a character is received 1 stop bit is sent to indicate the end of data reception. This design does not use a parity bit to check for parity. Once a character is sent then it will be processed properly in the processor entity.

Processor

The processor entity is used to tell which characters correspond to the specific place on the OLED display. The OLED display requires that each pixel in the display has its own 8-bit std_logic_vector signal. The characters received are in 8-bit std_logic_vectors for therefore there is no need to convert the signals to a different type. The initial design of this project was to read the character received and check if it compares to a character to create a message with only those characters. However, that is very restrictive on the user as it only allows for one specific order of the characters.

The current implementation of the processor works is based on the echo entity from Lab 3, which would receive characters from the Pmod then send them back to be displayed on the serial terminal. This processor is a similar implementation except we do not send the characters back to the serial terminal. The entity of the processor has a few ports. The inputs are clk, reset, en, ready, newChar, and charIn. The outputs are 16 8-bit std_logic_vectors. Similar to the

UART controller, clk is mapped to the system clock and enable is mapped to the same clock divider output as the UART. Rst is tied to the debounced reset button. Ready and newChar are mapped to the UART outputs corresponding to ready and newChar, while charIn is mapped to the UART output of charRec which holds the data for the character received. The processor uses a finite state machine that has 4 states, idle, busyA, busyB, and busyC. It also holds a count variable, of type natural initialized to 0, used in the finite state machine.

The processor starts with an synchronous reset that will clear all 16 output signals, the count variable and set the finite state machine's current state at idle. Otherwise, process is in the state machine. In the idle state, if newChar is asserted to '1' then the processor will check the count variable. When the count variable is equal to a specific integer between 0 and 15 then charIn will be stored in the output signal corresponding to the count's value, the count variable will increment by 1 and the current state will go to busyA. To give an example, if count = 8 before the newChar check once newChar is equal to 1, the count variable will be checked and then charIn will be stored into the char_8 signal, the counter will increment, and the current state goes to busyA. Each integer from 0 to 15 holds this control block. If the count variable is not within the range of 0 to 15 then the processor will not do anything until reset is asserted to '1'. Once rst is asserted to '1' then all the 16 output signals are cleared, the count is reset back to 0 and the current state goes back to idle. This processor allows for the user to send characters to the OLED display and reset whenever the user wants or after the screen is filled. The final elements of the processor are the busy states. If the current state is busyA then the state machine goes to state busyB, which will go to busyC. Once in busyC only if ready is asserted to '1' then the current state goes back to idle.

Oled Controller

The final component of the embedded system design for this project is the OLED Controller. The OLED controller has inputs of clk, rst, and the 16 std_logic_vectors from the processor's output. The OLED controller holds 7 outputs to drive the display. Chip select, Master-Out-Slave-In, SPI Clock, the Data/Command Control, Power Reset, Vbat Battery Voltage Control , and the Vdd Logic Voltage Control. The SPI protocol is how the OLED will receive data from the ZyboZ7 board. Once the CS line is low and it holds low then data is transferred serially on the DC line. The DC connection will make sure which data being transmitted to the display is used for the display or for controlling the display. The MOSI and the SCLK are used as control signals to make sure the DC line is sending the correct data at the correct time. VDDC and VBATC connections control the power supply of the Pmod. The VDDC connection controls the power of the logic driving the OLED display, so the SPI Protocol components. The VBATC connection controls the power of the OLED display. Finally, the RES output acts as a reset the controller when asserted to low.

The OLED controller has two components an OLED Init entity and an OLED Example entity. OLED init is used to control the initialization of the OLED display. Oled Example holds the control and logic of the display itself. This is the entity that holds how the display itself acts. In this entity the 16 8-bit std_logic_vectors are established as inputs. The screen is of type OledMem where OledMem is a 4 by 16 array of 8-bit std_logic_vectors. Therefore, the inputs can be placed into separate elements into the OledMem Array to be shown on the OLED display. OledEx has a state machine that will follow how to display the screens and how to update them. To hold a screen, the after_update_state, which is the state that the state machine goes to after the UpdateScreen state is finished, is set to the same screen. For example, if the screen state is called HelloWorld and if the screen needs to be held on HelloWorld and not clear then after_update_state will be set to HelloWorld state.

Testing and Usage

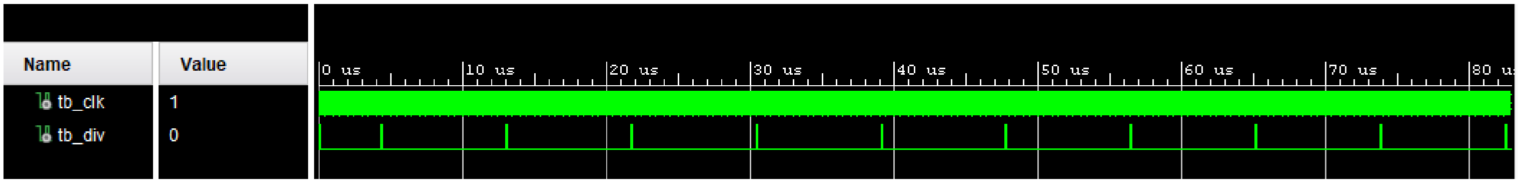
When all the components are separately tested the Top Level is designed. This top level connects all the components together using intermediate registers and structurally port mapping. The top level's entity has inputs of CLK, RST, and TXD. It outputs the 7 outputs explained in the OLED control entity and CTS and RTS which are set to ground. This file is synthesized and implemented. When a bitstream is generated the bit file can be used to program a Zybo Z7. Once the Zybo board has been programmed the Minicom terminal would need to be opened using the proper parameters and then the user can type messages into the terminal to get an output result. If the user would want to use the reset functionality to clear the screen then they would have to activate the virtual buttons and press the button that corresponds to the RST connection pin to perform a reset of the display and characters. By using Minicom, VLC media player and virtual buttons the design can transmit messages from Minicom terminal screen to the OLED display.

Requirements

The requirements of this project did not have too many restrictions apart from interfacing the Zybo Z7 at least 2 Pmods. The project while successful could have been improved in a few areas, specifically the features that the system had, the utilization of the resources and the power of the system. For features the project could have implemented ways to delete characters or included other control characters such as tabs. For utilization, the I/Os as shown in the post synthesis table and the power graphs are heavily utilized and improvements could be made to decrease the utilization of I/Os. Finally for power, a bulk portion of the power comes from I/Os so therefore a way to redcue the cost of the I/O resources could be very benefical to improving the design of the embedded system.

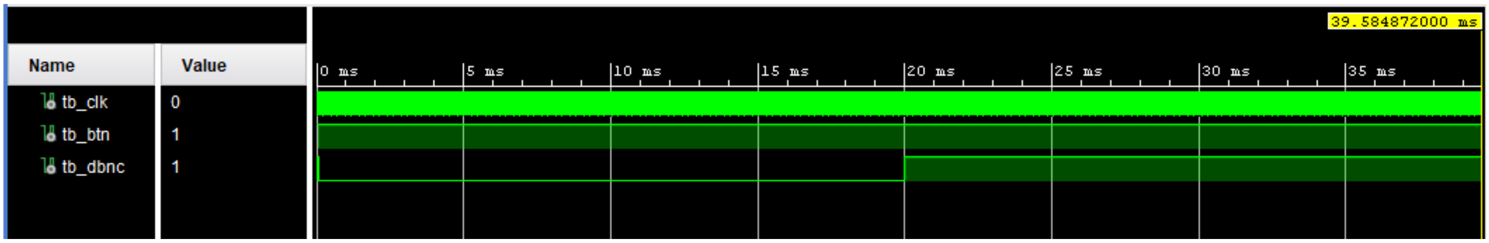
Simulation Waveforms:

Clk Div – 115200Hz



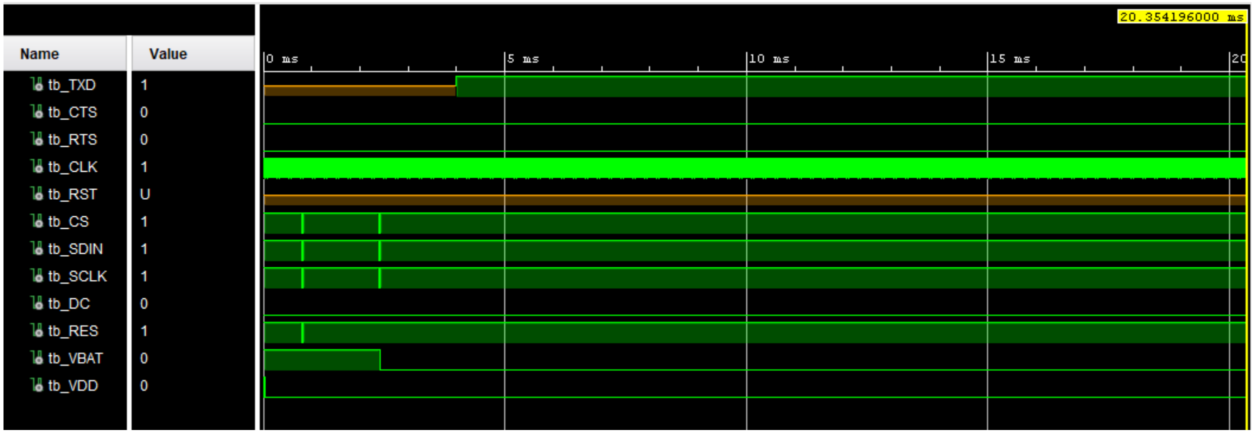
Name	Value
tb_clk	1
tb_div	0

Debounce – 20ms



Name	Value
tb_clk	0
tb_btn	1
tb_dbnc	1

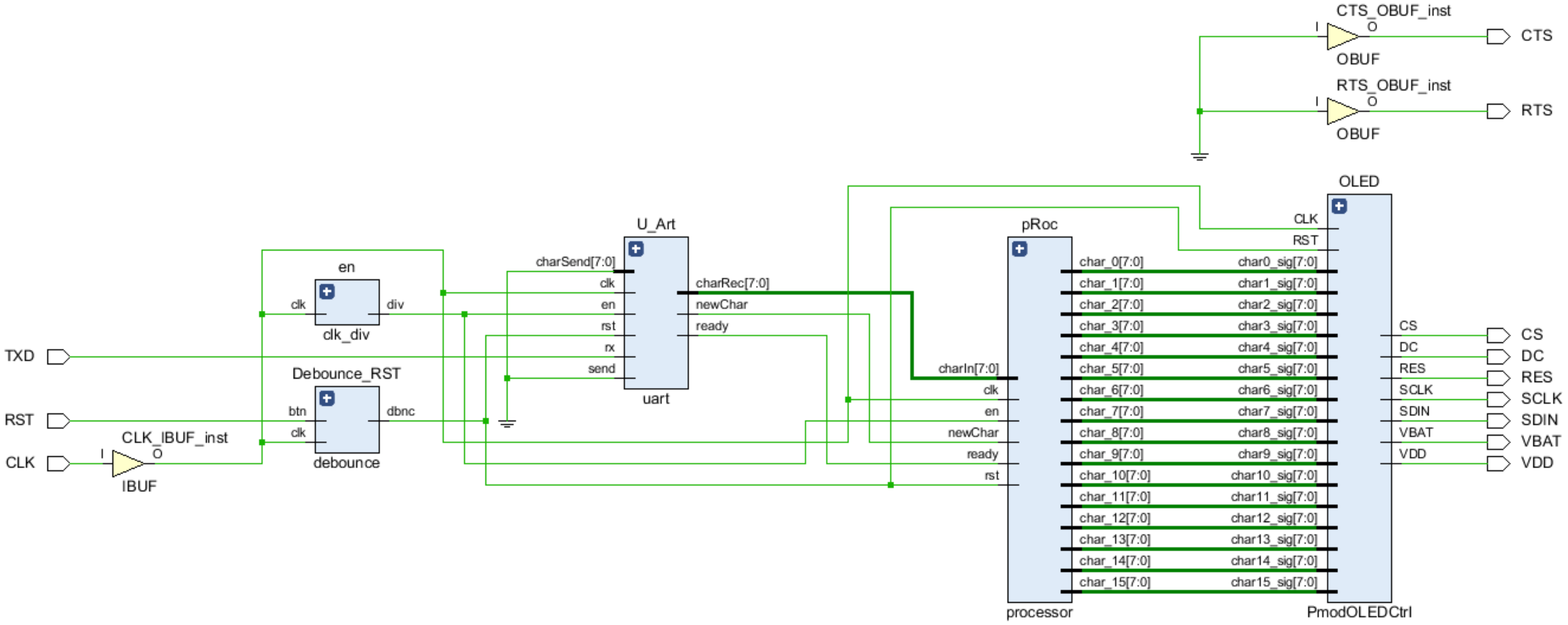
Top Level Design



Name	Value
tb_TXD	1
tb_CTS	0
tb_RTS	0
tb_CLK	1
tb_RST	U
tb_CS	1
tb_SDIN	1
tb_SCLK	1
tb_DC	0
tb_RES	1
tb_VBAT	0
tb_VDD	0

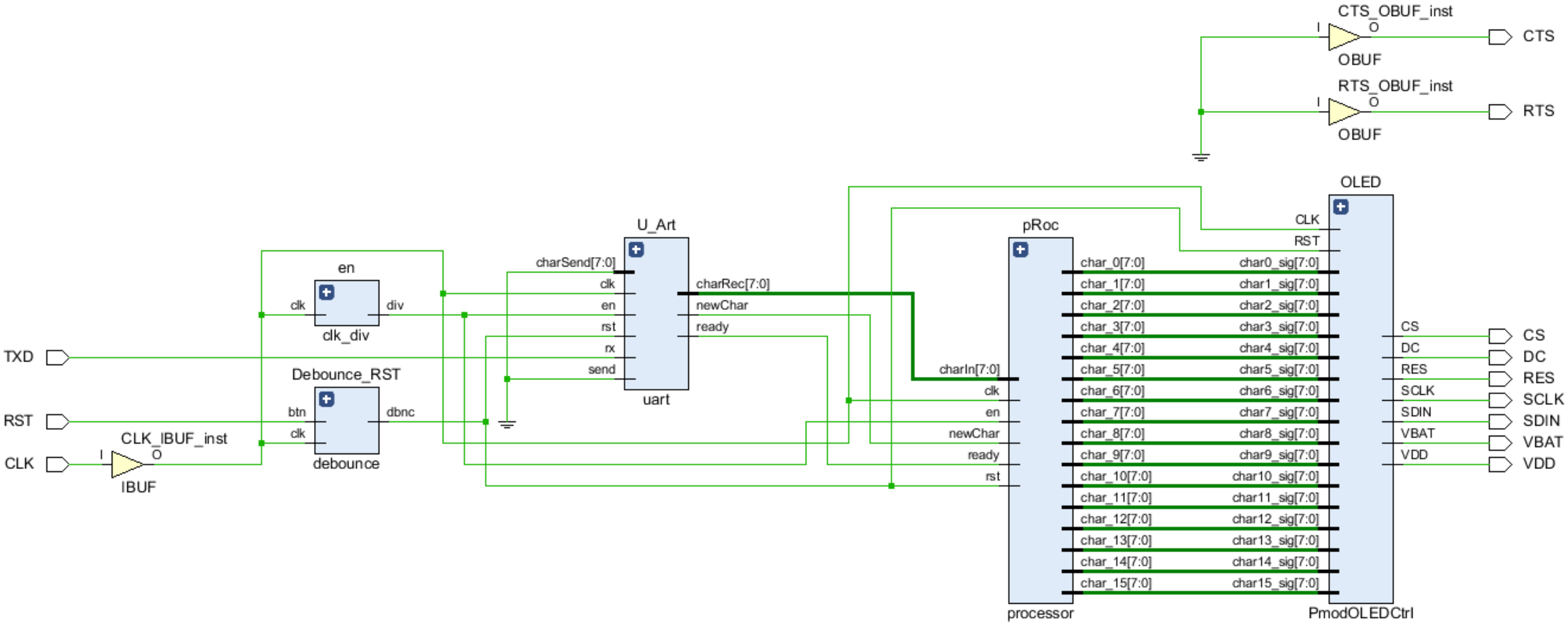
System Block Diagram

While the Elaboration Schematic is not really a block design it shows the diagrams well and shows how the components interact with each other. So the Elaboration Schematic is a sufficient representation of the black-box diagram of the system.

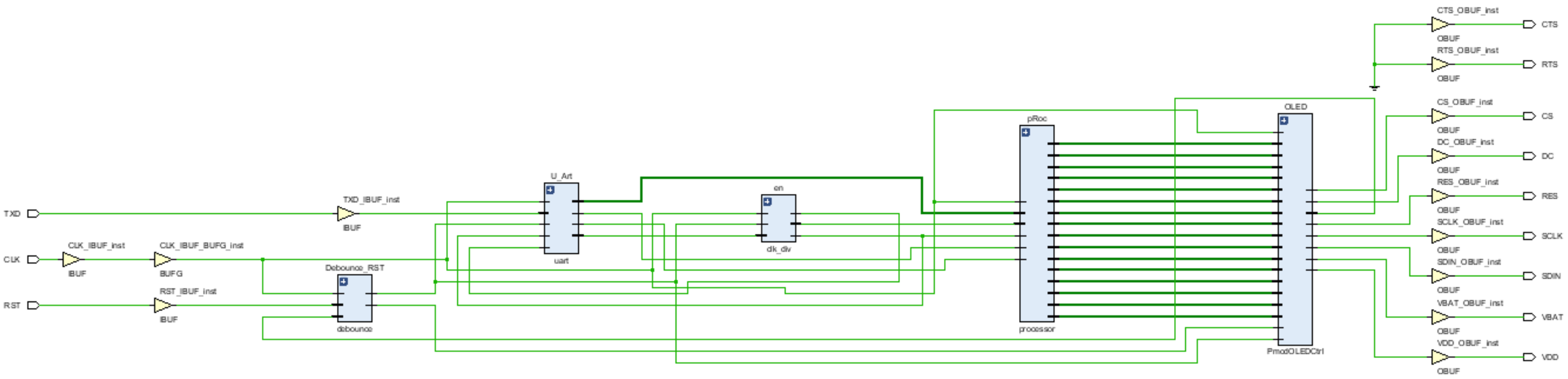


Vivado Schematics:

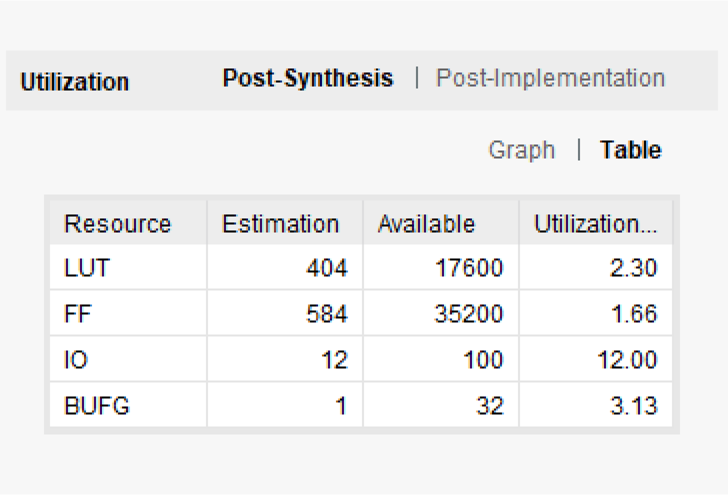
a) Vivado Elaboration Schematic



b) Vivado Synthesis Schematic



c) Post- Synthesis Utilization Table

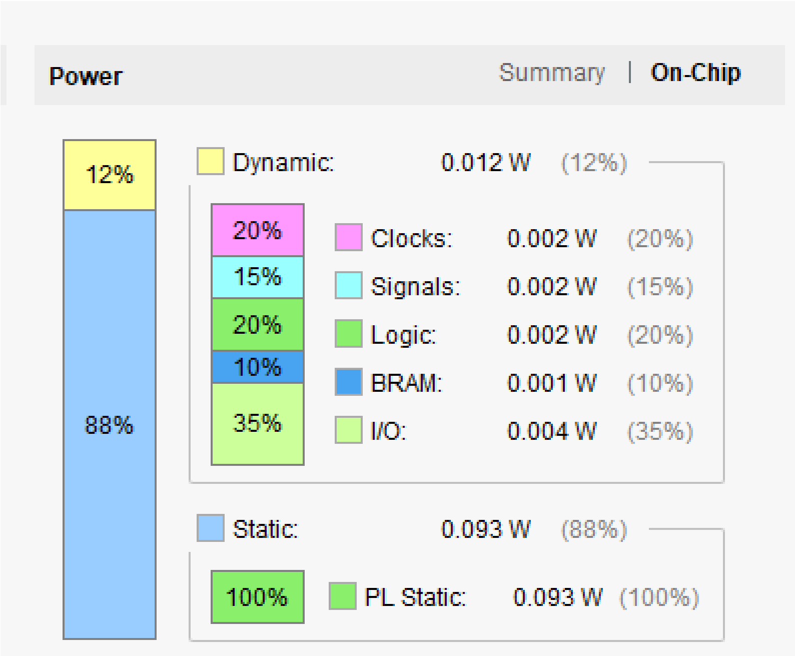


Utilization Post-Synthesis | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization...
LUT	404	17600	2.30
FF	584	35200	1.66
IO	12	100	12.00
BUFG	1	32	3.13

d) <u>On-Chip Power Graphs</u>



Dynamic:	0.012 W	(12%)
Clocks:	0.002 W	(20%)
Signals:	0.002 W	(15%)
Logic:	0.002 W	(20%)
BRAM:	0.001 W	(10%)
I/O:	0.004 W	(35%)
Static:	0.093 W	(88%)
PL Static:	0.093 W	(100%)

Constraint File

The constraint file required a few items to implement the design. The constraint file has a created clock with a period of 8ns and a waveform from 0 to 4, which is equivalent to the required 125MHz clock frequency. For the package pin number, if the board being targeted is the old Zybo Z7 the pin needed is L16. If the board being targeted is the new Zybo Z7 the pin needed is K17.

The UART Pmod has 4 connections. The connections are RTS, RXD, TXD, and CTS. The package pins for these connections are T20 , U20, V20, and W20, respectively.

The OLED Pmod has 7 connections. The connections are CS, SDIN, SCLK, DC, RES, VBAT, VDD. The package pins for these connections are T14, T15, R14, U14, U15, V17, and V18, respectively.

The final element in the .XDC constraint file is the RST connection. In the design RST is a button that toggles. Because this project is run on VLAB the package pin being used is K16 which is a virtual button that corresponds to a keyboard press.

Conclusion:

This project required knowledge in vast area of the concepts taught in the Embedded 1 course. I reinforced my understanding of a few concepts taught in the course while designing and implementing this project. I was able to interface multiple Pmods to get a working embedded system. The two most important pieces of this project were the two Pmods, the USBUART Pmod and the OLED Pmod. Without those components this project would not be able to happen. From the results shown in the demo and in the previous section, I feel I got the desired results of the project. I was successfully able to implement a way for the Zybo Z7 to receive characters from the computer input and then process that data such that the OLED display could display those characters serially. I also implemented a way for users to be able to clear the screen and reset back to the beginning of the display.

I think the best way to follow up this project is to implement two features to make the system's operation much more convenient. The first feature is a way to remove characters if they were added without having to fully clear the screen. To do this I would need to add a way to delete characters serially and I think one possible way of implementing this feature would be to have a button that toggles whether the user wants to add or delete characters. Once the button has been toggled then the user would be able to press a key to delete the previous characters and could change when they want to delete and when they want to add characters. The second feature that would be beneficial to this design is a way to move around the display to change characters individually. Right now, the display requires a certain order for the characters so by implementing for the user to dynamically update the characters on the screen would be beneficial. Overall, this project was successful, and the concepts used in this project have reinforced my understanding of embedded system design.

References:

I would like to make a note that the code used for the OLED controller was adapted from Digilent's reference for the PMOD on their website at https://reference.digilentinc.com/reference/pmod/pmodoled/start