Learning Objectives:

- Gain experience w/ the Vivado and the Nexys4 DDR Board
- Gain experience using IP Integrator to build embedded systems
- Gain experience doing C programming for embedded apps with Xilinx SDK
- Lay the groundwork for Project 2

PROJECT 1

Project 1 demo due on or before Wednesday-27-Jan-2016
Project 1 Deliverables due Friday 29-Jan-2016

Project: Pulse Width Modulation and Detection

This project is designed to give you experience generating an embedded system application using the Xilinx Vivado and SDK embedded system tools. The project will also add two useful functions to your embedded system arsenal – pulse width modulation and pulse width detection. As you will find out pulse-width detection (PWDET) is often used in applications that involve measuring time and frequency. Pulse width modulation (PWM) is used in a number of applications including servo control (like those used in R/C aircraft and cars). PWM can also provide analog voltages with proper filtering.

You will be provided with a step-by-step guide for building a target hardware platform for the project, an API (Application Program Interface) for using a Xilinx `axi_timer` in PWM mode and a sample application that can be the basis for your Project 1 application. You will be provided with custom peripheral and drivers (*`Nexys4IO`*) which makes the pushbuttons, switches and LEDs on the Digilent Nexys4 DDR™ board available to your application. You will augment the I/O devices on the Nexys4 DDR with two additional devices. You will use a PmodENC to add a rotary encoder and additional pushbutton to your application and a PmodCLP to provide character output. These additional devices are controlled by a second piece of IP, the *`Pmod544IO`* custom peripheral. Your job is to build a working hardware platform and application that meets the functional specification for the project. You will work individually on this Project.

Functional Specification

From a functionality point of view Project 1 is straightforward. You will generate a variable frequency, variable duty-cycle PWM output signal using a Xilinx `axi_timer` and you will detect the duty cycle of that PWM output by hardware you design and by implementing a pulse-width detection algorithm in your application software. There is value in providing both a hardware and software solution. A hardware pulse width detector can sense much higher frequencies than a pure software implementation but requires additional hardware and a mechanism for interfacing the pulse-width detector to a CPU. A software pulse width detector is simple to implement but cannot detect high frequency pulses.

Even though the PWM and PWDET functions are implemented as part of a single embedded system and application, the two functions operate independently. That is, the only connection between the two discrete functions is by wiring the PWM output to the input of the PWDET circuitry in your top level Verilog module (n4fpga.v in the reference design).

The slide switches, pushbuttons and rotary encoder are used as follows:

- Slide Switches[15:4] have no assigned functionality for this project

- Slide Switch[3] – This slide switch on the Nexys4 DDR selects between software and hardware pulse width detection. Hardware pulse width detection should be enabled when the switch is on (up) and software pulse width detection should be enabled when the switch is off (down)

- Slide Switches[2:0] – The three rightmost slide switches on the Nexys4 DDR are used to control the frequency of the PWM output as follows:
  - 000 – 100Hz 100 – 50000 (50KHz)
  - 001 – 1000Hz 101 – 100000 (100KHz)
  - 010 – 5000Hz (5KHz) 110 – 200000 (200KHz)
  - 011 – 10000Hz (10KHz) 111 – 500000 (500KHz)

  …or you can adjust the frequency selections to show off the speed of your HW implementation.

- Rotary Encoder – The rotary encoder knob on the PmodENC is used to alter the duty cycle (high time vs. low time) of the PWM output. Twisting the Rotary encoder knob to the right should increase the duty cycle (output is high for longer). Twisting the knob to the left should decrease the duty cycle (output is high for shorter). The minimum duty cycle should be 1%. The maximum duty cycle should be 99%.

- Pushbuttons – The pushbuttons have no assigned functionality for this project other than restarting the system. System reset is done by pressing the CPU Reset button (the button is red) on the Nexys4 DDR. This button is mapped to the sysreset_n (asserted low) and sysreset (asserted high) signals in the top level. Your hardware pulse-width detection and other modules you create may use either signal. The sample application uses the Rotary Encoder pushbutton (press the knob instead of turn it) to terminate the application. The same could be done in your program but this is not required. Typically an embedded application does not terminate, but instead runs in an infinite loop reading and processing inputs.

The character display on the PmodCLP should be used as follows:

- Top line (line 1) of display should show the PWM duty cycle and frequency selected by the switches and rotary encoder knob.

- Bottom line (line 2) of display should show the duty cycle and frequency detected by the PWDET logic.

In the ideal case the bottom line and the top line will be the same because the PWDET logic is accurately detecting the PWM output. A hardware implementation of PWDET should yield the ideal case but PWDET done in software may not be able to keep up at the higher frequencies.

The LEDs on the Nexys4 DDR board should be used as follows:

- LED[15] – Is used to provide a visual indication of the PWM output. The higher the duty cycle the brighter the LED. This functionality is implemented in n4fpga.v and needs no other support.
- LED[14:0] – These LEDS have no dedicated functionality for this project but you can use them to enhance your application. For example, my Project 1 solution implements a simple bar graph to indicate the duty cycle.
- RGB1 and RGB2 – The tri-color LEDs have no dedicated function for the project but RGB1 is used in the starter app (testpwm.c) to implement a pulse-width indicator in software. Since the indicator is done in software it may not work correctly at higher PWM frequencies.

The Seven Segment display on the Nexys4 DDR should be used as follows:

- Digits[7:4] (left bank) have no defined functionality for the project and are free for your application to use.
- Digits[3:0] should display the rotary encoder count in hex. This functionality is implemented in testpwm.c and can be retained in your application.

Deliverables (in checklist format)

- ☐ A demonstration of your project to the instructor or T/A.
- ☐ A 3-5 page design report explaining the operation of your design, most notably your hardware and software PWDET implementation.
- ☐ Source code for your C application. I expect you to base your application on testpwm.c but make it 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, that there are no debugging traces (like commented out lines) left, etc.
- ☐ Source code for any Verilog or VHDL that you write or modify. Be sure to include the Verilog for your hardware pulse width-detection solution including the changes you made to n4fpga.v. Personalize any code you copy/modify.
- ☐ 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:

- The functionality of your demo. (60 pts)
- The quality of your design expressed in your C and Verilog source code. Please comment 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 where you explain your design. (20 pts)

Hardware

The projects for this course will be implemented in the Artix Series 7 FPGA on a Digilent Nexys4 DDR board.

The rotary encoder functionality is provided by a Digilent PmodENC. The `n4fpga.v` file included in the reference design assigns the PmodENC pins to bottom row of the JD header on the board. If you find that the rotary encoder seems to be turning the "wrong" way (that is, turning to the right decreases a value) swap the JD[] inputs to *rotary_a* and *rotary_b*. The specification states that turning the knob to the right increases the duty cycle; you will lose points if this doesn't work correctly in your demonstration.

The display functionality is provided by a Digilent PmodCLP. The `n4fpga.v` file included the reference design assigns the PmodCLP databus to both rows of JA the PmodCLP control to the bottom row of JB on the Nexys4 DDR board. This can be changed in your design by adjusting the output port connections in n4fpga.v to match your configuration. Be sure that you also make necessary changes are made to the constraints file n4fpga.xdc if there are any.

NOTE: THE PROJECT RELEASE INCLUDES TWO CONSTRAINTS FILE. `N4FPGA.XDC` WORKS WITH THE ORIGINAL NEXYS4 BOARD. `N4DDRFPGA.XDC` WORKS WITH THE NEXYS4 DDR BOARD. BE SURE TO INCLUDE ONLY THE FILE THAT MATCHES YOUR BOARD. N4FPGA.V WORKS WITH EITHER BOARD BECAUSE THE PORT TO PHYSICAL PIN MAPPING IS DONE IN THE CONSTRAINTS FILE.

Project Tasks

1. Download the project 1 release package

Download the release package from the Projects section of the course website. This zip file contains a step-by-step "Getting Started" guide, several application examples and a repository (a .zip file) that provides the ECE544 IP (Nexys4IO and Pmod544IOR2).

2. 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 runing a sample application called *testpwm.c* on the SDK. *testpwm.c* demonstrates the *pwm_timer* API and the drivers for the custom peripherals. It is important to complete Task 0 before diving into the step-by-step instructions…that is, take the time to become familiar with Vivado and the IP Integrator. When you have successfully completed the tasks in the *Getting Started* guide you will have both a hardware platform and a software application that can be used as the basis for your project.

3. Modify the hardware platform to implement Project 1 (Vivado)

The functional specification for this project specifies that the application must do pulse width detection in both hardware and software. Doing this will require changes to both the hardware platform and software application.

- Software approach – in the software based approach the application needs to be able to read the level of the PWM output. The `n4fpga.v` from the getting started project provides two 8-bit general purpose I/O channels. `n4fpga.v` wraps the pwm output (pwmO) back into GPIO Channel 1 bit[0] to make it available to your application .
- Hardware approach – The hardware based pulse-width detection solution should be able to track much higher frequencies. To implement a hardware-based approach you need to develop pulse-width detection logic in Verilog or VHDL which takes the PWM output from the

`axi_timer` as an input and counts the high and low times (or the high time and the period). The circuit should indicate to the application that the new counts are available and should make those counts available to your application. A way to implement this is to create an external interrupt to the Microblaze and make the counts available through a 2nd GPIO instantiation (two 32-bit input ports). Doing this will require modifications to both `n4fpga.v` and the embedded system. Instead of an interrupt you could also poll the hardware pulse-width detector registers from your application.

IMPORTANT: THE BLOCK DESIGN FOR THE *GETTING STARTED* PROJECT INCLUDES THE CLOCK GENERATOR (CLK_WIZ_1) WHICH SYNTHESIZES THE CLOCKS. IF YOU PUT YOUR PULSE-WIDTH DETECTION LOGIC AT THE TOP LEVEL (E.G. IN N4GPGA.V) YOU CANNOT SIMPLY CONNECT IT TO SYSCLK. INSTEAD YOU NEED TO ADD A 2ND CLOCK OUTPUT FROM THE CLOCK GENERATOR AND BRING THAT OUT OF YOUR BLOCK DESIGN TO THE TOP LEVEL.

4. Implement the new application (SDK)

The `testpwm.c` application implements the PWM control as defined by the functional specification and can be modified to meet the spec for this project so it is worth reviewing the code and gaining and understanding of how it works.

- Software approach - Your application will need to sample the input at a much faster rate (preferably 5x) than the maximum PWM frequency and then count high time and low time. Once you have high and low time counts you can calculate the duty cycle and frequency. You may find it easier to implement the pulse-width detection by adding a 2nd timer counter peripheral to the embedded system and use it to count the times instead of counting purely in software.

- Hardware approach - Your application would have to change to respond to the interrupt from the PWDET hardware and read the counts, or to sample the registers and calculate the duty cycle and frequency as it changes.

5. Download your design to the board and debug/test it (SDK)

Verify that the hardware platform and application work according to the specification.

6. Demonstrate your project and submit the deliverables

Once you have your project working be prepared to demonstrate it to the instructors or TA. The demonstration can be given before the due date if you are ready.

References

[1] *Digilent Nexys4 DDR™ Board Reference Manual*. Copyright Digilent, Inc.

[2] *Digilent PmodCLP™ Parallel LCD Reference Manual*. Copyright Digilent, Inc.

[3] *Digilent PmodENC™ Reference Manual*. Copyright Digilent, Inc.

[4] *Samsung KS0066 Data sheet*. Copyright Samsung, Inc.

[5] *Nexys4IO and Pmod544IO Product Guides* by Roy Kravitz, 2014

[6] *Nexys4IO and Pmod544IO Driver User Guides by* Roy Kravitz, 2014

[7] *Getting Started in ECE 544 (Vivado/Nexys4)* by Roy Kravitz

[8] PWM timer/counter source code (`pwm_tmrctr.h` and `pwm_tmrctr.c`)