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 9S12's hardware subsystems.

WEEK 1
Pulse Width Modulation

Pre-Lab

1. Calculate the times asked for in Part 3.
2. Determine values you need to write to the PWM registers (except for the Duty Cycle register) to control the motor.
3. Calculate the values needed for the Duty Cycle Register to give the duty cycles needed for Part 4.

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 L293D motor controller 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 and *OUT2* will be low. 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.

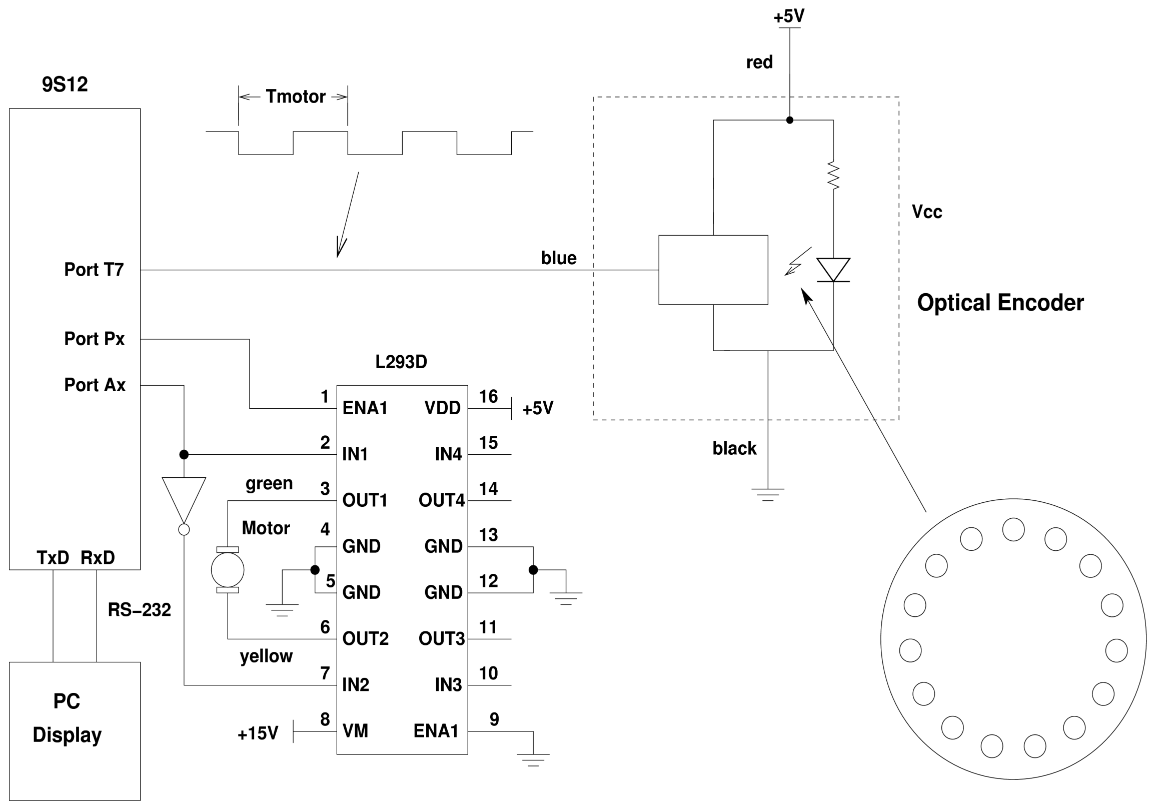

Figure 1. Using a pulse-width-modulated (PWM) signal to adjust the speed of a motor.

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 Vcc (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 9S12 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 L293D for this part.) Connect the Red lead from the optical encoder to Vcc, 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 9S12 to generate a PWM frequency of 5 kHz, with an active high PWM signal with a duty cycle of 50%. 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 9S12. 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 print the value to the terminal. (Note that you should use an interrupt to measure the period, and in the interrupt service routine set a global variable to the number of clock cycles between rising edges. You should print this number to the terminal in the main program loop, not in the ISR.)

   The motor speed at 100% duty cycle will be about 500 RPM. Plan to operate the motors between 50 RPM and 500 RPM. We will set the prescaler such that we 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 9S12 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 B:

PORTB	Duty Cycle	PORTB	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	81.255
0101	37.50%	1101	87.50%
0110	43.75%	1110	93.75%
0111	50.00%	1111	100.00%

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 9S12 print the number of timer pulses between rising edges to the terminal. 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

Pre-Lab

1. Write the program for Parts 1 and 2.
2. Calculate the expected 10-bit result when the voltage input to PAD8 is:
   2.1.0.0 V
   2.2.1.0 V
   2.3.2.0 V
   2.4.2.5 V
   2.5.3.0 V
   2.6.4.0 V
   2.7.5.0 VPre-Lab
   2.8.
   2.9.1.Write the program for Parts 1 and 2.
   2.10.2.Calculate the expected 10-bit result when the voltage input to PAD8 is:
   2.11.a)0.0 V
   2.12.b)1.0 V
   2.13.c)2.0 V
   2.14.d)2.5 V
   2.15.e)3.0 V
   2.16.f)4.0 V
   2.17.g)5.0 V

Introduction and Objectives

The analog to digital converter is described in the ATD Block Users Guide. The MC9SS12DP256B has two 8-channel 10-bit A/D converters, ATD0 and ATD1. The miniDragon board has a potentiometer connected to Bit AD7 of Port ATD0. We will use that potentiometer to put a variable analog signal into our 9S12. (Remember that Bits AD0 and AD1, part of ATD0, are used by D-Bug 12 at start up to determine whether to execute D-Bug12, 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 miniDragon 9S12 board, V_{RH} is connected to V_{CC}, and V_{RL} is connected to GND. When the 9S12 is set up to do 10-bit (1024) conversions, and input voltage of V_{RL} gives an output of 0x000, and an input of V_{HR} 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

$$voltage = \frac{measurements * (V_{RH} - V_{RL})}{1024} + V_{RL}$$

For example, if V_{RH} = 5 volts, and V_{RL} = 0 volts, and the measurement is 0x2B0 (688_{10}), 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 RTI interrupt. The RTI should generate an interrupt every 64 ms. The RTI interrupt service routine should set a global flag which tells the main program the interrupt has occurred. In your RTI interrupt service you should read the value from the A/D conversion of Port AN7 and save it to a global variable. Also, to verify that the RTI interrupt is working at the correct rate, display the sequence 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F on the seven-segment display connected to PTH. In the main program, write the 10-bit value from the value read from the A/D conversion to the terminal using the DBug-12 printf( ) function.

3. Vary the voltage on Bit AN7 by turning the potentiometer on the minDragon 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 10-bit value on the 7 segment LED display. In the main program, write the averaged 10-bit value to the terminal. Is the value more stable than it was when you displayed the unaveraged value?

SERIAL COMMUNICATIONS USING THE 9S12 SPI AND A REAL-TIME CLOCK

WEEK 3

Pre-Lab

1. Draw a schematic for Part 1. Make sure you label the pin numbers on the DS 1302, and the corresponding pin numbers on the 9S12.
2. Determine the SPI format needed to talk to the DS 1302: clock polarity, clock phase, clock speed, MSB or LSB first.
3. Determine the sequence of commands to set the time on the DS 1302.
4. Write the program for Par 2 to set the time on the DS 1302.
5. Determine the sequence of commands to read the time from the DS 1302
6. Write the program for Part 3 to read the time from the DS 1302, and display it on the computer terminal.

Introduction and Objectives

In this lab, you will interface the Dallas Semiconductor DS 1302 Real Time Clock to your 9S12. You will program your 9S12 to set the correct time in the DS 1302, and to display the time on your computer terminal. 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 1302 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 three-wire interface which is compatible with the 9S12's SPI.

1. Look at the datasheet for the DS 1302. Draw a schematic showing how to connect the DS 1302 to your 9S12. The DS 1302 is an eight-pin device. It has power (V_{CC2}) and ground pins, which should be connected to +5 V and GND, respectively. Two pins are used to connect to the 32.768 kHz oscillator. One pin (V_{CC1}) should be connected to the + terminal of the battery. The other three pins are used to communicate with the 9S12 via the SPI interface. You should connect a 0.1 μF capacitor between V_{CC2} and GND.
2. Program the 9S12 to set the time on the DS 1302.
3. Write a C program to display the time on the PC terminal. Program a Real Time Interrupt to interrupt the 9S12 at a rate of about 100 ms. In the Real Time Interrupt ISR, read the time from the DS 1302, and save the values (year, month, day of month, hour, minute, second, day of week) into global variables. In the main program loop, watch the value of the seconds variable. When this changes value, print the time to the PC terminal in the following form:

```
        Tue Mar 28 21:23:55 MST 2006
```

4. Run your program. Verify that it works. Turn power off, turn power back on, reload and rerun your program. Verify that the battery backup works --- that the DS 1302 kept the correct time through the power cycle.