Laboratory Worksheet #07
PWM - Frequency and Pulsewidth Exercise

On LMS, in the Lab 3 folder, the *worksheet_07.c* code is provided to demonstrate the operation of Pulse Width Modulation (PWM). The Pulse signals are characterized by two attributes, the period (T) of one cycle which is controlled by PCA_Start in the program and the pulse width (PW) which is controlled by PW in the program. A shorter period corresponds to a higher frequency. A high duty cycle, $DC = \frac{PulseWidth}{Period} \times 100\%$, corresponds to a relatively large pulse width.

Exercise 1: PCA When answering the following questions, refer to the *worksheet_07.c* code.

1) What is the size of the PCA counter (in bits)?

16

2) What triggers a count in the PCA?

void PCA_ISR(void) __interrupt 9

3) What is the interrupt priority of the PCA?

9

4) If $PCA\_Start = 47000$, how many counts will occur before the counter overflows? What is the period for this setting (time it takes to count from 47000 until it overflows)?

18536, 10.05 ms

5) Using the above start value, if PW = 3000, what is the pulse width in seconds? What is the Duty Cycle?

1.63ms 16.2%

Now, for a different example, determine PCA_Start and PW for a pulse train with a 3 ms period and a 35% Duty Cycle using a 16 bit counter triggered by SYSCLK/4.

PCA_Start = 48873

PW = 5806

Exercise 2: Hardware

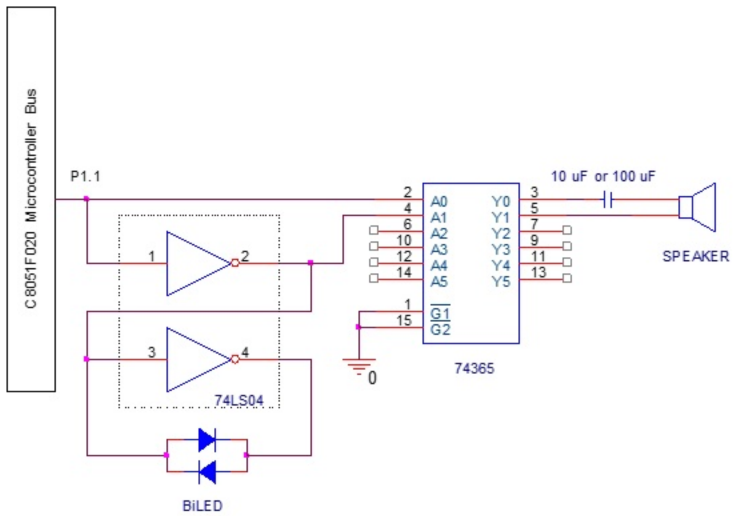

Figure 1: Potentiometer Circuit

1) Build the circuit as shown above. Note: you will need to obtain a speaker from the TAs. Speakers convert periodic electrical signals into corresponding tones. The buzzer you already have in your kits will NOT work since it only needs a voltage to provide a specific tone based on its internal circuit.

2) Download and run the sample program, Worksheet_07.c, from the LMS website.

a) Part A, changing duty cycle

a. Set *PCA_start* to 1000.

b. Change *PW*, the pulsewidth, and observe the effect on the LED..

At one extreme limit of the pulsewidth, the LED will be mostly green in color and at the other extreme limit, it will be mostly red in color. Explain this behavior.

When the duty cycle is high, the output of the upper inverter (2) can be treated as LOW ar

When the duty cycle is low, the output of the upper inverter (2) can be treated as HIGH an

b) Part B, changing duty cycle

a. Set the pulsewidth, *PW*, to 4000.

b. Change the PCA start value, *PCA_starth*, and observe the effect on the speaker output.

At one extreme limit of PCA_start, the frequency will be low and at the other extreme limit, it will be high. Explain this behavior.

The impedance of the capacitor depends on the frequency of the circuit. In low frequency,

In high frequency, the impedance of the capacitor is high, making the current lower and fre

3) When you use the logic probe to test your PWM output, how does the indicator light behave?

Red.

When complete, include Worksheet 7 with your Laboratory 3.1 Pre-lab submission.

Laboratory Worksheet #08
Crossbar Configuration Exercise

This worksheet will help you configure the crossbar for Lab 3, part 1. Refer to the notes from the professor's lecture on the crossbar. Review the example the professor went over in class on the crossbar. Also refer to the Priority Crossbar Decode Table in the handout.

Exercise 1: Reserved pins and Crossbar initialization

This problem is an example only, do not confuse it with the Crossbar configuration for Laboratory 3 (and later laboratories).

1) Assume the following are enabled: UART0, I2C (SMBus0), and the first four capture/compare modules associated with the PCA. Which port pins will be assigned to the following:

TX0 P0.0 ;

RX0 P0.1 ;

SDA P0.2 ;

SCL P0.3 ;

CEX0 P0.4 ;

CEX1 P0.5 ;

CEX2 P0.6 ;

CEX3 P0.7 ;

2) Determine the bit assignments for XBR0. Indicate assigned bits with a 0 or a 1, no bits will be unassigned (no X's).

XBR0 data sheet

bit	7	6	5	4	3	2	1	0
	0	0	1	0	0	1	0	1

3) Determine the command to initialize XBR0 based on the above bit assignments.

XBR0 =0x25 ;

Exercise 2: Laboratory preparation

1) What is the XBR0 setting indicated in Laboratory 3? 0x27

2) For each Laboratory 3.1 version, which Capture Compare Module is assiged.

Speed Controller CCM2;

Steering Servo CCM0;

LED CCM3;

When complete, include Worksheet 8 with your Laboratory 3.1 Pre-lab submission.

EVB Pin			Port Bit	Bit Addresses & Labels
1	2	1.		
		2.		
3	4	3.		
		4.		
5	6	5.		
		6.		
7	8	7.		
		8.		
9	10	9.		
		10.	P1.2	0x92 CEX2
11	12	11.	P1.3	0x93 CEX3
		12.	P1.0	0x90 CEX0
13	14	13.		
		14.		
15	16	15.		
		16.		
17	18	17.		
		18.		
19	20	19.		
		20.		
21	22	21.		
		22.		
23	24	23.		
		24.		
25	26	25.		
		26.		
27	28	27.		
		28.		
29	30	29.		
		30.		
31	32	31.		
		32.		
33	34	33.		
		34.		
35	36	35.		
		36.		
37	38	37.		
		38.		
39	40	39.		
		40.		
41 ⟷ 60				

Software Initializations

A) Port I/0

P1MDOUT = 0x0D;

B) Timers

C) Interrupts

EA = 1;
EIE1 |= 0x08;

D) A/D

E) PCA

PCA0MD =0x81;
PCA0CPM2 = 0xC2;
PCA0CN = 0x40;

F) XBAR

XBR0 = 0x27;

G) I2C

```
compile directives
   #include <studio.h>
   #include <c8051_SDCC.h>
   #include <stdlib.h>
declare global variables
   unsigned int PW_CENTER, PW_MIN, PW_MAX, PW, PCA_START, PCA_COUNTER.
   sbit CF (PCA 0 COUNTER OVERFLOW FLAG)
function prototypes
   void Port_Init(void);
   void PCA_Init(void);
   void XBR0_Init(void);
   void Interrupt_Init(void);
   void Drive_Motor(void);
   void PCA_ISR (void) __interrupt 9;
main function
   declare local variables
      (NONE)
   initialize system, ports and PCA
      Sys_Init();
      putchar(' ');
      Port_Init();
      XBR0_Init();
      Interrupt_Init();
      PCA_Init();
   set PW to 1.5ms and wait for 1s.
   reset PCA_COUNTER.
   begin infinite loop
      Run Drive_Motor(void) to calibrate motor speed
   end infinite loop
end main function

Functions
   void Port_Init()
      set output pin for CEX0, CEX2, and CEX3 in push-pull mode.
   End Port_Init()

   void XBR0_Init()
      configure the crossbar as directed in the labor manual.
   End XBR0_Init()

   void Interrupt_Init()
      Enable general interrupt
      Enable PCA overflow interrupts
   End Interrupt_Init()

   void PCA_Init()
      Enable SYSCLK/12 and enable interrupt.
      Enable CCM2 16bit PWM
      Enable PCA counter
   End PCA_Init()

   void PCA_ISR() __interrupt 9
      Increment PCA_COUNTER to count the number of overflows
```

```
    If PCA interrupt flag is set
        Clear the overflow flag
        Set PCA0 to PCA_START
    End if
    handle other PCA interrupt sources
End PCA_ISR() __interrupt 9

void Drive_Motor()
    declare local variables
        char input
    wait for key stroke
    if 'f', increase PW by 10
        if PW > PW_MAX, limit it to PW_MAX
    else if 'r', decrease PW by 10
        if PW < PW_MIN, limit it to PW_MIN
    else if another letter, give a reminder
    update speed command
        update lo byte of CCM 2
        update hi byte of CCM 2
    print PW
End Drive_Motor()
```

```
compile directives
   #include <studio.h>
   #include <c8051_SDCC.h>
   #include <stdlib.h>
declare global variables
   unsigned int PW_CENTER, PW_LEFT, PW_RIGHT, SERVO_PW, PCA_START.
   sbit CF (PCA 0 COUNTER OVERFLOW FLAG)
function prototypes
   void Port_Init(void);
   void PCA_Init(void);
   void XBR0_Init(void);
   void Interrupt_Init(void);
   void Steering_Servo(void);
   void PCA_ISR (void) __interrupt 9;
main function
   declare local variables
      char input
   initialize system, ports and PCA
      Sys_Init();
      putchar(' ');
      Port_Init();
      XBR0_Init();
      Interrupt_Init();
      PCA_Init();
   print beginning message.
   set SERVO_PW = PW_CENTER, 1.5ms.
   begin infinite loop
      Run Turn_Steer(void) to vary the steering angle.
   end infinite loop
end main function

Functions
   void Port_Init()
      set output pin for CEX0, CEX2, and CEX3 in push-pull mode.
   End Port_Init()

   void XBR0_Init()
      configure the crossbar as directed in the labor manual.
   End XBR0_Init()

   void Interrupt_Init()
      Enable general interrupt
      Enable PCA overflow interrupts
   End Interrupt_Init()

   void PCA_Init()
      Enable SYSCLK/12 and enable interrupt.
      Enable CCM0 16bit PWM
      Enable PCA counter
   End PCA_Init()

   void PCA_ISR() __interrupt 9
      Increment PCA_COUNTER to count the number of overflows
```

```
    If PCA interrupt flag is set
        Clear the overflow flag
        Set PCA0 to PCA_START
    End if
    handle other PCA interrupt sources
End PCA_ISR() __interrupt 9

void Steering_Servo()
    declare local variables
        char input
    wait for key stroke
    if 'r', increase PW by 10
        if SERVO_PW > PW_RIGHT, limit it to PW_RIGHT
    else if 'l', decrease PW by 10
        if SERVO_PW < PW_LEFT, limit it to PW_LEFT
    print SERVO_PW
    update Servo command
        update lo byte of CCM 0
        update hi byte of CCM 0
End Steering_Servo()
```