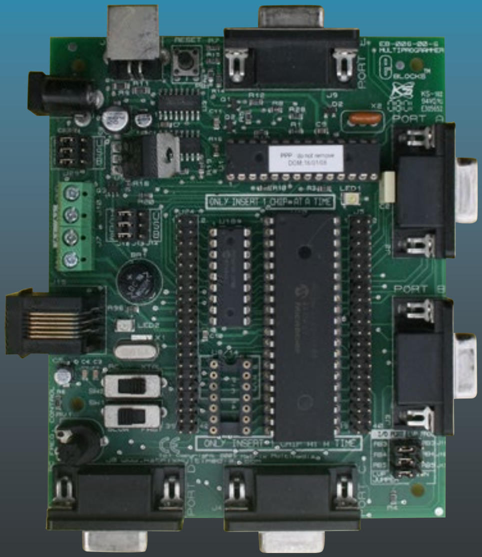
TECHNOLOGY IN ACTION™



C Programming for the PIC Microcontroller

Demystify Coding with Embedded Programming

—

Hubert Henry Ward

Apress®

C Programming for the PIC Microcontroller

Demystify Coding with Embedded Programming

Hubert Henry Ward

Apress®

C Programming for the PIC Microcontroller: Demystify Coding with Embedded Programming

Hubert Henry Ward
Lancashire, UK

ISBN-13 (pbk): 978-1-4842-5524-7 ISBN-13 (electronic): 978-1-4842-5525-4
https://doi.org/10.1007/978-1-4842-5525-4

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Natalie Pao
Development Editor: James Markham
Coordinating Editor: Jessica Vakili

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-5524-7. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Dedicated to my wife Ann

Table of Contents

About the Author

Hubert Henry Ward has over 24 years of experience in teaching students at the Higher National Certificate and the Higher Diploma in Electrical and Electronic Engineering. Hubert has a 2.1 Honours Bachelor's Degree in Electrical & Electronic Engineering. Hubert has also worked as a college lecturer and consultant in embedded programming. His work has established his expertise in the assembler language and C, MPLABX, and designing electronic circuit and PCBs using ECAD software. Hubert was also the UK technical expert in Mechatronics for 3 years, training the UK team and taking them to enter in the Skills Olympics in Seoul 2001, resulting in one of the best outcomes to date for the United Kingdom in Mechatronics.

About the Technical Reviewer

Leigh Orme is a graduate engineer at SSE plc in Greater Manchester, United Kingdom. He has an electrical and electronic engineering degree from Manchester Metropolitan University.

Introduction

This book looks at programming a PIC microcontroller in C. We'll study the following aspects of programming the PIC:

1. Looking at some of the background to the program language for micros
2. Creating a project in the Microchip IDE MPLABX
3. Configuring the PIC
4. Setting up the oscillator using the internal oscillator block
5. Setting up some digital inputs and outputs
6. Simulating a simple program using the simulator in MPLABX
7. Creating a simple delay and a variable delay
8. Using the ADC to accommodate an analogue input to the PIC
9. Using an LCD in both 4-bit and 8-bit mode to display data
10. How to make a header file to save writing the same instructions again in every project.
11. Using arrays and controlling how you step through an array

The Aim of the Book

The aim of this book is to introduce the reader to PIC microcontrollers and writing programs in 'C'. There is some background information starting with what a PIC is and some aspects of programming languages. It will then move onto what an IDE is and how to use MPLABX, one of the most common industrial IDEs. MPLABX is an IDE that is freely available from the Microchip web site. The 'C' compiler is their free compiler that again can be downloaded from their web site. Note that I use MPLABX version 5.2 and the XC8 compiler version 2.05 or 1.35. These can be downloaded from the archive section of their web site.

Then the text moves on to the exciting world of writing programs for microcontrollers. It is based around the range of microcontrollers, termed PIC micros, available from Microchip. It will show you how to write programs without buying any devices or equipment as you can use the MPLABX simulators that come free with the MPLABX IDE. If you have access to an ECAD package, such as PROTEUS or Tina, that has the ability to run 8-bit or 16-bit and so on micros, then it will show you how to use that software to run your programs, again without buying any equipment.

This book is based around the PIC18F4525 as it has the advantage of being a 40 pin dual in line package. This means it is quite easy for the hobbyist to create a practical circuit on vero board or even a small PCB.

My other books cover using the PIC to control a variety of DC motors such as simple DC motors using PWM to control the speed of the DC motor, stepper motors, and servo motors. I also have a short book looking at communications for the 18F4525. Apart from those books, I am writing another range of books on how to use a 32-bit PIC, but this is a surface mount device which makes it rather more difficult to build practical circuits. However, the 32-bit PICs have some very useful additions.

The PIC18F4525 is a very useful PIC with 5 ports giving us the use of 36 I/O. It has 4 timers and 3 external interrupt sources. It has a two-CCP module with the ability to provide two separate PWM outputs, and it has full bridge drive capabilities. There are more functions available, and they all make the PIC18F4525 a very useful microcontroller.

The Objectives of the Book

After reading this book, you should be able to do the following:

- Write PIC programs in C
- Use the main features of the MPLABX IDE
- Interface the PIC to the real world
- Design and create useful programs based around the PIC18F4525
- Enjoy delving into the exciting world of embedded programming

I hope you enjoy reading this book and find it very useful. I firmly believe that programmers should not just put together blocks of code, which perform the functions they want, to create a program. To be a good programmer, with the versatility to alter their programs to cope with the wide variety of microcontrollers and their different oscillator choices, you need to know how the code works. In my many years of teaching this subject, I have often been told that to create a 1-second delay, you simply write the instruction delay (1000). Well, that only works for a certain oscillator frequency and timer setting. To be able to create a delay using any oscillator, you need to understand how your timer counts and at what frequency it counts. Armed with that sort of deeper understanding, you will be a better programmer.

This book is aimed at giving you the full understanding of the fundamental aspects of the microcontroller and how it works. Then, with a deeper understanding of how the different control registers control the micro, you will become a programmer who will, with experience, fully control your device and not rely on bits of code, which you don't understand, doing the programming for you. It is essential that we have programmers who have this deep appreciation of their microcontrollers, and I hope that after reading this book, you are on your way to becoming one of those programmers.

The Prerequites for the Book

There are none really, but if you understand 'C' programming, it would be useful. Also, if you understand the binary and hexadecimal number systems, it would be an advantage, but there is a section in the Appendix that will help you with that.

CHAPTER 1

Introduction

This chapter covers some of the fundamentals of what a microprocessor-based system is and how a microcontroller is different. It then covers the historic development of the 'C' programming language for PIC controllers.

After reading this chapter, you should appreciate how the micro sees your instructions and understand the terms machine code, assembler, compiler, and linker.

Programmable Industrial Controllers

Programmable Industrial Controllers (PICs) is really just a trademark for the microcontrollers produced by Microchip, or so I have been led to believe. Some say it stands for Programmable Industrial Controllers, or Programmable Intelligent Controller, or Programmable Interface Controller. However, the term PIC is used by Microchip to cover an extremely wide range of microcontrollers produced by them. I will simply refer to the microcontroller as the PIC.

Each PIC will have all the components of a microprocessor-based system as shown in Figure 1-1, such as

- A microprocessor
- ROM, RAM
- An I/O chip
- The associated address, data, and control buses

https://doi.org/10.1007/978-1-4842-5525-4_1

However, all these parts are all on a single chip, not several as with older microprocessor-based systems. This means it is really a single-chip computer.

As well as all that, the PIC has much more circuitry on the single chip. This extra circuitry is used to make it into a control system with a micro at the heart of it. The extra circuit may include an ADC, opamp circuits for compare and capture, a PWM module, and a UART module. These extra circuits may not be on all PICs as PICs vary in their makeup for different applications.

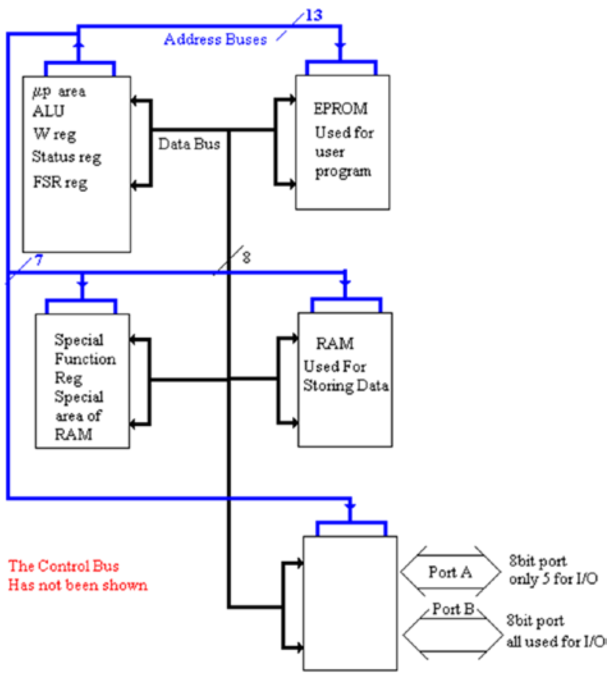


Figure 1-1. *The Basic Microprocessor System*

One more thing before we move on is that the PIC is a RISC chip as opposed to a CISC chip. RISC stands for reduced instruction set chip, whereas CISC stands for complex instruction set chip. Indeed, the instruction set for PIC micros ranges from 35 to 75 core instructions. However, the 18F4525 has an extended instruction set at your disposal. The Intel processor, which is a CISC chip, uses hundreds of instructions. So the PIC is pretty efficient.

Programming Languages

There is a wide variety of programming languages for microprocessor-based systems. However, all microprocessors only understand voltage levels, ideally 5V and 0V. These two voltage levels are how all microprocessors understand logic which has only two states which are "yes or no," 5V or 0V, and now 3.3v and 0v as with the 32-bit PICs.

It is because of this that the binary number system is commonly used in microprocessor-based systems. This is because binary only has two discrete digits '1' and '0'.

Consider the following binary number:

10101001

This really represents

5v0v5v0v5v0v0v5v

The 5v and 0v is really the only language that all microprocessors understand. However, we can easily use binary to represent the 5v and 0v as '1' and '0'. So writing in binary is easier than writing 5v and 0v.

Machine Code

This then is the birth of "machine code," the most basic programming language termed low level as it is at the level that the micro understands.

Now consider the following:

A9

This is a hexadecimal representation of the 8 binary bits 10101001. It is used to enable programmers to represent binary numbers in a less complicated manner to avoid mistakes, as its very easy to write a '0' instead of a '1'. However, the early programmers actually wrote their programs in the binary machine code to make them faster. There is a section in the Appendix that covers the binary and hexadecimal number systems which is something you need to understand. See Appendix 7.

Now consider the following:

> LDA#
>
> This is actually termed "mnemonics" which stands for an alphanumeric code used to represent the instruction.
>
> The mnemonic LDA# represents the instruction

LoaD the Accumulator immediately with the number that follows:

> 'LD' for load, 'A' for accumulator, and '#' for immediately.

It is fairly obvious that we, as humans, can learn to understand the mnemonics quicker than hexadecimal or the binary of the machine code. However, the microprocessor does not understand these mnemonics. Somehow the mnemonics has to be converted to the machine code.

Consider the following:

```
LDA#      A9     10101001
```

The first column is the code or mnemonics; the next two columns are the conversion to the machine code via hexadecimal and then to binary. Every instruction in the micros instruction set has its hexadecimal or binary equivalent. With the EMMA systems, the students actually converted the mnemonics code to the machine code, but this is very time-consuming.

The act of converting the mnemonics to machine code is called "compiling," and with the EMMAs, we get the students to compile the mnemonics. In real programming, we use a program called a compiler to do this.

Assembler Language

Different micros use different mnemonics to represent the instructions in their instruction set. All these different mnemonics are now collectively termed assembler language. There are different ones for different systems such as TASAM for TINA with the EMMAs, MASAM for Microsoft used in DOS programs, and MPLAB assembler from Microchip.

When using assembler language, all instructions have two parts:

- The OPCODE. This is the part that describes the operation (i.e., LDA Load The Accumulator).
- The OPERAND. Where the micro gets the data to be used in the operation (i.e., '#').

This means that the data is what follows immediately next in the micros memory.

As this book is based on the C programming language, there is no real need for the reader to understand the assembly language, but it is important to realize that all program languages, even visual basic, have to be converted to the machine code before being loaded into the micro. This process is called compiling, and it usually involves converting the program instructions into assembler before going into machine code.

C Programming Language

C and now C++ are generic programming languages that many programmers now study. As this has meant that there are a lot of engineers who can program in this language, then Microchip, the manufactures of PICs, have produced 'C' compilers that can be used to convert a 'C' program into the machine code for their PICs. Indeed, as the number of programmers who write in assembler have reduced and the number of 'C' programmers have increased, Microchip has stopped writing assembler compilers for their more advanced PICs such as the 32-bit PICs. Also, I believe that Siemens is now moving toward programming their PLCs in 'C.'

The more modern languages such as Python and C# have their roots in 'C.'

Different Programming Languages

Table 1-1 shows some of the more common programming languages.

Table 1-1. *Some Common Programming Languages*

Example Language
Machine code (binary 1s and 0s)
Assembly Language
Cobol
Fortran
C, Pascal
Ada 83
C++,
C#
Python
Basic
Visual Basic

The IDE

The term IDE stands for integrated development environment. It is actually a collection of the different programs needed to write program instructions in our chosen language. Then convert them to the actual machine code that the micro understands, and also link together any bits of program we may want to use.

The programs we need in the IDE are

- A text editor to write the instructions for the program. Note: The simple text editor "Notepad" could be used, but the text editor in MPLABX is by far a more advanced text editor.
- A compiler to change the instructions into a format the micro can understand.
- A linker to combine any files the programmer wants to use.
- A driver that will allow the programming tool used to load the program into the micro.
- A variety of simulation tools to allow the programmer to test aspects of the program.
- A variety of debug tools to allow the programmer to test the program live within the micro.

All these are in the IDE we choose; Microsoft has Visual Studio, Microchip has MPLABX, and Freescale uses CodeWarrior. Note that CODEBLOCK is an IDE for writing generic 'C' programs that will run on your PC. As this book is based on the PIC micro, it will concentrate on MPLABX. MPLABX has an improved text editor to give the text different color codes when we save the file as a .asm or .c for c program file such as light blue for keywords, light gray for comments, and so on.

There are some other organization programs within MPLABX such as the ability to write to the configuration registers for the PIC. There is also the ability to simulate your programs within the IDE. All this makes MPLABX a useful tool for programming PICs.

There is also a program called MCC Microchip Code Configurator. This will actually produce a lot of the code you need, to use various aspects of the PIC, for you. However, I firmly believe that you should produce the code you use yourself so that you fully understand the code you use. I will not cover the use of the MCC. Also, Microchip has not written the MCC for all their PICs, and the 18F4525 is one they have missed so far.

Really when asked who the programmer is, you should be able to say that you are and not the MCC. When you take the time to study how to write your own code, you will find it is not as hard as you first thought. Also, you will get a better self-reward if you write it all yourself.

The only aspect of the programs that I let Microchip do for me is to write the code configuration bits that set up the PIC. This is only because it is so simple to do this and it covers all the #pragma statements.

Summary

This chapter has given you some background information about microcontrollers. It has introduced some of the terms and given you an explanation of what they mean such as

- PIC
- IDE

The next chapter will take you through creating a project in MPLABX the IDE from Microchip. It will also allow you to produce your first PIC program.

CHAPTER 2

Our First Program

After reading this chapter, you should be able to create a project and write a program that uses inputs from switches and turns on outputs. We are going to start off by writing a program that will make the PIC wait until a switch connected to bit 0 of PORTA goes high. It will then light an LED on bit 0 of PORTB. The PIC will then wait until a second switch, connected this time to bit 1 of PORTA, goes high. When this happens, the LED on bit 0 of PORTB will be turned off. Note that both switches will be single momentary switches, that is, they will stay high only when they are pressed; when they are released, their logic will go low.

The PORTS of the PIC

Before I go any further, I think I should explain that the PORTS are the actual physical connections that the PIC uses to connect to the outside world. Note that the micros have used the analogy of the real ports, such as the Port of London or the Port of Liverpool, which actually connect the country to the outside world taking goods in for the country and sending goods out of the country.

These PORTS connect internally to registers inside the PIC. The registers are merely a collection of individual cells which we call bits. In the 18f4525 there are 8 cells or bits connected together to form a register. This is because the 18f4525 is an 8-bit micro. These bits are numbered from right to left as bit 0, bit 1, bit 2, bit 3, bit 4, bit 5, bit 6, and bit 7. This is shown in Figure 2-1.

H. H. Ward, *C Programming for the PIC Microcontroller*,
https://doi.org/10.1007/978-1-4842-5525-4_2

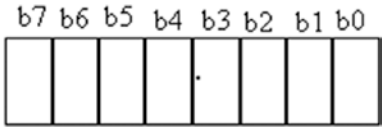


Figure 2-1. *An 8-Bit Register*

The bit 0 is sometimes referred to as the LSB or least significant bit, as this represents the units column or the ones column; whereas the bit7 is the MSB, most significant bit, as this represents the 128 column. Note that a 32-bit micro will have 32 bits in their registers and PORTS.

Good Programming Practice

All programs should be planned. The programmer should not just start writing code in the IDE. A good programmer should write an algorithm then construct a flowchart then write the program listing.

The Algorithm

This is really simply putting your thoughts, of how you are going to get the PIC to do what is asked of it, down on paper. The purpose is to focus your mind on how to complete the task. It will also allow you to choose the right PIC for the job. The algorithm should cover at least the following:

- You should explain the sequence of events you want to control.
- You should then identify all the input and output devices you will need.
- You should then create an allocation list for the control and identify any special inputs or outputs or controls you will need, such as analogue inputs, PWM outputs, and any timers.

The Flowchart

This is a diagram using standard symbols to show how the program will flow through the instructions and so complete the task.

Flowcharts are diagrams that show how any process flow through its constituent parts. They are very useful diagrams for designing computer programs. All flowcharts use five basic symbols; there are more, but the five most common symbols are shown in Figure 2-2.

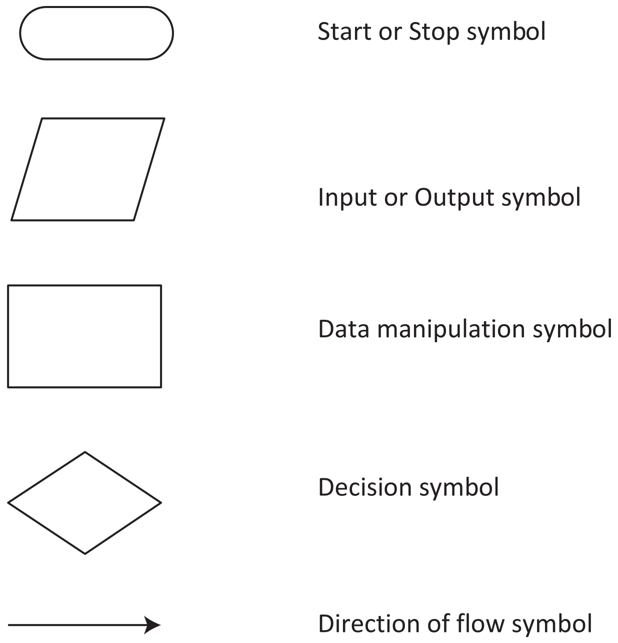


Figure 2-2. *The Main Flowchart Symbols*

The Program Listing

This is a list of the actual instructions written in your chosen language. If you have constructed your flowchart correctly, then each block in your flowchart will produce the correct lines of coding in your program listing.

Using MPLABX IDE

Before we go too far into the depths of MPLABX, I will discuss the use of MCC and MPLAB Harmony. Microchip has realized that there are many aspects of writing programs for the PIC that have to be carried out within every program. Therefore, they give you the facility to use their code-generating programs to write the code for you. MCC, MPLABX Code Configurator, is the program that does this for you. MPLAB Harmony does this for the 32-bit micros. Wow, isn't that great? Well yes and no. Using MCC creates a myriad of files and functions that are not easy to understand. If you write all the code for your program yourself, then you know where all the bits are and you understand how they work. Also this book teaches you how to use the datasheet to help write the instructions. You will learn how the PIC actually works and how it uses the simple logic '1's and '0's to control how it works. I firmly believe it is important for you, as the programmer, to understand what you are controlling and how your program instructions actually control it. If you use MCC straight off, then you risk losing this understanding and who the programmer is, you or Microchip. If you write all your own code, then you are the programmer.

MPLABX is the new IDE from Microchip. It is written in Java, and it has many improvements from the previous MPLAB. The book is written around using MPLABX version 5.2.

The text is based around using the PIC18f4525, but it can easily be adapted for any PIC micro. The 18F4525 PIC is a very versatile PIC in that it has

- 36 I/O
- 13 ADC channels
- 2 CCP modules as well as a UART and SPI

It has 48 kbytes of program memory as well as internal EEPROM.