Xiaobin Zhuang - 300519184
ECEN301 Lab 4
26/08/2020

Objectives

The lab introduced the example of using Capture mode of PCA counts and Timer 0. The purpose of this is to investigate the capabilities of different timers.

Methodology

Capture Interrupts

In this session of lab, I used PCA capture mode module 0. To use Capture mode in PCA module, it needed to be enabled, so CCAPM0 = 0x11. And of course, all interrupt and PCA interrupt needed to be enabled (EA = 1, EC = 1). Also, turn PCA counter on(CCON = 0x40). Set overflow flag off (CF = 0).

During interrupt routine, I set EA and EC back to 0 to prevent a new interrupt being trigged when counting the motor speed. If both the PCA counter flag and capture flag are on, overflow++. Save value into 'curr' (current state of PCA count), compare with previous state (count = cure - prev).

To find the relationship between 'count' and 'rpm', I counted rpms by my own with corresponding 'count' value, then made a table below:

count	rpm
11000	27
4600	65
2900	103
2100	143

Table 1

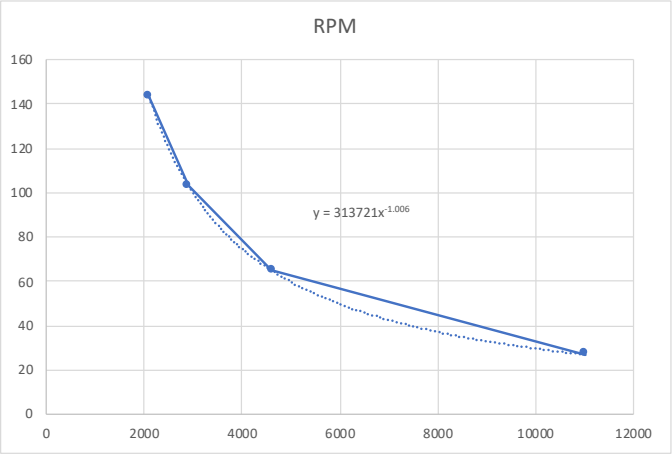
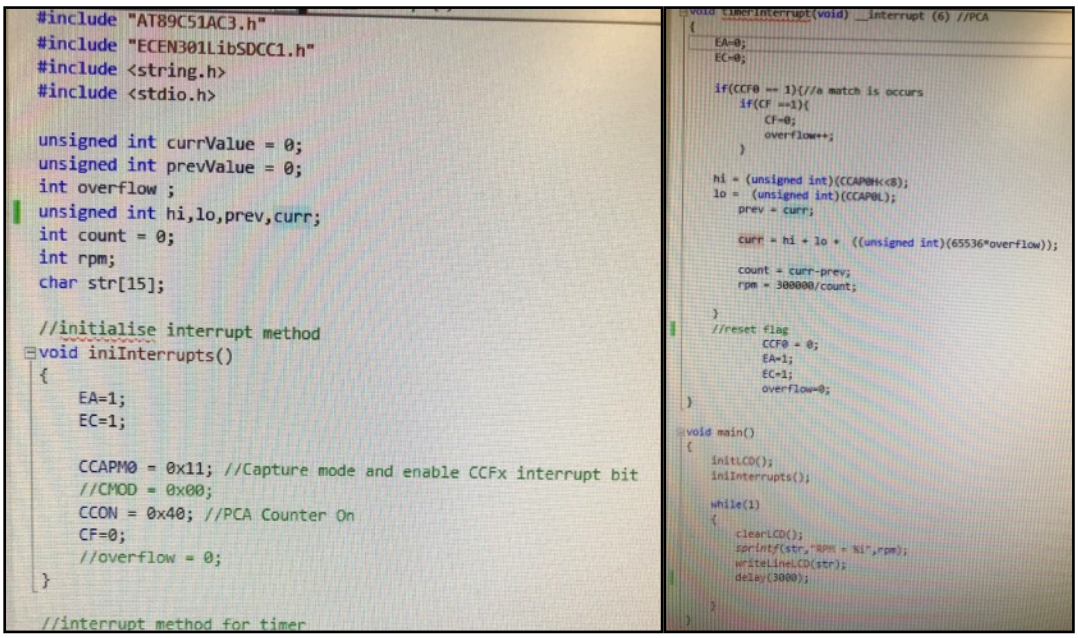


Fig. 1

I made an Excel graph (figure 1) basic on the data in table 1, and output a trend line.
$Y = 313721x^{-1.006}$; Y is rpm, x is count. To make trend line closer to the real value, I estimated that y = 300000x^(-1) = 300000/x

At last of interrupt routine, reset the flag and overflow, re-enable EA and EC to wait for next interrupt.
Program code shown in figure 2.



```
#include "AT89C51AC3.h"
#include "ECEN301LibSDCC1.h"
#include <string.h>
#include <stdio.h>

unsigned int currValue = 0;
unsigned int prevValue = 0;
int overflow ;
unsigned int hi,lo,prev,curr;
int count = 0;
int rpm;
char str[15];

//initialise interrupt method
void iniInterrupts()
{
    EA=1;
    EC=1;

    CCAPM0 = 0x11; //Capture mode and enable CCFx interrupt bit
    //CMOD = 0x00;
    CCON = 0x40; //PCA Counter On
    CF=0;
    //overflow = 0;
}

//interrupt method for timer
```

```
void timerInterrupt(void) __interrupt (6) //PCA
{
    EA=0;
    EC=0;

    if(CCF0 == 1){//a match is occurs
        if(CF ==1){
            CF=0;
            overflow++;
        }

    hi = (unsigned int)(CCAP0H<<8);
    lo =  (unsigned int)(CCAP0L);
        prev = curr;

        curr = hi + lo +  ((unsigned int)(65536*overflow));

        count = curr-prev;
        rpm = 300000/count;

    }
    //reset flag
            CCF0 = 0;
            EA=1;
            EC=1;
            overflow=0;
}

void main()
{
    initLCD();
    iniInterrupts();

    while(1)
    {
        clearLCD();
        sprintf(str,"RPM = %i",rpm);
        writeLineLCD(str);
        delay(3000);

    }
}
```

Fig. 2

Hardware connection shown in figure 3.
Note that for PCA module 0, external I/O pin is P1.3

After I set up everything, the issue I met was that if motor ran in low speed, rpm could be measured and shown in LCD display (shown in figure 3). However, if the motor ran in high speed, the rpm couldn't be displayed on LCD.

It wasn't the problem of code, because when I changed the display from 'rpm' to 'count',everything went well. No matter the motor ran in low speed or high speed, 'count' could be displayed. So I could't find the solution of this issue.

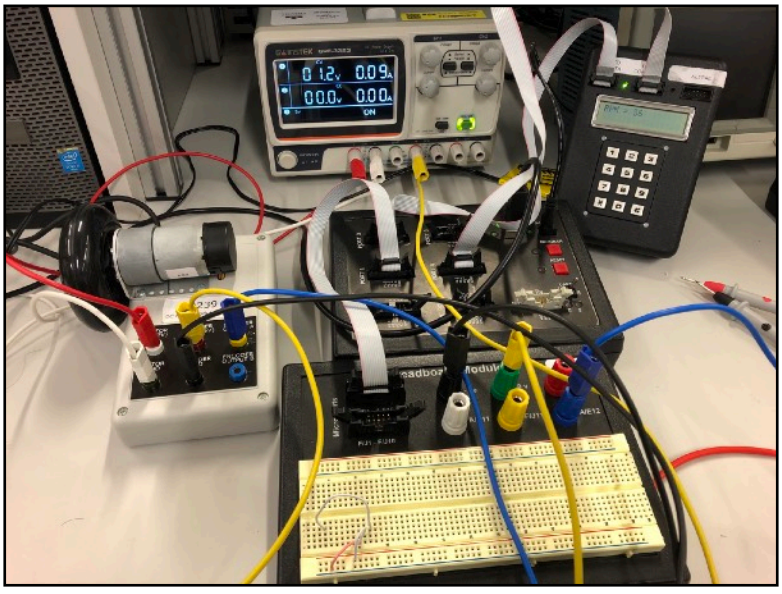


Fig. 3

Timer 0 and the calibrated delay

Software setting:

In the initTimer() function,

- Setup timer 0 as an 8 bit reload timer

1	M10	**Timer 0 Mode Select Bit** M10 M00 Operating mode 0 0 Mode 0: 8-bit Timer/Counter (TH0) with 5-bit prescaler (TL0). 0 1 Mode 1: 16-bit Timer/Counter. 1 0 Mode 2: 8-bit auto-reload Timer/Counter (TL0) (2) 1 1 Mode 3: TL0 is an 8-bit Timer/Counter TH0 is an 8-bit Timer using Timer 1's TR0 and TF0 bits.
0	M00	

Fig. 4

According to the red underline on figure 4, to set 8-bit reload timer, set TMOD = 0x2;

- Set TH0 and TL0 = 0x9B

During the lab, I found that the minimum delay of timer is 1.085us. If set both TH0 and TL0 = 1, the period of delay would be 255-1.085 ≈ 255us. So I set them into 9B(155), the output would be 255 - 9B ≈ 100us.

According to my test result, repeat initTimer() for 10000 times in an interrupts to get 500ms rate. (interrupt (1): Timer 0)

So when a delay started, it would not stop, until count = 10000. After 10000 counts, reset the interrupt routine then repeat it again.

Program code shown in figure 5.

```
*/
#include "AT89C51AC3.h"
#include "ECEN301LibSDCC1.h"

int count;
int delayf =0;
int countwant;

void initTimer(){
    TMOD = 0x2;
    TH0 = 0x9B;
    TL0 = 0x9B;
    EA =1;
    ET0 =1;
    TR0 =1;
}

void startDelay(unsigned int mesc){
    delayf = 1;//start delay
    count = 0;
    countwant = mesc;
}

void timerintt (void) __interrupt (1){
    count ++;
    TF0 = 0;//reset interrupt flag
    }

void main()
{
    startDelay(10000);
    initTimer();

    P1 = 0x00;

    if(delayf==1){
        while(1)
    {
            if(count == countwant){
                P1 = ~P1;
                count=0;
            }
    }

    }
```

Fig. 5

Hardware setting:

Connecting I/O module(Digital in) to Microcontroller(Port 1). I/O module power by 15V DC. Which shown in figure 6.

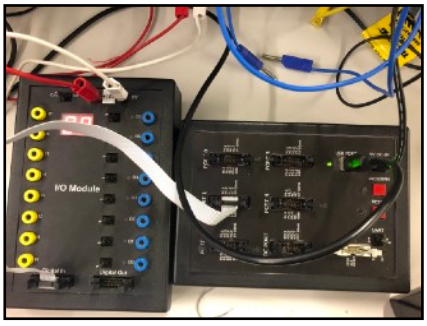


Fig. 6

Outcome:
The signal of square wave shown in oscilloscope (figure 7). One square had 500ms period, when signal was HI, LED on; when it was LO, LED off. Which fit the requirement (500ms on, 500ms off)

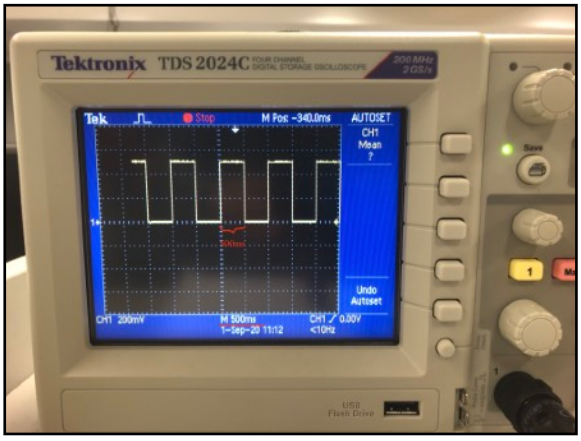


Fig. 7

Questions:

Q1: RPM = 300000/count.
Enable CAPPn, capture function triggered by a positive edge; enable CAPNn, capture function triggered by a negative edge.
Using 16-bits software mode to count from both encoders.
Q2: by changing from startDelay(10000) to startDelay(1), timer can be microsecond accurate. (100us on, 100us off). By making the value of TH0 and TL0 larger, timer can be more accurate than 100us.
Advantage: repeat the routine in short time, meet different requirement of timer delay.
Disadvantage: if the timer is too short, new interrupt might happen while the previous interrupt is still running.