

California State University, Northridge
ECE 520L - System on Chip
Fall 2022

Lab 5: Custom Vivado IPs



September 13, 2022
Instructor: Saba Janamian

Troy Israel

Introduction	3
Procedure	3 - 4
Results	5 - 6
Analysis/Conclusion	7

Introduction:

In this lab we will be learning how to design our own custom IPs using the IP integrator in Vivado and to implement them into our project. We will also learn how to add custom IPs created by other vendors and how to modify them for our own project. We will also learn how to add existing firmware code which can be written in either VHDL or Verilog and modify that HDL code to meet the design specific of our project.

Procedure:

Part 1:

For this part of the lab we will be using the IP integrator within Vivado. We will be designing a subsystem for our design. Firstly, we will download two files, one of which will be a file called IP Integrator and then because it is a zip file we will unzip the file inside our repo. One important thing that this part will show us is how we can include external IPs to vivado by going into the tools, settings and going into the repository. Inside the file system we will get VHDL files from the KCU105 folder that is from the downloaded IP_integrator file. Now we can create a block design. We will add the following IP blocks to our design: Utility_buffer, Uart_rx, led_ctl, Uart_baud_gen and three meta_harden blocks. The meta_harden blocks will all have their names changed to better suit the tasks that they will be doing. Next we will create all the external ports, we have four input ports and one output port. Now we can make all the internal connections and use the hierarchy tool to simplify the overall look of the design.

Part 2:

For this part of the lab we will learn how to package preexisting HDL code. We first have to download a file called Custom_IP and recreate the project inside Vivado. Initially there are some IPs that need to be upgraded. Then we will use the Create & Package IP tool to package the current project. Inside the File Groups we remove a lot of the extra files that came with the zip file and then after we make changes we merge them. We also configure the ports and interfaces.

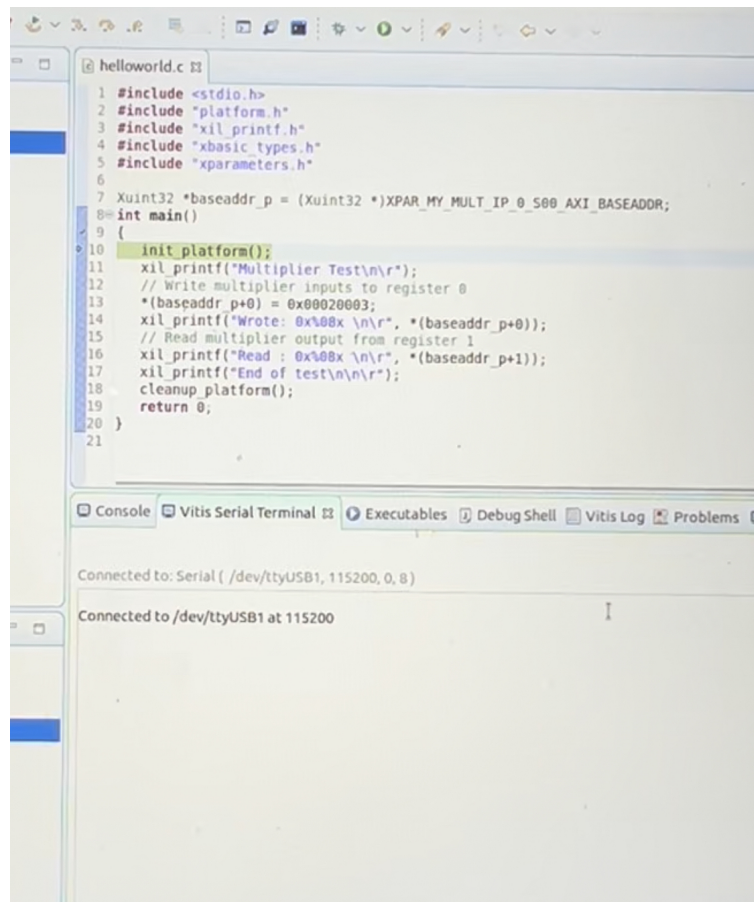
Each of the modules we choose an interface and map the physical ports. Once we are done we can save all the changes but while the file gets updated we cannot return to editing the package. Vivado forces us to re-edit the file again and we have to modify the name.

Part 3:

For the part of the lab we will create our own AXI IP and create drivers to communicate with the system. We will be using a prebuilt multiplier VHDL file. We use two 16 bit numbers but when it comes to the processor, it only works with 32 bit numbers. When the project is first created in Vivado, by default vivado sets the target language to Verilog and for this case we want to be using VHDL. This means that if we make the file initially it would make it in Verilog, so we change the target language in the project summary. For this part we will be using the IP generator and will be making an S00_AXI. We will make it a lite version and use 4 registers and it will be a slave interface. It will create this module in a VHDL file, and we will be modifying the generated VHDL files for the IP we are designing. We intinate the multiplier vhd file that we downloaded and add it to the design. After we make the changes to the VHDL files we then can go to the IP Package and merge the changes. Inside the address editor we will change the base address to 0x4000_0000.

For the Vitis section of this part, after we generated the bitstream and exported the XSA file we will create a C project. We will have various xilinx header files that will allow us to use specific functions and types so that we can do things like get the base address of the device, read and write to the device. All we are doing in the Vitis C project is reading and writing to two numbers and getting the result in the debugger console. While this is a simple task, this is a very critical task because we haven't had much experience with reading and writing on the device.

Results:



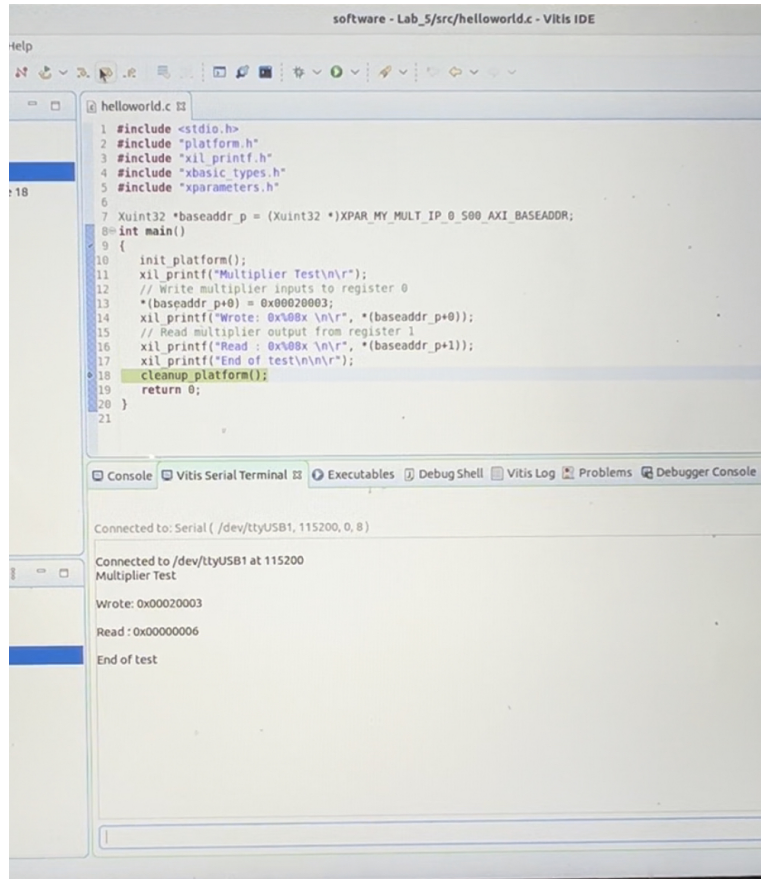
```
1 #include <stdio.h>
2 #include "platform.h"
3 #include "xil_printf.h"
4 #include "xbasic_types.h"
5 #include "xparameters.h"
6
7 Xuint32 *baseaddr_p = (Xuint32 *)XPAR_MY_MULT_IP_0_S00_AXI_BASEADDR;
8
9 int main()
10 {
11     init_platform();
12     xil_printf("Multiplier Test\n\r");
13     // Write multiplier inputs to register 0
14     *(baseaddr_p+0) = 0x00020003;
15     xil_printf("Wrote: 0x%08x \n\r", *(baseaddr_p+0));
16     // Read multiplier output from register 1
17     xil_printf("Read : 0x%08x \n\r", *(baseaddr_p+1));
18     xil_printf("End of test\n\r");
19     cleanup_platform();
20     return 0;
21 }
```

Console Vitis Serial Terminal Executables Debug Shell Vitis Log Problems

Connected to: Serial (/dev/ttyUSB1, 115200, 0, 8)

Connected to /dev/ttyUSB1 at 115200

Fig 5.1 Successful connection to the Zybo Z7



```
software - Lab_5/src/helloworld.c - Vitis IDE
helloworld.c
1 #include <stdio.h>
2 #include "platform.h"
3 #include "xil_printf.h"
4 #include "xbasic.types.h"
5 #include "xparameters.h"
6
7 Xuint32 *baseaddr_p = (Xuint32 *)XPAR_MY_MULT_IP_0_S00_AXI_BASEADDR;
8
9 int main()
10 {
11     init_platform();
12     xil_printf("Multiplier Test\n\r");
13     // Write multiplier inputs to register 0
14     *(baseaddr_p+0) = 0x00020003;
15     xil_printf("Wrote: 0x%08x \n\r", *(baseaddr_p+0));
16     // Read multiplier output from register 1
17     xil_printf("Read : 0x%08x \n\r", *(baseaddr_p+1));
18     xil_printf("End of test\n\r");
19     cleanup_platform();
20     return 0;
21 }
```

Console | Vitis Serial Terminal | Executables | Debug Shell | Vitis Log | Problems | Debugger Console

Connected to: Serial (/dev/ttyUSB1, 115200, 0, 8)

Connected to /dev/ttyUSB1 at 115200
Multiplier Test
Wrote: 0x00020003
Read : 0x00000006
End of test

Fig 5.2 Final result of after full program runs

Conclusion:

This lab has taught me a variety of useful features in Vivado and very important design strategies. Each part showed me various ways of implementing my own custom IPs and how to add IPs that were prebuilt. I learned how to simplify my block designs into a hierarchy to make the design layout in Vivado look a lot cleaner. I also learned how to make connections between each block is a much more efficient way and that is to use the drop menu and select each of the available ports by groups. Another very important thing I learned while adding a pre-design file is what to do when you get a message from Vivado saying the file is outdated. This is very crucial because maybe on the job we will use a lot of prebuilt files and a lot of times the original designers for these files have put the most up to date download link or where we download the file it does not have access to the most recent repository.

Appendix:

Github: <https://github.com/csun-ece/fa22-e520-lab5-troykerim>

Youtube: <https://youtube.com/shorts/K3warUbWlO4?feature=share>