CLOSED LOOP DC MOTOR CONTROL

Tejas Chavan and Rithvik Ballal

Portland State University

ECE 544 – Embedded System Design with FPGA

March 2, 2018

INTRODUCTION

The basic mechanism behind project - 2 is controlling DC motor using PID controller using FreeRTOS operating system. The peripheral used for interfacing the board to DC motor is Digilent's Pmod HB3 H-bridge driver with feedback inputs. A load motor is connected at the output with resistor and switch. Depending on speed of motor it will drive the load motor. Hall sensor will count the number of rotations of the motor and will give actual speed of motor.

Actual speed of motor is subtracted from desired speed to get an error signal which will be used for controlling duty cycle and hence speed of motor. FreeRTOS operating system is used for project-2 instead of standalone OS. The main thread basically Generates all the queues and creates required Task to perform required functionality.

The main function starts the task scheduler and enables microblaze interrupts. Initialization of all major peripherals and GPIOs is done in the do_init() task along with declaration of interrupt handlers. The design involves major interrupt handlers namely Sensor handler that increments the count each time the magnet crosses the sensor. The Second_Timer_Handler converts the pulses into rpm and provides the updated rpm every one second. The robustness of the application is achieved by including a watchdog time that forces a system restart if the application does not periodically restart the timer. The Watchdog timer IP is added to the MicroBlaze design The WatchDog Timer Handler Restarts the watchdog if it was not a force crash.

PWM GENERATION

The PWM generation is done using the Generate mode of the AXI Timer. As an input the AXI Timer has to be fed with appropriate Duty Cycle to generate proper PWM. Before PWM generation is performed, the basic PID controller is implemented by generating the errors and converting them into duty cycle before feeding them to AXI timer.

PWM DETECTION

The rpm of the motor is sensed using a hall sensor. Each time the magnet passes the hall sensor the magnet generates a low pulse and generates an interrupt. This interrupt is handled by the second_handler() and increments a counter. After each second, the handler converts the count value into rpm by multiplying the count by 60.

Note: The hall sensor has to be kept really close to the shaft of the motor and aligned with the axis of the magnet else the sensor won't provide correct readings. Moreover, the sensor is prone to noise, therefore any sudden rise or fall in rpm value needs to be eliminated. One can also take the average of the rpm for a known period of time, say 3 sec, and provide the average value to the main function.

****

CONTROL LOGIC

Proportional Control:

In case of proportional constant output of the controller is directly proportional to the error signal i.e. difference between actual and desired value of the system. And Kp will be a proportional constant of the P control.

Pout = Kp * e(t)

Pout: Output of Proportional Controller
Kp : Proportionality constant.
e(t): Error Signal (Set Value – Actual Value)

Derivative Control:

In case of derivative constant output of the feedback system is directly proportional to the derivative of error signal. And Kd will be derivation constant of derivative control.

Dout = Kd * (de(t)/dt)

Dout: Output of Derivative Controller.
Kd: Derivative Constant.
de(t)/dt : Change in Error signal (Previous Error – Current Error)

Integral Control:

In case of integral constant output of the feedback system is directly proportional to the integration of error signal. And Ki will be integration constant of integral control

Iout = Ki (Sum of Errors over time.)

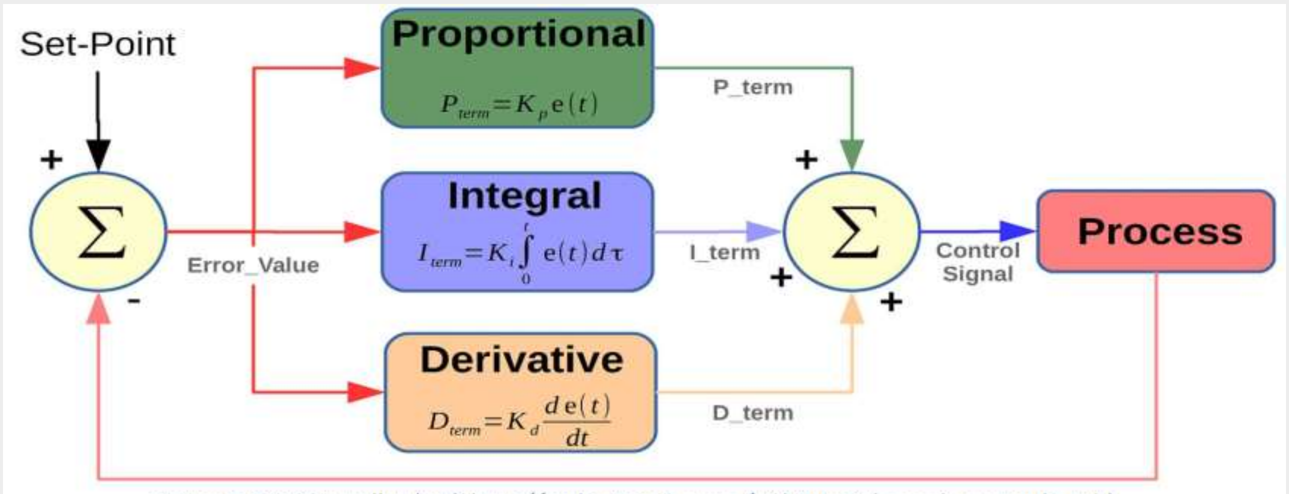
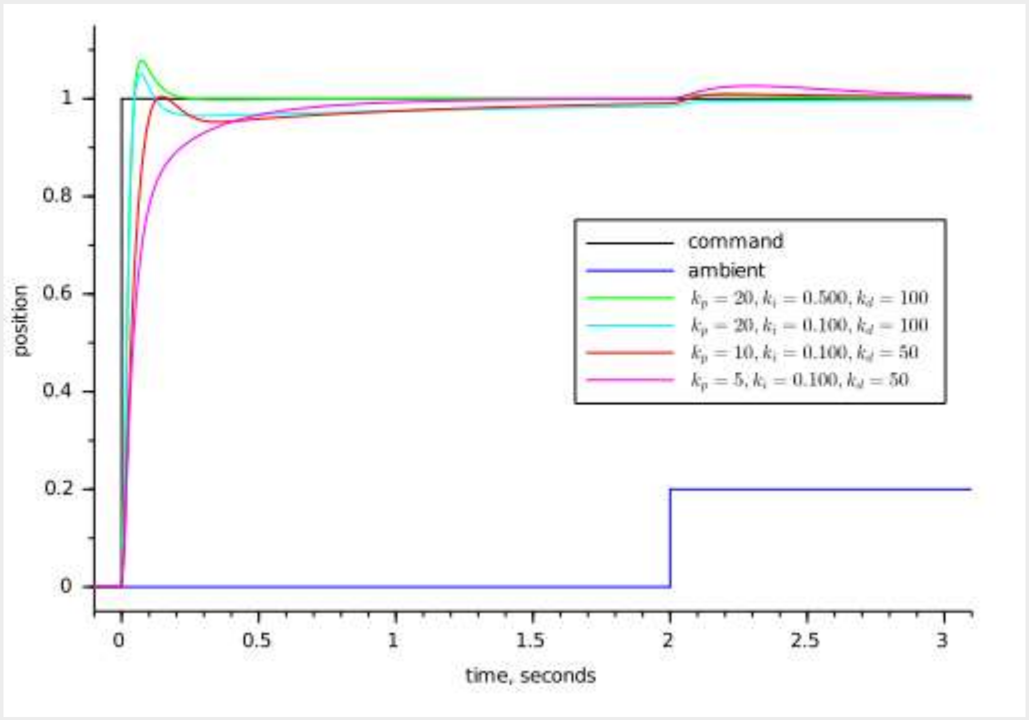


Figure 2: PID Controller (Ref: http://coder-tronics.com/pid-tutorial-c-code-example-pt1)



Note : 1<u>. Proportional Gain</u> basically helps to speed up the response of the system. If the value of the gain is not appropriately set, this may lead to oscillations in the system and the hardware might never come to saturation point.

<u>Lower Gain</u> : Cause Slow response. System takes time to reach the set value.
<u>Higher Gain</u>: Causes the system to become unstable and lead to oscillations around the set value.

2. <u>Integral Gain</u>: The Integral control is used to add long term precision to the system. That is, It reduces the steady state error present even after applying proportional control.

Ideally Integral gain has to be 100 times smaller than the proportional gain. If proportional gain is higher it would increase the oscillations.

3. <u>Differential Gain:</u> Differential Gain basically reduces the oscillations caused by the proportional controller. Differential Gain needs to be around 100 times of the proportional gain.

In Project 2 : Kp: 1.8, Ki = 0.01, Kd = 0

Code Snippet for motor Control:

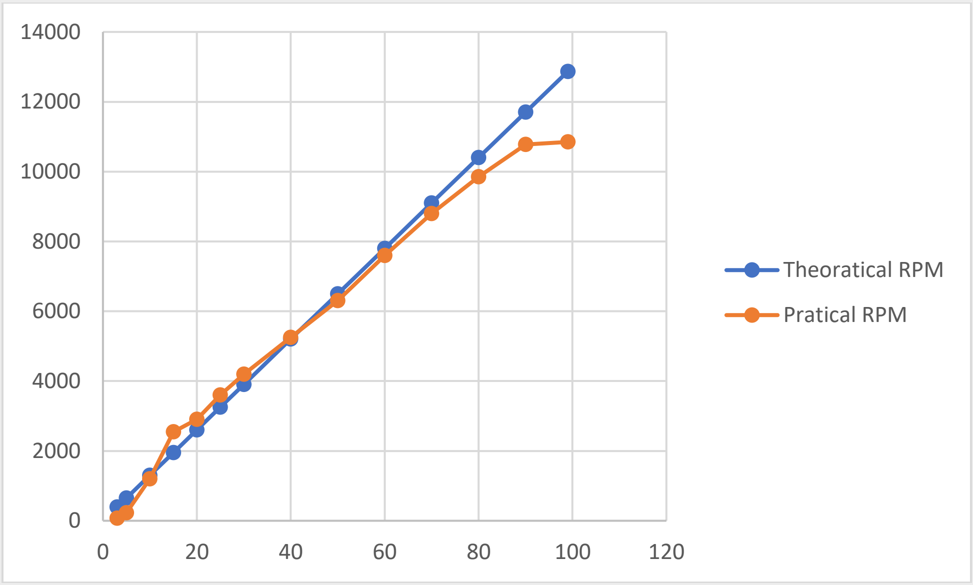
```
error = Rotary_cnt – RC_Process_Value;

Integrate_error = integrate_error + error;

diff_error = previous_eror – error;

final_rotary_count = Rotary_cnt + (Kp/10) error + kd*diff_eror + (Ki/100) * integrate_error.
```

***Kp and Ki are divided by 10 and 100 to achieve floating point precision.**

Cont.

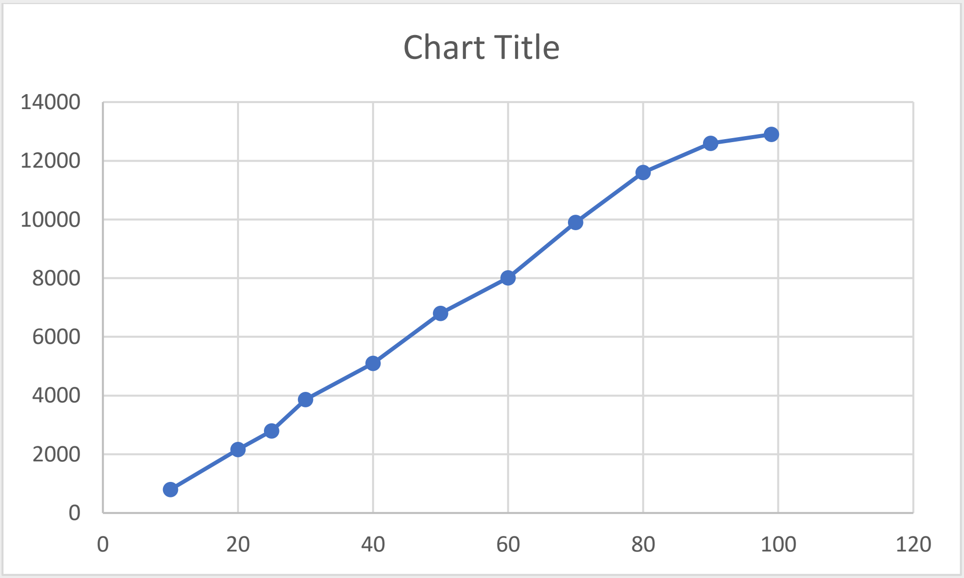
M OTOR SPEED RESPONSE

1. Motor Response on No Load Condition:



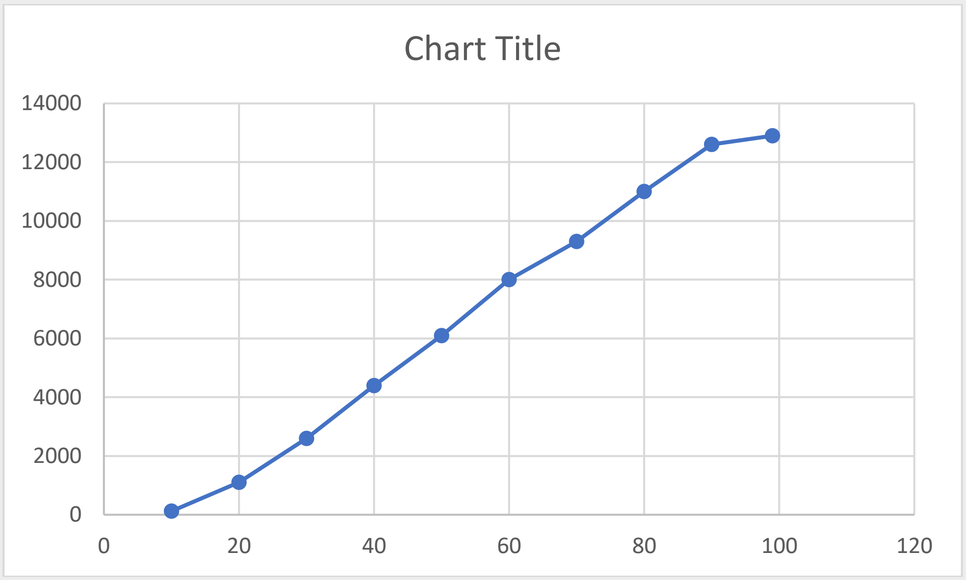
- The response of the motor must match the theoretical values provided in the data sheet. The motor that we used had a maximum rpm less than 14000. We could reach 13200

2. Motor Response With Dead Load of Generator motor.
Kp : 1.8 Ki = 0.01 Kd = 0



- This is the actual response of the control system with only the generator motor connected to the shaft of the drive motor. The Dive motor tries to correct the error using control loop and reaches the desired response that matches the ideal response.

3. Motor Response with Dead load of generator + Rl



- In the third case the a Load resistor of value 10K was connected in series with the generator motor. This added an extra load on drive motor which caused its speed to decrease in case of open loop response. But with close loop response, the motor corrected the error and tried to reach the ideal values.

NOTE: As the system designed was not perfect in terms of its immunity to noise and the exact rotation of the motor, one would not get perfect response. But the response must be linear when the control loop is connected.
The Exact values of rpm at different situations and its corresponding graph has been added to in the excel sheet in the documents.

Challenges faced:

- Faced a few difficulties in building the hardware for the system as we worked with indivisual AXI Gpio's we had to manually map the pins to the constraint files.
- It took a lot of effort to make the actual hardware steady and connect the shafts of the motor. We finally connect the shafts using a pen refill so that the shafts remained aligned and steady.

- Migration from Standalone to FreeRTOS took some time and understanding as the threading model had to be followed

Contribution of team members

Tejas Chavan : Created microblaze hardware and developed code to generate the PWM using AXI Timer. Worked on building the actual hardware and developing the control logic for PID. Adjusting the gain values and finalizing the standalone design

Rithvik Ballal : Implemented the PWM detection Logic using interrupts and AXI timer. Completely worked on implementing the design on FreeRTOS as well as implementing the logic for Watchdog timer. Finalizing the FreeRTOS implementation.

Conclusion:

Close loop feedback control for motor using PID control is implemented using FreeRTOS operating system. For values of KP, kI, and KD desired speed will match exactly with actual speed. But after taking Kp, kI and Kd collectively i.e. by using PID control actual speed matches with desired speed.

References:

- **PID without PHD by Tim Wescott.**
- **Xilinx Forum. Xilinx Documentations.**
- **FreeRTOS by Melot.**