9S12 Subsystems: Pulse Width Modulation, A/D Converter, and Synchronous Serial Interface

In this sequence of three labs you will learn to use three of the MC9S12's hardware subsystems.

WEEK 1

Pulse Width Modulation

Introduction and Objectives

The speed of a motor can be adjusted by powering it with a pulse width modulated signal. Figure 1 shows how this can be done. The SN754410 motor controller works as a switch. When the signal on the enable (*EN1*) input of the L293D is high the switch is closed, current flows through the motor, and the motor speeds up.
When the signal is low, the switch is open, no current flows, and the motor slows down. IN1 and IN2 determine the direction of the motor. When *IN1* is high and *IN2* is low, *OUT1* will be high (V_M) and *OUT2* will ground. Changing the logic levels of *IN1* and *IN2* reverse the voltage levels on *OUT1* and *OUT2*, and the motor changes direction. With a high enough frequency PWM signal the amount the motor speeds up and slows down in one period is negligible, and the motor turns at a constant speed. By adjusting the duty cycle the speed of the motor can be controlled. For the motors you will be using in this lab, use a PWM frequency of about 5 kHz.

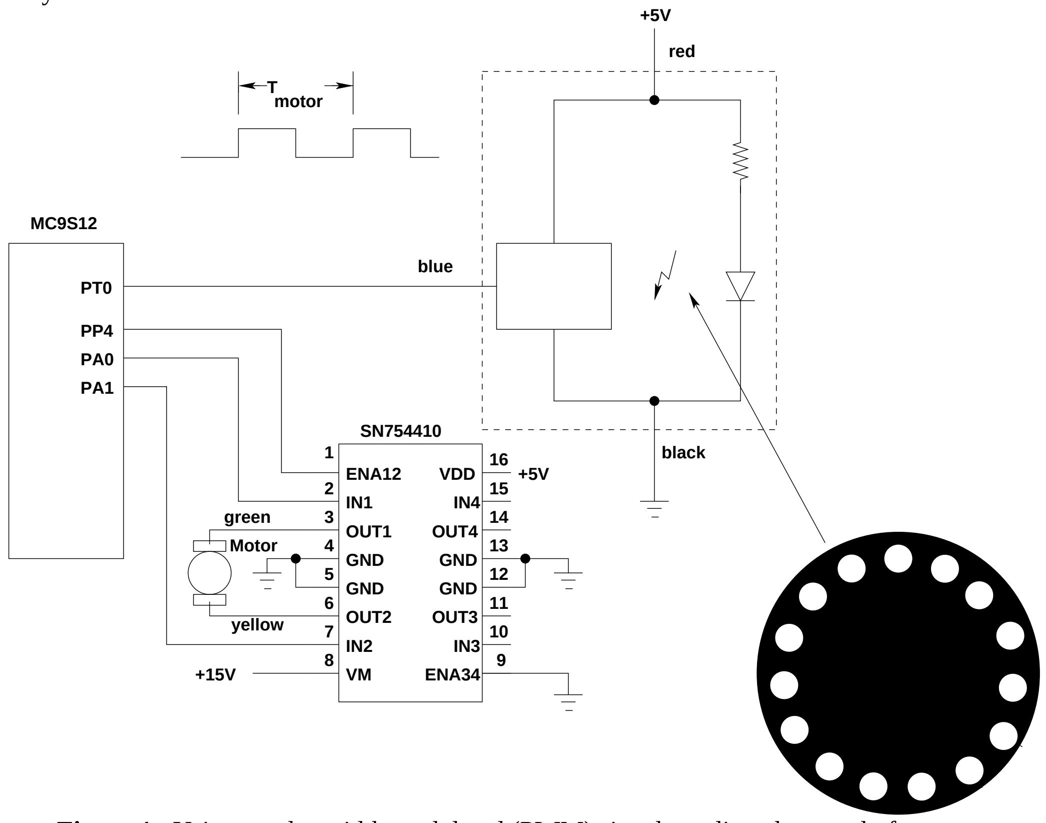

Figure 1. Using a pulse width modulated (PWM) signal to adjust the speed of a motor.

To make a motor turn at a desired speed it is necessary to know how fast the motor is turning. The motors you will use in this lab have encoders which will generate 15 pulses in a single revolution of the motor. A light emitting diode (LED) sends light through a wheel with holes in it which is attached to the shaft of the motor, and an optical sensor detects light. When the light shines through a hole and hits the sensor, the output of the optical sensor is V_{DD} (a digital 1); when the wheel blocks the light from reaching the sensor, the voltage goes to 0 V (a digital 0). Using this optical encoder you will be able to measure the speed of the motor. The motors we will use have two optical encoders, on opposite sides of the wheel. This allows you to determine both the speed and the direction of the motor, using a technique called quadrature detection. This will be discussed more in the final lab for the course. For this lab, use just one of the optical encoders. In this lab you will use the PWM subsystem of the MC9S12 to adjust the speed of a small motor, and use the input capture subsystem to measure the speed of the motor.

For the pre-lab do the calculations required for Parts 3 and 4. Write the program required for Parts 5.

1. Connect the motor directly between 0 and 15 V to verify that the motor turns (Do not use the SN754410 for this part.) Connect the Red lead from the optical encoder to V_{DD}, and the Black lead to ground. Have a TA or Instructor check your wiring before turning the power on. The motors and encoders are fairly expensive, and we do not want to burn up any of them. Use a logic probe to verify that the Blue lead from the optical encoder is pulsing, indicating that the encoder is working.

2. Program your MC9S12 to generate a PWM frequency of 5 kHz, with an active high PWM signal with a duty cycle of 50%. (Use Bit 4 of the PWM, since Bits 0 through 3 of Port P are used to enable the seven-segment LEDs.) Make Bit 0 of Port A high, and Bit 1 of Port A is low. Connect the circuit shown in Figure 1. Use your logic probe to verify that the optical encoder is working.

3. Connect the output of the optical sensor to one of the input capture pins of the MC9S12. To your program from Part 2, add code to use the input capture function to measure the period of the signal from the optical encoder, and to display the values on the seven-segment LEDs.

The motor speed at 100% duty cycle will be about 500 RPM. Plan to operate the motors between 50 RPM and 500 RPM. Set the prescaler such that you can measure speeds as low as 20 RPM (to avoid overflow of the TCNT register in case the motor slows down below the 50 RPM lower limit).

a) With a 500 RPM motor speed, how much time will there be between successive rising edges from the optical sensor?

b) With a 20 RPM motor speed, how much time will there be between successive rising edges from the optical sensor?

Set the prescaler of the MC9S12 so the rollover time of the TCNT register is the smallest time which is larger than the above two numbers. What value did you use for the prescaler?

4. Modify your program to set the duty cycle based on the state of four of your DIP switches. The duty cycle of the signal should be determined by reading the four DIP switch values into the four least significant bits of Port H:

Port H	Duty Cycle	Port H	Duty Cycle
0000	6.25%	1000	56.25%
0001	12.50%	1001	62.50%
0010	18.75%	1010	68.75%
0011	25.00%	1011	75.00%
0100	31.25%	1100	81.25%
0101	37.50%	1101	87.50%
0110	43.75%	1110	93.75%
0111	50.00%	1111	100.0%

Note that you should pre calculate the duty cycles as the integer you will write to the duty cycle register which will give duty cycles closest to the values in the table. (Do this for part of your pre-lab.)
Do not use floating point arithmetic in your C program.

5. Run the motor with each of the duty cycles from Part 4. Have the MS9S12 display the number of timer pulses between rising edges on the seven-segment LEDs. Record these numbers, and convert them to RPM. Record the speed in RPM for each duty cycle. Plot the motor speed as a function of duty cycle. How linear is it?

6. Compare your results (speed of the motor vs. duty cycle) to the results of at least two other groups. Do the motors all behave the same, or are there significant differences?

THE 9S12 ANALOG TO DIGITAL CONVERTER

WEEK 2

Introduction and Objectives

The analog to digital converter is described in the ATD Block Users Guide. The MC9SS12DP256B has two eight-channel ten-bit A/D converters, ATD0 and ATD1. The Dragon12-Plus board has a potentiometer connected to Bit AD7 of Port ATD0. We will use that potentiometer to put a variable analog signal into our MC9S12. (Remember that Bits AD0 and AD1, part of ATD0, are used by DBug 12 at start up to determine whether to execute DBug12, or to run code from EEPROM or the bootloader. Do not connect an analog signal to either of those two inputs.)

The A/D converter also uses two dedicated pins V_{RH} and V_{RL} for high and low voltage references respectively. On your Dragon12-Plus MC9S12 board, V_{RH} is connected to VCC (+5V), and V_{RL} is connected to GND. When the MC9S12 is set up to do ten-bit (1024) conversions, an input voltage of V_{RL} gives an output of 0x000, and an input of V_{RH} gives an output of 0x3FF. (This assumes that the DJM bit of ATD1CTL5 register is set, so that the data is right-justified in the results registers.) If we measure a voltage between V_{RL} and V_{RH}, we can compute the value by simple ratios

$$voltage = \frac{measurements * (VRH - VRL)}{1024} + VRL$$

For example, if V_{RH} = 5 volts, and V_{RL} = 0 volts, and the measurement is 0x2B0 (68810), then the measured voltage is

$$\frac{688 * (5 - 0)}{1024} + 0 = 3.359 volts$$

1. Set up the A/D converter so it measures in analog input on Port AN7. It should convert one channel (AD7), with a sequence of 8 conversions, and scan continuously.

2. Write a program which uses the TOF interrupt. The TOF should generate an interrupt every 174 ms. The TOF interrupt service routine should set a global flag which tells the main program the interrupt has occurred. In your TOF interrupt service you should read the value from the A/D conversion of Port AN7 and save it to a global variable. Use an RTI interrupt to display the A/D value on the seven-segment LEDs.

3. Vary the voltage on Bit AN7 by turning the potentiometer on the Dragon12-Plus board. Compare the value displayed on the terminal with multimeter measurements for several different input voltages.

4. The A/D conversion measurements can be improved by averaging the values in the registers ATD0DR0 through ATD0DR7. In your RTI routine, average the 8 values. Display the averaged ten bit value on the 7 segment LED display. In the main program, write the averaged ten bit value to the terminal. Is the value more stable than it was when you displayed the unaveraged value?

5. The Dragon12-Plus board has a light sensor in the form of a photodiode, Q1. The output of the light sensor is connected to Bit 4 of ATD0. Modify your program from Part 4 to display the output from Bit 4 of ATD0 rather than Bit 7 of ATD0. What is the voltage out of Q1 when the sensor is covered? What is the voltage out of Q1 when the sensor is exposed to a bright light?

SERIAL COMMUNICATIONS

The I^2C Bus and the Dallas DS 1307 Real Time Clock

WEEK 3

Introduction and Objectives

The Dragon12-Plus board has a Dallas Semiconductor DS 1307 Real Time Clock connected to the I^2C bus of the MC9S12 microcontroller. You will program your MC9S12 to set the correct time in the DS 1307, and to display the time on your LCD display. A crystal connected to the DS 1302 oscillates at 32.768 kHz. Note that 32,768 = 2^{15}, so dividing the oscillator by 2^{15} will result in a signal with a period of 1 second. The DS 1307 uses this 1 Hz signal to keep the date and time. It correctly takes care of leap years, and can be programmed to give the time in either 24 hour mode, or 12 hour mode with AM and PM indicators. It connects to a microcontroller using a two-wire I^2C interface.

1. Look at the datasheet for the DS 1307. Determine the addresses of the time registers.

2. Program the 9S12 to set the time on the DS 1307. The lecture notes from March 23 discusses how to write to and read from slave devices on the I2C bus. Note that to access the DS 1307, you need to initialize the I2C subsystem, send a byte which contains the I2C address of the DS 1307 (which is listed in the datasheet), and indicate whether you are going to write to the device or read from the device. To write the time to the DS 1307 you need to first send the address of the first register you want to program, followed by the data for that register and the subsequent time registers.

3. Write a C program to read the time and date from the DS 1307 and write it to the terminal using the DBug12->printf command.

4. Program the MC9S12 to continuously display the time and date on the LCD display. The date should be displayed in the first line, and the time on the second line, of the LCD display. The CD-ROM which came with the textbook has routines for accessing the LCD display. The program `eg07_08.c` in the directory `\programs\ch07` displays the string “hello, world!” on the first line of the LCD display, and the string “I am ready!” on the second line. All you have to do is to convert the time and data from the DS 1307 into two strings, one for the data and the other for the time. This is easy to do since the data from the DS 1307 is in BCD format. For example, the year will be of the form 0x09. Here is some C code to display the year part of the date, where the date is in the form “`09/03/30`” for March 30, 2009.

```
char s[16];

openlcd();                    // Set up LCD to write to first line
s[0] = (year>>4)&0xff + ‘0’;  // Get 4 MSB of year, convert to ASCII
s[1] = year&0xff + ‘0’;       // Get 4 LSB of year, convert to ASCII
s[2] = ‘/’;
...                           // Code for rest of date
s[8] = 0;                     // Terminate string with null character
puts2lcd(s);                  // Display on first line of LCD
```

The routines from the textbook use some delay routines in the file `delay.c`, which are found in the directory `\utils`. It also needs the `hcs12.h` include file, which can be found in the same directory. (The `hcs12.h` file on the EE308 web page had a small error in it. For the delay routines to work, copy the `hcs12.h` file from the CD-ROM, or download the updated file from the EE 308 web page.) You need to include the `delay.c` and `hcs12.c` files into your program, and copy the LCD functions from the file `eg07_08.c` into your program.

5. For extra credit, add the code to the start of your program needed to load the program into EEPROM. Go ahead and load the program into EEPROM and verify that it works. Your Dragon12-Plus board can now be used as a wrist-watch (as long as you carry around a battery to power it).