Lab 4 External Interrupt Control and Timer Counter Interrupt

EE138

Colin Chen & Daniel Graham

4/23/2015

Introduction

In this lab we will be integrating several of the concepts in previous labs to produce a closed loop brushless motor control system. New concepts such as the interrupt will also be used in this lab to replace the "wait" function from previous labs, allowing for better control of timing. The key objectives for the lab are,

- Interrupt & State Machine structure for controlling both speed and position
- Closed Loop control of the motor
- Real time display of speed (in RPM) and Position (in degree)

Procedure

Setup

The setup for this lab project required multiple peripherals to be properly enabled, port group A & B, Timer Counter, and the External Interrupt Controller. By using interrupt controllers, we would be reducing the amount of code that is placed in the "while(1)" loop and be primarily controlling the timing and flow of our program through the timer interrupts. This allows for a greater level of precision when it comes to timing as the timer interrupt can be accessed with a specific time interval in mind rather than relying on an inconsistent "wait" function that was used in previous labs. When a timer counter interrupt is flagged, the timer interrupt handler will be processed and all the things that we need to be processed will be properly handled in a timely fashion. For the External Interrupt Controller, the usage of this tool will allow us to not have to actively poll for encoder changes. By setting up the EIC to flag an interrupt whenever the encoder detects an edge, we made the EIC handler increment a counter to count the number of times that the encoder is "triggered". This in conjunction with a timer interrupt which is triggered at a set amount of time allows us to easily detect speed and change quite easily. The encoder itself has 400 ticks, so we can take into account that the amount counted by the EIC counter divided by the time period of the cycle at which the timer interrupt triggers can be easily translated to radians/second and rotations per minute.

Speed Control

For the control of speed, we first set up the way we recognize speed. Due to the fact that we finished coding the portion regarding speed before being introduced to control theory involving radian per second calculations, all our values are either scaled to rotations per minute or a factor of our duty cycle. The duty cycle, or maximum value for the duty cycle placed in the PER register of the TC counter, is 174. Every value that we used has been scaled to the range of 0 to 174 for simplicity. The fastest rotation possible towards one direction (in terms of duty cycle and speed) is assumed to be 0 and the fastest rotation towards the opposite direction is 174. The stop value is assumed to be around 87 with a dead zone to help accommodate for the fact that the POT attached to the ADC as we are not precise in hitting the middle value of the POT.

PID EQUATION

correctedvalue = lastcorrectedvalue + kp*error+ ~~ki*integrator~~ - kd*(error - last error)/(period of TC interrupt) ;

kp = 0.01;

kd = 0.0012;

ki = 0;

The control equation that we used for our control system was not in conjunction with the matlab simulation provided by Professor Ping Hsu. To easily set up his PID equation, we would have needed to have done all of our calculations in radians per second. Our PID setup was done using the method suggested by the control lecture in terms of "hacking" the control piece together. We first produced a kp that would solve our error but would produce some form of oscillation. Second, we tuned the differentiator equation so that it would reduce the error quickly but while trying to dampen out the oscillation. While implementing this system, we were met with many technical issues that will be discussed in the Result section.

Position Control

For position control, instead of measuring speed, measuring angle is slightly different than the process above. Instead of dividing the encoder change over the period of the timer interrupt, we will count the number of encoder ticks that pass after the "zero position" is set. Once the zero position Is set, any count on the encoder (both positive or negative) will be recorded and translated into an angle through a scalar factor. By using this angle, we assign this angle as the "angle error" and use a PID equation similar to speed to correct it.

PID equations

`correctedvalue = abs(angleerror)*kp;`

~~`integrator = integrator + .5*correctedvalue;`~~ `Not implemented in current code`

`derivitive = 0.007*((angleerror - lasterror)/0.005599);`

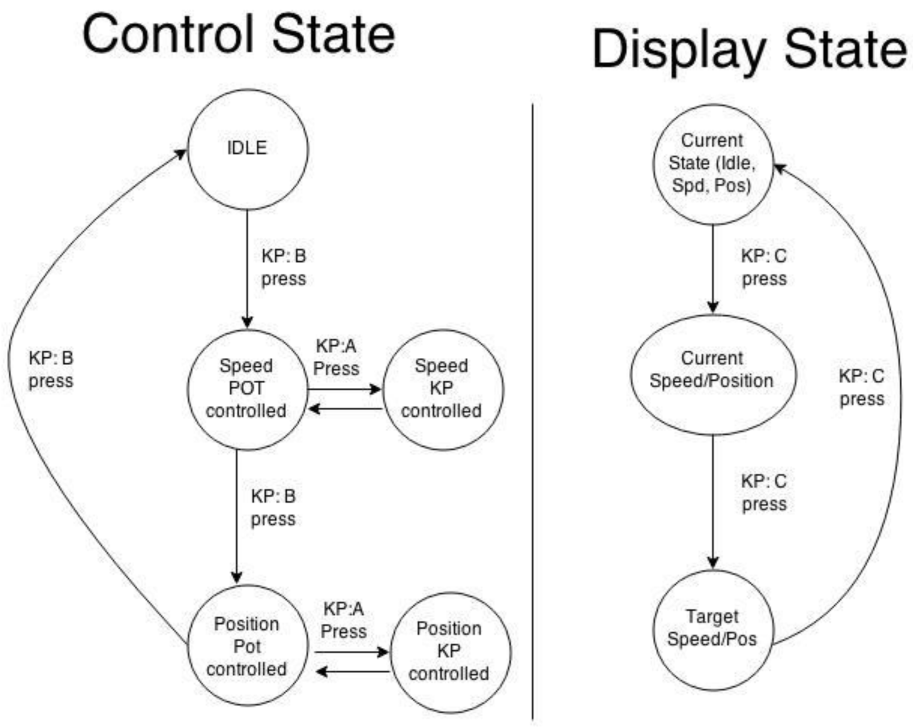
`duty1 = 87-correctedvalue -` ~~`ki*integrator*0`~~ `+ derivitive;`

`kp = 0.4;`

`kd = 0.007;`

For position control, we did a similar setup in terms of the PID equation to the speed control. We were able to implement the kp and the kd quite easily to produce a quick and stiff reaction, but the integral term would always result in the fuse burning out far too easily.

State Machine



Results

The result of our project was that we successfully completed a form of brushless motor PD (not PID) control that is somewhat oscillatory. Due to the timing constraint and the amount of fuses that we went through, we were not able to successfully integrate an integrating term in our control equation. We ran into a lot of trouble trying to add a integration portion into our control system due to it often trying too hard to correct the error, causing too much current to flow through the system. Even with safety precautions such as duty cycle limiters and values that would limit our error correction terms, the integrating portion of the equation still caused us to burn more fuses than practical for implementing a system to work in the small period of time given to us to program the system. Even without the integrating portion, our system itself is quite stiff in its response. The positioning response is extremely stiff and will attempt to harshly correct the position error even if it is only 15 or so degrees off.

Code:

http://pastebin.com/r2yaGbUQ