Portland State
UNIVERSITY

Department of Electrical and Computer Engineering

ECE 544 – Embedded System Design

Spring 2019

Project Report # 2

Microprocessor-based Closed-loop Motor Control

By
Surya Ravikumar
Kanna Lakshmanan

Microprocessor-based Closed-loop Motor Control

Project Report

Goal

- Gain experience with microprocessor-based closed loop control
- Gain experience with multi-tasking, synchronization, and inter-process communication (IPC) using FreeRTOS
- To successfully design a C code which implements PID closed loop control

Project Phases

- Getting started with the FreeRTOS to build a embedded system.
- Designing a simple embedded system implement basic motor control.
- Creating a custom IP for Pmod HB3 and using it for motor control.
- Implementing PID control for the motor.
- Interfacing the other Pmods and Nexys4 buttons to the control the motor parameters.

Achieved Goals

- Successfully designed a embedded system with custom IP.
- Implemented PID closed loop control for the motor
- Implemented motor control using Pmods and Nexys4 board.

The Hardware Circuit

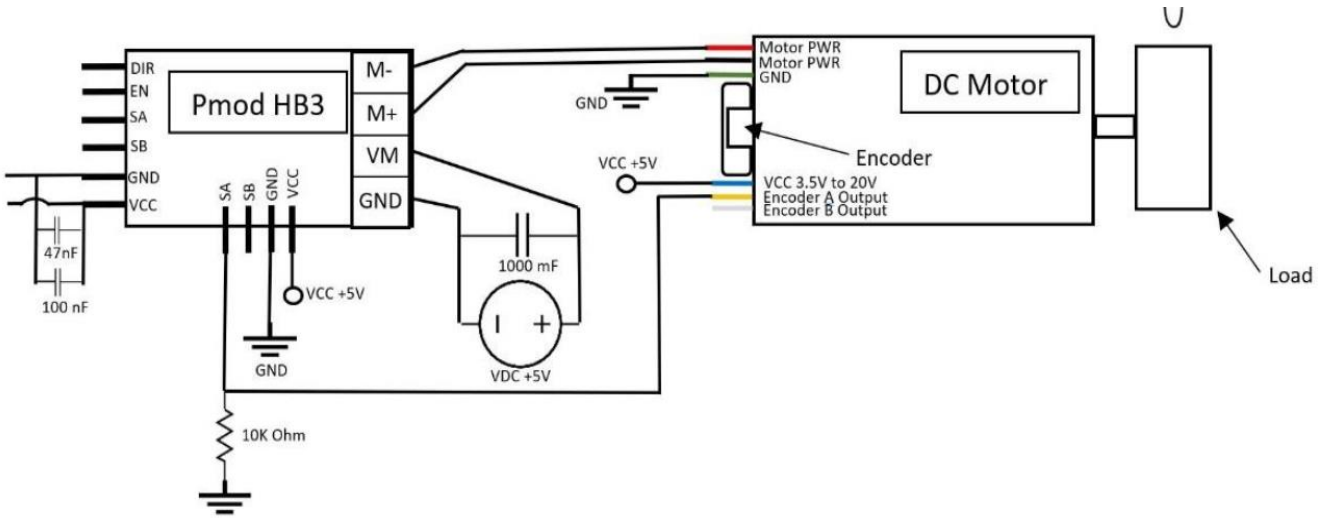


Figure 1: Control Circuit Schematic.

Above is the schematic of the system used to drive the motor. The motor is connected with the motor driver which drives the motor and has external pins to provide sensor feedback on the DC motor. 5V input is given to the motor to drive the motor. The pmod pins are connected to the JC port of the fpga to send and read values.

The Design

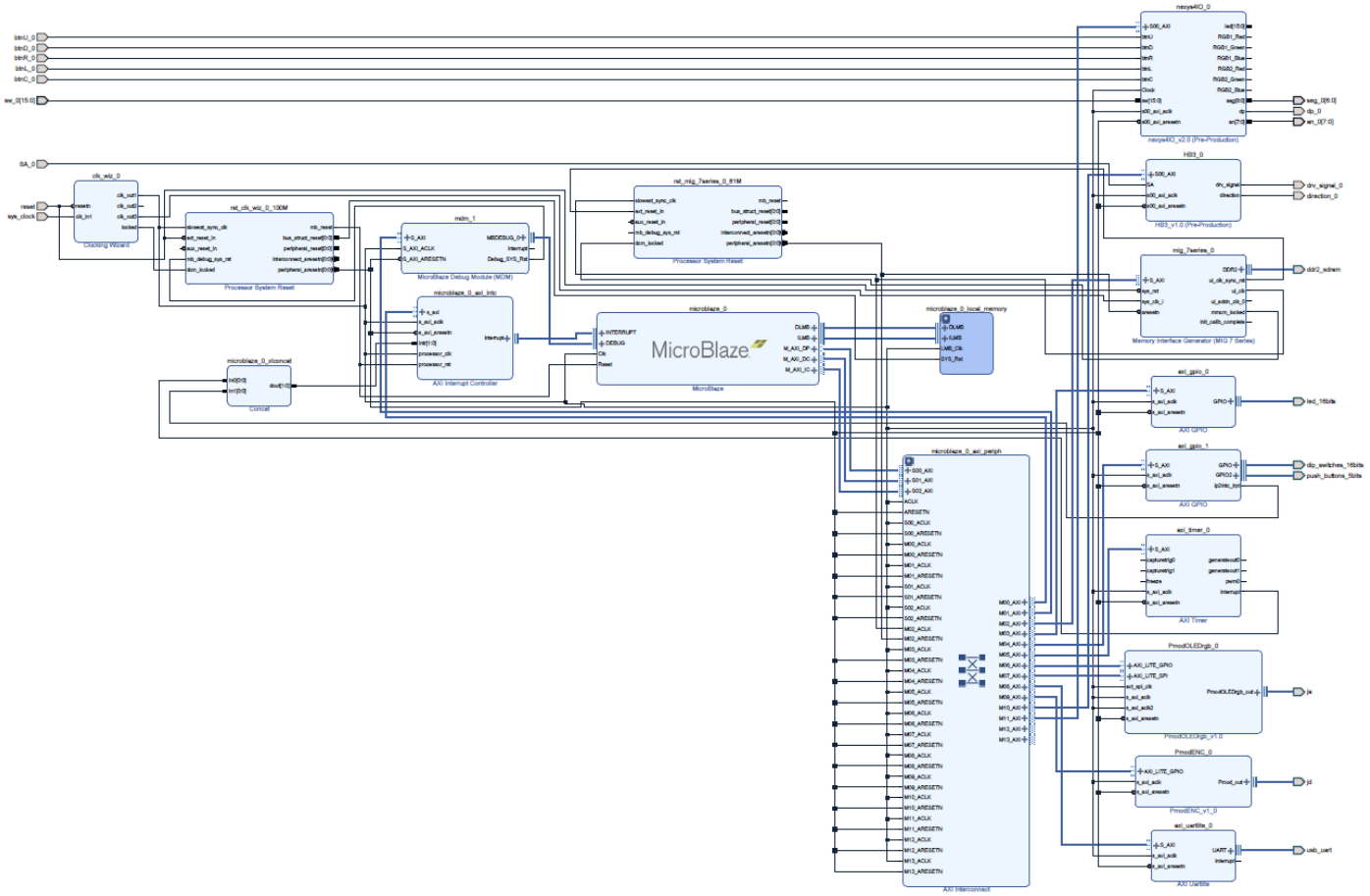


Figure 2: Embedded system design using Microblaze, Custom IP block and Nexys4 IO.

The design includes Pmod's, AXI GPIO ports, Clocking wizards, AXI uartlite, and Timer circuit blocks. The Pmod's are interfaced with the nexys and the microblaze to interact with the board. There are 2 GPIO ports in this design, one GPIO has the dip switches and push buttons mapped to it and the other gpio has the leds mapped to it

We are using a MIG 7 series ddr2 ram for the memory and there are two interrupt pins one from the AXI timer and the other interrupt is from the push buttons in gpio. The clocking wizard has three clocks 100Mhz, 200Mhz and 50Mhz. The 100Mhz clock drives the whole circuit and the 200Mhz clock is given to the mig7 series block. The Pmod Encoder and Pmod OLED are used for changing the pwm of the motor and displaying it in the OLED display.

The Hardware

Embedded System

Since we worked with FreeRTOS in this project, we had to add a memory module, interrupt controller and a timer to the embedded system. On top of this we added MicroBlaze, UART, GPIOs using the board files. Since the board files were used to create the system, constraint files were automatically generated. Pushbuttons and dip switches were connected to one GPIO module with interrupts enabled allowing us to work with button and switch presses in an interrupt handler while LEDs were connected to another GPIO module. UART was added to see print messages on the terminal. PmodOLEDRGB and PmodENC IPs were added to the design as well. These pmods were connected to the JA and JD ports automatically using the board file connectors. Later, Nexys4IO IP was added to embedded system which allowed us to use its drivers. The 7-segment display was also connected through Nexys4IO instead of a GPIO module.

Custom IP

A custom IP had to be created for Pmod HB3 to be included in the embedded system. The IP included to custom modules to generate PWM output based on a user defined value, and to detect RPM of the motor which was feedback to the system. Top module of the IP declared inputs – encoder output from the motor used to detect RPM, and outputs – driving signal (generated by the PWM generate module) and direction defined by the user. Top module also instantiated the modules.

8-bit PWM value written to a register in software is fed into PWM generate module along with the AXI CLK and AXI RESET inputs. PWM generate module then implements a simple counter on posedge of clock. If the counter is less than or equal to the 8-bit value fed in as input, the driving signal is 1, else 0. The counter resets to 0 when it overflows, therefore no resetting is needed unless there is a system reset. Since the embedded system operates at 100 MHz, and the IP has the same clock, motor speed was indistinguishable at high speeds. Therefore, the PWM input value is scaled by 4 such the period of the output signal increases.

SA signal from the Pmod is fed into RPM detect module This module just counts the number of pulses in 1 second (defined as a constant value). Since the counter is inside an always clock on clock edge, the SA signal could be missed. To overcome, another signal is defined such that the SA input is synchronized with the clock. This then allows us to check if there was a pulse (low to high transition) in between the current and previous clock edges. If this value is high, RPM counter is incremented. Once the time counter that counts on every posedge reaches the predefined 1 second count, the rpm counter is moved to the output variable and both counters are reset.

Once the RPM detect module return the RPM count every second, the value is stored in the register. After creating the IP, the new IP is added to the embedded system with SA input and Direction & PWM signal outputs are made as external signals.

Top module

After generating the block design, a top module is created. Since the HB3 signals were made externals, they had to connected to one of the PMOD ports. To do this, constraints file from project 1 had to included. Moreover, 7-segment signals from Nexys4IO also had to connected as outputs. Therefore, the constraints file from project 1 was edited such that it only contained pin definitions for JC pins, seg, an & dp pins. Pushbutton and dip switch inputs to the GPIO module were also connected to Nexys4IO in the top module.

The Software

FreeRTOS was used in this project. In the software code we used two different threads for the code. After initializing all the standard address ports. The interrupt function block has the interrupts defined if any push button is pressed the system treats it as an interrupt and gives it priority and the interrupt will be handled first.

Main thread contains the code for PID control. The PID is calculated by first reading the rpm of the motor from the hb3 module and then the target rpm is calculated by using the rpm array which contains a list of 255 values of motor rpm for each set value of the PWM. We give a default setpoint pwm value of 127 in the beginning of the code. This value is taken to calculate the target rpm from the array list.

The PID constants Kp, Ki, and Kd values are tuned and set as 1,1,0 respectively. These values were found be stable for the given system to get a smooth pwm change when the load is given. When the load is applied the current rpm- which has to be the new rpm is calculated by using these equations

Error = Target rpm- read rpm
Integral = integral+ error
Derivative = error- prev error.
Current rpm = target rpm+ Kp*error + Ki *integral + Kd* Derivative

The current rpm value is set to motor whenever the load is applied. This makes the motor increase the rpm whenever the load is applied compensating for the applied load.

The other parts of the code is for interfacing the other Pmod's and pins in the fpga. When ever the pmod switch is switched the motor changes the rotation side. It doesn't directly change direction the motor stops goes to zero and changes the direction.

Whenever the switches 3:2 is turned on if its 10 the push buttons enables us to change the values of the constant Kp if its 01 Ki and if it is 11 Kd changes. This way we can tune it accordingly while the motor is running.

The switches 1:0 enable us to change the multiplier count of the rotary encoder which when turned changes the pwm value set. When switch is 10 it increments by 1x and when the switch is 01 it increments by 5. By default the value of the multiplier is 1.

The Project

When the embedded system is turned on the motor spins at a set PWM of 127 rpm and the pmod OLED displays the value of the PID constants and the read rpm values. The seven segment displays also display the value of the constant that is selected to change and the read rpm values.

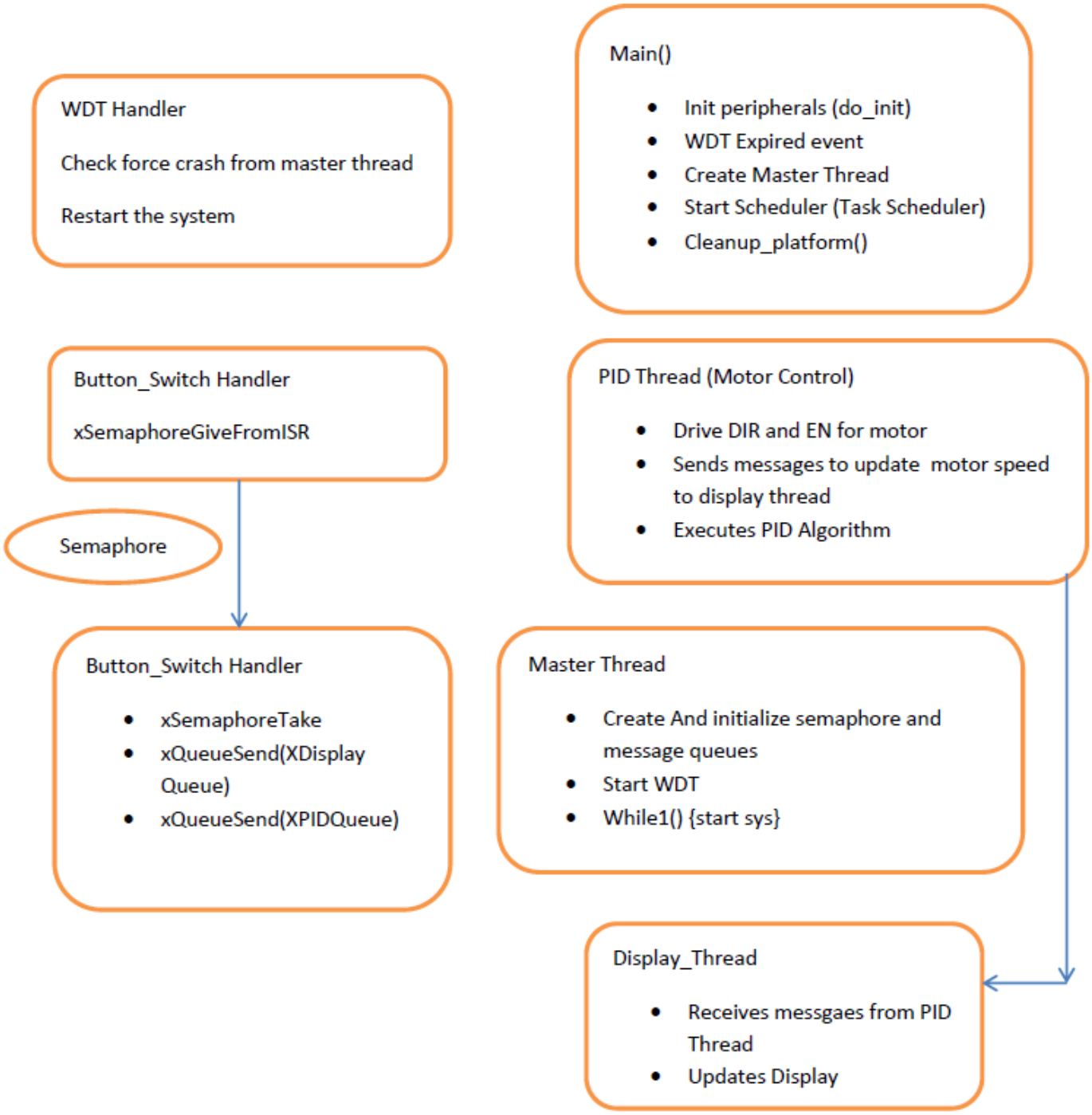
When the motor is given load the rpm of the motor increases and to compensate the load. The rpm increase can be seen on the display. The rotary encoder allows us to change the PWM of the motor by rotating in and the new pwm value also can be noted in the OLED display in the fpga.

We can change the pwm and also tune the motor if we need stabilizing by using the push buttons. The center push buttons sets the motor pwm to 0 and all the constants to zero.

We have included watchdog timer in the threading model, It can be implemented to have a extra functionality of the system. It can also be treated as an interrupt and stop the system when the timer is activated and it expires. Currently we have done it without the timer. We have added it in the threading model for future reference.

FreeRTOS threading model

Threading Model used with Free RTOS:



Challenges faced.

- Designing a custom IP was tricky it consumed a lot of time but we were able to figure out the pins and created the IP
- First we implemented the functionality in Standalone mode to see everything thing is working and then we switched to FreetRTOS
- FreeRTOS was a bit confusing but once we got the threading models right. The flow was seamless

Project Outcome.

We were able to successfully design a close looped embedded system with PID control. We got experience in designing a Custom IP and how to interface all together with a hardware system. WE also understood the working of freeRTOS and how the threading model works. We were able to generate a RPM of upto 5000 with max PWM of 255. We found stable values of PWM constats at KP =1 KI =1 and KD = 0.