IDC Technologies - Books - 1031 Wellington Street
West Perth
WA 6005

Phone: +61 8 9321 1702 - Email: books@idc-online.com,publishing@eit.edu.au

EB-E - Practical Embedded Controllers Troubleshooting and Design

Availability: In Stock

Price: $65.95

Ex Tax: $59.95

Short Description

From microwave ovens to alarm systems to industrial PLC and DCS control systems, embedded controllers are controlling our world. The microcontrollers that are at the heart of these and many more devices are becoming easier and simpler to use. But when these devices fail the solution to the problem needs to be found and the repairs have to be done quickly. This manual will help the technician, engineer and even the most casual user understand the inter-workings of microcontrollers along with the most common problems and their solutions.

Description

From microwave ovens to alarm systems to industrial PLC and DCS control systems, embedded controllers are controlling our world. The microcontrollers that are at the heart of these and many more devices are becoming easier and simpler to use. But when these devices fail the solution to the problem needs to be found and the repairs have to be done quickly. This manual will help the technician, engineer and even the most casual user understand the inter-workings of microcontrollers along with the most common problems and their solutions.

Embedded controllers are used in most electronic equipment today. Embedded controllers are intelligent electronic devices used to control and monitor devices

connected to the real world. This can be a Programmable Logic Controller (PLC), Distributed Control System (DCS) or a Smart Sensor. These devices are used in almost every walk of life today. Most automobiles, factories and even kitchen appliances have embedded controllers in them. This manual covers all aspects of embedded controllers but focuses specifically on troubleshooting and design. The manual covers design, specification, programming, installation, configuration and of course troubleshooting.

Table of Contents
Download Chapter List

Table of Contents

First Chapter

1. **Introduction to embedded controllers**

1 Introduction

Objectives

When you have completed this chapter, you will be able to:

- Describe the basic parts and functions of microcontrollers
- Explain what assembly language is and how it is used
- Describe memory mapping
- Describe the basics of inputs and outputs
- Describe what types of data communications controllers use
- Explain noise reduction and its relationship to good signals
- Describe potential grounding problems

1.1 Microcontroller introduction

Embedded controllers are used in most commercial and industrial electronic equipment. The sheer volume of embedded controllers used in the world drives us to understand how they work and then how to troubleshoot and repair them. The microcontrollers and support chips used in these controllers are becoming smarter and easier to use. This is bringing the design and use of embedded controllers to more and more engineers hence the need for a good understanding of what embedded controllers are and how to troubleshoot them.

Embedded controllers are intelligent electronic devices used to control and monitor devices connected to the real world. This can be a microwave oven, programmable logic controller (PLC), distributed control system (DCS) or a smart sensor. These devices are used in almost every walk of life today. Most automobiles, factories and even kitchen appliances have embedded controllers in them. As time goes on and electronic devices get smarter and smaller, the embedded controller will be in or associated with everything we touch throughout the day.

Early embedded controllers contained a CPU (central processing unit) and a multitude of support chips. As time went on, support chips were included in the CPU chip until it became a microcontroller. A microcontroller is defined as a CPU plus random access memory (RAM), electrically erasable programmable read only memory (EEPROM), inputs/outputs (I/O) and communications (Comms). The embedded controller is a microcontroller with peripherals such as keypads; displays and relays connected to it and is often connected to other embedded controllers by way of some type of communication system.

Figure 1.1
Embedded controller development board

Figure 1.2
Keypad for embedded controller

Electronic equipment is becoming more and more susceptible to noise and other outside influences that can cause catastrophic problems. To be able to troubleshoot and ultimately repair the embedded controller it is not only necessary to understand the inter-workings of the embedded controller but also the external forces that can affect the normal operation of the controller. This may be noise, bad connections or incorrect installation of the system. Often simple things like bad grounds or incorrectly made connections can cost the user hundreds, if not thousands of dollars in down-time. Although the embedded

controller ultimately can be a complicated device, when disassembled into its basic parts it becomes simple, clear and easy to understand.

1.2 Microcontroller design and functions

The microcontroller is a direct descendent of the CPU, in fact every microcontroller has a CPU as the heart of the device. It is therefore important to understand the CPU in order to ultimately understand the microcontroller and embedded controller.

The central processor unit (CPU) is the brain of the microcontroller. The CPU controls all functions and uses the program that resides in RAM, EEPROM or EPROM to function. The program may reside in one or more of these devices at the same time. Part of the program might be in RAM while another might be in EEPROM.

Figure 1.3
68HC11 CPU

A program is a sequence of instructions that tell the CPU what to do. These instructions could be compared to instructions a teacher may give to a student to get a desired result. The instructions sent to the CPU are very, very simple and it usually takes many instructions to get the CPU to do what is necessary to accomplish a task. Upper level programming languages like BASIC and C++ include multiple instructions in one command to speed up the process of programming the CPU. Just like the human brain the CPU is made up of regions that have specific functions. These components are controlled by the program instructions.

The main components of the microcontroller are as follows:

- CPU
- External address bus
- External data bus
- External control bus
- Internal RAM
- Internal ROM
- Internal ERPROM
- Internal EEPROM
- Internal registers

- Digital inputs
- Counter inputs
- Digital outputs
- Analog inputs
- Serial data communications
- Parallel ports

This may seem like a large number of components, but grasping the complete microcontroller system becomes very easy once each of the individual components is understood.

In a microcontroller, the CPU uses an internal parallel address and data bus to communicate with memory components like RAM, EEPROM and ROM. It also uses this internal bus to talk to communication systems, I/O ports and registers. The internal microcontroller memory components such as RAM, ROM, EPROM and EEPROMs are used to store (either temporary or permanently) data and program instructions. The internal registers are used to manage temporary bytes of data, like addressing for the program. The serial communications section lets the microcontroller communicate with other devices via a communication standard such as RS-232 or RS-485. The parallel ports such as A, B, C, D and E can be used to transfer data to and from external memory chips or devices. These ports can be used to read and write to devices like keyboards and LCDs. An external parallel data bus can also be used by the microcontroller to activate or read external devices like switches, relays, and LEDs. The digital I/O and analog inputs are used to bring inputs and outputs to and from the microcontroller.

1.3 Assembly language programming

Often when assembly language programming is mentioned programmers groan that it is all too hard and difficult. Assembly programming is actually easy and simple (almost too easy). The two best things about assembly language programming is the control it gives the programmer over the microcontroller and the minimal instructions needed to do the job. Using BASIC or C++ is compared by some to using a chain saw to peel an egg. From a functional point of view, using BASIC, C++ or some other high-level language is simple and straightforward but it does use a huge amount of memory compared to assembly language. This limits the size of the program that the programmer can load into the microcontroller. Chip manufactures have gone to great lengths to include RAM, ROM and EEPROM on board the microcontroller. This memory is usually

only hundreds of bytes. Programming the microcontroller without using external memory chips is almost impossible using BASIC or some other high level languages. Therefore, assembly language becomes the only option.

Programming is often compared to painting a picture. One difference though is that in art it is often unclear when the painting is finished. In programming the program is done when it does what it was designed to do. This can be defined and specified before the program is written. Strangely enough, this step of exacting specification is often overlooked and the program is just let to evolve. As in most endeavors, preparation is everything. The participants in the programming process should spend a large amount of time preparing for the writing of the program.

In its simplest form, the program is a sequential set or list of instructions that tell the microcontroller what to do. Each step in the process is done in a specific order. The process is divided up into separate individual sections called subroutines. A subroutine is a small program that performs some tiny function within the overall program. An example of this could be starting a car. The sequence of events that are used to start a car could be called a subroutine within the overall program of driving the car. It is a very specific and defined sequence of acts or instructions. It is stand-alone and can be repeated when necessary. In programming language, it would go something like...

Figure 1.5
Starting the car

Jump to ‘Start the Car’

Start the Car Put key in ignition

Started Turn key clockwise to the start position

Has the car started?

If the car has started, release the key and go to ‘End’

If not, continue to hold the key in the start position

Loop to ‘Started’

End Return to main program (i.e. drive the car)

Figure 1.6
Flow chart to start the car

Of course, this program is simplistic because we have not put in all the possibilities. Such as; if the car did not start the driver would run the battery down by continually holding the key in the start position. Also what are the parameters that define that the car has started? A main program is made up of many of these subroutines. This method of programming is simple and easy to troubleshoot by the programmer. Also notice the flow chart in Figure 1.6. This is an easy way of designing the program before writing any code. This helps the programmer see the program in an overall form and therefore see mistakes before they happen. One thing that is not shown in the above example is where in the memory map of the microcontroller is the ‘Start the Car’ program located.

A memory map is a list of the address locations where the program, ports and various other devices reside in the microcontroller system.

The memory map can be separated into three parts:

- Address locations of RAM, ROM, EPROM and EEPROM
- Address locations of ‘vectored’ jump locations
- Address location of input, output and communications locations

Note: A vector is the location of the beginning of a subroutine or function of the program. A vector could be a memory location, where a jump is located, that branches to a keypad subroutine, (more about this later).

The programmer uses the memory map in the same way a road map would be used by a driver to find his/her way to the destination. The road map might indicate that the location of a town is at A/3. The driver (assuming that the driver wants to go to the town) would look on the map and find A/3. The driver would then take the road that goes to that town. The memory map of a microcontroller might say that the external RAM is located at $C000. This address is a hexadecimal address that the programmer puts in the start of the program. Once the program is loaded into RAM memory location at $C000, a subroutine could jump or ‘vector’ to this location at any time and the program would start there.

1.4 Inputs and outputs

Digital inputs and outputs on the microcontroller are located within the ports A, B, C, D, or E. Some of these ports are defined as fixed inputs or outputs while others are bi-directional. Ports that can be setup within the program as either inputs or outputs are called bi-directional I/O. The ports have registers that the programmer uses to set up the bi-directional port. A single bit changed from a 0 to a 1 in a particular register can determine whether a line on a port in an input or an output. The programmer stores a hex number in the register to set the I/O line in the port to be an input or output. This type of port is called a definable port.

Figure 1.7
Typical inputs and outputs

The definable I/O is accessed by setting up a register located at unique addresses in the memory map. Registers are usually 8 bit devices where each bit has a special function. A typical example would be the register at $1009. This is the data direction register of port D on a HC11 microcontroller. If the programmer was to store #$10 or 00010000 in binary to this register, bit 4 of port D would be defined as an output. If the programmer sent #$00 or 00000000 to $1009 then port D bit 4 would be an input. The programmer could then store a hex value in port D and depending on the value stored the line would be on or off. Remember in digital electronics a one or zero can be either ‘ON’ or ‘OFF’ depending on the way it has been designed. (In fact, in most systems a zero is ‘ON’.)

Analog inputs are sometimes included on the microcontroller, but most of the time they are a function of external chips to the microcontroller. Even microcontrollers that have analog inputs on board usually have very few and therefore the designer must use external chips for more inputs. An analog input measures voltage and then stores in memory as a binary number. The rate at which the microcontroller reads or samples the voltage is called the sample rate. The amount of numbers that define the voltage is called the resolution. The binary number that represents the voltage is transferred to memory and ultimately to a database. This database is then displayed, printed or used by other devices for control.

1.5 Data communication

RS-232, 422, and 485 are slowly giving way to USB, Firewire and Ethernet. Because of the limitations of this book, the author has confined the discussion

here to the first set. In the near future USB, Firewire and Ethernet will probably be used extensively to communicate to microcontrollers, but as of this writing RS-232, 422, and 485 are still the most common methods of interconnecting embedded controllers.

Figure 1.8
RS-232 comm port on a computer

Serial asynchronous and synchronous communications are two of the most popular types of communication used in industry today. RS-232, RS-422 and RS-485 voltage standards are usually asynchronous communications systems. Because asynchronous is very simple and convenient, it is still very common in data communications. This will continue for the next few years or decades. Asynchronous does have its problems, such as slow speeds and large overheads, but often its ease of use overcomes these limitations. In industry the catch phrase is ‘if it works and it’s cheap then use it’. Asynchronous is used because every computer has an RS-232 port and the interface chips that connect to the microcontroller for RS-232 are cheap, easy to use and readily available.

Synchronous systems are becoming popular because of the need for higher data communication speeds. Synchronous data communications use clocking, start characters and error checking to maintain high-speed communications. Along with the lack of start bits, stop bits and other overheads, synchronous systems can transfer data thousands and even millions of times faster then asynchronous systems. The most common voltage standards using asynchronous communication systems are RS-422 and RS-485. The two fastest growing synchronous data communication systems in use today are the USB and Ethernet. One day they may take over from RS-232, RS-422, and RS-485.

Figure 1.9
USB connector

1.6 Noise reduction

Noise reduction in electronic circuits is fast becoming a high priority in printed circuit board and system design. There are two issues with respect to noise reduction in controller systems. One is preventing noise being transmitted from the device into the outside world, and the other is installing systems that are less

susceptible to noise from outside sources.

The simplest way to transmit noise is with fast changing current flowing through an exposed conductor. As electronics on the board become faster and faster the chances that the PCB will radiate EMI frequencies and noise levels will increase. The PCB can therefore be thought of as a radio transmitter of noise. The typical PCB has many different high-speed currents flowing through exposed conductors on the board. All the PCB needs is an antenna (input and output wires) and it becomes a noise transmitting device.

Figure 1.10
Noise reduction on a printed circuit board

PLCs, DCSs and other control systems are very susceptible to noise from external sources. The most common way noise gets inside a controller is through the wiring in the cable run. The wire connecting the controller to sensors, PCs and other equipment acts like an antenna to the noise created by other electrical and electronic equipment. The wire that connects to controllers can be thought of as both a transmitting and receiving antenna. It is important therefore to look at noise from the controllers' point of view as both a conveyor and recipient of noise.

We find that the reduction of noise can be as easy as either moving the offending transmitting wire away from the victim wire or moving the victim wire away from the broadcasting wire. In the past, noise reduction, troubleshooting and repair was done by using oscilloscopes and filters. Since the advent of the digital revolution the rules have changed and now we find that not only is equipment more susceptible to noise but traditional methods of troubleshooting and repair do not work. When repairing noisy circuits, filters should be kept at a minimum as they often can make the problem worse. This is because filters reduce the separation between our equipment and ground. Often noise is coupled to our equipment through the ground connection.

1.7 Grounding solutions

Grounding, with respect to noise reduction and proper operation of equipment can be divided into two areas; PCB track grounds and equipment ground. Grounding practices in some ways has changed a lot in the last twenty years and in other ways they have stayed the same. The greatest changes have been in the area of the new EMC requirements of electronic devices and especially in high-

speed digital equipment.

In PCB design there are four areas of noise reduction:

- Placement of components
- Track placement
- Ground planes
- 1D and 3D Faraday boxes

Figure 1.11
Faraday box on a PCB

Each of these areas has gone through substantial changes of late and will continue to evolve over the years as noise reduction requirements change. The need for increased noise reduction from a PCB/EMI radiation point of view is universally expected to increase in the future. Proper placement of components has become critical when it comes to chip to chip noise transfer on a PCB. Track placement, track spacing and track size becomes an issue on both internal and external EMI. Ground planes have become an important tool for the designer in the reduction of noise on PCBs.

On the other side, once the equipment has been designed and installed, it is necessary to do everything possible to protect it from noise and external high voltages such as lightning. Grounds were once seen as the best protection against noise in electrical systems, but since the introduction of highly sensitive digital electronics, grounds have become noise conduits into digital equipment. The problem is that on one hand ground can be a noise source, but it is highly necessary for lightning and static voltage protection. This conflict has caused a lot of controversy in the controller and electrical industry. Having said that, it is possible through proper installation to build systems that give a high level of noise and lightning protection.

1.8 Installation techniques

Installation of controllers, sensors and wire systems is an important part of the overall quality of a system. The best-designed system will fail if the installation is not done correctly. It has been proven that approximately 60% of failures in working equipment are due to bad connections. These failures can usually be

traced back to improper installation with only a small percentage of that 60% being part failure. Proper installation is a very subjective thing and although there are many standards; most installers rely on their experience and personal training. Unfortunately, as technology evolves, installers don't often have the opportunity to keep up with those changes.

Proper installation of connections and terminations is often an overlooked or undervalued skill in the reduction of failures in electronic systems. If screw connectors are under or over tightened, the connection will fail. Soldering can be used to increase the quality of a connection, but sometimes it will add to the possible failure of a connection. Using crimp connectors can be fast and good connections, but if installed incorrectly they can cause problems. The two most common causes of bad connections and terminations are not following the correct installation procedures or using the wrong crimp tool.

Figure 1.12
A good installation

Cable runs and conduit systems are used to hold the wires that connect the equipment. This at first doesn't sound too important, but often the type and placement of the cable runs can affect the noise quality of our system. The cables in a cable run can be thought of as antennae connected to the equipment. The cables connecting the equipment are the largest part of the system and this is where most noise is transferred from one system to another. If large voltage and current carrying cables are placed next to highly sensitive signal wires, problems will be inevitable. Conduits made of steel will have a different and better effect on the reduction of noise than, say, one made of PVC.

1.9 Conclusion

Although it is impossible to cover every detail associated with the subject of embedded control systems, it is hoped that this book will give the reader some hard hitting practical knowledge concerning the troubleshooting and design of embedded controllers. This chapter started with the make up of typical microcontrollers, then moved to functional methods of troubleshooting. Repair of microcontroller systems was discussed and then an introduction to real techniques in installation. The reader should come away with practical introduction to controller systems.

Although the reader may never design the hardware or software associated with an embedded controller, this book should give the reader an overview of the inter-workings of the microcontroller. This understanding can help in the specification, use or even the sale of controller equipment. To troubleshoot an embedded microcontroller system it is important to understand the inputs, outputs and the way the controller communicates. Noise reduction and proper installation are important subjects from the point of view of making a system work properly. As time goes on the microcontroller will become an increasingly important part of our lives. It is to this end that the author hopes the reader finds this book of some assistance.