Learning Objectives:

- Gain experience with microprocessor-based closed loop control
- Gain experience creating custom peripherals and drivers
- Apply the concepts learned in Project 1
- Flex your hardware-building muscles w/ "real" circuits
- Control small scale electric machinery with "Proportional" control

Project 2

Project 2 demos will be conducted on Thu 18-May-2017. Project 2 Deliverables are due to D2L by 10:00 PM on Sat, 20-May-2017. This project will be done in teams of two.

Project: Microprocessor-based Closed-loop Motor Control

This project is designed to give you practical experience with a common, and often critical, 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. An example of a control system is automotive cruise control. The driver sets a target speed, engages the cruise control and takes his or her foot off the accelerator. The cruise control does its best to keep the car moving at the target speed (called the *setpoint*) even as the car goes up and down hills. The control circuit for this project is much simpler than cruise control but the same general principles apply.

You will be provided with a schematic and Bill of Materials (BOM) for a simple motor circuit. You will mount the motor on a board of your own choosing, leaving enough mechanical space to allow for a second motor (the second motor will be used in Project #3). You will also mount a Hall sensor on the same board as the motor(s) to count the number of times a small magnet mounted to the motor shaft passes by the sensor when the motor is turning (a device that measures motor speed is called a tachometer). A Digilent PmodHB3 (2A H- bridge circuit) will be used to drive the motor and drive the output of the Hall sensor back to the Microblaze via one of the PMOD connectors on the Nexys4 DDR.

The control circuit will make use of PWM generation functionality from Project 1 to drive the PmodHB3 and subsequently control the speed of the motor. The PWM circuit, an additional output to set the direction of motor rotation and the tachometer will be combined (by you) into a custom peripheral that you will create/package with the Vivado Create/Package IP wizard. The configuration of a single magnet and single Hall sensor can be used to detect the speed of the motor but not the rotation direction. Determining which way the motor is turning would require a second magnet and Hall sensor (which would be a nifty addition for extra credit if you can get it working).

You can purchase the components you need for this project from the LID (Laboratory for Interconnected Devices), FAB 84-20 (Chris Clark). The motor mounting board and possible proto-board should be your own design. The motor needs a solid mount because it will speed up and slow and you don't want it coming loose. Don't forget to leave room for the second motor.

For Project #2 you will implement a "user-friendly" closed loop control platform to experiment with different parameters for microprocessor-based control of an electric motor. There are several pieces to this project:

- Build the motor mount and the circuit to drive the motor and detect the rotation speed
- Create/package a custom IP block that provides the interface between the PmodHB3 and a Microblaze-based embedded system
- Write an application to select and maintain a motor speed using closed loop control
- Tune the control loop to optimize the response of your circuit

The inputs and outputs of the PmodHB3 control circuit (including power and ground) are brought to/from the Nexys4 DDR though one of the PMOD expansion connectors. As with Project 1, this assignment makes use of the PmodEnc (Rotary Encoder, switch, and pushbutton) and some of the LEDs, buttons, and switches on the Nexys4 DDR. The PmodOLEDrgb is optional.

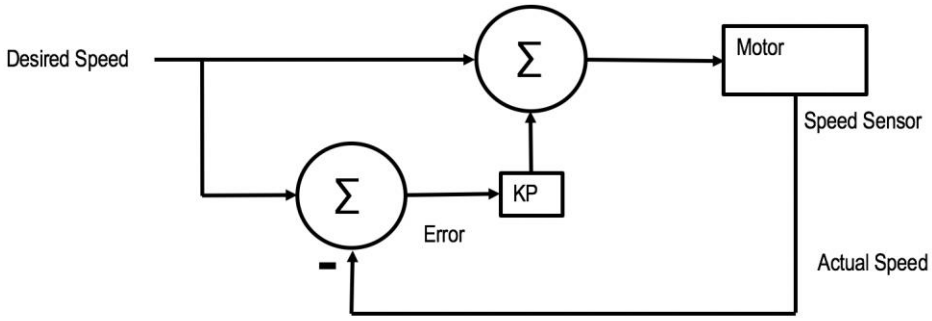
Time is short (less than 2 weeks) so it's best to get started right away.

Application

You will develop an application (C program) that demonstrates closed loop control using your custom IP block. The application will send data to a PC using a UARTLite serial interface. You are interested in capturing, and transmitting, the values of the motor control signal(s), the motor speed, and the "error", and "KP" of the control loop. You can upload and store the data in a text file on the PC using a terminal emulator such as PuTTY and export it to your favorite graphing tool to produce interesting graphs. The application will also display information about the system on the Seven Segment display and the LEDs on the Nexys4 DDR.

You will implement the P portion ("proportion") of a PID controller with a constant KP to provide closed loop control of the motor. The user can tune the control circuit by using the top and bottom buttons to slowly increase (top button) or decrease, (bottom button) KP. Consider using fixed point arithmetic or floating point to provide one or two decimal points of precision. For example, say you want to vary KP from 0 to 100. Instead of incrementing by 1 between 0 and 100, increment between 0 and 1000. Divide the count by 10 and you have an implied decimal point – this is fixed point arithmetic. If adding a single decimal point to KP does not give you fine enough resolution to tune your control loop than consider implementing two decimal points by counting between 0 and 10,000 and dividing by 100.

The desired control signal should go directly to the H-bridge driving the motor. An "error" signal is generated if there is a difference between the desired and actual speed; that "error" signal is multiplied by KP to adjust the PWM value to the PmodHB3. Use floating point to ensure the best results.



An important design decision is how to represent the motor speed or rotational velocity (both desired and actual) and the proportional control constant, KP. The motor speed will be determined by an 8-bit unsigned (0-255) quantity because we are implementing 8-bit PWM. That 8-bit PWM value should scale over the entire motor speed range as much as possible. Also, you will need convert the Hall sensor pulses into a comparable speed measure that corresponds to the PWM value. For example, if the maximum motor speed is 6000 RPM (PWM = 255) then the half-speed (PWM = 127) should be 3000 RPM.

The Proportion constant, KP, is also represented by an 8-bit quantity. What happens when you multiply the error signal (which is the difference between desired and actual speed) by KP? Multiplying two 8-bit numbers gives you a 16-bit number. To convert back to 8-bit, you have to decide how many bits to throw out to get a resulting signal that is appropriate.

Since the "error" signal can be positive (motor is turning slower than the set point) or negative (motor is turning faster than the set point) it is possible for the sum of the desired speed and (KP X "error") to overflow or underflow. Since the command to the motor can only be positive (i.e. you can't set PWM to a value < 0) you should use saturation arithmetic, where overflow ends up with the maximum number (PWM = 0xFF = 255) and underflow results in zero (PWM = 0x00 = 0). You also need to be careful of how to handle maximum and minimum speeds with respect to the "error" signal. Ultimately, the magnitude of the error signal will determine how you scale your control and speed representation. You will have to experiment with what is the best magnitude of the correction signal, error, and KP.

Nexys4 DDR Device Mapping

We are not specifying functionality of the PmodOLEDrgb and the LEDs for this project so we encourage you to think up additional function that could earn you extra points. One example is to use the PmodOLEDrgb like an oscilloscope, showing the last few seconds of motor operation, actual speed and desired speed signals in different colors.

The minimal required user interface for your application is as follows. Feel free to innovate and improve on it. Just keep in mind that you will use Project 2 as a starting point for Project 3.

PmodENC:

- Rotary Encoder – The rotary encoder knob is used to select the motor speed. Twisting the knob the clockwise increases the motor speed (like volume control). Twisting the knob counter clockwise decreases motor speed.
- Switch – The switch on the PmodENC changes the direction of the motor. Looking at the PmodENC with the switch on the top, with the switch in the left position, the motor turns counter-clockwise, with the switch in the right position, the motor turns clockwise. This command only takes effect when the rotor is stopped (0 RPM). Make sure that the direction is not changed when the motor is running

Nexys4 DDR Slide Switches:

- Switches[15:6] – not assigned
- Switches[5:4] – controls the amount the KP is incremented or decremented on each press of the BtnU or BtnD buttons. These switches are only needed if you implement one or two decimal fractions for KP. Pick whatever constants you need for your algorithm but one suggestion is to implement the following:
  - 00: If both switches are open, the Rotary Encoder changes the motor speed by ±1
  - 01: If Switch[4] is closed, change the KP value by ±5 on each button press
  - 1x: If Switch[5] is closed, regardless of Switch[4] position, change the KP value by ±10 on each button press.
- Switches[3:2] – Reserved for project #3. Will support full PID control in Project #3
- Switches[1:0]
  - 00: If both switches are open, the Rotary Encoder changes the motor speed by ±1
  - 01: If Switch[0] is closed, the Encoder changes the speed by ±5
  - 1x: If Switch[1] is closed, regardless of Switch[0] position, the Encoder changes the speed by ±10

Nexys4 DDR Pushbuttons:

- BtnC – sets desired motor speed to 0 and turns off the PWM signal to the motor. The Proportion control constant needs to be reset to a non-zero value.
- BtnU – Increment the Proportion control constant
- BtnD – Decrement the Proportion control constant

Nexys4 LEDs:

- LED[15:3] – not assigned
- LED[2:0] - Reserved for project #3. Will support full PID control in Project #3

Nexys4 Seven Segment Display:

- Digits[7-4] – displays the value of KP. Note, in project #3 it will be possible to select for display and modification each of the three constants: KP, KI, and KD.
- Digits[3-0] displays the RPM of the motor. The actual value should be scaled so that maximum motor speed uses all four digits.

It should be clear that this application is decidedly more complex than the one you created in Project 1 so best get started as soon as possible. You need to think through your control algorithm very carefully since it is possible to issue commands that damage the motor. In embedded computing hardware and/or software bugs can damage hardware.

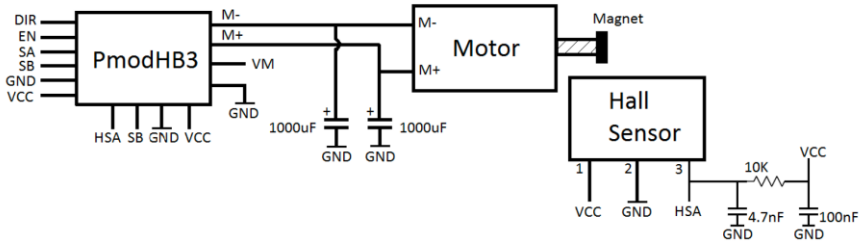
Running a test

- User: Press BtnC to set the desired motor speed to 0 and KP to 0 and start the test.
- Increment and decrement motor speed with different combinations of clock and counter clockwise rotation.
- Increment and decrement the Proportion constant control and study how well it stabilizes the motor.
- Press BtnC to reset the control and stop the motor.

Hardware

Control Circuit

The Project 2 release package contains a schematic, a PmodHB3 Reference Manual, and Bill of Materials (the BOM) for the suggested control circuit. The motor needs to be mounted on a solid platform, proto-board or small piece of plastic. The motor will be changing speeds quickly and should be mounted securely. For Project 3 you will have a load motor which will put even more stress on the drive motor. The Hall sensor and its circuit needs to be mounted securely close to the motor's drive shaft. The magnet needs to be mounted on the motor's drive shaft so when the shaft rotates it passes near the Hall sensor to allow it to detect the magnet's presence.



VM - Connect it to VCC
VCC - FPGA 3.3V output from Pmod Connector.
GND - FPGA Pmod Connector ground.

The motor output and Hall sensor input need to be connected to the PmodHB3 as shown in the schematic.

When setting up your system, please keep in mind that for Project 3 you will be mounting a second motor whose shaft will connect (small tubing works well) to the first motor's shaft. You need to keep that in

mind with respect to the Hall sensor placement. Also in Project 3 there will be a circuit (resistor and switch) connected to the drive circuits of the 2nd motor.

The control circuit and hardware can be built in whatever manner you are comfortable with. You can use a proto strip (the white plastic boards with rows of holes for components and wires) or you can build something a bit more lasting on a piece of perf board (the board containing rows of solder plated holes) or you can design, fabricate, and assemble a printed circuit board in the LID (Laboratory of Interconnected Devices).

Note: Although we believe the motor can draw enough current from the 3.3V pin on the PMOD Connector to safely drive the motor under all conditions it would be prudent to drive the motor from a separate power supply (either a bench supply in the lab or some batteries).

Custom IP Block

You will create a new block design and top level module using Vivado and package the block design as new (user generated) IP. This IP includes the interface between the PmodHB3 and the Nexys4 DDR. There two aspects of this IP: First, the IP should generate the PWM and direction signals to the H-Bridge on the PmodHB3. Second, the IP block should implement a tachometer function.

Although you can refactor your hardware pulse-width detection from Project 1 as a tachometer, we recommend an alternate approach that may be more suitable for calculating RPM (Revolutions Per Minute). The Hall sensor produces a pulse every time the magnet mounted to the motor shaft passes by the surface of the sensor. The higher the rotational speed of the motor, the closer together the pulses will be. In effect you are determining an unknown frequency for a known interval (seconds or minute). As we discussed earlier in the term, a way to do that is to count pulses for a known amount of time.

With that in mind it should be fairly easy to implement a Verilog module that detects a rising edge on the Hall sensor and counts the number of rising edges in a 1 second interval. Multiply that count by 60 and you have RPM's. Your IP block can either produce a count of rising edges in 1 second that can be read by the Microblaze via a register and converted to RPM's in your application or, it occurs to us, that you could do the multiply by 60 in the IP block and return RPM's directly via a register. The Series 7 FPGA on the Nexys4 DDR contains a number of hardware multiply blocks that can perform fast multiplications. Clocking the edge detection logic with the 100MHz AXI clock should provide plenty of margin for detecting a rising edge on an under 50 KHz signal from the Hall sensor.

Embedded System Configuration

Use Vivado and the IP Integrator to create an embedded system with this minimum specification. You can add additional hardware as you see fit:

- Microblaze, mdm, interrupt controller, etc. Configure hardware floating point in the Microblaze. That should keep the program size reasonable and provide fast computation for the control loop.

- Program/Data memory (board-dependent):
    - Local (BRAM) memory of 128KB for program/data memory
    - No external memory and no caches. Since the program/data memory is in BRAM there isn't much to be gained by adding caches since they are also made from same type of BRAM
- FIT timer set to generate 5KHz interrupts: The interrupt can be used to debounce the push button input, generate delays used in the logic, and for capturing the frequency output from the PmodHB3 IP. It can also be used to filter the output of the PmodHB3 frequency detection to achieve a smooth curve while plotting the graph.
- GPIO, one port, 4 bits wide. This is an output port that you can place debug information on.
- Nexys4IO and PmodEnc – used to provide access to the NexysDDR4 devices and the PmodENC.
- UartLite peripheral - Used to send the signals to PC to be plotted on a graph. Configure the UART Baud Rate to 15200, N-8-1.
- PmodOLEDrgb is not required for this project, but there are potential extra credit uses of that device so you may want to include it in your system. If you decide to use the PmodOLEDrgb you will need to add that IP.

There is no circuitry in the FPGA other than the embedded system so you could have the IP Integrator generate a system wrapper file, make that your top level of your synthesized design, and edit the constraints file to match those port names. Alternatively, you could use the same hierarchy as Project #1 and instantiate the embedded system in a top level module (`n4fpga.v`) and connect the ports from your embedded system to the module ports in `n4fpga.v`. Either approach is acceptable.

You may want to change the port/pin assignments in the constraints file to move the PmodOLEDrgb (if you include it) from the JA connector to the JB connector. Doing so will free the JA connector for your PmodHB3 control circuit interface. The top level `n4fpga.v` and the constraints files are not included in the Project #2 release package. You can create your own by editing the files provided with Project #1.

Embedded System Drivers and Link Map

Your Project 2 application will use the drivers and standalone OS provided by Xilinx. The following additional drivers and files should be included in your software platform:

- Nexys4IO and PmodEnc drivers as used in Project 1.
- Your Level 0 and Level 1 drivers for the PmodHB3 interface, PWM control and Hall sensor input.

Project 2 Tasks Summary

Select a partner

You will work in teams of two for Projects 2 and 3. We have set D2L up for self-enrollment. We will assign teams for any students who have not indicated a team within a few days of project assignment. To self-enroll in a group:

- One member of your team should claim an unused group (Group #) and add himself/herself to the group
- Same person should email, text, or whatever the other member(s) of the team with the group number
- Other member(s) of the team should self-enroll in the same group
- When a group is full I will rename the group to match the team members

While there are many ways to split the work on this project one suggestion is that one partner creates the embedded system and builds the control circuit hardware while the other partner creates the control application and peripheral. Both partners could (and should) team up on the control system design, system integration and data gathering and reporting. While you may collaborate with other teams, all of the work you submit must be your own and those you collaborated with should be named in your report.

Download the Project 2 release package

Download the Project 2 release package from the course website.

Mount the motor, build the control circuit and connect it to your FPGA development board.

The Project 2 release package includes the schematic and BOM for the suggested control circuit, including the motor. (The BOM also contains the necessary parts for Project 3 too.)

You may be able to use the PWM and pulse-width detection circuitry from Project #1 to test your control circuit by connecting the PWM output from your Project 1 to the motor. This allows you to start operating the motor without control feedback. In this case make sure that the direction is set to a constant logic level before the PWM output is applied to the Enable signal of the PmodHB3. In fact, you can characterize the control circuit by sweeping the PWM value from 0 to 255 and observing the output of the Hall sensor on an oscilloscope or logic analyzer. Doing so will provide the minimum and maximum speed of the motor and the offset needed to start the motor rotating. Adding some delay between PWM values should give the circuit a chance to settle.

Create your motor control and Hall sensor frequency detect peripheral and write a driver package for it

Design the motor control interface and Hall sensor-based tachometer circuits in Verilog and create a Xilinx custom peripheral for it. Create a software driver package for the new peripheral. The driver package should most likely include these functions:

- Initialize the peripheral being used into the correct mode.
- Drive a PWM signal from your hardware IP to the PmodHB3.
- Capture Hall sensor frequency (RPM) readings

Package the IP when the peripheral and drivers are finished and debugged. Add the IP block to a new respository or to the repository that includes PmodENC and Nexys4IO.

Build the embedded system and top level module for the project

Build your embedded system using the IP Integrator and instantiate it in your top level module. Connect the embedded system ports to the top level ports and make any necessary changes to the constraints file. Your embedded system should include all of the peripherals your application needs, including your IP block.

Don't forget to configure the FIT timer and connect it to the system clock, peripheral reset and the interrupt controller. The FIT timer interrupt should be the highest priority interrupt in your application.

Synthesize, implement, generate bitstream and export your hardware.

Integrate the hardware and software and tune your control system response

This application is more complex than the application in the first project. You may implement the user interface as described in this write-up or you can innovate and develop a user interface of your own design. The important thing is to implement the Proportional control algorithm in a way it can be quickly and effectively demonstrated.

Implement the control application

Once you have built the control circuit and your hardware system, you are ready to integrate the system and run your application. As part of your experimentation you should determine the value of KP that gets the motor speed up to the setpoint quickly and then stabilize on, or as near to, the set point as is possible.

Demonstrate your project and submit the deliverables

Once you have your project working be prepared to demonstrate it to the instructor or T/A. Please try to demonstrate this project on or before the deadline. Submit your deliverables to the D2L Dropbox in a single .zip or .rar file of the form <yournames>_proj2.zip. Only one submission per team is necessary. We will grade the submission with the latest timestamp if there are more than one.

Deliverables

- A demonstration of your project to the instructor or TA. You must bring your own control and motor system.
- A five to seven (5 to 7) page project report explaining the operation of your design, most notably your control algorithm and user interface. Please include at least one "interesting" graph showing the results from the control algorithm. List the work done by each team member. Be sure to note any work that you borrowed from somebody else.
- Source code for your C application(s). Please take ownership of your application. We want to see your program structure and comments, not ours.
- All files regarding your new motor control and speed measurement system (IP), including the Verilog hardware, and driver code.
- Your constraint and top level Verilog files.
- 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. Part of this will be how well your demo works, it must be perfectly functional. Also, we do expect you to take initiative and add some bells and whistles. We will look at other "quality" of design issues such as unnecessary display flicker, slow response of the system to buttons and control changes. (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 you. (20 pts)
- The quality of your project report. (20 pts)

Extra Credit Opportunities

Project 2 offers several opportunities to earn extra credit points. Here are some suggestions but we are willing to be amazed and amused:

- Innovate on the user interface –you want to be able to easily "tinker" with the control loop
- Use the PmodOLEDrgb as a plotter to plot the last few seconds of both the desired RPM and the current RPM. The time scale should be selectable with a switch or two. Also, you need to think about if you want any "magnification" in the amplitude scale.
- Enhance the tachometer functionality to be able to detect direction of rotation in addition to the speed.

References

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

[2] *Digilent PmodHB3 H-Bridge Motor Control Reference Manual*. Copyright Digilent, Inc.

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

[4] *Nexys4IO and PmodEnc Product Guides and Driver User Guides* by Roy Kravitz, 2014

[5] *Getting Started in ECE 544 (Vivado/Nexys4 DDR)* by Roy Kravitz, et. al.

[6] *Digilent PmodOLEDrgb* User Manual (optional)

Revision History

Rev 1.0	20-Feb-2017	DH	Major Revision to Project #2. This project does closed loop Proportional control to maintain the speed of a small electric motor. Many of the concepts (closed loop control, create/package custom IP, etc.) are borrowed from the previous project concept but the application is different.
Rev 1.1	28-Apr-2017	RK	Changed pulse width detect from high/low count to counting rising edges for a one second interval. This algorithm is better for using a tach pulse from a Hall Effect sensor to calculate RPM. Also changed KP factor selection to provide greater precision using fixed point. Organization, typographical and writing style changes.