Lab 3 (part 1): Pulse Width Modulation

Preparation

Reading

Lab Manual

Chapter 4 - The Silicon Labs C8051F020 and the EVB (sections on the *C8051 timer functions, interrupts, and the programmable counter array*)

Chapter 5 - Circuitry Basics and Components (The buffer)

Chapter 6 - Motor Control (DC Motors)

Objectives

In Lab 3, your team of three students will each have an individual assignment. One partner will be responsible for the ultrasonic ranger and the drive motor. The second partner will be responsible for the electronic compass and the steering servo. If the team is a group of three, the third partner will be responsible for the light sensor and a blinking LED. As with Lab 1, the 3 individual assignments should **not** be combined into a single file and each should be checked off separately. This should continue through all 3 parts of Lab 3.

General

Implement the servo, speed controller or LED hardware and write a calibration/test code. The code should allow the user to create different pulsewidths for the servo, speed controller or LED.

Hardware

1. Ultrasonic Ranger Task – Configure the speed controller to control the car speed.
2. Electric Compass Task – Configure the servo to control the car steering angle.
3. LED Task – Set up an LED as a duty cycle indicator.

Software

1. Ultrasonic Ranger Partner - Develop software to allow the user to set the drive motor speed by varying the pulsewidth of the signal being sent to the drive motor. When the user presses "f", the car moves "f"aster and when the user presses "s", the car moves "s"lower.
2. Electric Compass Partner - Configure the servo to control the car steering angle. Develop software to allow the user to calibrate the steering by finding the pulsewidth for the center position as well as the pulsewidths for extreme left and extreme right positions. When the user presses "l", the car turns "l"eft and when the user presses "r", the car turns "r"ight.
3. LED Partner - Develop software to dim or brighten a LED by varying the pulsewidth of the signal being sent to the LED. When the user presses "d", the LED becomes "d"immer and when the user presses "b", the LED becomes "b"righter.

Motivation

You will learn to use the electronic compass and the ultrasonic ranger on the *Smart Car*. Later, you will be using these sensors on the *Gondola*. Since these labs form an evolutionary developmental sequence for the Gondola, you may wish to familiarize yourself with the objectives of all remaining labs as an aid in planning for each.

An individual partner of the 3-student teams will be responsible for one task - someone will be working with the electronic compass, another will be working with the ultrasonic ranger, and the third will also be working with the ultrasonic ranger. The team will merge the software for the electronic compass and ultrasonic ranger into a single code controlling the speed and direction of the *Smart Car* in Laboratory 4.

Lab Description and Activities

1. The laboratory 3 projects will be merged in Lab 4. Given this, it is in your best interest to modularize your codes and have discussions with the other members of your team to make sure that there won't be any software or hardware conflicts. This includes such issues as repeated variable names and output pin assignments.
2. Refer to the sample code. Modify this code so that you have configured the PCA to reflect which capture compare you will use. You will also need to modify the other functions so that you can produce the appropriate pulsewidth output.

Hardware

The hardware schematics for the electronic compass and the ultrasonic ranger are shown in *Figure 3-1.1* and *Figure 3-1.2*, respectively. The ranger and compass will be mounted on the small proto-board on the front of the car. Compass developers will complete the circuit in *Figure 3-1.1* and the ranger developer will complete the circuit in *Figure 3-1.2*. The servo motor is hard-wired for +5V and ground. The control signals for the servo and the speed controller are outputs on the Port pin determined by XBR0. A 74F365 buffer chip serves as the interface between the EVB and the servo motor/speed controller.

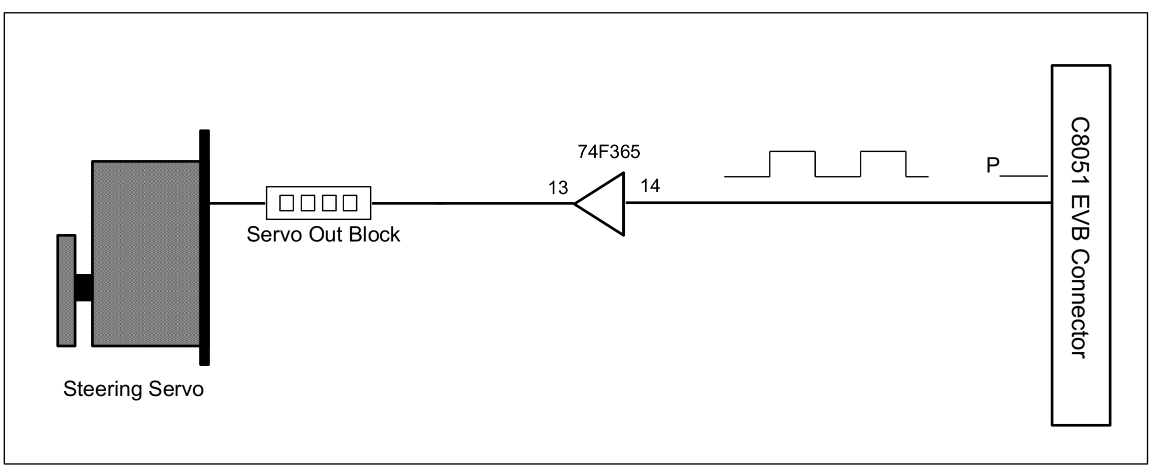

Figure 3-1.1 - Car steering servo motor control circuitry for Lab 3 (part 1) - Electronic Compass

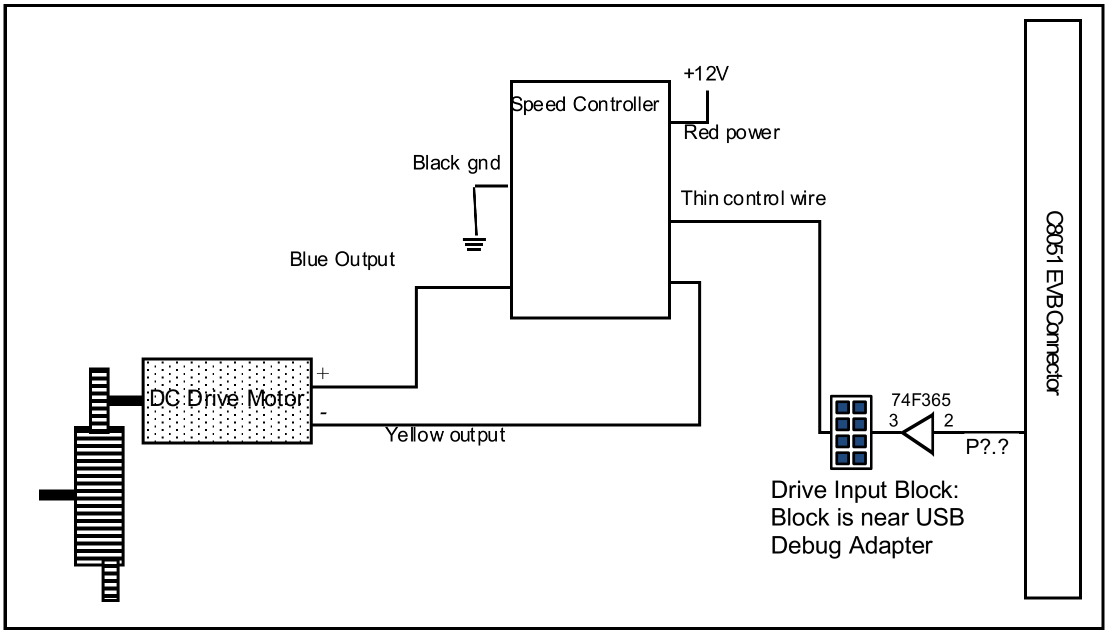

Figure 3-1.2 - Car drive-motor circuitry for Lab 3 (part 1) - Ultrasonic Ranger.

NOTE: the Drive Input Block is mounted next to the USB Debug Module, the connection used to download code into the 8051, not the 12V battery-monitoring block mounted near the +5V and Ground blocks.

Software - Ultrasonic Ranger

The Ultrasonic Ranger Partner will use a Crossbar setting XBR0=0x27 with a pulsewidth signal on CEX2

1. Write code that sends a pulse width modulated signal to the speed controller. The period must be 20ms and the pulsewidth must be initialized to be 1.5ms. Configure the PCA to use a 16-bit counter with SYSCLK/12.
2. In order to properly initialize the speed controller, the code must set the pulsewidth to the 1.5ms value and leave it at that value for about 1 second. To implement one second elapsed time, put a counter in the PCA ISR, similar to what was done in Laboratory 2, except use PCA overflows to trigger the count. A count of 50 20ms periods is 1s. During this initialization time, the speed controller should make a whining sound. Contact a TA if it doesn't. The 1.5 ms pulsewidth initially puts the car in neutral.
3. After initialization, the code prompts the user to adjust the pulsewidth to control the motor speed. The code must limit the minimum and maximum pulsewidths to be 1.1ms and 1.9 ms respectively.
4. For a longer pulsewidth, the car should be move forward and for a shorter pulsewidth, it should move backward. If the opposite happens, then switch the blue and yellow wires at the connection block on the car.
5. As a user, adjust the pulsewidth to confirm proper operation; that the motor can be run at variable speeds from full reverse to full forward and that the motor is off when the pulsewidth is 1.5ms.

Software - Electronic Compass Partner

The Electronic Compass Partner will use a Crossbar setting XBR0=0x27 with a pulsewidth signal on CEX0

1. Write code that sends a pulsewidth modulated signal to the servo. The period must be 20 ms and the pulsewidth must be initialized to be 1.5 ms. Configure the PCA to use a 16-bit counter with SYSCLK/12. <u>Initial estimates of the left and right limits are 0.9 ms and 2.1 ms]</u>
2. The code must then receive inputs from the user to either increase or decrease the pulse width. A step size of 10 is reasonable when changing the pulsewidth.
3. The code prompts the user to adjust the pulsewidth until the steering is centered. The code will print this value and store it as a variable.
4. The code then prompts the user to adjust the pulsewidth until the car steering is at the maximum left. This should be a pulsewidth that doesn't stress the steering mechanism. The code prints this value and stores it as the minimum pulsewidth.
5. The code then prompts the user to adjust the pulsewidth until the car steering is at the maximum right. Again the code prints this value and stores it as the maximum pulsewidth.
6. After determining the minimum pulsewidth (left turn), center pulsewidth, and the maximum pulsewidth (right turn), the code will allow the user to vary the steering angle but will limit the pulsewidths to stay within the range determined by the minimum and maximum found in Steps 4 and 5 above.

Software - LED Partner

The Ultrasonic Ranger Partner will use a Crossbar setting XBR0=0x27 with a pulsewidth signal on CEX3

1. Write code that sends a pulsewidth modulated signal to an LED. The period must be 20ms and the pulsewidth must be initialized to be 10 ms. Configure the PCA to use a 16-bit counter with SYSCLK/12. <u>The pulsewidth should never be less than 1 ms or greater than 19 ms.</u>
2. The code must then receive inputs from the user to either increase or decrease the pulse width. A step size of 200 is reasonable when changing the pulsewidth.
3. The code prompts the user to find a threshold maximum pulsewidth such that the LED is considered to be the dimmest. The code prints this value and stores it as the minimum pulsewidth.
4. The code then prompts the user to find a threshold minimum pulsewidth such that the LED is considered to be the brightest. The code prints this value and stores it as the maximum pulsewidth.
5. After determining the minimum pulsewidth (dimmest), center pulsewidth, and the maximum pulsewidth (brightest), the code will allow the user to vary the brightness but will limit the pulsewidths to stay within the range determined by the minimum and maximum found in Steps 4 and 5 above.

	Pulsewidth	PCA_Start
Neutral		
Maximum Forward		
Maximum Reverse		

Table 1: PWM - Acceleration

	Pulsewidth	PCA_Start
Center		
Maximum Left		
Maximum Right		

Table 2: PWM - Steering

		PCA_Start
Center		
Dimmest		
Brightest		

Table 3: PWM - LED

Lab Check-Off: Demonstration and Verification

1. Record the PCA settings in the tables and put the results in your notebook
2. By making use of the C code developed for this lab, demonstrate that the C8051 can change the drive motor speed or steering angle. Also demonstrate that pulsewidth limits have been implemented.
3. Your TA will ask you to explain sections of the C code and/or circuitry you developed in this laboratory. All partners will need to understand the entire system.

Grading-Preparation and Checkoff

Prior to the starting the laboratory you must complete

1) The appropriate Worksheets (Worksheets 6 and 7).
2) The Pin-out form (Port, Interrupt, XBR0, PCA initializations
3) The Pseudocode (Revision when finished)

When you are ready to be checked off, the TAs will be looking at the following items

4) That your project performs all the indicated requirements (defined above in ***Lab Description & Activities***)
5) Appropriately formatted and commented source code
6) Clean and neat hardware, with appropriate use of colors for source and ground connections

Additionally, you will be asked a number of questions. The questions will cover topics such as

7) Understanding algorithms in the code, identifying locations in the software that perform specific actions, understanding the hardware components, understanding the test equipment

The final item that will be included in the Laboratory grade is your

8) Notebook.

The above 8 items each have an individual contribution to your Laboratory grade.

Sample C Code for Lab 3 (part 1) - Ultrasonic Ranger Partner

```
/* Sample code for speed control using PWM. */

#include <stdio.h>
#include <c8051_SDCC.h>
#define PW_MIN _____
#define PW_MAX _____
#define PW_NEUT _____

//-----------------------------------------------------------------------------
// 8051 Initialization Functions
//-----------------------------------------------------------------------------
void Port_Init(void);
void PCA_Init (void);
void XBR0_Init(void);
void Interrupt_Init(void);
void Drive_Motor(void);

unsigned int MOTOR_PW = 0;

//-----------------------------------------------------------------------------
// Main Function
//-----------------------------------------------------------------------------
void main(void)
{
   // initialize board
   Sys_Init();
   putchar(' '); //the quotes in this line may not format correctly
   Port_Init();
   XBR0_Init();
   PCA_Init();
   Interrupt_Init()

   //print beginning message
   printf("Embedded Control Drive Motor Control\r\n");
   //set initial value
   MOTOR_PW = PW_NEUT;
   //add code to set the servo motor in neutral for one second
   while(1)
      Drive_Motor();
}

//-----------------------------------------------------------------------------
// Drive_Motor
//-----------------------------------------------------------------------------
//
// Vary the pulsewidth based on the user input to change the speed
// of the drive motor.
//
void Drive_Motor()
{
   char input;
   //wait for a key to be pressed
   input = getchar();
   if(input == 'f') //if 'f' is pressed by the user
   {
      if(MOTOR_PW < PW_MAX)
      MOTOR_PW = MOTOR_PW + 10; //increase the steering pulsewidth by 10
   }
   else if(input == 's') //if 's' is pressed by the user
   {
      if(MOTOR_PW > PW_MIN)
      MOTOR_PW = MOTOR_PW - 10; //decrease the steering pulsewidth by 10
```

```
    }
    PCA0CPL2 = 0xFFFF - MOTOR_PW;
    PCA0CPH2 = (0xFFFF - MOTOR_PW) >> 8;
}

//-----------------------------------------------------------------------------
// Port_Init
//-----------------------------------------------------------------------------
//
// Set up ports for input and output
//
void Port_Init()
{
    P1MDOUT = ________   ;//set output pin for CEX2 in push-pull mode
}

//-----------------------------------------------------------------------------
// Interrupt_Init
//-----------------------------------------------------------------------------
//
// Set up ports for input and output
//
void Interrupt_Init()
{
    // IE and EIE1
}

//-----------------------------------------------------------------------------
// XBR0_Init
//-----------------------------------------------------------------------------
//
// Set up the crossbar
//
void XBR0_Init()
{
    XBR0 = __________   ;  //configure crossbar with UART, SPI, SMBus, and CEX channels as
                           // in worksheet
}

//-----------------------------------------------------------------------------
// PCA_Init
//-----------------------------------------------------------------------------
//
// Set up Programmable Counter Array
//
void PCA_Init(void)
{
    // reference to the sample code in Example 4.5 - Pulse Width Modulation implemented using the PCA (Programmable Counter Array, p. 50 in Lab Manual.
    // Use a 16 bit counter with SYSCLK/12.
}

//-----------------------------------------------------------------------------
// PCA_ISR
//-----------------------------------------------------------------------------
//
// Interrupt Service Routine for Programmable Counter Array Overflow Interrupt
//
void PCA_ISR ( void ) interrupt 9
{
    // reference to the sample code in Example 4.5 -Pulse Width Modulation implemented using the PCA (Programmable Counter Array), p. 50 in Lab Manual.
}
```

Sample C Code for Lab 3 (part 1) - Electronic Compass Partner

```
/* Sample code for Lab 3.1. This program can be used to test the steering servo */
#include <c8051_SDCC.h>
#include <stdio.h>
#include <stdlib.h>
//----------------------------------------------------------------------------
// 8051 Initialization Functions
//----------------------------------------------------------------------------
void Port_Init(void);
void PCA_Init (void);
void XBR0_Init();
void Steering_Servo(void);
void PCA_ISR ( void ) interrupt 9;

unsigned int PW_CENTER = _____;
unsigned int PW_RIGHT = _____;
unsigned int PW_LEFT = _____;
unsigned int SERVO_PW = 0;

//----------------------------------------------------------------------------
// Main Function
//----------------------------------------------------------------------------
void main(void)
{
   char input;
   // initialize board
   Sys_Init();
   putchar(' '); //the quotes in this line may not format correctly
   Port_Init();
   XBR0_Init();
   PCA_Init();
   Interrupt_Init()

   //print beginning message
   printf("Embedded Control Steering Calibration\n");
   //set initial value for steering (set to center)
   SERVO_PW = PW_CENTER;
   while(1)
      Steering_Servo();
}

//----------------------------------------------------------------------------
// Port_Init
//----------------------------------------------------------------------------
//
// Set up ports for input and output
//
void Port_Init()
{
   P1MDOUT = ________   ;//set output pin for CEX0 in push-pull mode
}

//----------------------------------------------------------------------------
// XBR0_Init
//----------------------------------------------------------------------------
//
// Set up the crossbar
//
void XBR0_Init()
{
   XBR0 = __________   ;  //configure crossbar with UART, SPI, SMBus, and CEX channels as
                          // in worksheet
```

```
}

//-----------------------------------------------------------------------------
// PCA_Init
//-----------------------------------------------------------------------------
//
// Set up Programmable Counter Array
//
void PCA_Init(void)
{
    // reference to the sample code in Example 4.5 -Pulse Width Modulation implemented using
the PCA (Programmable Counter Array), p. 50 in Lab Manual.
}

//-----------------------------------------------------------------------------
// Interrupt_Init
//-----------------------------------------------------------------------------
//
// Set up ports for input and output
//
void Interrupt_Init()
{
    _____ = ________;       //Enable interrupts
    _____ = ________;
}


//-----------------------------------------------------------------------------
// PCA_ISR
//-----------------------------------------------------------------------------
//
// Interrupt Service Routine for Programmable Counter Array Overflow Interrupt
//
void PCA_ISR ( void ) interrupt 9
{
    // reference to the sample code in Example 4.5 -Pulse Width Modulation implemented using
the PCA (Programmable Counter Array), p. 50 in Lab Manual.
}

void Steering_Servo()
{
    char input;
    //wait for a key to be pressed
    input = getchar();
    if(input == 'r') //if 'r' is pressed by the user
    {
        if(SERVO_PW < PW_RIGHT)
        SERVO_PW = SERVO_PW + 10; //increase the steering pulsewidth by 10
    }
    else if(input == 'l') //if 'l' is pressed by the user
    {
        if(SERVO_PW > PW_LEFT)
        SERVO_PW = SERVO_PW - 10; //decrease the steering pulsewidth by 10
    }
    printf("SERVO_PW: %u\n", SERVO_PW);
    PCA0CPL0 = 0xFFFF - SERVO_PW;
    PCA0CPH0 = (0xFFFF - SERVO_PW) >> 8;

}
```

The LED Partner should choose to modify the Ultrasonic Ranger for their Sample C Code

Recommended Procedure for Laboratory Development

The following recommendations will make software development easier, allowing you to debug execution errors quickly. Once again, you are strongly encouraged to take an incremental approach to Laboratory development rather than trying to do everything at once.

1) Build the circuit.
   This circuit is very straightforward. At this point, you should be familiar with the 74365 Buffer chip (including remembering the Output Enable pins). Use the XBR0 setting to identify the appropriate Port Pin for input to a buffer gate.
2) Copy the requisite Initialization routines from Laboratory 2 and edit them for Laboratory 3.
   With the exception of Timer0, you will use all the current initializations routines in the coming Laboratories. A/D conversion is not used in Laboratory 3, but will be in Laboratory 5 and 6.
3) Add the PCA and XBR0 initializations.
   Refer to the Laboratory description for the appropriate settings. Recall, the motors operate at a 50 Hz or 20 ms cycle.
4) Code a CCM threshold value and verify that the appropriate Port Pin has a pulsed output.
   Print both the Start value of the PCA and the Pulsewidth. Use the logic probe to check for pulsed output.
5) Write a subroutine that can be used to calibrate the lower and upper bounds of the Pulsewidth for your option (Speed controller or Servo motor).
   This routine should respond to user input for calibration. Use a `getchar()` command to read one character from the keyboard (refer to `homework1.c`)
6) Write a subroutine that can be used to manually steer or drive the car.
   This routine should be a minor revision of your calibration routine, with the Pulsewidth limits embedded in the code.

Having developed the above routines, refine the code to account for the Laboratory specifications.