

PIC18F452 Software Exercises

David Rye

You are to work in a group of two students to write, debug and demonstrate a series of small assembly language and C programs that meet the task requirements listed below. All code is to run on a Microchip PICDEM 2 PLUS Demo Board. In some cases, it will be necessary to breadboard a small amount of electronic circuitry to interface to the Demo Board. The hardware required is very simple.

*Please be very careful when interfacing to the Demo Board: in all cases check the maximum input current and voltage specifications of each pin of the PIC18F452 that you are using, and the maximum current capability of outputs. Study the Demo Board circuit diagram. Power the ICD 3 debugger **before** connecting it to the Demo Board. Power the board before injecting any signals into it. Review techniques for interfacing between discrete transistors, TTL, and CMOS – see the Notes on Interfacing Logic Families. Remember that capacitors and inductors store energy that can discharge into inputs (or outputs!). Use input buffers wherever possible to protect components on the Demo Board and output buffers where necessary to ensure full output voltage swing.*

Reasonable quantities of the electronic components necessary will be provided on request. Note that one would usually use 74HCxx (high-speed CMOS) or 74ACxx (advanced CMOS) components in PIC18F452 designs. Circuits may be patched on the solderless breadboards available in the laboratory, and stored in the lockers between labs.

You are required to complete a Lab induction in Week 1 before you will be permitted to work in the Lab. You must read the documents *Mechatronics Lab: Introductory Notes* and *Mechatronics Lab: Notes on Facilities* before starting work in the Lab. These two documents are available from the Lab website, <http://sydney.edu.au/engineering/aeromech/MTRXLAB/>

Completing the Exercises

Initial reading and familiarisation will be required to understand each exercise, and to write and debug the code. Do not wait until a topic has been covered by lectures: study the material yourself and work as fast as you can. The week given in square brackets for each exercise is not a due date, but indicates where we think you should be up to. Note that you don't have to do the tasks in order. To complete all of the tasks, we suggest that you work as a team of two, assisting and discussing problems within your team as required; you must obtain solutions independently of the other groups. Everyone is expected to have an understanding of the solutions to each of the eight exercises by the end of the assignment. Both lab partners will be questioned as each exercise is signed off.

Remember that there will always be more than one way of solving each problem. Remember also that the principal objective in this group of tasks is for you to learn how to use the resources of the PIC18F452, and to become familiar with the Demo Board, the assembler/compiler, and the MPLAB X IDE.

Exercise Signoff and Assessment

You must demonstrate your working software and hardware to the tutors as each exercise is completed. Do not wait until the end of the exercise set and expect to demonstrate them all.

You should aim to complete all of the eight exercises below, including the extensions. The maximum mark that can be obtained for an exercise that has an extension is 80%. In such a case, the exercise extension must be completed for a (potential) mark of 100% to be reached or exceeded.

Due Dates

The eight exercises do not have individual due dates; you should complete them as soon as you can. All lab work for the exercise set will finish at **5pm on Thursday 10 September 2015**; the functioning of the software/hardware for each task must be demonstrated to the tutor/lecturer during one of your formally scheduled laboratory times *before* this date. Your reports are due in electronic form via Blackboard on or before **10:00 am, Monday 14 September 2015**. Reports or demonstrations, whether fully functional or not, will not be accepted after the applicable due date.

Incremental Development

Some of the exercises have a moderate degree of complexity. You are encouraged to adopt an *incremental* approach to designing and building code and hardware to solve them. That is, *do not* attempt to write, test and then debug all of the code in one go. Begin with a simple subset of the exercise that you are trying to complete. Write the code and test it. Then design and add additional code to increase the functionality of your already-working program. Test again, including code you have previously written, to make sure that the new code is working and the old code has not been broken. This is a simple model of incremental development. For example, in Exercise 1, you might first get the A/D converter running and just write an 8-bit natural binary number to PortB. Then in the second step you may add code to give the specified bit pattern on PortB once you know that the A/D converter is running and your code writes to PortB.

Note also that some of the later exercises reuse code (e.g. serial I/O) from earlier exercises. It is therefore to your advantage for you to make sure that the code you write is easy to use and thoroughly tested!

Lab Books

Each student must keep a hardbound lab book that is brought to every lab session and used to record your calculations, circuit diagrams, tests, results and observations. A lab book should ideally be about A4 size, hardcover, and with pages that are numbered; these features are essential in professional practice. Lab books must be completed in pen and each day's entry initialled and dated. You must not leave any gaps between entries for successive lab sessions. Code listings can be pasted in. A lab book is to be regarded as a working document that is completed in the lab, not as something to be "filled in" when you are back at home. You must make notes in your Lab Book that are sufficiently detailed to allow your group to complete a brief report on each exercise. Assessment of the quality of your lab book will count for 15% of your mark for this assignment. Lab books can (and will) be inspected at any time without further warning for assessment.

Reports

Each group must complete and submit for assessment a single report containing a reflective commentary on the design of the code and circuitry for each exercise: why this particular design was chosen; observations that you made during development, debugging and testing; and responses to particular questions in the exercises. The report should be *brief*; no more than two pages for each exercise *in addition to* circuit diagrams and code listings. Reports will be assessed only if they are accompanied by a completed and signed version of "Cover Sheet Template.doc" that will be available via Blackboard.

Cheating

If you use any code or circuit diagrams from a reference source or fellow student *without acknowledgement* this will be interpreted as an attempt to increase your marks by using (i.e. stealing) someone else's work. Such action will result in harsh sanctions that may include setting marks to zero.

Mark Breakdown

Marks will be awarded for the functioning software and hardware, the Lab Book, and the report. Each student's mark for this exercise set will be calculated as follows

- Software/hardware: 60%;
- Lab book: 15%;
- Group report: 25%.

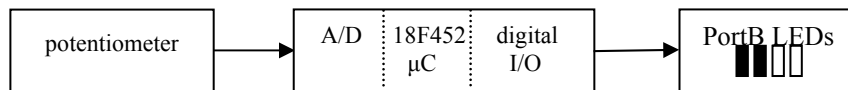
The marks of individuals will be moderated on the basis of individual effort and understanding in the laboratory, as assessed by the tutors and lecturer.

Exercise 1: Analog Input and Digital Output

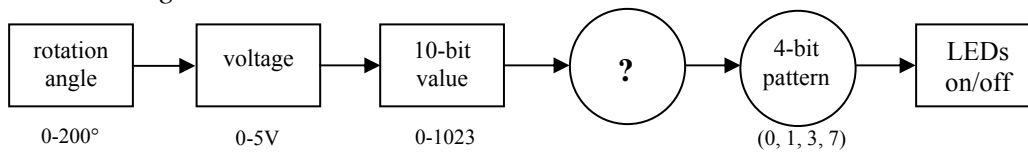
Task: (a) Write an assembly language program that reads the analogue voltage generated by potentiometer R16 on the Demo Board, converts it to a 4-bit number, and writes that number to the PortB LEDs to give a ‘light bar’ bit pattern as the potentiometer is turned.

Study: Peatman Ch. 10; *PIC18FXX2 Data Sheet* and *PICMicro 18C Family Reference Manual*; A/D converter interfacing; I/O Port B; Instruction set; Demo Board circuit diagram, MPLAB IDE, MPASM Assembler.

Context (Block) Diagram:



Data Flow Diagram:



Exercise 2: Analog Input and PWM Output Using CCP PWM Mode

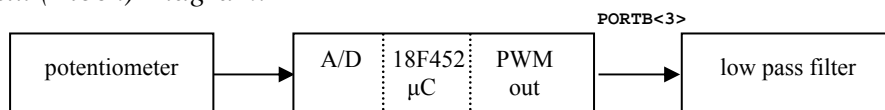
Task: (a) Write an assembly language program that converts an analog voltage generated by potentiometer R16 to equivalent pulse-width modulated output on **PORTB<3>** using the PWM mode of **CCP2**.

(b) Filter the PWM output with a passive low-pass (RC) filter to produce an analog signal that is (near-) identical to the voltage input from the potentiometer.

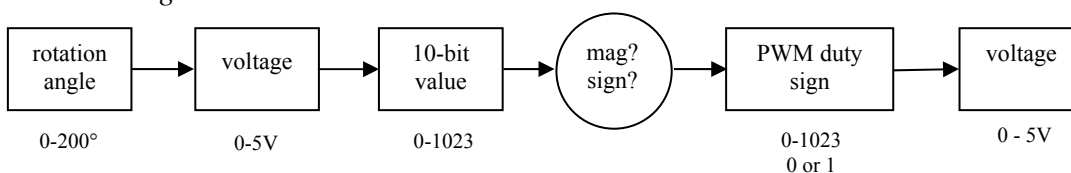
(c) What effect does changing the low-pass filter time constant ($T = 2\pi RC$ sec) have?

Study: *PIC18FXX2 Data Sheet* and *PICMicro 18C Family Reference Manual*, A/D conversion; CCP PWM mode; I/O ports; Interfacing; Low-pass filter; Oscilloscope.

Context (Block) Diagram:



Data Flow Diagram:



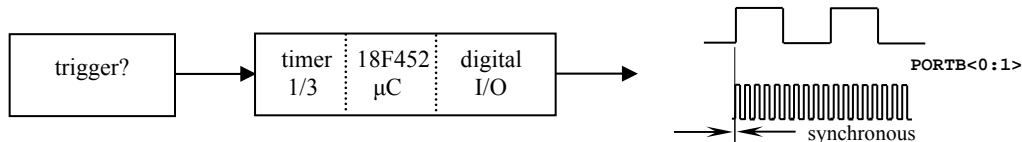
Exercise 3: CCP Timing and Interrupts

Task: (a) Write assembly language code that uses the Compare function of **CCP1** and/or **CCP2** together with the PIC's interrupt system to implement suitable delays. Generate a 0.500 Hz square wave on **PORTB<2>** and a synchronous 50.00 Hz square wave on **PORTB<3>**. Both square waves must have 50% duty cycle.

What is the timing resolution that can be achieved?

Study: Peatman Ch. 5, Ch. 9, Ch 13; *PIC18FXX2 Data Sheet* and *PICMicro 18C Family Reference Manual*, Hardware timing; CCP; timers, Interrupts, Oscilloscope.

Context (Block) Diagram:

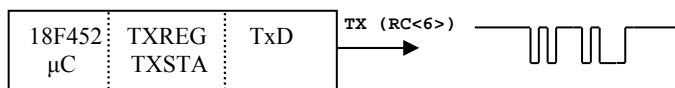


Exercise 4: RS232 Serial Data Transmission

Task: (a) Write an assembly language program that transmits a null-terminated string of ASCII characters (your choice!) on a 3-wire serial (RS-232) line at 9600 baud. The code must be in the form of a subroutine called **tx232** that is passed a pointer to the start of the string to be transmitted. Indirect addressing (pointers) must be used to access characters in the string. Your code must use the null terminator to determine when to stop transmitting the string. Display the transmitted characters on a PC using the *Tera Term* terminal emulator.

Study: Peatman Ch. 9, Ch. 18, *PIC18FXX2 Data Sheet* and *PICMicro 18C Family Reference Manual*, Serial data transmission; Data structures: Strings, tables & queues - see Peatman, *Design With Microcontrollers*, ch. 5, McGraw-Hill, 1988, 621.3981 6; addressing modes.

Context (Block) Diagram:



Extension: (b) Add an interrupt-driven serial receive subroutine called **rx232** that places a received string in a buffer. Good practice would be to use a circular buffer, and access it using an insertion pointer and a removal pointer. When the string terminator is received, use the subroutine **tx232** to transmit the following string

You said: <rxstr><cr><lf>

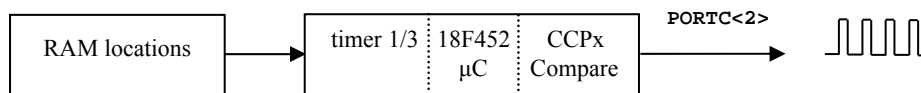
where **<rxstr>** is replaced by the string that was received, and **<cr><lf>** represent the ASCII carriage return and line feed characters 0x0D and 0x0A. Note that one cannot explicitly send the null terminator **<nul>** (0x00) from the keyboard using *Tera Term*. Instead, you should treat the **<cr>** as the string terminator, and replace it with **<nul>** on receive.

Exercise 5: Generalized Variable Output Using CCP Compare Mode

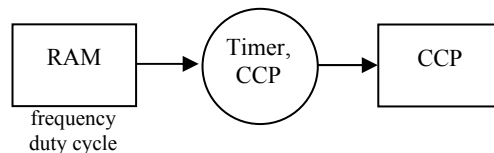
Task: (a) Write an assembly language program that generates a PWM signal on `PORTC<2>` that has adjustable base frequency and duty cycle. The base frequency must be adjustable between 200 Hz and 10 kHz, and the duty cycle between 0% and 100%. Store the values of the base frequency in Hz and the duty cycle (*not* base period and on period) as two numbers in fixed locations (i.e. “global” variables) in RAM, and use MPLAB to change their values while the assembler program is running. The program is *not* to use the CCPx PWM mode.

Study: Peatman Ch. 14, Ch. 16; *PIC18FXX2 Data Sheet* and *PICMicro 18C Family Reference Manual*, Timers; CCP Compare mode.

Context (Block) Diagram:



Data Flow Diagram:



Extension: (b) Add an interrupt-driven serial receive subroutine that receives commands to set the base frequency and duty cycle, parses the arguments out and writes them into the global variables, so that the tone can be adjusted from a terminal emulator, such as *Tera Term*.

Exercise 6: Analog Input and Digital Output, C Code

Task: (a) As for Exercise 1 (potentiometer to single LED), but write the code in C. Take care to make your C code as simple as it can be to help with the extension!

Study: *PIC18FXX2 Data Sheet* and *PICMicro 18C Family Reference Manual*, *MPLAB C18 C Compiler Getting Started*, *MPLAB C18 Compiler User's Manual*, C start-up code source examples.

Extension: (b) Make a studied comparison between the code generated by the compiler and the assembly code that you wrote in Exercise 1. Compare the size of code generated, the addressing mode(s) used and the ease of understanding. Make the comparison both with all compilation optimizations off, and all on.

Exercise 7: RS232 Serial Data Transmission, C Code

Task: (a) As for Exercise 4 (RS232 serial data transmission), but code in C.

(b) In addition, add an interrupt-driven serial receive function `void rx232Isr(void)` that places a received string in a buffer. Good practice would be to use a circular buffer, and access it using an insertion pointer and a removal pointer. When the `<cr>` terminator is received, use another function with the prototype

```
int status tx232C( unsigned char * txPtr );
```

to transmit the string

```
You said: <rxstr><cr><lf>
```

where `txPtr` is a pointer to the string that is to be transmitted; `status` is a return status code; `<rxstr>` is replaced by the string that was received, and `<cr><lf>` represent the ASCII carriage return and line feed characters. The function `tx232C()` may (and should) use interrupts.

Exercise 8: Synchronous Serial Expansion via SPI Bus

Task: (a) Write a C language program that reads an 8-bit number from a 74HC165 parallel-in/serial-out shift register and writes that number to a 74HC164 serial-in/parallel-out shift register. The value written must be updated at a frequency of 30 Hz.

You will need to construct the 74HC164 and 74HC165 circuits on solderless breadboard. Use switches (or wires) to set the 8-bit number that is read, and LEDs to display the value written. Please be aware that Microchip provides libraries of C functions that support its hardware.

Study: *PIC18FXX2 Data Sheet* and *PICMicro 18C Family Reference Manual*, SPI and interrupts, 74HC164 and 74HC165 data sheets.