Learning Objectives:

- Gain experience with Pulse Width modulation and detection
- Gain experience incorporating interrupt handling into an application
- Lay the groundwork for Projects #2 and your final project

Notes:

Revisions to support both the Digilent Nexys A7 (Nexys4 DDR) and the RealDigital Boolean Board.

- The Boolean board only has 4 pushbuttons instead of the 5 buttons on the Nexys A7. We will not use `btnC` on the Nexys A7. Buttons are mapped as follows (`Nexys A7(Boolean)`):
  ```
  {btnC(N/A), btnu(BTN0), btnd(BTN3), btnl(BTN2), btnr(BTN1)}
  ```
- Both FPGA target boards have 2 RGB LEDs but they are numbered differently (`Nexys A7(Boolean)`):
  ```
  {RGB1(RGB0), RGB2(RGB1)}
  ```
- There is no `btnCpuReset` on the Boolean Board. We will implement this on Boolean with:
  ```
  btnCpuReset = ~(BTN0 & BTN1)
  ```
- Both FGPA target boards have 16 LEDs and 16 switches and an 8-digit 7-segment display.

The text in this document uses the Nexys A7 names to be consistent with past releases.

Project Overview

This project builds on the experience you gained in the Getting Started project. Project #1 replaces the MicroBlaze MCS and IOModule we used in our embedded system for the Getting Started project with a MicroBlaze/AXI-based system that includes Xilinx IP, Digilent IP, and ProfRoyK IP including the `rgbPWM` IP used to drive the RGB LED on the target FPGA platform in the Getting Started project. The functionality added to this project uses the Digilent `PWMAnalyzer` IP to detect the duty cycles of the PWM signals driving the RGB1 LED. You will work individually on this project.

Functional Specification

Summary

Functionally, Project #1 is straightforward. The application will write the switches on the target FPGA platform to the LEDs and use the pushbuttons to set the duty cycles for the red, green, and blue segments of RGB1 LED. For Project #1 we will connect the RGB1 PWM inputs (outputs from the `rgbPWM` IP block) to 3 instances of the Digilent PWM Analyzer IP block and display the red, green, and blue duty cycles (0%…99%) on the digits of the 7-segment display. In addition,

the RGB2 LED will be written with the detected red, green, and blue duty cycles; this will provide a visual indication that the PWM Analyzers are correctly capturing the RGB LED1 duty cycles. If all is working correctly the two RGB LEDs should look pretty much the same.

The pushbuttons, switches, LEDs, 7-segment display, and RGB2 LED are controlled with the `Nexys4IO` IP block and driver instead of being handled by the Digilent or RealDigital board files; in fact, you should not use the board files – you can directly select the FPGA on the NexysA7 (`XC7A100T-1CSG324C`) or Boolean Board (`XC7S50-CSGA324`).

Your application will:

- Take inputs from the switches and display them on the 16 green LEDs on the board,
- Take inputs from the pushbuttons to control the duty cycles for the 3 LED segments on the RGB1 LED.
- Measure the duty cycles of the PWM inputs to RGB1 LED and display information about the Red, Green, and Blue PWM duty cycles on the 7-segment display.
- Drive RGB2 LED from the duty cycles detected by the PWM Analyzer

Nexys4IO device connections:

For Project #1 we will be using the available peripherals of the board in the following manner:

Slide Switches:

- The 16-slide switches are written to the 16 green LEDs.

Pushbuttons:

- The `btnCpuReset` button is used to reset the system.
- The `btnR` and `btnU` and `btnD` buttons are used to cycle through the range of duty cycles for the RGB1 LED segments. Pressing and releasing `btnR` cycles the duty cycle of the BLUE segment. Pressing and releasing `btnU` cycles the duty cycle of the GREEN segment. Pressing and releasing `btnD` cycles the duty cycle of the RED segment.
- The `btnL` button resets the duty cycles of all 3 RGB1 LED segments to 0. Press and release `btnL` to reset all three duty cycles.

RGB LEDs:

- RGB1 LED – Connected to red, green, and blue PWM channels from the `rgbPWM` IP module.
- RGB2 LED – Connected to the RGB2 outputs from `Nexys4IO`. The duty cycles for each segment should be written to the same duty cycle(s) as detected by the `PWMAnalyzer` blocks using `Nexys4IO` driver functions.

7-segment display:

- `Digits[7:6]` should display the duty cycle detected for the RED channel of RGB1
- `Digits[5]` is left blank
- `Digits[4:3]` should display the duty cycle detected for the GREEN channel of RGB1
- `Digits[2]` should be left blank
- `Digits[1:0]` should display the duty cycle detected for the BLUE channel of RGB1

- `DP[0]` – (decimal point to the right of Digit[0] blinks on and off. The value is toggled whenever the FIT (Fixed Interval Timer) interrupt handler is entered.
- `DP[7:1]` – no assigned value but you may want to use them for debug

Embedded system configuration

You can create the embedded system the same way you did so for Project #1 by creating the block diagram in the IP Integrator and modifying the target FPGA platform top level and constraints files to accommodate the addition of the Nexys4IO inputs and outputs. Specify the FPGA part number for your target FPGA platform instead of using the board files.

The embedded system should have this minimum configuration. You can add additional hardware marked "(Optional)" as you see fit:

- Microblaze, `mdm`, etc. Configure the Microblaze for 64 kB of local (BRAM) memory. Configure the `mdm` (debug port) to include a UART. Doing this will make it possible to display messages from your application without including a separate `UARTLite` in your embedded system. Enable the AXI bus and interrupts.
- `Nexys4IO`: Add a Nexys4IO IP block to your embedded system. This IP block and driver interfaces to the green LED's, switches, pushbuttons, RGB LEDs, and 7-segment display on the target FPGA platform.
- `AXI Timer/counter`: Add a 32-bit AXI timer. This AXI timer should be initialized in "Generate" mode in the application. Application-wise, the timer should be loaded with a Timer Interval of 1998. (Timing Interval = (1998 + 2) * (10^{-8}) -> 50 KHz/PWM Period = 512) => 98Hz which is fast enough that the LED will not blink)
- `AXI FIT` (Fixed Interval Timer): Add an AXI FIT timer. Set the Number of Clocks to 50_000_000. This setting will cause the FIT interrupt handler to be called about every 0.5 second (2 Hz) which is fast enough to be responsive to pushbutton and switch input changes without consuming an inordinate amount of CPU time. As mentioned in class, it is desirable to limit the functionality of the interrupt handler to process the switches, LEDs, and pushbuttons.
- `AXI_GPIO`: Add a 32-bit, output-only general-purpose IO port. This output port should be connected to the Control Register input to the `rgbPWM` IP block.
- `AXI Interrupt Controller` & `xl_concat` block with number of interrupt sources set to 1 and connected to the FIT timer interrupt output.
- `rgbPWM_r2`. Add a `rgbPWM` IP module to your block design. This is done the same way you did it on the Getting Started project. This `rgbPWM` IP block will be driven by a 10MHz clock from the `clk_wiz blk`. As recommended in the Getting Started project, add a Processor Reset module to synchronize the reset signal to the input clock to avoid a Critical Warning.

  You will need to customize the `rgbPWM` IP block to enable the clock divider and set the max count for the divider to 50. Doing this will result in the RGB1 LED being updated about 97Hz (fast enough to avoid blinking).

- Digilent `PWM_Analyzer_1.0`. Add three (3) instances of the PWM Analyzer. Connect the RGB outputs from `rgbPWM` to these three pulse-width detection detectors. Note that the PWM Analyzer looks for edges on the PWM input. A duty cycle of 0% has no edges.

Refer to the embedded system schematic included in the release for details and to the outline for building the embedded system included in the release.

Deliverables (in checklist format):

- ✓ A live demo to Victoria, Daniel or myself. We will announce lab hours for demos at the Capstone Lab in FAB and, if there is demand, at WCC before or after class. We recommend including a video demo in your deliverables, too.
- ✓ A 3–5-page design report explaining the operation of your design, most notably a description of your application code, problems encountered and how you solved them, and suggestions for how the project could be improved.
- ✓ Source code for your project #1 application. I expect you to base your application on the `Getting Started` and the `test_nexys4IO` code, but there is additional functionality to implement. Please make the application code your own. For example, my name appears in the comments at the top of the program… but this is your program, not mine. Make sure the headers and comments are up to date and that there are no "artifacts" (like commented-out lines) in the code. The readability of your code counts. We will look kindly at code that is organized and easy to follow. We will not look kindly at code we have to slog through to make sense of it.
- ✓ Source code for any HDL that you write or modify. Be sure to include the code for your top-level module since there is a possibility that you will make changes to it. Personalize any code you copy/modify.
- ✓ Constraint file(s) if you made any changes to the provided `.xdc` file(s) or any additional constraint files you added to your project.
- ✓ A schematic for your embedded system. You can generate this from your block design by right- clicking in the diagram pane and selecting *Save as PDF File…*

Grading:

You will be graded on the following:

- (60 pts) The functionality of your demo.
- (20 pts) The quality of your design expressed in your C and SystemVerilog source code. Please add comments to your code to help us understand how it works. The better we understand it the better grade we can give it.
- (20 pts) The quality of your design report. Keep the report concise, but remember, the overarching goal is to help us understand your implementation and what went well, or not so well.

Project Tasks:

1. Complete the tasks in *Getting Started in ECE 544* (Vivado/SDK)

Getting Started in ECE 544 provides a step-by-step guide to building a target hardware

platform with Vivado and running a sample application called `gsproj_app.c` under Vitis. The application writes the switches to the LEDs and uses the pushbuttons to control the intensity of the 3 segments in one of the RGB LEDs on the NexysA7. When you have successfully completed the tasks in the *Getting Started* guide you will have a working knowledge of the design flow for creating embedded systems in Vivado and writing/launching/debugging an embedded application in Vitis.

2. Open Vivado and add the ECE 544 IP repository (Vivado)

You will use several IP blocks during the term. Vivado needs to know the location of the repository containing user-provided IP available to a Block design. Repositories can be added at the project level or at a default level in Vivado which will make the IP in the repository available whenever a new Vivado project is created. Placing the repository at the default (tool) level will only work when a new project is created. Start Vivado and select *Tools/Settings/IP Defaults* from the *Quick Start* screen. Click **+** in the IP Catalog dialog and add an entry with the path to `ece544ip_w25` folder on your PC. Click *OK*. This only needs to be done once.

3. Create the Project #1 embedded system and generate the output products

The functional specification in this write-up specifies the requirements for the embedded system for this project. There are many ways to create the block design and I encourage you to experiment. I have included an outline of the steps I followed in the release.

4. Generate a bitstream that includes the embedded system for the project (Vivado)

Synthesize, Implement, Associate ELF files, Generate the bitstream and export the hardware and bitstream. These are the same steps you took for the Getting Started project.

5. Execute the `test_nexys4io` application (Vitis)

Export the hardware and bitstream from Vivado and open Vitis. Create a platform project, a system project, and an application. Debug/execute the test application. The test application is interrupt-based, like Project #1 and exercises the devices managed by Nexys4IO (switches, pushbuttons, RGB LEDs, LEDs, and 7-segment display.

6. Implement the new application (Vitis)

The `test_nexys4io.c and gsproj_app` applications can be used as starter code, but you are going to have to develop new code on your own. Create and execute a new application that meets the project specification in this write-up.

7. Schedule a live demo for your project. We also recommend including a video of your working application with your deliverables.

Once you have your Project #1 hardware and application working to your satisfaction you will demonstrate it to Victoria, Daniel, or me. We will publish a demo schedule a few days before the due date for the project

8. Upload your code and report (including your “embsys" schematic) to Canvas.

Bundle your deliverables into a single `.zip` (or `.rar` or `.tgz` or `.7zip`) file and

upload it to your Project #1 dropbox. You may submit more than once – we will grade your latest submission so make sure that each upload is complete.

References:

1. *Digilent Nexys A7 Reference Manual* and schematics. Copyright Digilent, Inc.
2. *RealDigital Boolean Board Reference manual and schematics. Copyright RealDigital*
3. *Getting Started in ECE 544 (Vivado/Nexys A7)* by Roy Kravitz.
4. *Outline for Creating your Project #1 embedded system* by Roy Kravitz.
5. *Previous documentation* by Roy Kravitz.