ECEN301 Embedded Systems Lab 3
PWM, LDRs, Interrupts & Timers Submission

Daniel Eisen 300447549

September 10, 2020

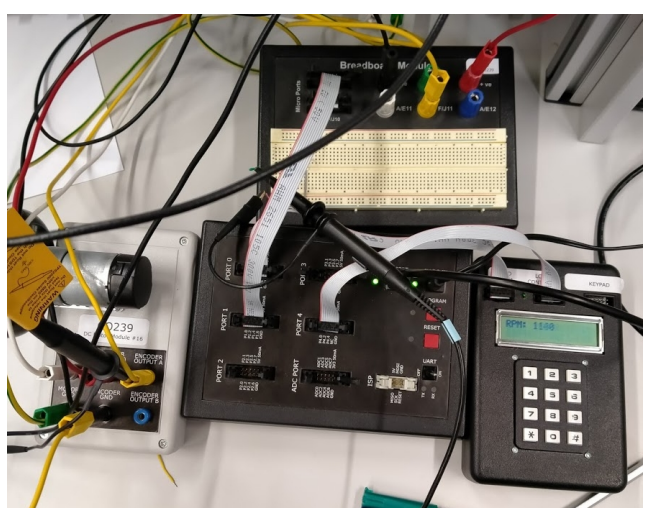
1 Objectives

This lab covered both the introduction to internal and externally triggering interrupts, for external measurements. As well as use of the timer 0 peripheral in creating a better, more stable delay functions that doesn't really of idle machine cycles. Refines use of ISR's and using different interrupt modes and inputs

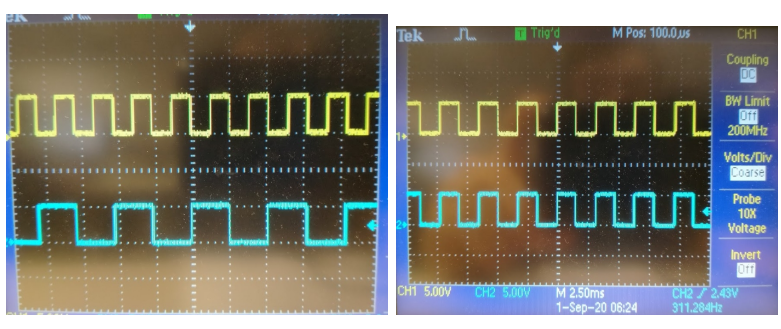
2 Methodology

2.1 Capture Interrupts

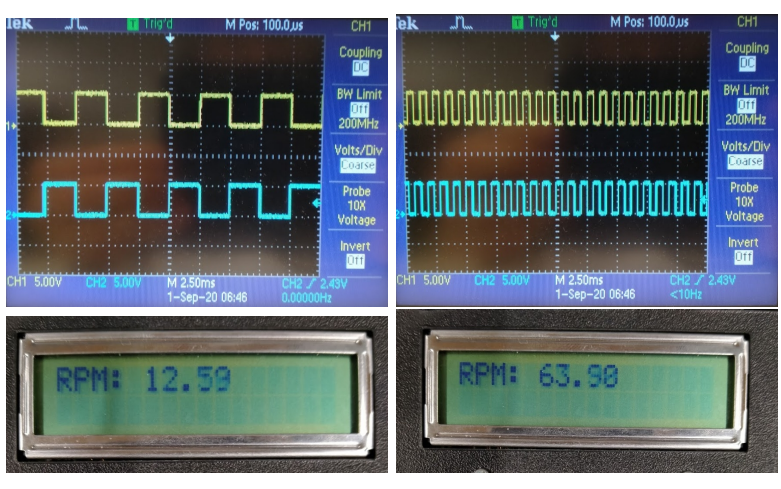
To measure the speed of the motor it first must be powered, this was just done with simple voltage control. The onboard encoder output was then connected to the input capture-compare pin so the generated pulses could trigger an interrupt.



The software can then be configured to generate an interrupt from the external pulses, either on a single edge (left) resulting in a halved signal or on both the rising and falling edge. This is configured with CAPMn.5&4.

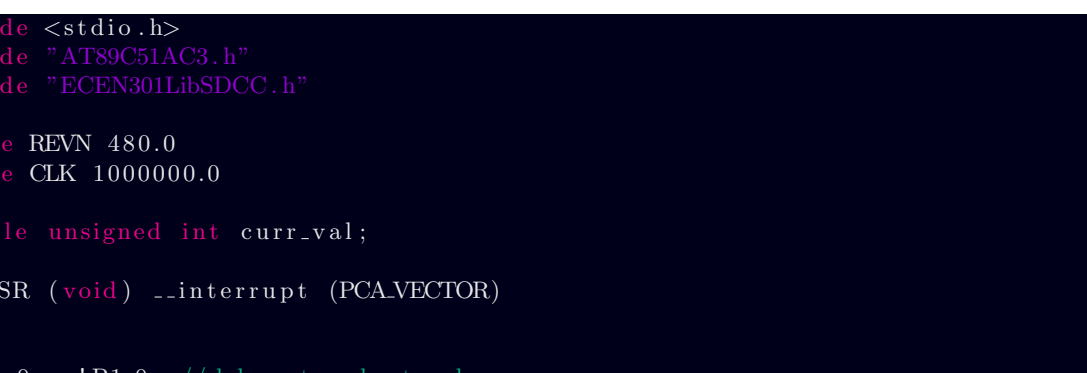


With the PCA timer enabled, the ISR can then use it to sample the timer between interrupts (ie pulses) and use the known values of the counter clock speed, encoder steps per revolution, and counter value to determine a revolutions per minute value to display to the LCD.




```c
#include <stdio.h>
#include "AT89C51AC3.h"
#include "ECEN301LibSDCC.h"

#define REVN 480.0
#define CLK 1000000.0

volatile unsigned int curr_val;

void ISR (void) __interrupt (PCA_VECTOR)
{

    P1_0 = !P1_0; //debug toggle to show
    curr_val = (CCAP0H << 8) | CCAP0L; //read the timer value
    CCON &= 0b11111110; //clear flag

    CH = 0; //reset the values
    CL = 0;
}

void main(void)
{
    IEN0 =   0b11000000; //enable interrupts all and PCA
    //setup capture
    CCON =   0b01000000; //set CR to turn in PCA timer
    CCAPM0 = 0b00110001; //set to double edge trigger (match encoder output), and
    enable CCF0 intrpt bit

    initLCD();
    char str[16];
    while (1) {
        /*  Encoder has 480 holes per rev (REVN) that trigger the interrupt
            Clock speed is at 1Mhz so clock time is 1/1Mhz in seconds (1/60 to get to
    minutes)
            curr_val stores the timer count at the time of interrupt*/
        clearLCD();
        unsigned int rev_m = (60.0 / (2.0*REVN*(float)curr_val*(1.0/CLK)))*100.0;
        sprintf(str, "RPM: %u.%u", rev_m/100, rev_m%100);
        writeLineLCD(str);
        delay(10000);
    }
}
```

2.2 Timer 0 and Calibrated Delay



To improve upon the previous delay methods of just entering a set number of empty loops, the timer peripheral can be used to set an actual time value for the delay. Above shows the results of setting to 10mS and 2mS and below the is the implementation.
To achieve a set time of a 1mS accurate the timer (in 8 bit mode) was preloaded at 6 ($2^8 - 6 = 250uS$) assuming that the clock is at 1Mhz. I found that mine was in X2 mode and running at 2Mhz for had to count 8 overflows to get to 1mS.

```c
#include <stdio.h>
#include "AT89C51AC3.h"
#include "ECEN301LibSDCC.h"
/*
    timer clk is 2Mhz in X2 mode
    timer increment is 0.5uS (1/2Mhz)
    For a time delay of 100 uS the timer has to make 200 increments.
    2^8 = 256 is the maximum number of counts possible for a 8 bit timer.
    If the timer is started at 6, then the overflow time is 125uS (0.5*250) to get a 1
   mS value with x8 scale
*/

volatile unsigned long overflows = 0;

void ISR (void) __interrupt (TF0_VECTOR)
{
    ++overflows;
}

void timer_delay(unsigned long t) //t in mS
{
    overflows = 0;
    TR0 = 1; //turn on timer 0
    while(overflows/8 < t);
    TR0 = 0;
}

void main(void)
{
    IEN0 = 0b10000010; //enable interrupts all and TIMER0
    TMOD |= 0b00000010; //sets in 8bit mode

    TL0 = 0; //clear timer 8bit
    TH0 = 6; //preset auto-reload to 6, ie for 250 increments

    while (1) {
        P1_0 = 1;
        timer_delay(500);
        P1_0 = 0;
        timer_delay(500);
    }
}
```

3 Questions

1. ***Given that you know the number of seconds in a minute; the number of counts occurring per second, the number of counts occurring per hole and the number of holes per revolution, find a formula for the RPM, the revolutions per minute.***

   Known values are:

   480 holes per revolution of the encoder (double if using double edge trigger), PCA clock speed of **1Mhz** (1/6 of cpu clock by default) so 1uS increment time, and N as the increment count between interrupts. Therefore...

   $$RPM = \frac{60}{2 \cdot 480 \cdot N \cdot 1 \times 10^{-6}}$$

2. ***Can you make the timer more accurate? To the millisecond? To the microsecond? What are the advantages/disadvantages of doing this?***

   The current implementation is currently millisecond accurate and relies and counting a set amount of timer overflows to achieve higher time intervals. To get a 'faster' delay function, say a single timer increment accurate the timer can be set into 16 bit mode and the timer duration is set by preloading the timer with a determined value and the overflow defined the end of the delay. Doing this increases the accuracy but in order to uncap the upper limit of $2^16 \times 1uS$ a second timer is needed.