

Introduction

Every year at Halloween, there is a special “The Simpson’s” episode in which spoofs are made of different horror and science fiction movies and stories. One of my favourite one involves Mr. Burns, the evil nuclear plant operator, building a robot to replace his employees. The only thing missing from his design is a controller. The quick solution to this problem is to use Homer Simpson’s brain, which has its own quirks and priorities. Despairing, Mr. Burns abandons the project and pulls Homer’s brain from the robot.

While this is a seemingly trivial introduction to the topic of programming robot controllers, this spoof actually illustrates what I see happening all the time with robot developers. I have found that most people trying to create their own robots do not put as much thought, consideration and planning into their robot’s controller as they do for its mechanical structure, sensors and drive train. As part of my research for this book, I discovered many beautifully designed and built robots that languished on the shelf because their developers had not planned properly for the controller or did not design the application in such a way that the controller could be modified and enhanced.

The goal of this book (along with the other books in the “Robot DNA” series) is to provide a robot designer with the knowledge and tools that will help guarantee that their robot will perform to expectation and specification as well as be easily modified. I consider the controller to be at least as important as the other major subsystems that make up the robot and they should be specified and designed in to the robot as it begins to take shape in the designer’s mind, not something that will be “gotten around to”.

In this book, I will introduce you to how robot controllers are integrated into mobile robots. An important part of this is showing how different sensors, outputs, controllers and peripherals can be wired to a microcontroller as well as interfaced to. Along with introducing you to the different subsystem functions that the controller will have to perform, I will go into quite a bit of detail about the programming techniques for integrating the different functions together as well as creating a “high level” control program for the robot.

While there are a number of robots that are controlled by PC motherboards, PC laptops, specially designed controller boards, neural networks and simple light sensors, I will focus on using microcontrollers for robot control. This does not mean that my examples cannot be scaled up or down to different controller methodologies and hardware, just that I would like to take advantage of the amazing power and ease of use that is available from these devices. Along with this capability, modern CMOS devices are remarkably robust and able to survive an astonishing amount of abuse. You will find that virtually all of the example circuits presented in this book can be built in five minutes or less.

Along with introducing you to high level control strategies for robots, I will also be working through quite a few common and different input/output (“I/O”) methodologies that you can use as part of your robot. I will be describing the basic theory behind the different devices as well as sample projects to help you understand how the different interfaces are actually implemented in a microcontroller. I have broken up the different I/O devices into three categories (output devices, input devices/sensors and motor control) and I have designed the controlling code in such a way that different devices can be integrated together with a minimal opportunity for problems.

I will be focussing on the Microchip PICmicro PIC16F627, an eighteen pin device, providing 1K instruction space and 68 bytes of variable memory. The chip has up to 16 I/O pins available (some of which can act as voltage comparators) along with two eight bit timers, a sixteen bit timer, serial communications and simple single vector interrupt capability. The PIC16F627 is “Flash” based which means that it can be erased and reprogrammed without an ultra-violet light

Introduction

source and can be programmed using very simple hardware. Included on the CD-ROM is a simple programmer circuit that I have developed which is quite fast and takes the output of the software development tools directly. Despite the apparently modest capabilities of the microcontroller, it is extremely popular with an amazing amount of resources available for it free of charge on the Internet as well as there being quite a few books written about it. In an informal survey of user developed robots, the PIC16F84 and the obsolete PIC16C84, which are precursors to the PIC16F627, were clearly the most popular microcontrollers used in home built robots.

The sample interfaces and robot control application code presented in this book was designed for easy "porting" to other microcontrollers with a minimum of changes. There are literally hundreds (and maybe <I>thousands</I>) of different microcontrollers that can be used to control your robot. The "C" code used in this book is designed to avoid using assembly language or hardware features that are specific to one microcontroller and instead use capabilities that are common to many different chips. While the concept of real time operating systems ("RTOS") are introduced in this book, the example code presented is designed for a simple, single interrupt driven environment. The PIC16F627 cannot implement a "true" RTOS (which is also true for many other microcontrollers) but there are other programming techniques that provide similar ease of application development. My goal for this book is to give you the skills/knowledge/sample software to allow you to "mix 'n match" interfaces in your robot and allow the applications to work without significant problems.

Making the PIC16F627 even more attractive as a demonstration/experimental tool is the availability of Microchip's outstanding "MPLAB" Integrated Development Environment (IDE) as well as HI-TECH Software's "PICC Lite" C programming language compiler which integrates with the MPLAB IDE. These two tools, which are provided on the CD-ROM that comes with this book, gives you a development capability that would cost a thousand dollars or more if a different microcontroller was used. I'm sure you will find that the two tools will make robot software development a lot easier than what is available with other tools that are not as well integrated.

In describing how software is written for robots, I work at compartmentalizing the work into one of three different spectrums that I call "biologic", "mechalogic" and "elelogic", with each term representing what the software is doing. The different spectrums, along with the functions they provide are shown in {Reference "**Robot Spectrums**" Graphic}.

{Put in "**Robot Spectrums**" Graphic}

"Biologic" programming is the high-level decision making being done by the robot's controller. Commands from this code spectrum are not required to respond exceptionally quickly – responses to multiple inputs that take 50 msecs (1/20th of a second) or more are not an issue. This code provides the high-level "artificial intelligence" demonstrated by the robot. It is often the most challenging code and requires the most computing resources.

"Biologic" programming can be compared to the functions provided by the central nervous system in the human body and assumes that the actual hardware interfaces are provided by other functions. In the human body, the peripheral nervous system that controls muscles, process inputs and reflexes can be compared to the "mechalogic" and "elelogic" code spectrums.

The "mechalogic" programming spectrum, executes in the range of 100 usecs to 100 msecs and is primarily responsible for controlling the mechanical devices built into the robot. The primary devices controlled by mechalogic programming including motors (both DC and servo) along with their speed controls and collision sensors ("whiskers" connected to micro switches). Mechalogic programming is generally the simplest of the three spectrums and, depending on the controller selected can use built in interfaces to simplify the circuit design.

Introduction

In this book, I will tend to represent “mechalogic” code as being interrupt driven. The reason for this is that while the PIC16F627 has some advanced hardware features that simplifies the task of controlling motors or providing other hardware functions, they may not all be available to a robot developer for a specific application.

The last programming spectrum, “elelogic” provides inter-computer communications as well as some interface and output functions. “Elelogic” code runs at the full speed of the processor to minimize data transfer times and care must be taken to ensure that the elelogic code does not “starve” the other spectrum’s code of processor cycles. Along with this, elelogic code should be designed to be tolerant of interrupts required for the “mechalogic” functions. Elelogic code is generally not very difficult to develop, but care must be taken to make sure it does not interfere with the other processes executing in the robot’s controller.

The three spectrums (“biologic”, “mechalogic” and “elelogic”) are terms that I have come up with and I have never seen it in any robot literature. Breaking up the software into these three groups is my way of rationalizing how robot controller applications are designed and helps me visualize different software requirements within the robot and how they interact. Note that I consider that the “biologic” code is the high level operation of the robot and treats the “mechalogic” and “elelogic” code as functions and subroutines that can be called without regard to how the tasks are actually carried out. Breaking the software into the three parts allows for code to be reused simply between robots and allows for quite simple experimentation with the biologic code.

As part of this book, I will go through much of the process of developing a complete robot application. This information will be from the perspective of the robot’s controller, but as I indicated above, I believe that all the major parts of the robot should be specified and designed together. This means that the techniques I discuss in this book can be applied to the information provided in other books in the “Robot DNA” series.

I also discuss how to find and resolve the problems that will inevitably come up as you work with your robot. I believe very strongly in testing your application code in a simulator before trying it out in actual hardware. The MPLAB IDE simulator will allow you to observe exactly how the application is executing and if there are any errors in programming or the strategy that was used to implement the robot functions.

For information on building robot structures and learning more about the electrical characteristics of motors, I recommend “Constructing Robot Bases” by Gordon McComb and “Building Robot Drive Trains” by Dennis Clark and Michael Owings, two other parts in the “Robot DNA” series. The three books have been written to complement each other and by working with them, you will have all the material you need to create a mobile robot with a minimum of problems.

I wouldn’t be surprised if you were wary that a simple microcontroller like the PIC16F627, which has less memory and processing power as the first “Apple” computer, can effectively control many different peripheral functions in a robot as well as its high level control. Before going on, I just want to assure you that a simple microcontroller like this one can control a robot more effectively than Homer Simpson’s brain.

Prerequisites for this book

Like the other books in this series, “Programming Robot Controllers” was written for robot developers with some experience in the skills needed to develop robots. I will not be going into detail explaining basic programming, electronics or PC operation, but you will have to be familiar with these areas of study. If you can write and execute a simple program on your PC (as simple as “Hello World!”) and wire a circuit consisting of some digital electronic chips, you should not have any difficulty with the concepts and information presented in this book.

Introduction

All the software presented in this book will be written in the “C” high level language. I have worked at making the code as device independent as possible – you should be able to take the code presented in this book and port easily to another compiler (or language) and microcontroller. The basic requirement for a microcontroller to be able to execute the applications presented here is just a simple interrupt (based on timers as well as changes in I/O pin input) capability. The timer interrupt function of the microcontroller is used to sequence the different electronic and mechanical interfaces and allow the biologic code to execute as if it were the only application running in the microcontroller (and the interfaces are all hardware, rather than software based).

If you aren't familiar with C, you should get a copy of “The C Programming Language” by Brian Kernighan and Dennis Ritchie – this book is considered the “C bible” and explains the operation of the language, which is implemented faithfully in the PICC Lite compiler. There are a variety of C compilers available for other microcontrollers and there is a version of the “GCC” (GNU, open source/open license) C compiler available for the PC that you can download for free to better understand how C programs are written.

The C programming language has a reputation for being difficult to learn and work with – I have seen it derisively called the “universal assembly language” because of its strength in manipulating low-level data. While “C” is not an ideal language for writing data base queries or Internet web page applications, it is ideally suited for providing control of hardware interfaces. C is a relatively simple language that is very strongly structured and typed which makes it very attractive for electronics programming in general and robot programming in particular.

There is one aspect of C that can be abused, making it very difficult for people to read and understand and other people's code. C has the ability of combining statements, to lessen the amount of data entry required for an application. For example, the three statements:

```
A = A + 1;  
B = A * 4;  
if (B == 16) {
```

could be written as:

```
if ((B = (++A * 4)) == 16) {
```

The second statement is identical to the first three (“A” is incremented, “B” is assigned the value of “A” (after “A” was incremented) times four and conditional code is executed if “B” is equal to 16). In this book, I will refrain from aggressively combining statements as this hurts the readability of the code and makes it difficult to port the code to another language like “BASIC”.

Another concern with C is its use of pointers. Right now, there is a trend away programming languages that implement pointers – this is one of the “selling points” of Java. While I believe that it is important to remember that C strings are implemented as a pointer to an array of characters, there are few requirements for pointers when creating robot applications.

To build the sample applications presented in this book, you will have to have some basic understanding of electronics. The reasons for saying this is because some of the parts I have used in developing the applications came from junk boxes, while some others came from surplus stores and others came from electronic distributors only available in North America. I doubt you will be able to get exactly all of the same parts that I have used. This should not be an issue because I have explained the theory and use of these devices as well as any issues that you should be aware of when using them in robot applications.

If you have an understanding of the basic DC voltage/current laws (Ohm's law, Kirchoff's law, Thevinin's equivalence), along with some transistor theory and an understanding of how to work

Introduction

with digital electronic devices, then you will not have any problems with the circuits presented in this book.

Of greater importance than understanding basic electronic theory is the ability to build prototype electronic circuits. For many of the example circuits presented in this book, I have shown how they can be wired on a simple "breadboard". For an actual robot, I recommend that soldered or wire-wrapped circuitry is used on your robot for reliability.

You will have to be comfortable with loading applications and developing code on your PC. I highly recommend that the files for different interfaces and applications be kept in separate folders (or subdirectories) on your PC's hard disk. Before starting to work through the applications presented in this book, you should be comfortable with navigating and using the different resources available within the PC including the ability to work with a typical Windows based editor like "WordPad".

Lastly, you should have access to the Internet. Not only will it be useful for accessing my web page (for updates and errata) but also to be able to look up information on different components and circuits. There are quite a few different web sites, list servers, bulletin boards, FAQs, etc. devoted to robots and this additional information will help you in deciding how you should design your robot.

Conventions used in this book

{Editor: Text in chevrons ("Greater Than" and "Less Than" Characters) describes a special character that should be used. Please contact the author if you have any questions.}

{Editor: Please convert the text immediate following "*" to a superscript (ie it is a mathematical "power" to the value to the left of "***") and delete the "***"}**

<Ohm Symbol>	- Ohms
k	- k<Ohm Symbol> (Thousands of ohms)
M<Ohm Symbol>	- Millions of Ohms
<muSymbol>F	- microFarads (1/1,000,000 Farads)
pF	- picoFarads (1/1,000,000,000,000 Farads)
secs	- seconds
msecs	- milliseconds (1/1,000 of a second)
<muSymbol>secs	- microseconds (1/1,000,000 of a second)

{Editor: Please convert all instances of "usecs" to <muSymbol>secs in the text}

nsecs	- nanoseconds (1/1,000,000,000 of a second)
Hz	- Hertz (Number of cycles per second)
kHz	- kiloHertz (1,000 cycles per second)
MHz	- MegaHertz (1,000,000 cycles per second)
GHz	- GigaHertz (1,000,000,000 cycles per second)
####	- Decimal number
-####	- Negative decimal number
0x0####	- Hexadecimal number
0b0####	- Binary number
n.nn x 10**e	- Real number. The number n.nn times ten to the power "e"

Introduction

- [] - Optional parameters within bold square brackets
- | - Either or parameters
- _ - Underscore indicating that statement is continued to next line
- _Label - Negatively active signal/bit
- Register.Bit - Specific bit in a register
- Monospace Font - Example code
- // - Text/Code/Information that follows is commented out
- : or ... - "And So On". This is put in to avoid having to put in meaningless (and confusing to the discussion) text

& - Two Input Bitwise AND

- Truth Table:

Inputs		Output
A	B	
0	0	0
0	1	0
1	0	0
1	1	1

AND, && - Logical AND

| - Two Input Bitwise OR

- Truth Table:

Inputs		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	1

OR, || - Logical OR

^ - Two Input Bitwise XOR

- Truth Table:

Inputs		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

XOR - Logical XOR

! - Single Input Bitwise Inversion

- Truth Table:

Input	Output
-------	--------

Introduction

A		
-----	+	-----
0		1
1		0

NOT	- Logical Inversion
+	- Addition
-	- Subtraction or Negation of a Decimal
*	- Multiplication
/	- Division
%	- Modulus of two Numbers. The modulus is the "remainder" of integer division
<< #	- Shift Value to the Left "#" Times
>> #	- Shift Value to the Right "#" Times
->	- Mouse click the specified pull down/button from the previous pulldown/control

The CD-ROM that comes with this book

Once you've you are ready to start going through it and build the projects I have presented in here, I highly recommend that you put the CD-ROM that comes with this book into your PC and take a look at what's available to you on it. The CD-ROM contains a lot of information that you will help you to better understand the data as well as quickly build the sample circuits presented in this book.

The CD-ROM contains the following information and materials:

- All software source files
- Development tools, including copies of HI-TECH Software's "PICC Lite" "C" compiler for the Microchip PIC16F627 microcontroller, Microchip's "MPLAB" Integrated Development environment and my "El Cheapo" PICmicro MCU programmer circuitry and software.
- PDF datasheets for the PICmicro MCU used in the book as well as the manuals for the PICC Lite compiler and the MPLAB IDE.
- Web resources including page references to different information

The CD-ROM also includes the necessary instructions and tools for loading the files from the CD-ROM as well as instructions for installing the tools included on the CD-ROM.

Registered Marks

Microchip is the owner of the following trademarks: "PIC", "PICmicro", "ICSP", "KEELOQ", "MPLAB", "PICSTART", "PRO MATE" and "PICMASTER". microEngineering Labs, Inc. is the owner of "PicBasic". "PICC" and "PICC-Lite" are trademarks owned by Microchip but licensed exclusively to HI-TECH Software. "HI-TECH C" is a trademark of HI-TECH Software. Microsoft is the owner of "Windows/95", "Windows/98", "Windows/NT", "Windows/2000" and "Visual Basic". All other copyrights and trademarks not listed are the property of their respective manufacturers and owners.