EE2A2

Introductory Material: Using the C language on a PIC Processor

1.0 Aims

(i) To take a bare microprocessor and to convert it into a working micro-controller system programmed using the C language.

(ii) To use IO-mapped data structures to talk to interface pins.

(iii) To consider the basic functionality of a synchronous finite state machine and the equivalent implementation on a processor.

(iv) To combine multiple, disjoint program fragments together into one package that can be rapidly demonstrated.

(v) To implement logical bit-wise operations in software

(vi) To implement look-up tables on both inputs and outputs (lambda logic) of synthesised finite state machines.

This experiment is designed to allow students to familiarise themselves with developing a microcontroller system programmed with the C language. It will include programming the device using an in-circuit programmer, developing IO-mapped structures, the use of bit-wise logic operators and the relationships between a microcontroller and a synchronous finite state machine.

2.0 Learning Outcomes

Deliverables:

i) A working system.

ii) Well-structured and -commented PIC programs that interface to the outside world according to the specification given in Section 3.1.

iii) A log book containing any preparatory work undertaken outside the lab, design decisions, test procedures and a discussion of the issues involved.

iv) Ability to generate your own mark scheme and rigorous evaluation procedures (can you pre-empt a cynical boss?).

What will be assessed:

i) The program source code, with particular emphasis on adequate comments, program layout and elegance.

ii) A practical demonstration.

iii) The adequacy of the log book (lab books not written up during the experiment will incur penalty marks).

3.0 Activities

3.1 Specification

3.1.0 Demonstrate that clock microprocessor is running at 4 MHz

The PIC 18F27K40 may be clocked at speeds from DC to 64 MHz. A high-precision internal oscillator block (HFINTOSC) may be used to select frequencies range up to 64 MHz. Examine the include file (18F27K40.h) to find out which parameters and commands may be used. A clock signal at a frequency of one quarter of the main frequency may be configured to appear on pin RA.6 (Hint: #use delay(internal=64MHZ,clock_out)).

3.1.1 Demonstrate that clock microprocessor is running at 8 MHz

Repeat 3.1.0 with the main clock running at 8 MHz Ideally, find out how to change the clock speed once the main program has started. Even better, read Section 4 of the data sheet to see how manufacturer's define their products and look at the 'lst' file to see how your code is converted into register settings.

3.1.2 Demonstrate that clock microprocessor is running at 16 MHz

Repeat 3.1.0 with the main clock running at 16 MHz.

3.1.3 Single Pin Output

A synchronous finite state machine is to be simulated where the output connected to Port C0 toggles on each clock cycle (the equivalent of a T-type flip-flop with the T input connected high). You are to simulate a 2 Hz clock applied to the synchronous finite state machine, such that the output changes at a 1 Hz rate. A single-bit memory-mapped IO structure, or variable, should be used. Ensure that the output rate remains constant as the clock frequency is varied repeatedly (4 MHz, 8 MHz, 16 MHz)

3.1.4 Multiple Pin Output (i)

A synchronous finite state machine is to be simulated where the output connected to Port C0..5 implements a 6-bit pure-binary up-counter. You are to simulate a 1 Hz clock applied to the synchronous finite state machine. A six-bit memory-mapped IO structure, or variable, should be used.

3.1.5 Multiple Pin Output (ii)

A synchronous finite state machine is to be simulated where the output connected to Port C0..5 implements a 6-bit pure-binary down-counter. You are to simulate a 1 Hz clock applied to the synchronous finite state machine. A six-bit memory-mapped IO structure, or variable, should be used.

3.1.6 Multiple Pin Output with output logic

A synchronous finite state machine is to be simulated where the output connected to Port C0..5 implements a 6-bit Gray-code up-counter. This is to be achieved by using the equivalent of 'output logic' to convert between a binary value and a Gray code value – this is called a look-up or hashing table. You are to simulate a 1 Hz clock applied to the synchronous finite state machine. A six-bit memory-mapped IO structure, or variable, should be used.

3.1.7 Multiple Output Groups with output logic

A synchronous finite state machine is to be simulated where the output connected to Port C0..2 implements a 3-bit pure-binary up-counter, whilst Port C3..5 implements a 3-bit Gray-code up-counter. You are to simulate a 1 Hz clock applied to the synchronous finite state machine. A suitable memory-mapped IO structure should be used. Note, multiple structures mapped to the same IO locations will be required.

3.1.8 Multiple Input Groups – demonstration of bit-wise AND operation

Connect the 8-way DIP switch to the pint of port B. Connect the other side of the switches to 0V. Configure all pins of Port to be inputs with weak pull-up resistors (read section 15 of the PIC 18F27K40 manual, the CCS compiler manual and the header file to understand about the WPU register). Configure the IO mapping structure such that a two-bit structure-member variable is mapped to pins B0..1 and a second two-bit structure-member variable is mapped to pins B2..3. Bit-wise AND the two two-bit variables together an output the result to a structure-member variable mapped to pins C0..1.

3.1.9 Multiple Input Groups – demonstration of bit-wise OR operation

Repeat 3.1.8 with the AND operation replaced by an OR operation.

3.1.10 Multiple Input Groups – demonstration of bit-wise NOT operation

Repeat 3.1.8 with the pins C0..3 being the inverse (NOT) of pins B0..3.

3.1.11 Multiple Input Groups – demonstration of bit-wise Exclusive-OR operation

Repeat 3.1.8 with the AND operation replaced by an Exclusive-OR operation.

3.1.12 Demonstration of Set-Reset flip-flop simulation

Assume that B0 is the set input and that B1 is the reset input of a set-reset flip flop. Pin C0 should be assumed to be the Q output – implement the bistable element.

3.1.13 Demonstration of input look-up tables

Use the four inputs derived from pins B0..3 to drive the indices of an input look-up table. Connect eight outputs to pins C0..7 and implement either a running-light or bar-graph type display. The input pins will select one of 16 non-linearly spaced sampling rates. Make sure that your readership understands why input look-up tables are so good at covering all possible input conditions in a single line of code.

3.1.14 Demonstration of 'bright-eyed and bushy tailed capability'

Be creative and play – demonstrate something interesting (you have actually been learning about inputs, outputs and mapping multiple structures to the same IO pins)

3.1.15 Combined all above experiments into one program

Use pins B4..B7 to drive a 'case' statement (or sixteen 'if' statements) to combine all the sub-sections into a single program that will allow you to demonstrate all the individual component programs extremely rapidly.

That's it – now do it!

4.1 Preparatory Work

The following sections aim to guide you through some of the design processes.

4.1.1 Hardware

The PIC 18F27K40 is ideal for this laboratory experiment as it contains an internal RC oscillator (thus saving two pins, a crystal and two capacitors). Pins '*Vss*' are connected to 0V and pin '*Vdd*' is connected to +5V (Decouple this supply with a 100 nF capacitor as close to the IC pins as possible). Overall, there are 28 pins on the device of which probably 20 may be freely used during your experiments, see Figure 1.

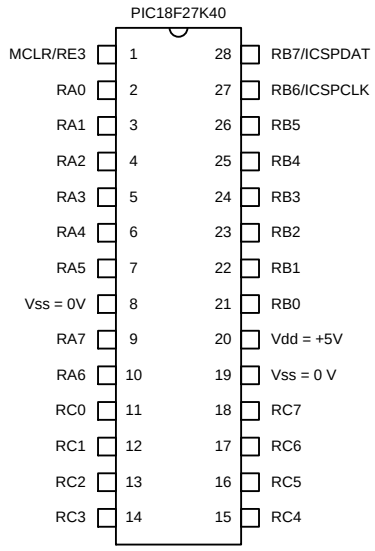


Figure 1: PIC 18F27K40 microcontroller

You will use the PICKit 3 to program the device. This contains a 6-way socket of which pins 1 to 5 will be used – Pin 1 is identified by a triangle. The interconnections are shown in Figure 2 and the PICKit 3 requires power derived from an external source (pins 2 and 3).

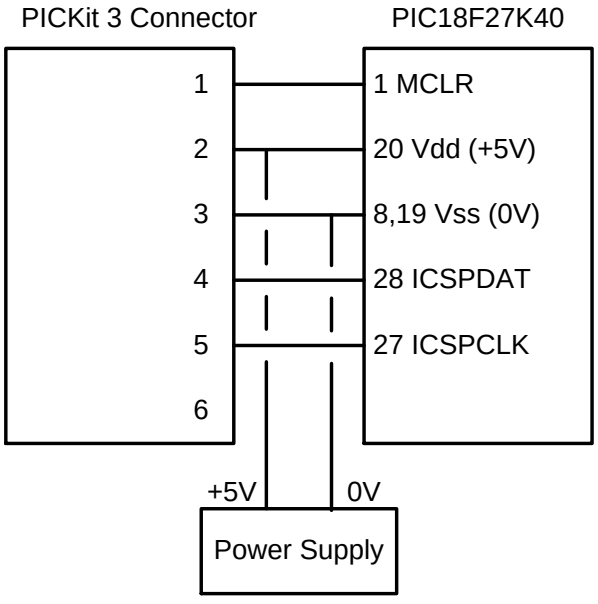


Figure 2: In Circuit Serial Programming Interface

It is suggested that you connect the 8 single-pole, single-throw (SPST) switch module to pins RB0..RB7. The other side of these switches should be connected to 0V, such that when the switch is closed a logic '0' is applied to the relevant IO pin and when the switch is open a logic '1' is applied as a result of the internal pull-up resistors within the PIC. Eight LEDs should be connected

to pins on Port C0..7, each being protected by a series 270R series resistor. Connect pins RB6, RB7 and RE3 to the PICkit 3 programming interface.

PW1. Draw a complete schematic circuit diagram of the PIC 18F27K40, programmer and the interface components. This will be the circuit diagram that you will wire up within the laboratory. All power supply connections, LEDs, resistors and switches should be clearly labelled.

5.1 Laboratory Activities

Starting with a bare microcontroller integrated circuit is going to be scary for the first time as there are so many things to go wrong. The development flow chain is shown in Figure 3.

First, we enter the C code using a text editor. I prefer to use the embedded editor within the CCS Integrated Development Environment (IDE) as this provides the most rapid code development cycle. Other exceptionally good editors that give you assistance as you type include Notepad++.

The flow chain now needs a compiler to generate machine executable code. This compiler will be specific to the device you are integrating and will be very different in nature to a standard PC compiler. Examples include the CCS PCWH which will cost you $500 per machine.

Having generated the executable code, this must be transferred to a suitable programming software environment. To keep things minimally simple, I tend to use the MPLAB Integrated Programming Environment. For these laboratory sessions, you will connect a PICKIT 3 programmer via a USB interface to your PC.

Finally, the programmer will transfer data to your microcontroller using a serial interface sometimes known as an In Circuit Serial Programmer (ICSP).

More details about each of these software suits is provided in Appendix 1.

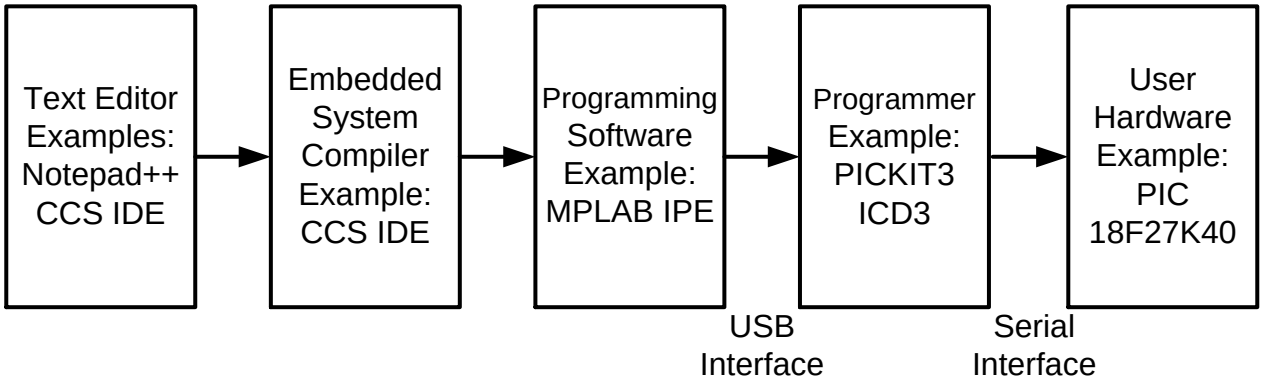


Figure *3*: Development flow chain

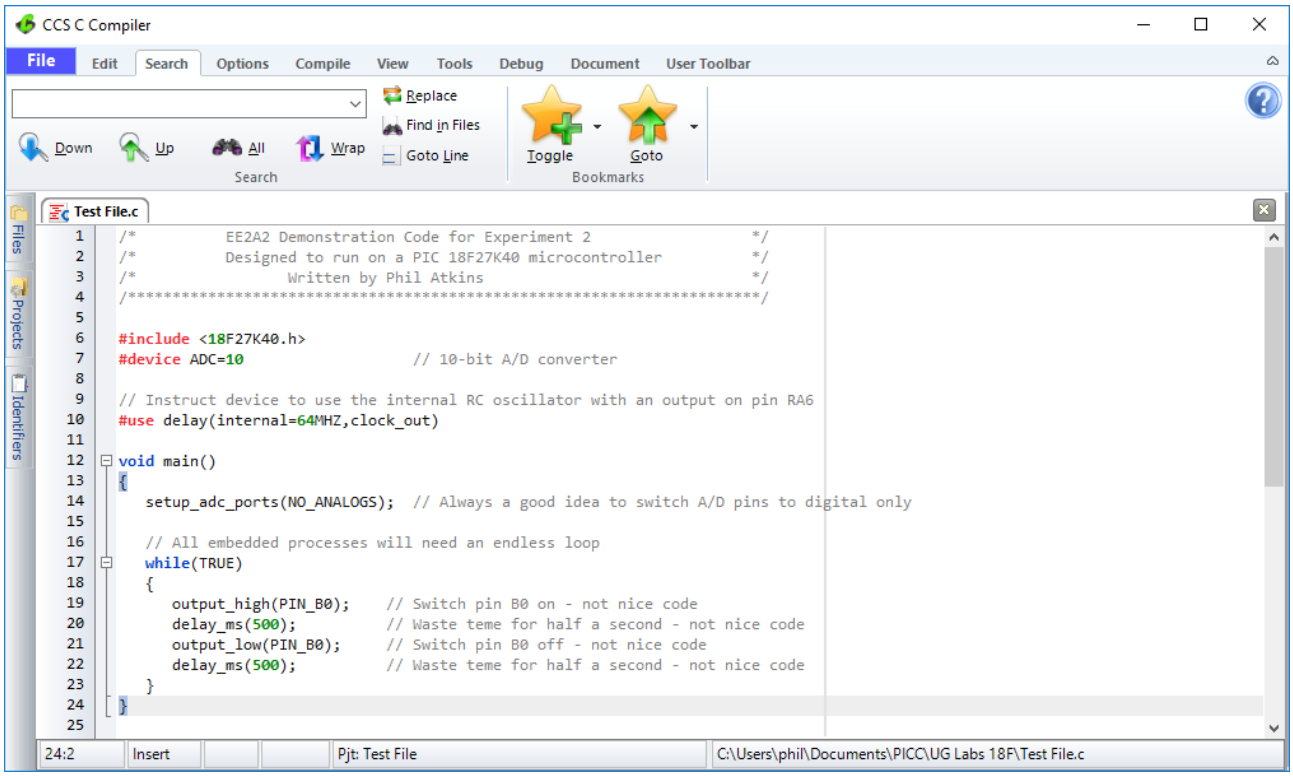
Appendix 1

Development Flow Chain

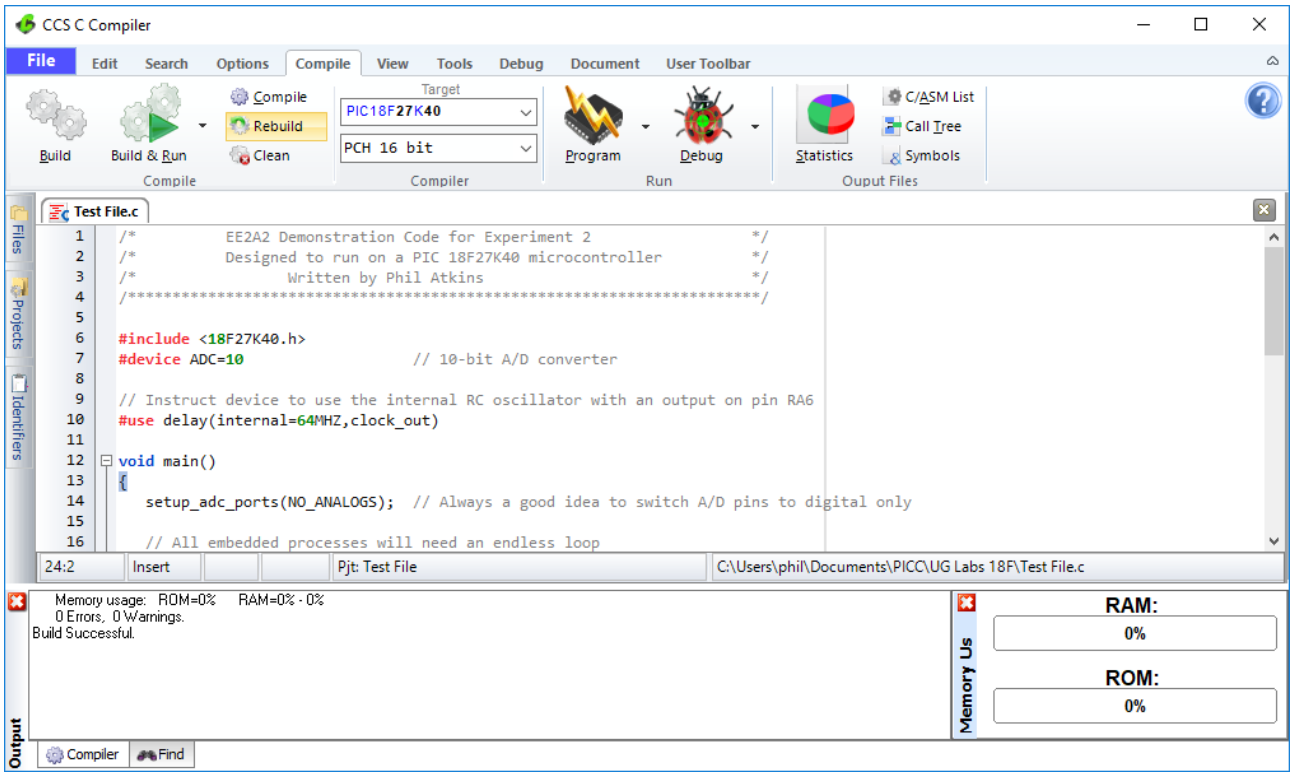
CCS Environment

Type ‘PIC C Compiler’ in the Windows program search box. It is likely that the compiler will open with the ‘Search’ tab in focus. Click the ‘File’ tab, then the ‘New’ item pull-down selection and finally click on ‘Source File’. A ‘Save As’ dialogue box will appear, navigate to you Y drive files and enter a suitable filename such as ‘EE2A Lab 2.c’.

You can now enter a few lines of test code. Most experienced designers enter a very small number of lines between compiles (typically five new lines) and then check that they function as expected. You may get disheartened trying to fix bugs in large programs entered in one go.

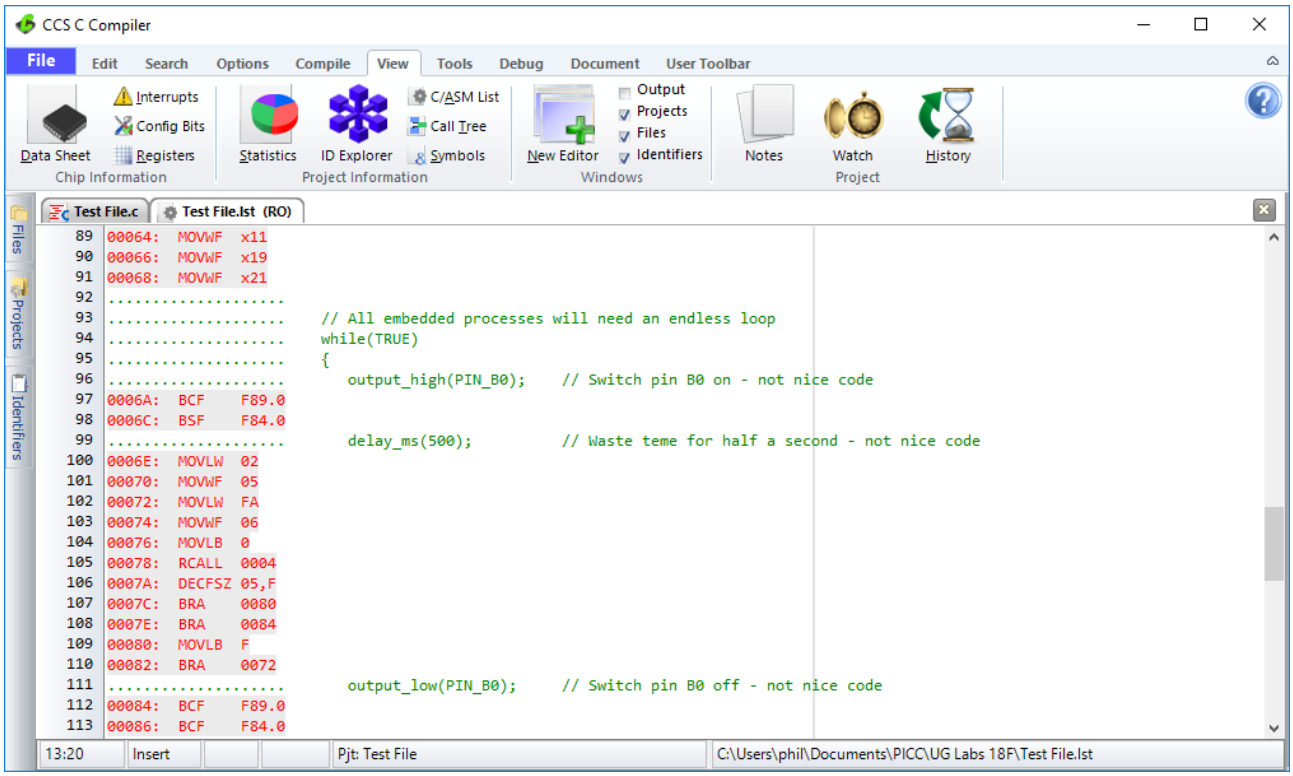


Having entered the code, save it and click the ‘Compile’ tab. Now click ‘Build’. A pop-up menu will appear to inform you of success, or failure. After a few seconds the screen should revert to something like:



If there are errors, hopefully some information is provided about the type and location of the source of the error. However, an error such as a missing semicolon at the end of a line may have occurred many lines before.

You have now generated some operational code, experienced programmers will always keep an eye on the assembly code generated. Do this by clicking the 'View' tab and the 'C\ASM List' icon. You will be able to scroll down through your original C code statements and the assembly code that has been generated to implement your code – compilers do make mistakes and you will be expected to find them in the workplace.



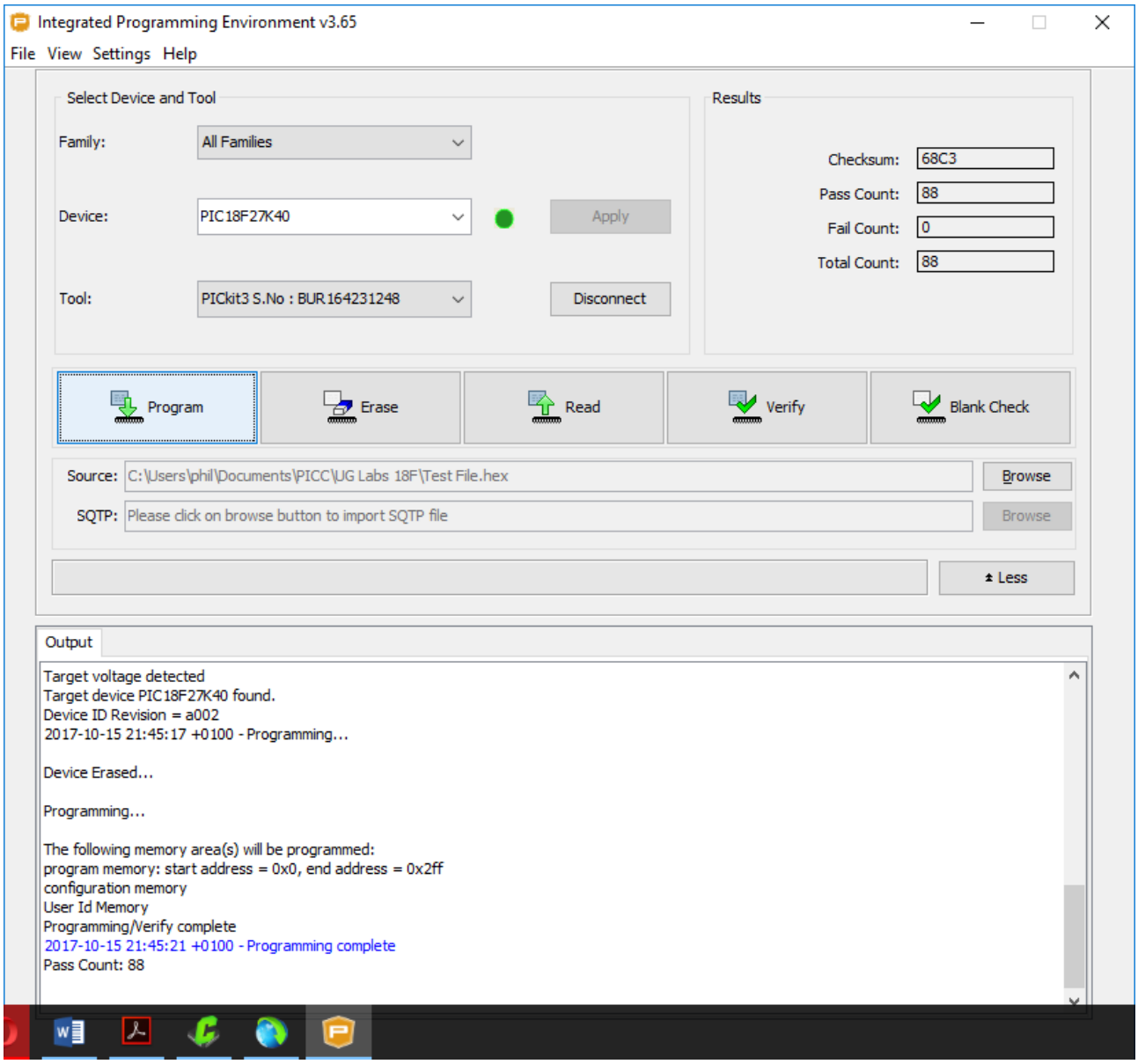
The MPLAB IPE Environment

Type ‘MPLAB IPE’ in the Windows program search box. Run the latest version of the Integrated Programming Environment installed on your PC (this is freely downloadable from Microchip along with some restricted C compilers). First select the ‘Device’ to be a PIC18F27K40. Now click the ‘File’ table, followed by ‘Import’ and then ‘Hex’. Navigated to your Y drive where your source file is stored. Always display the ‘details’ of when the file was created – the most common error is to accidentally have two copies of the same filename on your machine. One file gets updated, whilst the other (unchanged file) gets repeatedly loaded to the device. You should obtain a message in the output panel that the hex file has been loaded successfully.

Click the ‘connect’ button. If the programmer detects a valid device and that it is powered, you will obtain a series of messages such as:

Target voltage detected
Target device PIC18F27K40 found.
Device ID Revision = a002

Click the ‘program’ button. If everything is OK, you will obtain a screen such as that shown below.


Integrated Programming Environment v3.65
File View Settings Help
Select Device and Tool
Family:
All Families
Device:
PIC18F27K40
Apply
Tool:
PICkit3 S.No : BUR164231248
Disconnect
Results
Checksum:
68C3
Pass Count:
88
Fail Count:
0
Total Count:
88
Program
Erase
Read
Verify
Blank Check
Source:
C:\Users\phil\Documents\PICC\UG Labs 18F\Test File.hex
Browse
SQTP:
Please click on browse button to import SQTP file
Browse
Less
Output
Target voltage detected
Target device PIC18F27K40 found.
Device ID Revision = a002
2017-10-15 21:45:17 +0100 - Programming...
Device Erased...
Programming...
The following memory area(s) will be programmed:
program memory: start address = 0x0, end address = 0x2ff
configuration memory
User Id Memory
Programming/Verify complete
2017-10-15 21:45:21 +0100 - Programming complete
Pass Count: 88

Checklist to Obtain a 'W' during Demonstration

Switch Code	Comment	Seen
0000	Demonstrate that clock microprocessor is running at 4 MHz, pin RA6 produces 1 MHz	
0001	Demonstrate that clock microprocessor is running at 8 MHz, pin RA6 produces 2 MHz	
0010	Demonstrate that clock microprocessor is running at 16 MHz, pin RA6 produces 4 MHz	
0011	Demonstrate T-type flip flop in toggle mode. Pin RC0 produces 1 Hz	
0100	Demonstrate that Port C0..5 implements a 6-bit pure-binary up-counter.	
0101	Demonstrate that Port C0..5 implements a 6-bit pure-binary down-counter.	
0110	Demonstrate an output look-up table via a 6-bit Gray-code up-counter	
0111	Demonstrate multiple output variables: 3-bit pure-binary up-counter and 3-bit Gray-code up-counter.	
1000	Demonstrate bitwise AND operation.	
1001	Demonstrate bitwise OR operation.	
1010	Demonstrate bitwise NOT operation.	
1011	Demonstrate bitwise Exclusive-OR operation.	
1100	Demonstrate set-reset flip-flop synthesis.	
1101	Demonstrate input look-up table.	
1110	Demonstration of 'bright-eyed and bushy tailed capability'	
	All the above controlled by four switches and a 'switch/case' construct.	
	Note: The above tasks will require multiple overlaid structures mapped to the same IO pins.	

Name:

EE2A Experiment 2

PIC Introduction

Feedback Mark Sheet – Paste into the logbook

The following will be assessed during the laboratory session (PGTA to circle, date and sign)

Circuit construction and demonstration:	Yes	Partially	No
Demonstration of sixteen programming constructs linking finite state machines and embedded controllers (35%)	**W**	**WP**	**NW**

Inspection Mark For Log Book:	Could not be better	Good attempt	Room for improvement	Appalling
A 'flick-test' of the log book will be carried out. This will be based on the purpose of the log book – to convey useful information to other engineers and to allow others to carry on with the work.	**4**	**3**	**2**	**1**

At the end of the year, the log-book will be handed in and assessed using the following criteria:

Preparatory Work:
Evidence of adequate preparatory work undertaken outside laboratory (15%). Students to have decided what is 'adequate' as part of their work.
Source Code:
Existence and completeness of the program header (author, date, filename, target device, fuse settings, program function) (5%). Appropriateness and clarity of comments, labels and variable/constant definitions (5%). Efficient use of code (i.e. no redundancy) (5%). Program Elegance (5%). Technical content (10%). Code print-outs are assumed.
Reflective journal:
Care and neatness of preparatory work (written up outside lab) (5%). Sensible attempt at keeping a log-book (written up at the time) to convey engineering information to other professionals. (10%). If this student was a professional engineer who left his/her organization today, could another engineer pick up the pieces in six months time ? Conclusions- sensible executive summary & record of the learning experiences gained by the student during this experiment. (5%).

Verbal feedback will be provided during the laboratory.

Timeliness (Autumn Term)

Week in which experiment was demonstrated to supervisor (shaded blocks show overrun)

Week 4	Week 5	Week 6	Week 8	Week 9	Week 10	Week 11
	Excellent Progress	Excellent Progress	Excellent Progress	Good Progress	Average Progress	Getting Worried