ECEN 449 Lab 3: Creating a Custom Hardware IP and Interfacing it with Software

02/13/2020

Jason Gilman

**Introduction:**

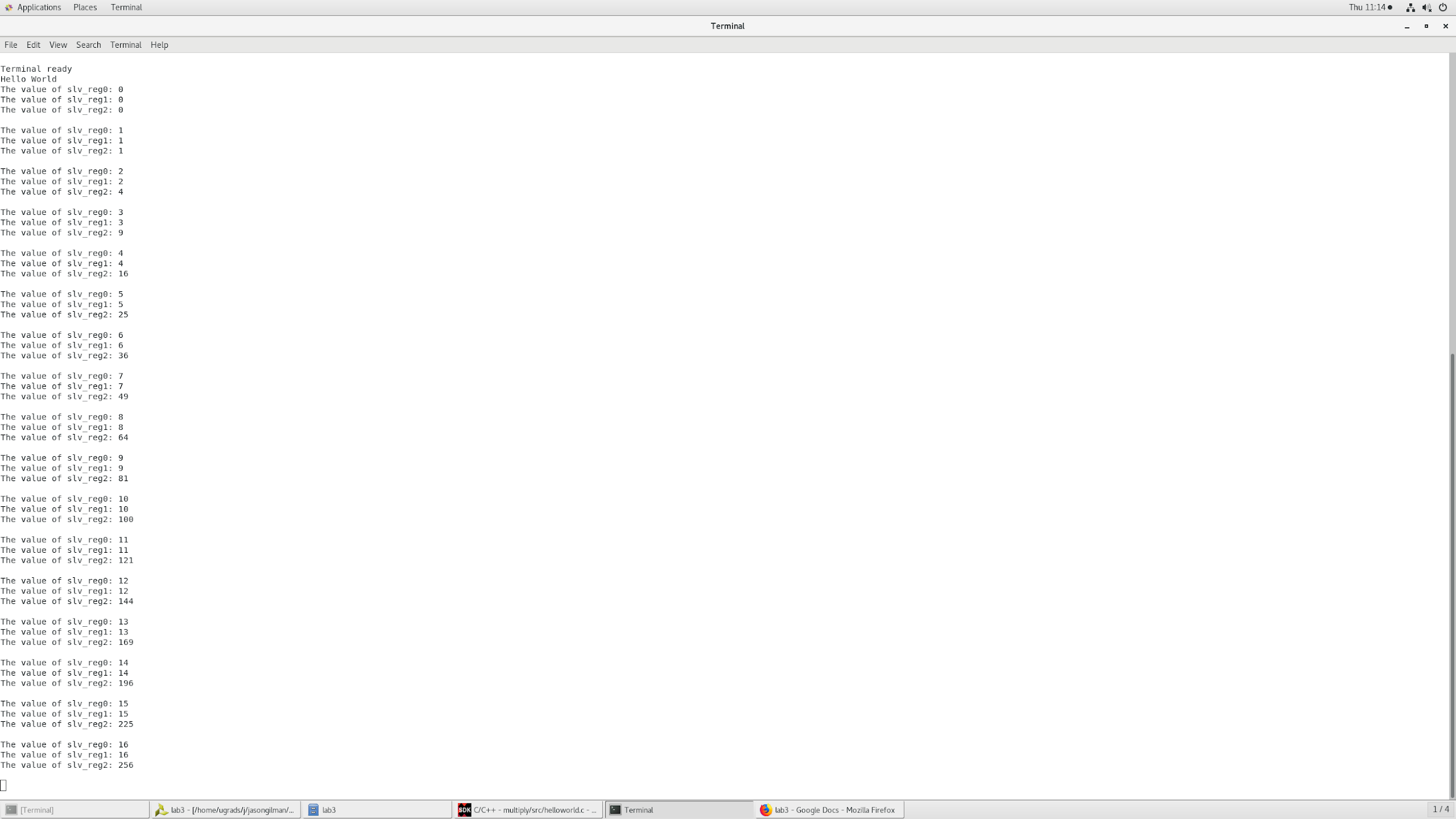
In this lab, we were tasked with creating a custom IP module and implementing it with a Zynq Processing System. The platform was then pushed to the SDK, where software was developed to aid interaction between the hardware and software.

**Procedure:**

* Initially, the system was set up with a Zynq processor system. This generated the block diagram for the system.
* A new IP module was then made and added to the block diagram.
* Logic was added to the multiply IP so that the first two registers would be multiplied and the product would be stored in the third register.
* The hardware was then pushed to the SDK, where software was implemented to facilitate the interaction between the peripheral and the hardware.
* By using the ‘xparameter.h’ and ‘multiply.h’ headers, software was implemented to write a series of numbers to the first two registers.
* A for loop was used to iterate through 0 to 16 and write each number to the first two registers, so that they would be multiplied.
* The product was then read from reg2, which effectively was the square of the input.
* All three numbers were printed to the linux terminal.

**Result/Output:**

The following was the final output of the code. The pattern of the output is such that the value of reg0 and reg1 is displayed, the the product of the two is displayed in reg2. The input iterated from 0 to 16.



**Conclusion:**

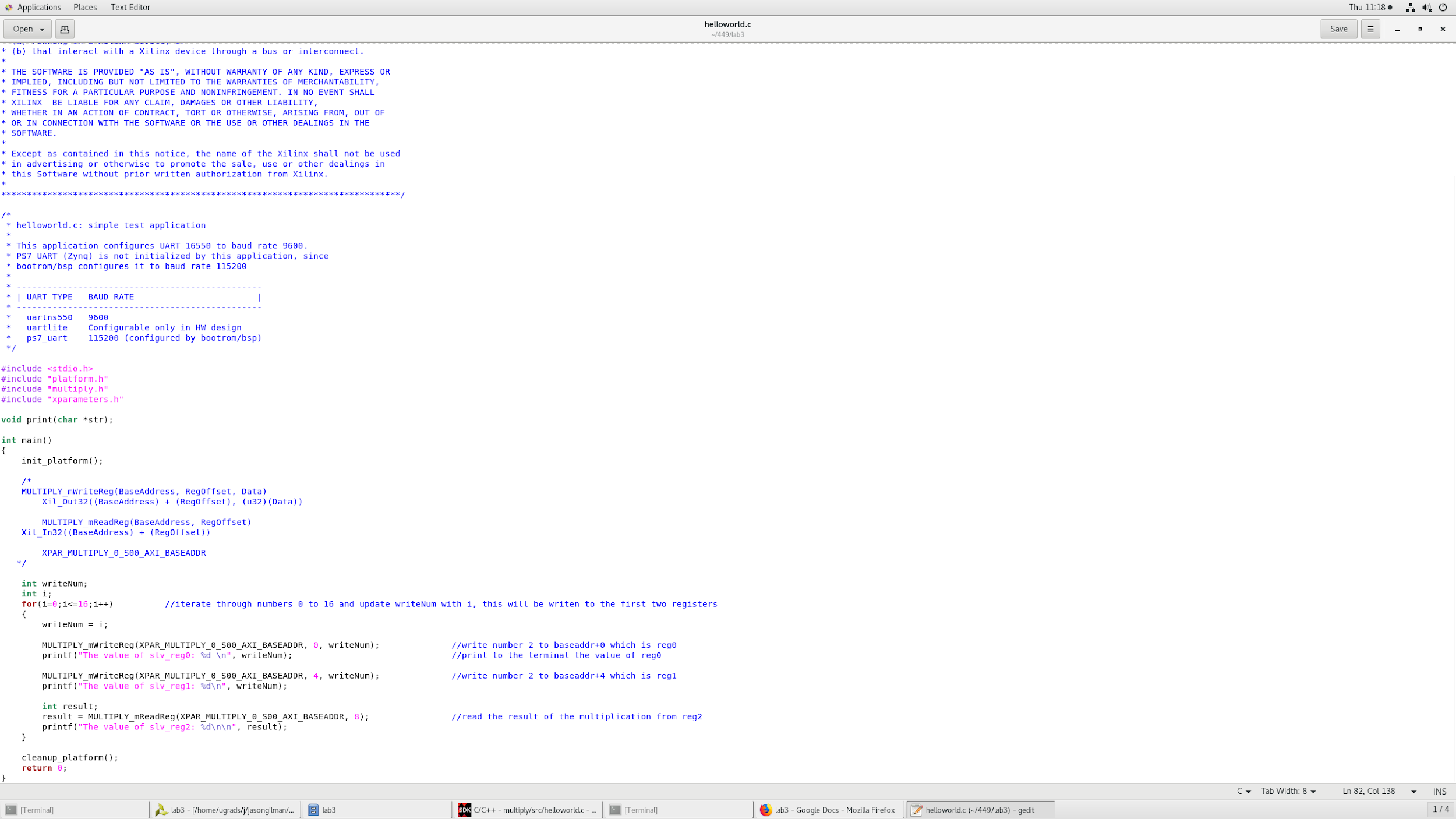
By creating a custom IP and adding it to the Zynq Processing system, we were able to implement hardware that allowed us to multiply two numbers that were stored in registers. The hardware was able to read the registers and compute the multiplication. The software was used to populate the registers with inputs, and receive the output from reg2. The software also allowed us to print the values that the registers contained to the linux terminal, using the printf() command.

**Post Lab Questions:**

1. The tmp\_reg is used to store the the product of the first two registers, then used to store it in reg2. Beyond this, the tmp\_reg allows for extra delay to be added, because the tmp\_reg is assigned on CLK 1, for example, and reg2 is assigned the value of tmp\_reg on CLK2. This facilitates debugging, when necessary, because you can see the sequential order of statements executing, and their outputs, in relation to the clock signal. If you were to remove the tmp\_reg, the product would be calculated on the same clock signal that it is stored in reg2.
2. The only issue I could see happening, is that if the product of the multiplication of reg0 and reg1 is over 32 bits wide. This would cause an overflow error. In changing the peripheral to accommodate this, I would make the registers 64 bits wide instead of 32. In this lab, each register was 32 bits wide, so it the instantiation look something like: [0:31] slv\_reg0; -- instead I would make it: [0:63] slv\_reg0;

**Appendix:**

Helloworld.c (updated with multiplication logic)



multiply\_V1\_0\_AXI.v

