ECEN 449 Lab 2: Using the Software Development Kit (SDK)

02/06/2020

Jason Gilman

**Introduction:**

In this lab, we were tasked with implementing a function using the Software Development Kit, as opposed to using an HDL as in the previous lab. This involved creating a block diagram of the hardware components that will run the code developed.

**Procedure:**

The first part of the lab involved setting up the block diagram that would contain the plans for the hardware that would allow code to run. In this case, we used the MicroBlaze processor. Most of the setup was done, although the input/output needed to be configured to fit our needs. This step involved adding a GPIO block to the diagram. This allowed us to specify what input and output are interacting with our hardware. The GPIO was then connected to the constraint file, .xdc, to determine where the input and outputs would be going and coming from on the connected piece of hardware - the FPGA. An HDL wrapper was then created, which is basically a top level module of the hardware. From here, the SDK was then involved in the project. The first function to be implemented was a simple counter that was displayed on the LEDs. This required mapping the LED pins to the .xdc file as output. The FPGA was then programmed with the c code developed to produce the desired output. The second function implemented was a more complex counter that involved inputs. To allow the hardware to function with inputs, a second GPIO block was added to the hardware. This one was 8 bits wide, with 4 designated to the buttons on the FPGA, and the rest for the switches on the FPGA. The .xdc file was updated to reflect the new inputs and outputs. The c code was programmed to the FPGA such that button0 would increment a global count, button1 would decrement the count, button2 would display the switch values on the LEDs, and button3 would display the count on the LEDs.

**Result/Output:**

The following text is the output of the TCL console as a result of demo two. When the code initially starts, there is a single output message stating ‘START’. Next, depending on the value of the button that is pushed, the console will show what is going to happen. For example, if button0 is pushed, the console will show ‘INCREMENTING COUNT’ and then it will display the updated count value.

START

INCREMENTING COUNT

Value of LEDs = 0x1

INCREMENTING COUNT

Value of LEDs = 0x2

INCREMENTING COUNT

Value of LEDs = 0x3

INCREMENTING COUNT

Value of LEDs = 0x4

DECREMENTING COUNT

Value of LEDs = 0x3

DECREMENTING COUNT

Value of LEDs = 0x2

DISPLAY COUNT

DISPLAY SWITCH STATUS

INCREMENTING COUNT

Value of LEDs = 0x3

INCREMENTING COUNT

Value of LEDs = 0x4

DECREMENTING COUNT

Value of LEDs = 0x3

DISPLAY COUNT

**Conclusion:**

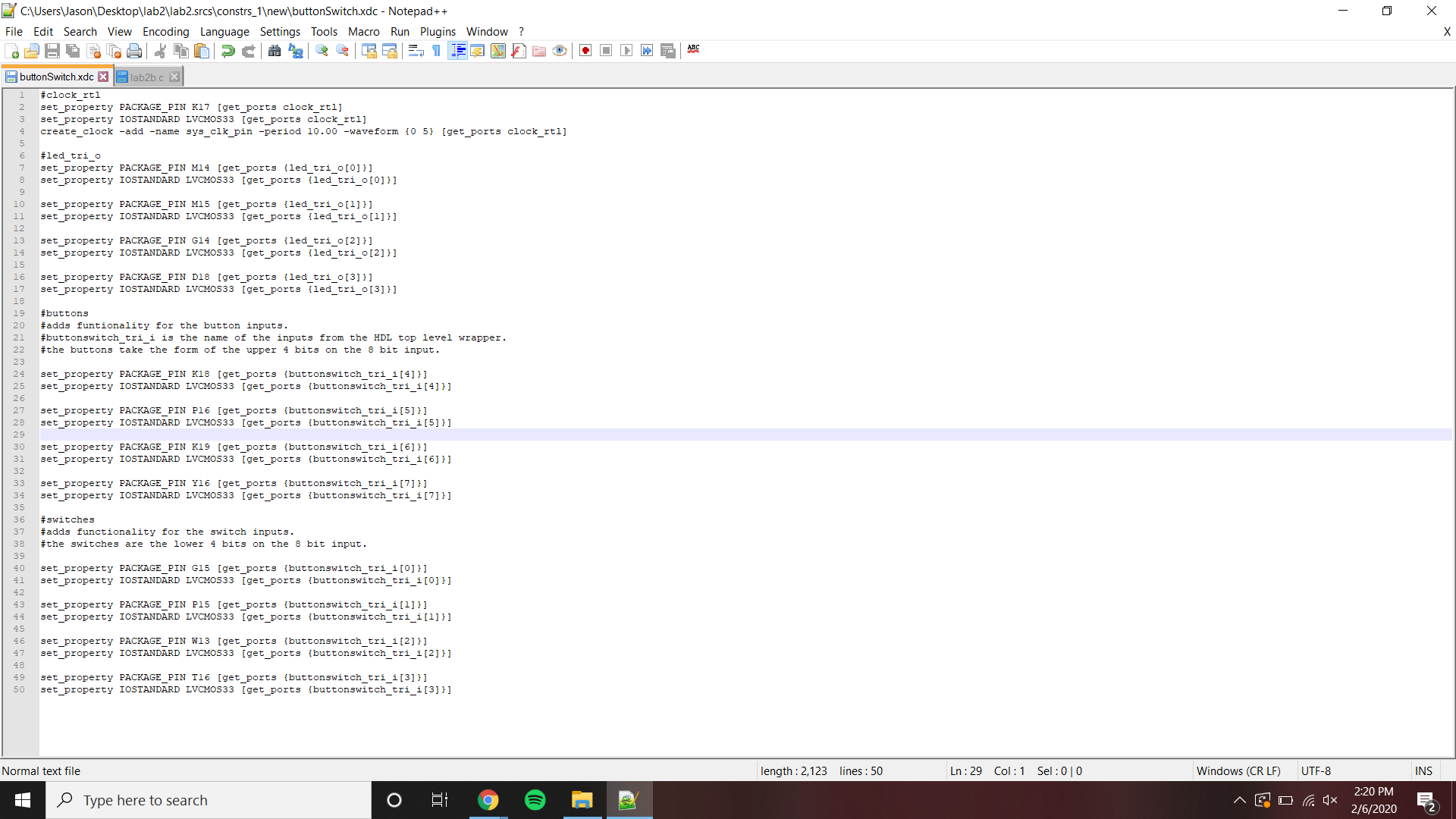
Using the Software Development kit, you are able to create a program that can be then pushed onto a processor. Combining the SDK with vivado, you are able to create a block diagram of a system of hardware, which will then run any function you program onto it. In this case, we used the inputs and outputs of the FPGA in combination with the MicroBlaze system to create the desired output of the function.

**Post Lab Questions:**

1. The count for the second lab was less than the count in the first lab for the clock divider. This is because the first lab required division of the fpga clock directly. The second lab was going through a processor, and the c code that was run took more than 1 clock cycle per iteration. The while loop I used in my code took multiple clock cycles per iteration, thus requiring less division of the ‘fast’ clock.
2. The count variable is declared as volatile, because the variable value may change at any time during the code.
3. This allows the code to run indefinitely, until the code is terminated by the user. The 1 is logically equivalent to true, so while(true), it runs forever, until stopped.
4. I believe that the software implementation was easier. I feel like implementing a function using c code is easier to work with than a hardware implementation. Although when using an HDL you can have multiple lines of code running at the same time, which can make detection of changing i/o easier.

**Appendix:**

buttonSwitch.xdc



Lab2b.c

