Xiaobin Zhuang - 300519184

ECEN301 Lab 3

26/08/2020

Objectives

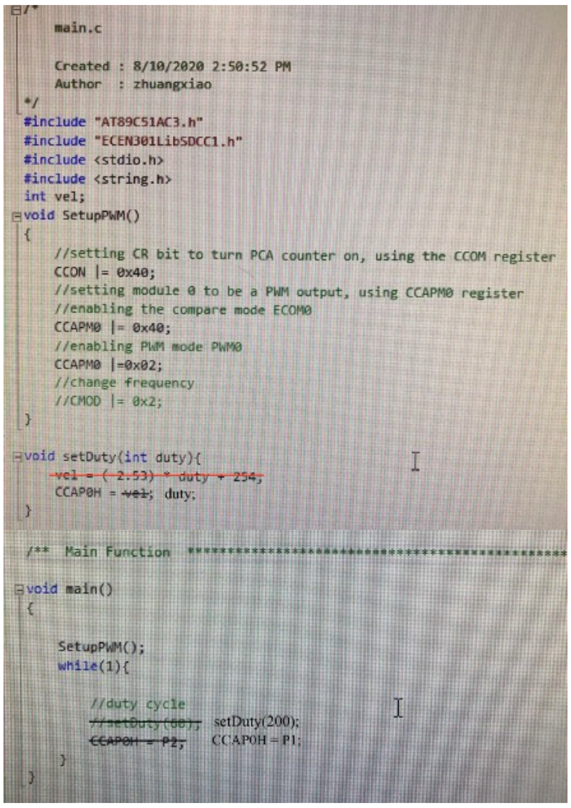
The purpose of this lab session is to be familiar with the PWM output, External interrupts, The interval timers and Internal interrupts.

Methodology

PWM output

In the program, to set up PWM, first using the CCON register, setting CR bit to turn PCA counter on. Second, using CCAPM0 register to set module 0 to be a PWM output. Last, enable PWM mode. Next, assigning duty to 200. The code shown in figure 1.

The External I/O pin of Module 0 is P1.3. So I assigned CCAP0H to P1.3 and set up the connection which shown in figure 2.



```
/*
    main.c

    Created : 8/10/2020 2:50:52 PM
    Author  : zhuangxiao
*/
#include "AT89C51AC3.h"
#include "ECEN301LibSDCC1.h"
#include <stdio.h>
#include <string.h>
int vel;
void SetupPWM()
{
    //setting CR bit to turn PCA counter on, using the CCOM register
    CCON |= 0x40;
    //setting module 0 to be a PWM output, using CCAPM0 register
    //enabling the compare mode ECOM0
    CCAPM0 |= 0x40;
    //enabling PWM mode PWM0
    CCAPM0 |=0x02;
    //change frequency
    //CMOD |= 0x2;
}

void setDuty(int duty){
    ~~vel = ( 2.53) * duty + 254;~~
    CCAP0H = ~~vel~~; duty;
}

/** Main Function ***********************************************

void main()
{

    SetupPWM();
    while(1){

        //duty cycle
        ~~//setDuty(80);~~  setDuty(200);
        ~~CCAP0H = P2;~~   CCAP0H = P1;
    }
}
```

Fig. 1

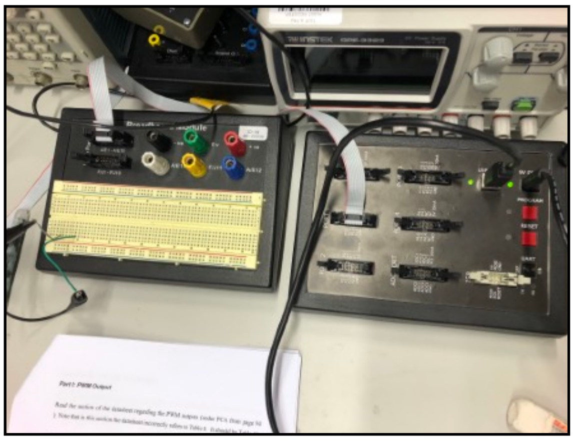


Fig. 2

The output shown in figure 3. When 8-bits register value increased, the square wave would be wider, vice versa. I tasted 5 different values from 0 to 255 and made a line graph which shown in figure 4.

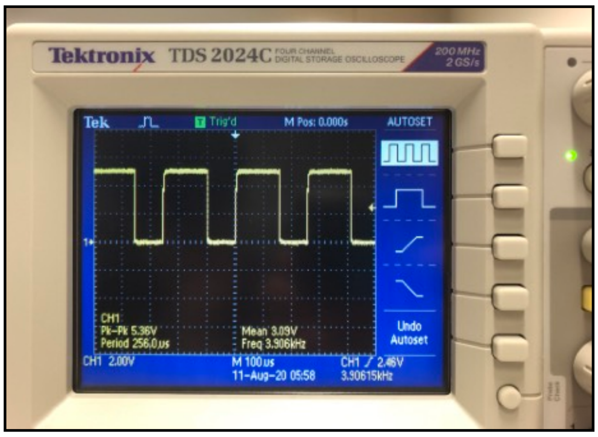


Fig. 3

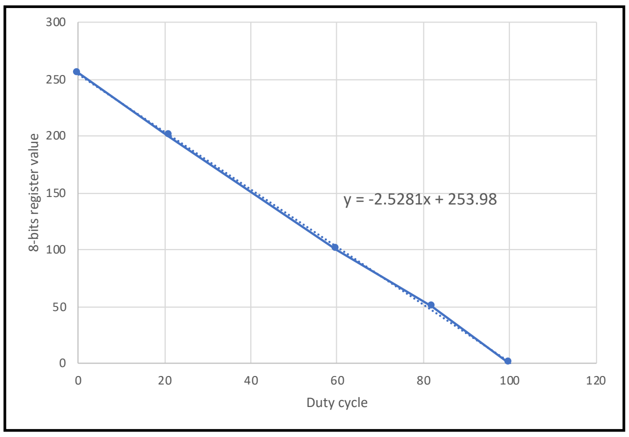
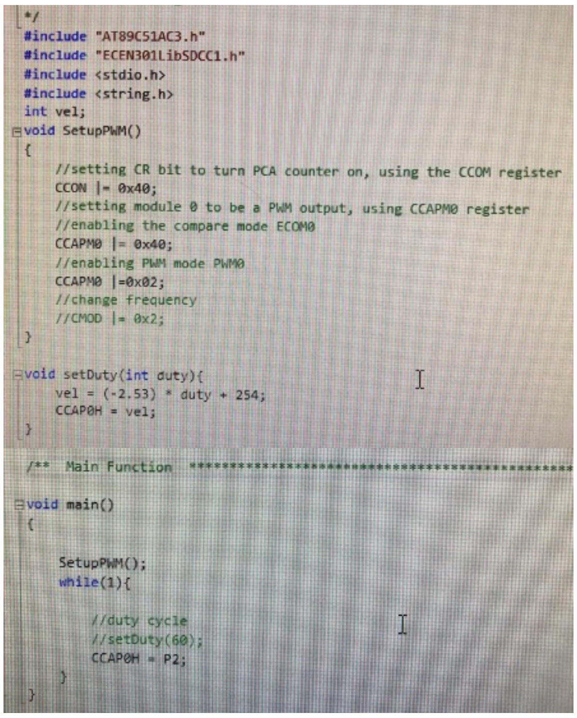


Fig. 4

The method to calculate the duty was: for example, in figure 3 one period took about 2.2 grids length. There were 10 grids length in total, so duty is 2.2/10 * 100% = 22%
According to this line, generated a function: y = -2.53x + 254. With this function, the value could be assigned by I/O module. So the speed of the motor can be changed by using I/O module via connecting to P2. Program code shown in figure 5. Connection shown in figure 6.



```
*/
#include "AT89C51AC3.h"
#include "ECEN301LibSDCC1.h"
#include <stdio.h>
#include <string.h>
int vel;
void SetupPWM()
{
    //setting CR bit to turn PCA counter on, using the CCOM register
    CCON |= 0x40;
    //setting module 0 to be a PWM output, using CCAPM0 register
    //enabling the compare mode ECOM0
    CCAPM0 |= 0x40;
    //enabling PWM mode PWM0
    CCAPM0 |=0x02;
    //change frequency
    //CMOD |= 0x2;
}

void setDuty(int duty){
    vel = (-2.53) * duty + 254;
    CCAP0H = vel;
}

/** Main Function ***********************************************

void main()
{

    SetupPWM();
    while(1){

        //duty cycle
        //setDuty(60);
        CCAP0H = P2;
    }
}
```

Fig. 5

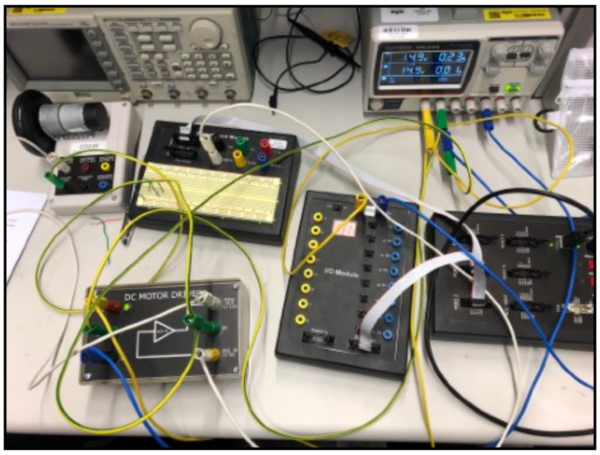


Fig. 6

LDRs and External Interrupts

Resistance range of LDR

Light Condition	Resistance (k ohm)
Full light	0.4
Room light	2.5
Dark	43
No light	Open

Table 1

About the LM311 set up, used two voltage dividers as inputs of LM311. In input 3, 2.5V divided by two 1k ohm resistances. In input 2, the other 2.5V divided by a LDR and resistance box. In order to give LDR a big range of voltage (0V-4V), I set resistance box as 30k ohm.
On the page 3 of AT89C51AC3 datasheet, the External Interrupt 0 (INT0) is in P3.2, so this pin would be the output of LM311.
To specify the circuit, I draw a diagram shown in figure 8

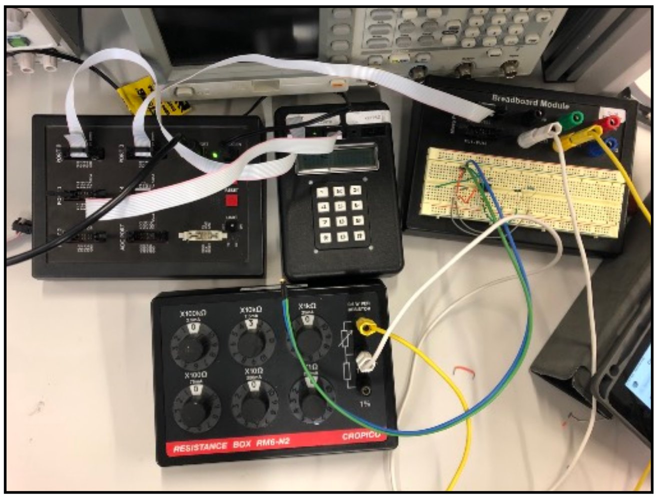


Fig. 7

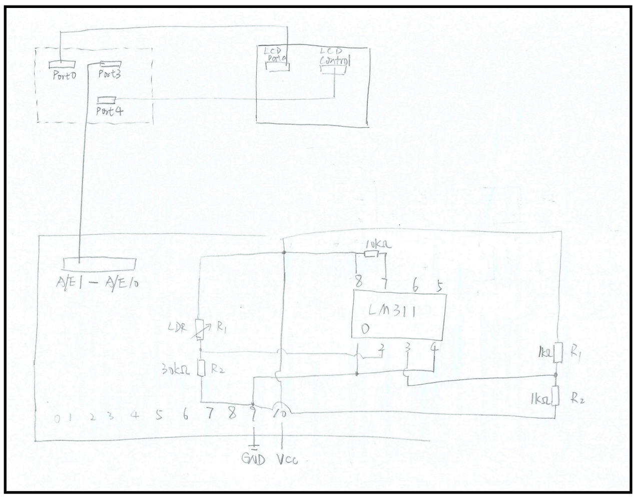
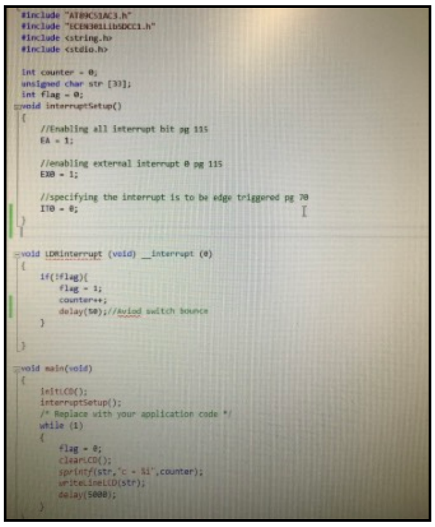


Fig. 8

In the program code, IT0 = 0, mean level trigger was enable. When signal was high, it would trigger a count.
The issue I met was a count bounce. When there was a trigger, counter would count more than one. To solve this, I set flag = 1 in the interrupt program to make sure the program would stay in interrupt until it has done. Which shown in figure 9.
In terms of level trigger, when LDR was covered (LO), there would be no count, when it wasn't (HI), there would be a count.



```
#include "AT89C51AC3.h"
#include "ECEN301Lib5DCC1.h"
#include <string.h>
#include <stdio.h>

int counter = 0;
unsigned char str [33];
int flag = 0;
void interruptSetup()
{
    //Enabling all interrupt bit pg 115
    EA = 1;

    //enabling external interrupt 0 pg 115
    EX0 = 1;

    //specifying the interrupt is to be edge triggered pg 70
    IT0 = 0;
}

void LDRinterrupt (void) __interrupt (0)
{
    if(!flag){
        flag = 1;
        counter++;
        delay(50);//Aviod switch bounce
    }

}

void main(void)
{
    initLCD();
    interruptSetup();
    /* Replace with your application code */
    while (1)
    {
        flag = 0;
        clearLCD();
        sprintf(str,"c = %i",counter);
        writeLineLCD(str);
        delay(5000);
    }
```

Fig. 9

Questions

Q1: Bit-addressable objects are objects that may be addressed as words or as bits. Only data objects that occupy the bit-addressable area of the 8051 internal memory fall into this category.

Q2:

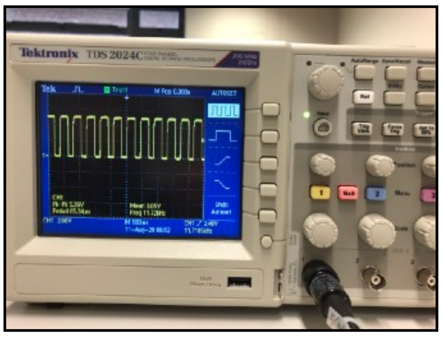
CCAPMn Registers:

Bit Number	Bit Mnemonic	Description
7	-	**Reserved** The Value read from this bit is indeterminate. Do not set this bit.
6	ECOMn	**Enable Compare Mode Module x bit** Clear to disable the Compare function. Set to enable the Compare function. The Compare function is used to implement the software Timer, the high-speed output, the Pulse Width Modulator (PWM) and the WatchDog Timer (WDT).
5	CAPPn	**Capture Mode (Positive) Module x bit** Clear to disable the Capture function triggered by a positive edge on CEXx pin. Set to enable the Capture function triggered by a positive edge on CEXx pin
4	CAPNn	**Capture Mode (Negative) Module x bit** Clear to disable the Capture function triggered by a negative edge on CEXx pin. Set to enable the Capture function triggered by a negative edge on CEXx pin.
3	MATn	**Match Module x bit** Set when a match of the PCA Counter with the Compare/Capture register sets CCFx bit in CCON register, flagging an interrupt.
2	TOGn	**Toggle Module x bit** The toggle mode is configured by setting ECOMx, MATx and TOGx bits. Set when a match of the PCA Counter with the Compare/Capture register toggles the CEXx pin.
1	PWMn	**Pulse Width Modulation Module x Mode bit** Set to configure the module x as an 8-bit Pulse Width Modulator with output waveform on CEXx pin.
0	ECCFn	**Enable CCFx Interrupt bit** Clear to disable CCFx bit in CCON register to generate an interrupt request. Set to enable CCFx bit in CCON register to generate an interrupt request.

COMD Registers:

Bit Number	Bit Mnemonic	Description
7	CIDL	**PCA Counter Idle Control bit** Clear to let the PCA run during Idle mode. Set to stop the PCA when Idle mode is invoked.
6	WDTE	**WatchDog Timer Enable** Clear to disable WatchDog Timer function on PCA Module 4, Set to enable it.
5	-	**Reserved** The value read from this bit is indeterminate. Do not set this bit.
4	-	
3	-	
2-1	CPS1:0	**EWC Count Pulse Select bits** CPS1 CPS2 Clock source 0 0 Internal Clock, FPca/6 0 1 Internal Clock, FPca/2 1 0 Timer 0 overflow 1 1 External clock at ECI/P1.2 pin (MAX. Rate = FPca/4)
0	ECF	**Enable PCA Counter Overflow Interrupt bit** Clear to disable CF bit in CCON register to generate an interrupt. Set to enable CF bit in CCON register to generate an interrupt.

Q3: 8 bits, 2^8 = 256. So T = 256us, f = 1/T = 1/256us = 1/256 * 10^6 = 3.9kHz. Which is same with the display on oscilloscope. (refers to figure 3)
For changing the frequency, add code CMOD | = 0x2 (CPS1 = 0, CPS2 = 1, 1/2 the PCA clock frequency.) It was 1/6 the PCA clock frequency before, so the frequency would be 3 times bigger. (≈ 11.7kHz)
The figure on the right shows frequency = 11.72kHz, which is around 3 times bigger than 3.9kHz.



Q4: Yes, the switch bounce occurred. I set flag = 1 in the interrupt program to make sure the program would stay in interrupt until it has done. Then set flag = 0 in the main when interrupt finished.