Maseeh College of Engineering and Computer Science

PORTLAND STATE UNIVERSITY

ECE 544 Embedded System Design with Programmable Logic

Closed Loop Control System Using PID Controller in Nexys4DDR

Author:
Michael Escue
Ramprakash Baskar

Supervisor:
Prof. Roy Kravitz

20th May 2019

Project Output

YouTube Reference:

https://www.youtube.com/watch?v=b8_c7qHz_3M&feature=youtu.be

Contents

Chapter 1

Introduction

The Objective of the project is to implement closed loop control system by using PID controller to control a DC motor using Nexys4 FPGA. The closed loop control system automatically varies the motor speed according to the load provided on the shaft of the motor. The Project also includes a Watchdog timer to implement recovery action in case the application slips into infinite loop.

Chapter 2

Hardware Implementation

The Hardware Implementation involves building a Pulse Width Modulation logic and Revolutions Per Minute (RPM) logic built in Verilog and implemented as a Intellectual Property (IP) block in the Embedded System Design. The IP block is built using Vivado IP Packager and then added into the Embedded System Block Design. Software drivers are then used to refer to the registers in the block to write and read data. The Outputs of the Block Design is then routed into the top module to drive the Pmod ports to which the Pmod HB3 moudle is connected.

Block Design

The Embedded System is implemented using Block Design wizard available in Xillinx Vivado. The Block Design includes the following components:

- Clock wizard
- Interrupt controller
- Microblaze Debug Module
- Nexys4IO
- AXI Timer
- PmodENC
- PmodOLEDRGB
- GPIO Port

In addition to utilizing the components directly some of the components were customized to suit the project requirements.

Clock Wizard

The Clocking Wizard is used to generate 100 MHz and 50 MHz signal for the Embedded System and PmodOLEDrgb.

GPIO

There is one GPIO module which is used to read the input of switches and push buttons.

Custom IP Block

A Custom IP block is used to control the Motor and to read the value of RPM from the encoders available in the motor. The IP block is packaged as a AXI peripheral with one input port to detect the pulses from the DC Motor and two output ports to get the PWM signals generated and the Direction

Hardware Designs

The Hardware design implemented in the project is attached below:

Embsys Block Design

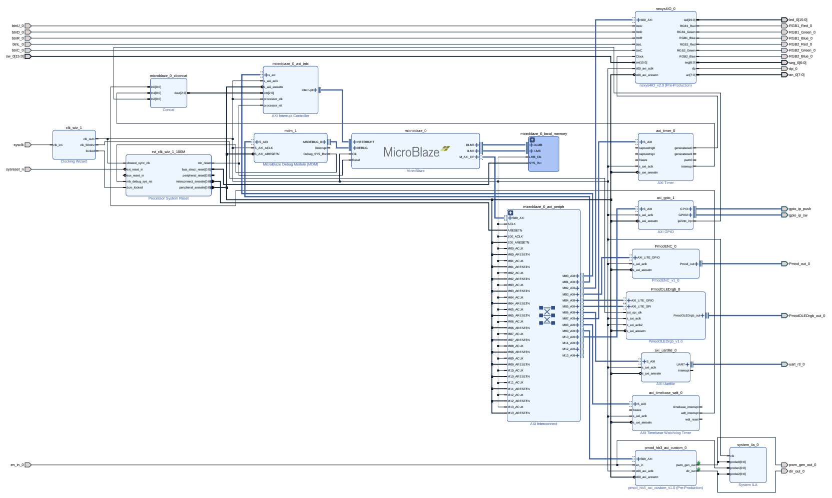

Figure 2.1: Embsys

Elaborated Design

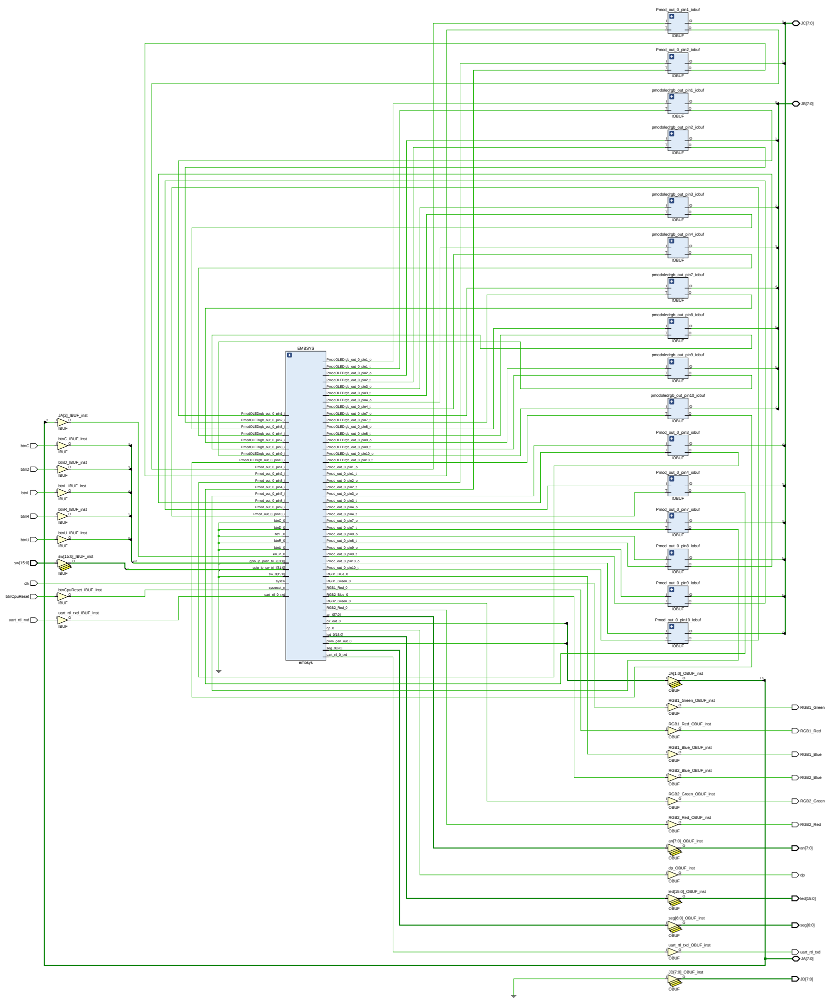

Figure 2.2: Elaborated Design

RPM Control

The RPM value of the motor is calculated by counting positive edge of the input signal for a known duration of 1 second. We know that that the motor produces 12 pulses for 1 revolution of the motor, so the calculating the pulses for 1 second and dividing the value by 12 gives us the Revolutions per Second. On multiplying the output by 60 we can count the Revolutions per Minute.

The Verilog Code Snipped is attached below:

```verilog

always @(posedge clk) begin
    if(counter == 0)
        counter <= 500000000 - 1; // for 100mhz
    else
        counter <= counter - 1;
end

// simple flip/flip to store last value of signal
reg last = 0;
always @(posedge clk) begin
    last <= signal;
end

// when counter == 0, register current count and reset edge counter
// otherwise increment edge counter for each rising edge of the
   pulse
reg [C_PWM_AXI_DATA_WIDTH-1:0]edges = 0;
reg [C_PWM_AXI_DATA_WIDTH-1:0]outreg = 0;
always @(posedge clk) begin
    if(counter == 0) begin
        outreg <= edges;
        edges <= 0;
    end
    // detect edge, was low and now high
    else if(~last & signal)
        edges <= edges + 1;
end

always@(posedge clk)
    rpmout = outreg; //for  100Mhz
```

Listing 2.1: Hardware Pulse Width Detection

Chapter 3

Software Implementation:

The Software implementation involves the creation of a feedback loop which takes in the RPM value from the custom IP block and based on the load tries to increase or decrease the speed of the motor. In addition to this to overcome the effect of program falling into a infinite loop; watchdog timer module has been implemented using Xillinx Watchdog timer IP package.

Four threads are used in this application; Master, Input, Display, and PID threads. Thread priority was determined by experimentation, eventually placing the PID thread at priority 3, while all other threads were placed at a priority of 2. This seemed to provide the best performance (relative) for the program. The purpose of the Master task was the generation of the 3 other tasks, then monitoring switch 15 for an asserted signal to force a crash of the watchdog timer.

The Input task responded to the interrupt of the GPIO which was tied to the buttons and switches. It also polled the ENC state to check for rotation or ENC switch throws. Updates of the button and switch states were enabled by release of a semaphore from the GPIO interrupt handler just as in the example. Updates in values as responses to siwtches and buttons were passed along a queue to the display and pid threads.

The display thread receives messages through a queue which only blocked for a single tick before making display updates. If the message sent through queue by the input thread has not changed, then the updates will not change. SSEG, LED, and OLED updates were performed in this tasks. This thread did transmit any data other than to the visual displays.

The PID thread also received messages from the input task via a queue which was different than the one used to send data to the display thread, though it contained the same information. The queue in this task similarly only blocked for 1 tick before continuing to update the PID control. This code could have been better improved were there more time to spend on it. Feedback to this task was provided by a FIT interrupt handler which sampled the slave registers of the HB3 IP.

Chapter 4

Input/Outputs

The Inputs to the Control System is provided using PmodENC, Switches and Buttons. The PmodENC is used to increase the speed or decrease the speed by rotating clockwise or anticlockwise. The Switch available in the pmodENC is used to change the direction of the rotation. The Switches are used to bring change in the parameter Proportional, Integral or Derivative controller value and also to implement a Watch-Dog Timer. The Buttons available is used to increase(BtnU) and decrease(BtnD) the value of PID. The Outputs are displayed using DC Motor, LED, Seven Segment and pmodOLEDrgb.

- The Switch 1 selects between Hardware and Software implementation.
- Switches 2 and 3 selects between Red, Green and Blue signals. The
- Push Button Up increases the Value and down decreases the value
- Push Right increases the saturation and left decreases the saturation value.
- Encoder rotate right increases Hue and rotate left decreases the value.

Chapter 5

Results:

- RPM Feedback from DC Motor encoder was implemented.
- Psuedo control was acheived, but performance was decimated by low sampling rate.
- Best control settings were a Porportional Gain of 2 and an Integral gain of 1, with no Differential gain.

Chapter 6

Issues and Fixes:

- Pulse width detection: Many issues with converting pulse count to RPM. Timing was not met in our implementation due to long carry chains. This may have been fixed with a shift add multiplier rather than behavioral design of multiplication.
- PID: Converting from RPMS back into motor control without crashing thread. Calculations took several cycles to complete, inducing latency in the task, and possibly violating WDT.
- OLED: Would not instantly display written characters and required system reset before displaying. Unresolved without reset.

Chapter 7

Work Done:

- Embedded System Configuration: ME
- Software: ME
- RPM HDL: ME/RB
- IP packaging: RB
- Report: RB/ME

Chapter 8

References

- Getting Started Guide
- Nexys4IO Product Guide
- SDK Documentation
- Lecture Notes