How to Give Inputs to FPGAs Remotely

How FPGAs are programmed physically:

FPGAs use incredible amount of programmable logic gates along with clever engineering design to give quick and stable digital processes. The input/output pins (which can be LEDs, 7-segment displays, PMOD headers, switches, buttons, clock signals, etc) can be used to exchange signals with the FGPA internally or externally.

The routing FPGA pin is translated to a name to match the code input/output/other which we will call a port. These ports are expressed in code and are especially useful for anyone who wants to give inputs and receive outputs from your electronics.

The example below is code, demonstrating how a simple On/Off switch works with a single LED

```
module switchExample(SW, LED);

    input [2:0] SW;
    output [2:0] LED;

    assign LED[2:0] = SW[2:0];

endmodule
```

From the code above, 3 input switches and 3 output LED ports are used from the .XDC file. Each input is assigned to an output, when one input is ON (Changed from 0 to 1) the respective output is ON (Changed from 0 to 1). This means that turning on the switch turns on the LED.

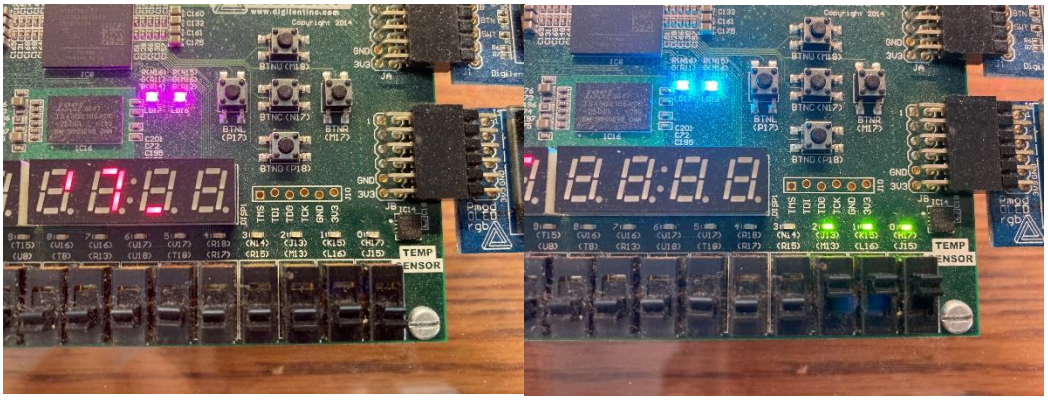

Figure 1: See how the 3 input switches are routed to the 3 output LEDs when the FPGA is programmed

The reason for naming the switches and LEDs to “SW” and “LED” was because it was the default name in the provided constraint files (they may have their name changed though). Below are snippets of code for the constraint .XDC file.

```
##Switches

set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { SW[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { SW[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]

## LEDs

set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { LED[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { LED[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { LED[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { LED[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
```

Figure 2: XDC Descriptive Routing Pins (Blue) and Ports (Red)

As seen to program the FPGA, circled in red the right ports had to be called, so the Verilog code would require the proper names.

However, the FPGA itself does not understand by the port name, instead every time the port name is called it is translated to the "PACKAGE_PIN" numbering instead circled in blue. Looking at the previous FPGA board example, when looking in-between the switches and the LEDs on the actual board, there are some naming conventions written there, the top row represents the LEDs, and the bottom row represents the switches. Every time an input or output is referenced on the board it gets translated from the physical pin to the port name and vice-versa.

How FPGAs will now be programmed remotely:

As the physical buttons can no longer be accessed to test your program, a remote implementation has been added below for the user to use remotely. While the Verilog code will remain the same, the constraints have been modified.

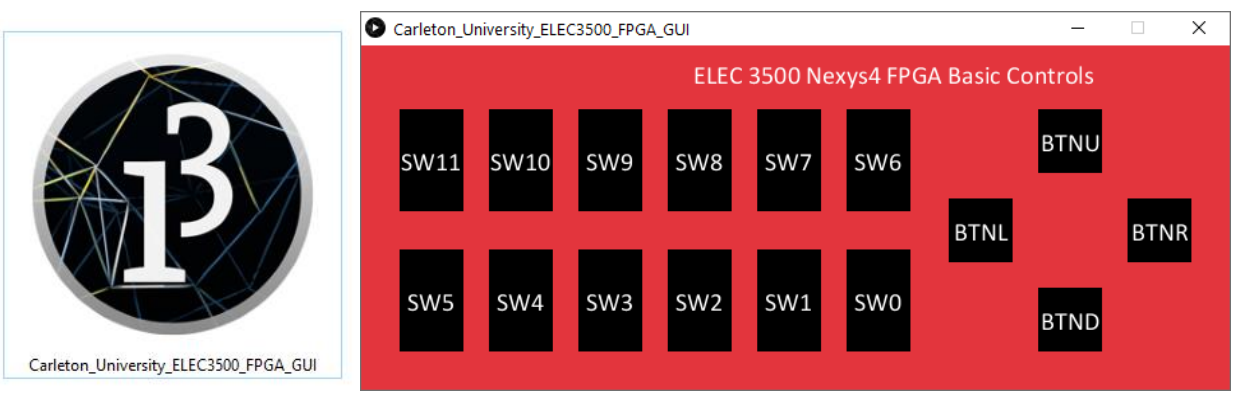

Figure 2: FPGA GUI Application (Left), PMOD Pin Switches and Buttons (Right)

Instead of sending an input signal from the physical switch, the signal is sent from the GUI to a microcontroller then to the FPGA. Below is an image of a microcontroller connected to the FPGA through two sets of several pins. Each set is for a single PMOD interface, where small I/O modules can be connected to the FPGA for extra features, e.g. LCD displays, external sensors, or a Bluetooth interface. However, in this case it would simply be used to take in a single signal and represent a single switch or button input. As two PMOD interfaces are used 12 switches and 4 buttons can be used to work with the FPGA board.

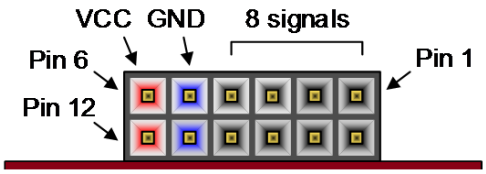

Figure 3: PMOD Pin Map Uses

With this implementation the Verilog code would have to be changed to represent the proper incoming port names. However, a modified constraint file has been provided switching the PACKAGE_PIN naming with the switches and the PMOD headers. This provides the students a seamless programming experience where they do not have to worry about the extra conventions that would be needed programming physically or remotely. Looking at the sample code below shows the swap from the switch's pins, to the PMOD header pins.

```
Previously:
##Switches
#set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { SW[0] }]; #IO_L24N_T3_RS0_15 Sch = SW[0]
#set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch = SW[1]
#set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { SW[2] }]; #IO_L6N_T0_D08_VREF_14 Sch = SW[2]
#set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch = SW[3]
...

##Pmod Header JC
...
#set_property -dict { PACKAGE_PIN H2    IOSTANDARD LVCMOS33 } [get_ports { JD[7] }]; #IO_L15P_T2_DQS_35 Sch = JD[7]
#set_property -dict { PACKAGE_PIN G4    IOSTANDARD LVCMOS33 } [get_ports { JD[8] }]; #IO_L20P_T3_35 Sch = JD[8]
#set_property -dict { PACKAGE_PIN G2    IOSTANDARD LVCMOS33 } [get_ports { JD[9] }]; #IO_L15N_T2_DQS_35 Sch = JD[9]
#set_property -dict { PACKAGE_PIN F3    IOSTANDARD LVCMOS33 } [get_ports { JD[10] }]; #IO_L13N_T2_MRCC_35 Sch = JD[10]
...


Currently:
##Pmod Header Switches JC
set_property -dict { PACKAGE_PIN P18    IOSTANDARD LVCMOS33 } [get_ports { SW[0] }]; #IO_L23N_T3_35 Sch = SW0
set_property -dict { PACKAGE_PIN M17    IOSTANDARD LVCMOS33 } [get_ports { SW[1] }]; #IO_L19N_T3_VREF_35 Sch = SW1
set_property -dict { PACKAGE_PIN P17    IOSTANDARD LVCMOS33 } [get_ports { SW[2] }]; #IO_L22N_T3_35 Sch = SW2
set_property -dict { PACKAGE_PIN M18    IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L19P_T3_35 Sch = SW3
...
```

The Arduino is controlling the PMOD headers JC and JD

Just remember when you are programming remotely, you are not controlling the physical buttons or switches, but rather using the PMOD interfaces that have taken the name of switches and buttons.

FPGA_GUI Mapping to Pins

The JC and JD PMOD headers are already routed in a specific way. If you would like to make a circuit to include the switches and pins from the FPGA GUI, you should use the following descriptive text for your XDC file:

```
##Pmod Header Switches JC

set_property -dict { PACKAGE_PIN P18    IOSTANDARD LVCMOS33 } [get_ports { SW[0] }]; #IO_L23N_T3_35 Sch = SW0
set_property -dict { PACKAGE_PIN M17    IOSTANDARD LVCMOS33 } [get_ports { SW[1] }]; #IO_L19N_T3_VREF_35 Sch = SW1
set_property -dict { PACKAGE_PIN P17    IOSTANDARD LVCMOS33 } [get_ports { SW[2] }]; #IO_L22N_T3_35 Sch = SW2
set_property -dict { PACKAGE_PIN M18    IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L19P_T3_35 Sch = SW3
set_property -dict { PACKAGE_PIN T13    IOSTANDARD LVCMOS33 } [get_ports { SW[4] }]; #IO_L6P_T0_35 Sch = SW4
set_property -dict { PACKAGE_PIN T8     IOSTANDARD LVCMOS33 } [get_ports { SW[5] }]; #IO_L22P_T3_35 Sch = SW5
set_property -dict { PACKAGE_PIN U8     IOSTANDARD LVCMOS33 } [get_ports { SW[6] }]; #IO_L21P_T3_DQS_35 Sch = SW6
set_property -dict { PACKAGE_PIN R16    IOSTANDARD LVCMOS33 } [get_ports { SW[7] }]; #IO_L5P_T0_AD13P_35 Sch = SW7


##Pmod Header Switches and Buttons JD

set_property -dict { PACKAGE_PIN R13    IOSTANDARD LVCMOS33 } [get_ports { SW[8] }]; #IO_L21N_T3_DQS_35 Sch = SW8
set_property -dict { PACKAGE_PIN U18    IOSTANDARD LVCMOS33 } [get_ports { SW[9] }]; #IO_L17P_T2_35 Sch = SW9
set_property -dict { PACKAGE_PIN T18    IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L17N_T2_35 Sch = SW10
set_property -dict { PACKAGE_PIN R17    IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L20N_T3_35 Sch = SW11
set_property -dict { PACKAGE_PIN J15    IOSTANDARD LVCMOS33 } [get_ports { BTNU }]; #IO_L15P_T2_DQS_35 Sch = BTNU
set_property -dict { PACKAGE_PIN L16    IOSTANDARD LVCMOS33 } [get_ports { BTNL }]; #IO_L20P_T3_35 Sch = BTNL
set_property -dict { PACKAGE_PIN M13    IOSTANDARD LVCMOS33 } [get_ports { BTNR }]; #IO_L15N_T2_DQS_35 Sch = BTNR
set_property -dict { PACKAGE_PIN R15    IOSTANDARD LVCMOS33 } [get_ports { BTND }]; #IO_L13N_T2_MRCC_35 Sch = BTND
```

The port name is not as important as the pin address in this case, so feel free to change the names of SW[-] and BTN-

Adding the Constraint File

If you want to find the .xdc file for your project, find the Constraint file through your class' CULearn, or through Nagui's Website in your Course folder, you should be able to find the Constraint folder, and the .xdc file.

To add the Constraint File to your project, choose **Project Manager ▸ Add Sources ▸ Add or create constraints ▸ Add Files ▸ Select the .xdc file ▸ Finish**. You should be able to find and edit your constraint file in the Project Manager's Sources. You can change which pins/ports are being used.

References

[1] Digilent Documentation. "What is a Constraints file". Digilent. https://reference.digilentinc.com/learn/software/tutorials/vivado-xdc-file (accessed Oct 2020)

[2] Nexys A7 Digilent Documentation. "Nexys A7 Reference Manual". Digilent. https://reference.digilentinc.com/reference/programmable-logic/nexys-a7/reference-manual#pmod_ports (accessed Oct 2020)