Brief Review

- We should all be familiar and capable of
  - Compiling C code with the MPLAB XC8 compiler
  - Simple debugging – TBD
  - Know where the header files are and something about the configuration bits and where to find out more information about them
  - Have at least looked at the datasheets for the microcontroller chips and user guides for the training boards

Brief Review

- How to setup a line for digital input or output
- How to setup a line for analog input
- How to read from or write to a digital input line
- How to access the results of an analog to digital conversion and scale it appropriately
- How to use the interrupt scheme of the PIC 8 bit microcontrollers
- How to write a C subroutine to handle interrupts
  - In notes or the XC8 compiler users guide

After This Week

- How to use the PIC 8 bit microcontroller timers, both 8 bit and 16 bits
  - Using polling
  - Using interrupts
- At least three ways to interface an LCD
  - 8 bit direct connection (8 data lines, 2 or 3 control lines)
  - 4 bit direct connection (4 data lines, 2 or 3 control lines)
  - 8 bit indirect
    - 1 data line, 1 control line
    - Using an I/O expander and SPI interface

What's Left

- Serial communications
  - Input from devices, such as sensors and storage media
  - Out to devices, such as storage media
- CCP (Compare, Capture, PMW)
  - PMW – Pulse Width Modulation
    - Way to change the output from the microcontroller by manipulating the frequency, duration, and percentage of time the microcontroller outputs high/low
    - One important use is electric motor control
    - Generating sound using piezo buzzer

What's Left (Approximately)

- Application program tying it all together
  - Timers (external crystal for more precise timing)
  - Analog input (potentiometer)
  - LCD output
  - Digital input (pushbuttons)
  - Digital output (LEDs)
  - Digital input/output (EEPROM)
  - Interrupts
  - State management of inputs/outputs
  - Digital/analog input (temperature sensor)
  - PWM (piezo buzzer, electric motor)

- And then, robotics

Today's Class

- Read chapter 13 – serial I/O
- Look on the Internet for information on microcontroller serial I/O
- Read the EUSART and MSSP sections of one of the datasheets (PIC18F8722 or PIC18F46K22)

Serial I/O

- Serial I/O is used to transfer 1 bit at a time, versus parallel I/O transferring multiple bits at a time
- Why would we want to transfer data using serial transfer versus parallel transfer
  - Hardware associated with serial transfer is typically simpler than parallel
    - For parallel need to worry about cross contamination between data wires
  - Can run at higher speeds
  - Less wires
    - Single data line for serial
    - Multiple data lines for parallel (one for each bit in parallel)

Serial I/O Protocols

- EIA-232 (used to be called RS-232)
  - Most often used with PCs
  - Originally 25 wires, typically 7 – 9 today
- Serial Peripheral Interface (SPI)
  - Microcontrollers and peripheral devices, 4 wires
- Inter-Integrated Circuit (I2C, I^2C)
  - Microcontrollers and peripheral devices, 2 wires
- One-wire Bus (1-Wires®)
  - Network control, 1 wire
- Controller Area Network Bus (CAN)
  - Automative applications
- Local Interconnect Network (LIN)
  - Automative applications

Synchronous/Asynchronous

- Synchronous
  - Receiver and transmitter use the same clock
  - Used in serial communications between microcontrollers and peripheral devices
  - Transmission starts with sync characters and then followed by the data
- Asynchronous
  - Receiver and transmitters use an agreed upon clock speed
  - Transmission begins with start bits, followed by data, and ending with stop bits
  - Typical application is PC and modem

Simplex/Duplex

- Simplex
  - Data flows in a single direction
  - Microcontroller to a peripheral device or PC to a peripheral device
- Half Duplex
  - Data flows in both directions, but only one direction at a time
  - Push to talk communicationss – CB radio
- Full Duplex
  - Data flows in both directions at the same time
  - Telephone conversation

Transmission Rate

- In asynchronous communications, the agreed upon clock defines the rate of data flow
  - Bits per second (BPS)
  - Baud – signaling rate
    - Changes in transmission medium per second in a modulated signal

PIC18 Microcontrollers

- Supports serial I/O via
  - Enhanced Universal Synchronous Asynchronous Receiver Transmitter (EUSART)
  - Master Synchronous Serial Port (MSSP)
    - Serial Peripheral Interface (SPI)
    - Inter-Integrated Circuit (I2C)

  - We will be using SPI & I2C serial communications with the EEPROMs

EUSART

- Supports
  - Asynchronous full duplex
    - Communicate with PC terminal
  - Synchronous
    - Master half duplex
    - Slave half duplex
    - Communicate with peripheral devices
      - A/D or D/A integrated circuits
      - Serial EEPROMs
- Current plan is to not implement code associated with this type of serial I/O
  - This may change though

MSSP

- SPI
  - Four wire interface
    - Clock (SCK)
    - Data in (SDI)
    - Data out (SDO)
    - Chip select ($\overline{CS}$)
      - The master can connect to multiple devices
  - Multi device example
    - Connect multiple devices to same wires for SCK, SDI, SDO
    - Connect individual devices to $\overline{CS}$
  - Synchronous interface with the master providing the clock and selecting the device to communicate with
    - Master initiates data transfer

MSSP

- SPI Registers
  - SSPxCON1
    - SSPEN – enables SPI
    - SSPM – sets mode and clock
    - CKP – sets clock polarity
  - SSPxSTAT
    - SMP – sets when to sample the data
    - CKE – sets when transmission occurs
    - BF – buffer full in receive mode
  - SSPxBUF
    - Data being sent/received
  - SSPxIF
    - Interrupt flag signifying data transfer complete
  - PIE/IPR/PIR
    - Enable interrupt, set the priority, flag the interrupt

MSSP

- I2C
  - Two wire interface
    - Clock (SCL)
    - Data (SDA)
  - Synchronous interface with the clock controlled by the master
    - Master initiates transfer
  - Single master and multiple slaves
  - Multiple masters
  - Transfers are framed
    - Start bit
    - Data bits
    - Stop bit
    - Acknowledgement

MSSP

- I2C Registers
  - SSPxCON1
    - SSPEN – enables I2C
    - SSPM – sets mode and clock
    - CKP – Hold/release clock in slave mode
  - SSPxCON2
    - ACKSTAT – acknowledge status bit
    - ACKDT – acknowledge data bit
    - RCEN – receive enable bit
    - PEN – stop condition enable bit
    - RSEN – repeated start condition enable bit
    - SEN – start condition enable/stretch bit

MSSP

- I2C Registers
  - SSPxSTAT
    - SMP – slew rate control
    - CKE – SMBus select bit
    - D/A – data/$\overline{\text{address}}$ bit
    - P – stop bit
    - S – start bit
    - R/$\overline{\text{W}}$ – read/$\overline{\text{write}}$ information bit
    - UA – update address bit
    - BF – buffer full or empty
  - SSPxBUF
    - Data being sent/received
  - SSPxADD
    - Low byte of address
  - SSPxIF
    - Interrupt flag signifying data transfer complete
  - PIE/IPR/PIR
    - Enable interrupt, set the priority, flag the interrupt

I2C and SPI

- Since SPI has two data wires, serial data in and serial data out, it can support full duplex
  - Send and receive during a single data transfer
  - Requires the additional wire though
- Since I2C has only one data wires, it can only support half duplex
  - Send or receive during a single data transfer

EEPROM Plan

- The PICDEM DEM2 and PIC 18 boards both have an external EEPROM on them
- PIC 18 uses SPI interface and is working on my board with XC8 compiler
- DEM2 uses I2C interface and is working on my board with XC8 compiler (wasn't last year)
- The Mechatronics board doesn't have external an EEPROM
  - We will be using this board for PWM and powering the electric motor

Where Are We Today

- Lab 4
  - Who has the timer at least partially functional
- If you don't have any timer functionality, try
  - Set timer0 to 16 bit mode
  - Set timer0 clock source to internal
  - Set timer0 to use the prescale
  - Set timer0 prescale to 256
  - Make sure clock is running at 4MHz or 8MHz
  - Clear TMR0IF
  - Turn timer0 on
  - Infinite loop
    - If TMR0IF == 1
      - Toggle an LED (should toggle every 4 – 8 seconds)
      - Clear TMR0IF