Learning Objectives:

- Gain experience with PID control and driver development
- Gain experience with FreeRTOS
- Lay the groundwork for 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 is designed to give you practical experience with a common, and often crucial, embedded system function – closed loop control. Closed-loop control involves using feedback from one or more sensors to adjust the input parameters that control the output of the circuit.

In this project you will gain practical experience working with FreeRTOS and interfacing to an external sensor. Project #2 uses a MicroBlaze AXI based system which includes an AXI I2C IP. The AXI I2C module is used to get a Lux measurement (The light intensity falling on a surface) from a TSL2561 sensor.

Project #2 is a team-based project (teams of 2). You may self-enroll in teams. One team member should pick an unassigned group, assign themselves to it, and let their partner know which group to join.

Functional Specification

Summary

Functionally, Project #2 implements firmware using FreeRTOS on a Microblaze-based system to

get the lux measurement from a TSL2561 sensor. The lux reported by the sensor should match the target lux given to the application by the user. A closed-loop PID controller responds to varying light levels by adjusting the PWM duty cycle to the LED to maintain the target light level.

You will be creating a TSL2561driver to support the I2C interface to the TSL2561 Luminosity sensor and then build an application that implements a PID controller for the LED. The user will adjust the luminosity setpoint and tune the control circuit by using the buttons and switches to increase or decrease the setpoint and PID control constants (Kp, Ki, and Kd). The switches will be used to select which parameter is being modified, and the buttons will be used to increase/decrease the values.

In order to evaluate how well the PID control circuit can maintain the desired luminosity, you will intentionally introduce disturbances into the environment. This control system can be challenged by increasing or obstructing light. Introducing an obstruction in front of the light sensor can test how quickly and accurately the controller can adjust the LED output to compensate for the reduced light. Increasing the light in the sensor's field of view can test how well the controller can decrease output.

These disturbances can be introduced using items you may already have around your home. Parchment paper/transparency paper can be placed between the sensor and the LED to create an obstruction. A flashlight or other light source directed at the sensor can be used to increase the lighting.

You may need to experiment a bit to find the ideal configuration for your sensor and LED.

Additional Materials

- TSL2561 Luminosity Sensor (https://www.amazon.com/dp/B01HTC5OYI?ref=ppx_yo2ov_dt_b_fed_asin_title)
- White LED
- Resistor / Breadboard / Jumper Wires
- Box or other enclosure to isolate circuit from ambient room lighting
- Material to obstruct LED from sensor
    - Ex: parchment paper, tissue paper, tracing paper, etc.

Nexys4IO device connections:

For Project #2 we will be using the available I/O peripherals of the board in the following manner:

Slide Switches:

- Switches [15:8]
    - Not assigned
- Switches[7:6]
    - Used to select which of the 3 PID control constants to change.
        - 01: If Switch[6] is closed, the buttons will increment and

decrement the Kp value
        - 10: If Switch[7] is closed and Switch[6] is open, the buttons will increment and decrement the Ki value
        - 11: If both switches are closed, the buttons will increment and decrement the Kd value
- Switches[5:4]
    - Setpoint and control constant multiplier. Controls the amount the selected setpoint or control constant is incremented/decremented with each press of BtnU or BtnD buttons. You may choose the base increment that best suits your application (ex: ± 1, ± 0.1, etc)
        - 00: If both switches are open, change the setpoint/Kp/Ki/Kd value by ± 1 increment on each button press
        - 01: If Switch[4] is closed, change the setpoint/Kp/Ki/Kd value by ± 5 increments on each button press
        - 1x: If Switch[5] is closed, regardless of Switch[0], change the setpoint/Kp/Ki/Kd value by ± 10 increments on each button press
- Switch[3] is used to select a change in setpoint
    - 1: If switch[3] is closed, the buttons will increment and decrement the setpoint
    - 0: If switch[3] is open, no change in setpoint will occur on button press
- Switch [2:0] are used to enable and disable the PID control constants so you can experiment with different control algorithms
    - Switch[2] Derivative Control
        - 0 Disable D control
        - 1 Enable D control
    - Switch[1] Integral Control
        - 0: Disable I Control
        - 1: Enable I Control
    - Switch[0] Proportional Control
        - 0: Disable P control
        - 1: Enable P control

Pushbuttons:

- BtnU – Increment the selected parameter (setpoint/Kp/Ki/Kd)
- BtnD – Decrement the selected parameter (setpoint/Kp/Ki/Kd)
- You may choose to include the other buttons in your design, but it is not required

Pushbuttons are connected to a 16-bit GPIO input port in the top-level module. The buttons are connected in the following order: {11'h000, btnC, btnU, btnD, btnL, btnR}.

7-segment display:

- `Digits[7:5]` should display the setpoint
- `Digits[4]` is left blank

- `Digits[3:1]` should display the current lux as read from the TSL2561 sensor
- `Digits[0]` is left blank

Embedded system configuration

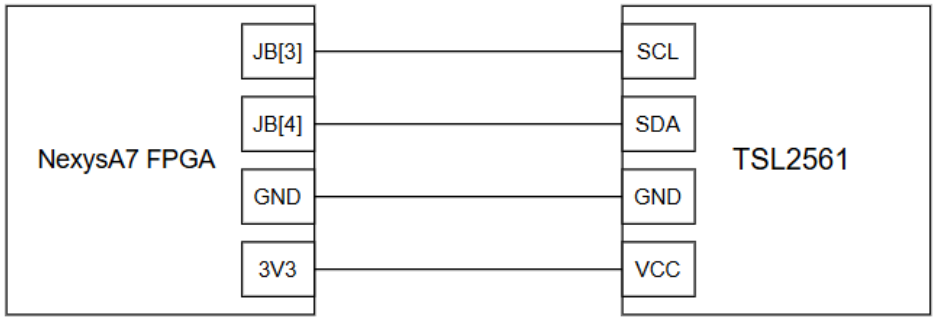
The project release contains .xsa and .bit files that can be used to jumpstart your design process. You are not required to use the provided files, and may opt to create your own embedded system. The embedded system that was included in the project release has the following specifications:

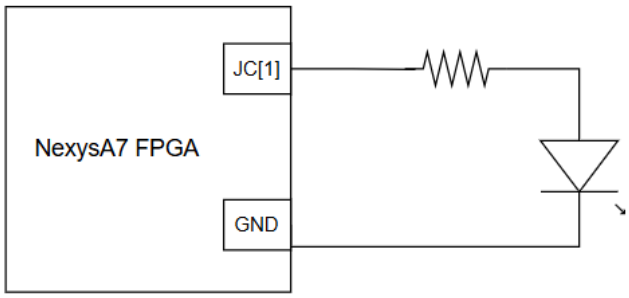
- **`Microblaze`**: Microblaze is configured for 128 kB of local (BRAM) memory. The `mdm` (debug port) includes UART. AXI bus and interrupts have been enabled.
- **`Nexys4IO`**: Make signals external, but note that the signals are configured differently than previous projects. The RGB1 Blue output is being used to drive the PWM signal to the external LED, instead of the onboard RGB1 LED. This connection is made in the top-level module. The other RGB signals have been deleted, as this project does not require use of the RGB LEDs
- **`AXI GPIO`**: configured with two ports (input only). Port one is for the pushbuttons, and the second port is for the switches. This AXI GPIO is configured with interrupts and added as a source to the interrupt controller. The FreeRTOS sample program and the suggested tasking model are based on the buttons and/or switches being interrupt-driven.
  - The buttons and switches are also connected as inputs to Nexys4IO, but Nexys4IO does not support the interrupt driven inputs that the example program/tasking model require
- **`AXI Timer/counter`**: Used to generate the "systick" for FreeRTOS
  - Configured:
    - 32-bit, both timers enabled
- **`AXI Interrupt Controller & xl_concat`** block connected to the timer interrupt output and GPIO interrupt.
- **`bidir module`** (2 instances)- I/O buffers for I2C bidirectional buses.
  - Connected as follows:
    - Bidirec_0/Outp to axi_iic/scl_i
    - Bidirec_0/oe to axi_iic/scl_t
    - Bidirec_0/inp to axi_iic/scl_o
    - Bidirec_1/ Outp to axi_iic/sda_i
    - Bidirec_1/oe to axi_iic/sda_t
    - Bidirec_1/inp to axi_iic/sda_o
  - After connecting, the "bidir" ports were made external
- **`AXI IIC –`** Used to interface the TSL2561 sensor.
  - Configuration**:**
    - SCL Clock Freq (in Khz) = 100
    - Address Mode = 7 bit
    - SCL/SDA Inertial Delay = 0
    - Active State of SDA = 1
    - General Purpose Output Width = 1
    - Default GPIO Output Value = 0x00
- **`AXI Uart Lite:`** We will be using an external serial terminal to collect data sent to a PC using UART. Data sent over UART will be used to create graphs showing system response.
  - Configuration:

- AXI CLK Freq = 100.0, Baud Rate = 115200, Data Bits = 8, No Parity

Hardware Setup (LED and TSL2561 Luminosity Sensor)

Your system will interface with two external components: a white LED and the TSL2561 luminosity sensor. The Nexys A7 board is equipped with four general purpose PMOD connectors that we will utilize to connect these components. The white LED can be connected to logic and GND pins in the PMOD JC port. The TSL2561 sensor can be connected to logic, VCC, and GND pins on the PMOD JB port.





If you would like to change the connections to the Nexys A7 board you will need to modify the top level module and constraints files accordingly.

Creating the TSL2561 Driver

Review the TSL2561 datasheet to understand what registers need to be referenced and how to structure your driver code. We recommend that you create two files: `tsl2561.c` and `tsl2561.h`. Include your driver files in the `src` directory of your application project.

`tsl2561.c` should contain the code for the functions described below:

- `void tsl2561_init(XIic *i2c);`
- `uint16_t tsl2561_readChannel(XIic *i2c, tsl2561_channel_t channel);`
- `float tsl2561_calculateLux(uint16_t ch0, uint16_t ch1);`

Function Name	Function Description
`tsl2561_init`	Power on the sensor and read the ID register to verify sensor presence. Configure the timing register, you may use default values, or experiment with what works best for you
`tsl2561_readChannel`	Issue command to access the control register and reads the provided channel number
`tsl2561_calculateLux`	Calculate the lux using coefficients and equations from data sheet

Review the data sheet application information section for samples of how to implement this functionality. You are welcome to add any additional functions as you see fit.

`tsl2561.h` should contain any structure declarations, constants, and function prototypes used in `tsl2561.c`

Firmware setup in Vitis:

- While creating the Platform Project in Vitis, select FreeRTOS instead of the Standalone OS.
- You will likely need to configure FreeRTOS to use a smaller heap. If you need to make any adjustments to the FreeRTOS settings (heap size, stack size, etc) navigate to:
    - Board Support Package -> Modify BSP Settings -> Overview -> freeRTOS10_xilinx
- The project release contains an example FreeRTOS application. The application uses two tasks, a queue, and a semaphore to blink the LEDs using a GPIO interrupt.

The project release includes a possible tasking model for your application. You can implement this model or create one of your own. Be sure to document and describe your model in your project report

Deliverables (in checklist format):

- ✔ A live demo to Victoria, Daniel or Roy. 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 your control algorithm and application. Please include a few graphs showing the results from your control algorithm. List the contribution of each team member. Describe and challenges you faced and how they were overcome. Be sure to note any sources used.
- ✔ Source code for your project #2 application. Please make the application code your own. 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.

Extra Credit Opportunities:

- Add watchdog timer functionality to your project.
    - Add AXI Timebase Watchdog Timer peripheral to your embedded system. You can configure the timer interval to about 2 or 3 seconds.
    - Application should include some kind of recovery action

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. *TSl2561 Datasheet*