2 Live Lab Session - Advanced Interfacing

These instructions will lead you through the following activities:

- Design and write ASM/C programs for a specific target and a specific assembler/compiler that can configure, read and/or write to a built-in peripheral in the context of a specific circuit
- Utilise the features of the MPLAB-X environment and the SNAP debugger to test, debug, and verify the operation of an ASM/C program.
- Verify and characterise "correct" operation of a micro-controller driven circuit using standard laboratory tools (oscilloscope, signal generator, multi-meter, power supply)

To complete these activities you will need:

- a breadboard that has been set up and tested as described in the build instructions
- the ASM program and routines that you wrote in the last lab practice exercise.

We will make use of the built-in peripherals in this lab - specifically:

- Serial Port peripheral - EUSART
- PWM peripheral - CCP2

2.1 EUSART peripheral

We will use the USB-Serial adapter to communicate with the computer at 1200 baud, 8 bits, no parity.

1. Locate the HIPO in the PIC16F18877 datasheet for Asynchronous Transmission. What page number is it on?
2. Locate the calculation for the value to be placed in the Baud Rate generator registers SPBRGH:SPBRGL. What page number is it on?
3. Write down the settings for the bits BRG16 and BRGH for a multiplier of 64.
4. The value to be placed in the registers SPBRGH:SPBRGL for a multiplier of 64, with a 2MHz oscillator, for a baud rate of 1200, will be 25_{10}. What will the error in the generated baud rate be?
5. Create ASM code routines labelled `lkupASCII`, `sendSerial` and `cfgSerial` to match the descriptions provided.

Configure Serial port for 1200 baud 8N1 Transmission

Label: cfgSerial

INPUT PORTC pin 6 is connected to the USB-Serial Adapter RX line

OUTPUT The EUSART is configured for Serial Transmission via PORTC pin 6 at 1200 baud 8N1

PROCESS Use bit-operations and move operations

cfgSerialStep1 configure PORTC pin 6 as an output using TRISC

cfgSerialStep2 configure PORTC pin 6 as a digital pin using ANSELC

cfgSerialStep3 configure pin RC6 as the Serial TX by putting 0x10 in register labelled RC6PPS

cfgSerialStep4 put the 16-bit value equivalent to decimal 25 into the pair of 8 bit registers SPBRGH:SPBRGL

cfgSerialStep5 clear the BRGH (register BAUD1CON), BRGH and SYNC (register TX1STA) bits

cfgSerialStep6 set the SPEN (register RC1STA) and TXEN (register TX1STA) bits

cfgSerialStep7 leave routine

Send a character

Label: sendSerial

INPUT EUSART is configured and enabled for TX, the character to be sent in the the working register

OUTPUT The character has been transmitted

PROCESS When the EUSART is free, copy the contents of working register to TXREG

sendSerialStep1 wait for TXIF bit to be set

sendSerialStep2 copy value from the working register to TXREG

sendSerialStep3 leave routine

Find ASCII value associated with a hexadecimal digit

Label: lkupASCII

INPUT An 4-bit value in the lower nibble of the working register

OUTPUT The equivalent ASCII code for the character associated with the hexadecimal digit in the working register

PROCESS Computed-goto (Lookup table)

seglkupStep1 Mask the value in the working register to clear all bits in the upper nibble (Hint: AND with 0x0F)

seglkupStep2 Add the value in the Working Register to the PCL register, answer in the PCL register (Remember to BANKSEL first!!)

seglkupStep3 leave routine with value based on the value that was in the working register (Hint: use a list of retlw statements)

6. Use the program that you produced at the end of the previous practice Session. Add the additional routines, and modify your main loop so that the values in the registers labelled `CntHi` and `CntLo` are transmitted as an ASCII string to a computer on each loop. (Hint: you will need to send the information by converting each of the four nibbles to an ASCII character - if you need to have a newline for each value - transmit the value 0x13 (ASCII code for carriage return). N.B. You will need to connect the USB-Serial adapter, and use CuteCOM or another terminal emulator to observe the transmitted strings.

2.2 CCP2 peripheral

The built in CCP2 peripheral can be used to generate a PWM signal on pin RC1. Timer 2 is used to generate the PWM period, and the CCP registers are used to specify the duty cycle (ratio of ON-time to the PWM period).

7. Locate the HIPO in the PIC16F18877 datasheet for configuring the CCPx peripheral for PWM. What page number is it on?

8. We wish to have a PWM frequency of 200Hz. To do so, we will configure Timer0 clock to be Fosc/4 and set the PR2 register to decimal 19. Use the formulas from the datasheet to determine the actual PWM frequency generated.

9. The duty cycle ratio is defined as: CCPR1H/(PR2 + 1). For a 50% duty cycle, what value should be placed in CCPR2H?

10. Create ASM code routines labelled `setPWMdc` and `cfgPWM` to match the descriptions provided.

Configure CCP2 port for PWM period of 5ms (PWM frequency 200Hz)

Label: cfgPWM

INPUT PORTC pin 1 is connected to the base of the transistor that switches the motor

OUTPUT CCP2 is configured to provide a PWM signal of period 5ms to PORTC pin 1 with an initial duty cycle of 100%

PROCESS Use bit-operations and move operations

cfgPWMStep1 Configure PORTC pin 1 as an output using TRISC register

cfgPWMStep2 Configure PORTC pin 1 as a digital pin using ANSELC register

cfgPWMStep3 Configure pin RC1 as CCP2 output by putting 0x0A in the RC1PPS register

cfgPWMStep4 Configure the CCP module for PWM by loading the value 0x9F into the CCP2CON register, and the value 0x05 into the CCPTMRS0 register

cfgPWMStep4 Load the PR2 register with the PWM period value of 5ms (decimal 19).

cfgPWMStep5 Set the duty cycle to 100% by putting the value 0xFF in the CCPR2L and CCPR2H

cfgPWMStep6 Configure the clock source for Timer 2 by placing the value 0x01 into the T2CLKCON register

cfgPWMStep7 Configure Timer 2 for a prescaler of 128 and enable the timer by putting the value 0xF0 into the T2CON register

cfgPWMStep8 leave routine

Set Duty Cycle

Label: setPWMdc

INPUT The duty cycle to be set on CCP2 (a value between 0 and 20_{10}) is in the working register; CCP2 is already configured for PWM with a PR2 value of 19_{10}

OUTPUT CCP2 duty cycle has been set to the value from the working register

PROCESS Verify if the value is valid, and adjust CCPR2H accordingly

setPWMdcStep1 if the value in the working register is greater than 20_{10}

setPWMdcStep1.1 TRUE:

setPWMdcStep1.1.1 goto setPWMdcStep2

setPWMdcStep1.2 FALSE:

setPWMdcStep1.2.1 goto setPWMdcStep3

setPWMdcStep2 put the value 20_{10} in the working register

setPWMdcStep3 copy the value inthe working register to CCPR2H

setPWMdcStep4 leave routine

11. Add the additional routines, and modify your main loop so that the duty cycle is controlled by the value (0 - 15) produced by the 4-bit DIP switches. Adjust your main loop so that the Counts N.B. You will need to connect the USB-Serial adapter, and use CuteCOM or another terminal emulator to observe the transmitted strings.

12. While the previous program is running, use an oscilloscope to view the signal from the fan motor sensor when the fan is running at different speeds (adjust the DIP switches to vary speeds). What do you notice? Is the frequency at which the sensor is changing related to the motor speed in RPM? (Hint: the motor datasheet can be found at `https://bit.ly/39eeRS2`)

13. Challenge: Use the logging feature to capture data from your program for 3 different PWM settings. What do you observe about the way in which the number CntHi:CntLo increases? Could this data be used to estimate the average RPM of the fan motor? If so, explain the algorthm. If not, explain why the data is unsuitable.