Lab 2 Pre-lab Notes

Purpose:

- The purpose of the lab this week is to develop a software alternative to the Verilog solution from last week. You will be using the Vivado Block Design Builder to create a MicroBlaze processor system (an environment where you can run the software you create).
- This is kind of like a microcontroller which you can run software on and have GPIO pins (in this case the switches or LEDs) to communicate with.
- We are programming the FPGA with a Microblaze processor bitstream that you will generate throughout the lab.
- The SDK is used to write your C code that will run on the processor that is synthesized to the FPGA.
- The lab is relatively straightforward, just follow the instructions **carefully**.

Lab Steps and Tips:

- Be sure you do not add a source file like you did in the previous lab. We are not programming Verilog in this lab.
- Make sure you check 'All Outputs' when you see the Re-customize IP window for the AXI GPIO block.
- For part 1, if your console is not outputting any message when the program is running, you need to change all the messages in xil_printf() from single quote to double quote (aka, ' -> ").
- For part 2, its recommended that you start over by creating a new project. This avoids the possible problem of files not being generated properly.
    - The new 8-bit GPIO block should be all inputs.
    - For those of you wondering why the definitions of switches and buttons are not showing up in the xparameters.h: you need to regenerate the bitstream after completing the design & constraints. If you follow the steps like you did in part 1, you'll see the peripheral for switches and buttons will appear.
    - The operating of the switches and buttons will be similar to those being done on LEDs, but you are performing "reading" from GPIO instead of "writing" to GPIO.
    - Make sure you demo part 2 to the TA.

```
void XGpio_SetDataDirection (XGpio *      InstancePtr,
                             unsigned     Channel,
                             u32          DirectionMask
                            )
```

Set the input/output direction of all discrete signals for the specified GPIO channel.

Parameters:

- `InstancePtr` is a pointer to an XGpio instance to be worked on.
- `Channel` contains the channel of the GPIO (1 or 2) to operate on.
- `DirectionMask` is a bitmask specifying which discretes are input and which are output. Bits set to 0 are output and bits set to 1 are input.

Returns: None.

Note: The hardware must be built for dual channels if this function is used with any channel other than 1. If it is not, this function will assert.