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, you will form team of two pairs of students each. On each team, one pair will specialize on the electronic compass and heading (steering) control. The other pair will specialize on the ultrasonic ranger and altitude (speed) control.

General

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

Hardware

1. Ultrasonic Ranger Pair - Configure the speed controller to control the car speed.
2. Electric Compass Pair - Configure the servo to control the car steering angle.

Software

1. Ultrasonic Ranger Pair - 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 Pair - 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.

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 *Blimp*. Since these labs form an evolutionary developmental sequence for the Blimp, you may wish to familiarize yourself with the objectives of all remaining labs as an aid in planning for each.

The class will be divided into two groups - one group will be working with the electronic compass and the other group will be working with the ultrasonic ranger. For the upcoming laboratories, the groups will form into teams of four. The teams will merge the software for the electronic compass and ultrasonic ranger into a single code controlling the speed and direction of the *Smart Car.*

Lab Description and Activities

1. For Laboratory 3, the ultrasonic ranger and the electronic compass pair of a team will work independently on hardware and software. Their 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 pair on 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 control the steering.

Hardware

The hardware schematics for the ultrasonic ranger and the electronic compass are shown in *Figure C.4* and *Figure C.5*, respectively. Ranger pairs will construct the circuit in *Figure C.4* and the compass pairs will construct the circuit in *Figure C.5* on separate protoboards. 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.

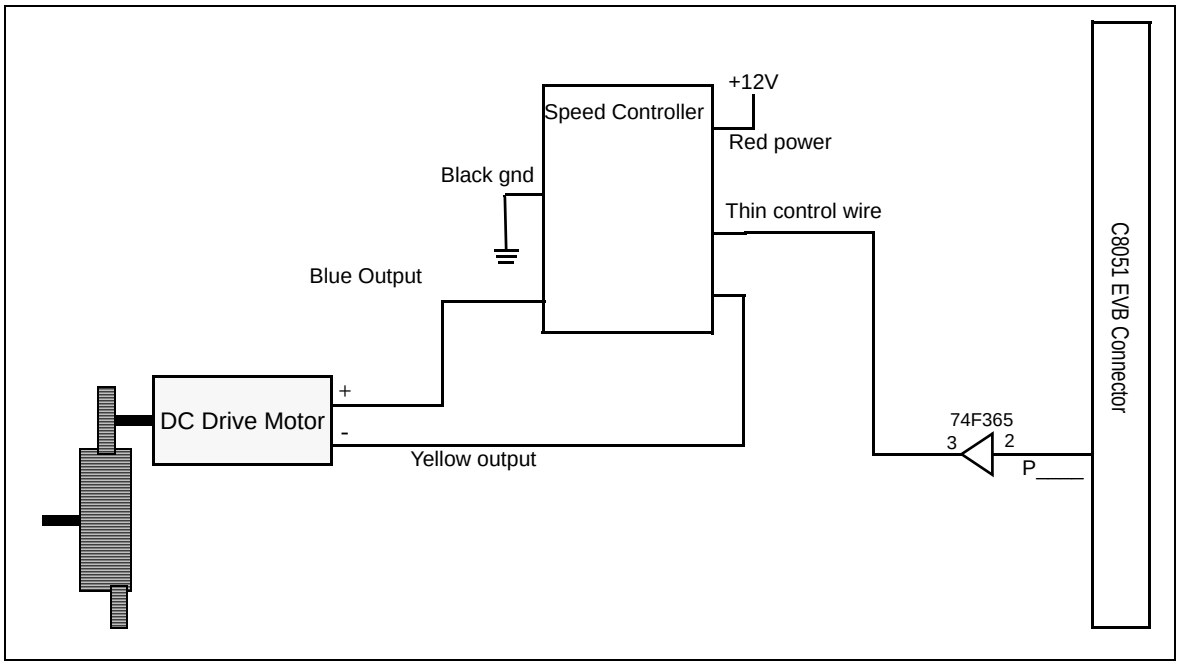

Figure C.3 - Car drive-motor circuitry for Lab 3 (part 1) - Ultrasonic Ranger.

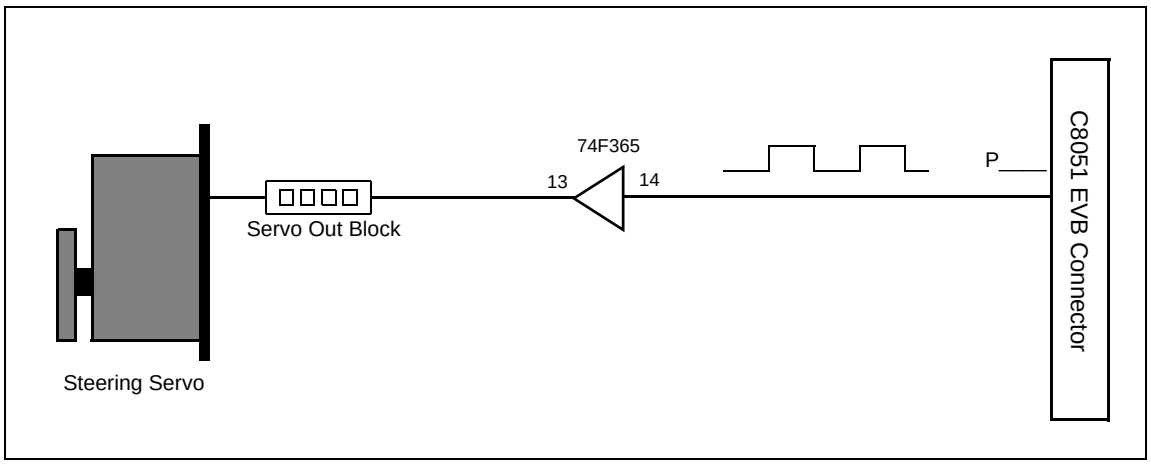

Figure C.4 - Car steering servo motor control circuitry for Lab 3 (part 1) - Electronic Compass

Software - Ultrasonic Ranger

The Ultrasonic Ranger Pair 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. 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 Pair

The Electronic Compass Pair 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 20ms and the pulsewidth must be initialized to be 1.5ms. Configure the PCA to use a 16-bit counter with SYSCLK/12. Initial estimates of the left and right limits are 0.9 [ms] and 2.1 [ms].
2. The code must then receive imputs 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.

	Pulsewidth	PCA_Start
Neutral		
Maximum Forward		
Maximum Revers		

Table 1: PWM - Acceleraation

Lab Check-Off: Demonstration and Verification

1. Complete the pseudocode and fill out the pinout form
2. Record the PCA settings in the tables and put the results in your notebook
3. 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.
4. Your TA may ask you to explain how sections of the C code or circuitry you developed for this exercise works. To do this, you will need to understand the entire system.

	Pulsewidth	PCA_Start
Center		
Maximum Left		
Maximum Right		

Table 2: PWM - Steering

Writing Assignment - Lab Notebook

Continue to record your notes as you did with the previous laboratories.

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

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

#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 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();

   //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
}

//-----------------------------------------------------------------------------
// 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).
    // 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).
}

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

```
/* Sample code for Lab 3.1. This program can be used to test the steering servo */
#include <c8051_SDCC.h>
#include <stdio.h>
#include <stlib.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();

   //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).
}

//-----------------------------------------------------------------------------
// 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).
}

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;

}
```