



**Practical**

# **Embedded Controllers**

**Design and Troubleshooting with the Motorola 68HC11**



**John Park**



# **Practical Embedded Controllers: Design and Troubleshooting with the Motorola 68HC11**

## **Titles in the series**

*Practical Cleanrooms: Technologies and Facilities* (David Conway)

*Practical Data Acquisition for Instrumentation and Control Systems* (John Park, Steve Mackay)

*Practical Data Communications for Instrumentation and Control* (Steve Mackay, Edwin Wright, John Park)

*Practical Digital Signal Processing for Engineers and Technicians* (Edmund Lai)

*Practical Electrical Network Automation and Communication Systems* (Cobus Strauss)

*Practical Embedded Controllers* (John Park)

*Practical Fiber Optics* (David Bailey, Edwin Wright)

*Practical Industrial Data Networks: Design, Installation and Troubleshooting* (Steve Mackay, Edwin Wright, John Park, Deon Reynders)

*Practical Industrial Safety, Risk Assessment and Shutdown Systems for Instrumentation and Control* (Dave Macdonald)

*Practical Modern SCADA Protocols: DNP3, 60870.5 and Related Systems* (Gordon Clarke, Deon Reynders)

*Practical Radio Engineering and Telemetry for Industry* (David Bailey)

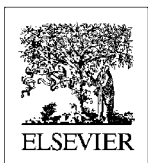
*Practical SCADA for Industry* (David Bailey, Edwin Wright)

*Practical TCP/IP and Ethernet Networking* (Deon Reynders, Edwin Wright)

*Practical Variable Speed Drives and Power Electronics* (Malcolm Barnes)

# Practical Embedded Controllers: Design and Troubleshooting with the Motorola 68HC11

**John Park** ASD, IDC Technologies, Perth, Australia



AMSTERDAM • BOSTON • HEIDELBERG • LONDON • NEW YORK • OXFORD  
PARIS • SAN DIEGO • SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Newnes is an imprint of Elsevier



Newnes  
An imprint of Elsevier  
Linacre House, Jordan Hill, Oxford OX2 8DP  
200 Wheeler Road, Burlington, MA 01803

First published 2003

Copyright © 2003, IDC Technologies. All rights reserved

No part of this publication may be reproduced in any material form (including photocopying or storing in any medium by electronic means and whether or not transiently or incidentally to some other use of this publication) without the written permission of the copyright holder except in accordance with the provisions of the Copyright, Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licensing Agency Ltd, 90 Tottenham Court Road, London, England W1T 4LP. Applications for the copyright holder's written permission to reproduce any part of this publication should be addressed to the publisher

### **British Library Cataloguing in Publication Data**

A catalogue record for this book is available from the British Library

ISBN 07506 58029

For information on all Newnes publications, visit our website at <a href="http://www.newnespress.com">www.newnespress.com</a>
---

Typeset and Edited by Vivek Mehra, Mumbai, India  
([vivekmehra@tatanova.com](mailto:vivekmehra@tatanova.com))

Printed and bound in Great Britain

Special thanks to

Industrial Automation  
[www.cs.jcu.edu.au/~gregory/hc11/](http://www.cs.jcu.edu.au/~gregory/hc11/)

All photos in this book courtesy of

Cursor Magic  
[www.cursormagic.com](http://www.cursormagic.com)  
[photos@cursormagic.com](mailto:photos@cursormagic.com)

# Preface

From microwave ovens to alarm systems to industrial programmable logic controllers (PLCs) and distributed control systems (DCSs), embedded controllers are running our world.

Embedded controllers are used in most items of electronic equipment today. They can be thought of as intelligent electronic devices used to control and monitor devices connected to the real world. This can be a PLC, 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.

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 repairs done quickly.

This book will help technicians, engineers and even the casual user understand the workings of microcontrollers, along with the most common problems and their solutions.

This book covers all aspects of embedded controllers but is biased towards troubleshooting and design. The book also covers design, specification, programming, installation, configuration and troubleshooting.

After reading this book we hope you will have learnt how to:

- Design, set up and program a complete embedded controller development system
- Apply the latest techniques in programming these versatile devices
- Apply troubleshooting tips and tricks for microcontrollers
- Apply the best techniques for installation of microcontrollers
- Fix problems due to electrical noise and interference
- Design correctly the first time to avoid grounding and EMC problems
- Choose and configure the correct software

Typical people who will find this book useful include:

- Electronic technicians and engineers
- Instrumentation and control engineers and technicians
- Process control engineers and technicians
- Electrical engineers
- Consulting engineers
- Process development engineers
- Design engineers
- Control systems sales engineers

A basic knowledge of electrical principles is useful in understanding the concepts outlined in the book, but the contents are of a fundamental nature and are easy to comprehend.

The structure of the book is as follows.

**Chapter 1: Introduction.** This chapter gives a brief overview of the main components of a microcontroller.

**Chapter 2: Microcontroller basics.** A review of the basics of this device with a discussion on number systems, Boolean logic, accumulators, registers, data communications, power systems, crystals and oscillators, is done in this chapter.

**Chapter 3: Microcontroller programming.** A review of the simple techniques involved in programming a microcontroller with a discussion on the various programming issues such as programming structures, addressing modes, operations and finally a short comparison of C++ and BASIC, is done in this chapter.

**Chapter 4: Microcontroller memory.** The main types and techniques in the effective use of memory such as user RAM, BUFFALO routines, interrupts, control registers, and EEPROM are assessed here.

**Chapter 5: Microcontroller inputs and outputs.** Analog and digital inputs, keypad and LCD interfacing are described here.

**Chapter 6: Data communications.** This important topic is broken down into a discussion on the fundamentals, the OSI model, modes of communication and RS-232 and RS-485.

**Chapter 7: Noise reduction.** This chapter gives an overview of noise reduction and a discussion on conductive, capacitive, and magnetically coupled noise.

**Chapter 8: EMC grounding solutions.** The most important features of grounding (and protection from lightning) to protect the microcontroller from the effects of EMC are discussed here.

**Chapter 9: Installation and troubleshooting.** This chapter is a short discussion on connections, cable runs and trays, wire management and troubleshooting techniques.

**Chapter 10: End notes.** A wrap discussion on the issues discussed in the earlier chapters with a few words on assembly language programming, memory, inputs and outputs, data communication, noise reduction and grounding solutions and finally installation techniques.

# Contents

Preface

xiii

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Microcontroller introduction	1
1.2	Microcontroller design and functions	3
1.3	Assembly language programming	5
1.4	Inputs and outputs	7
1.5	Data communication	8
1.6	Noise reduction	9
1.7	Grounding solutions	10
1.8	Installation techniques	11
1.9	Conclusion	12

---

<b>2</b>	<b>Microcontroller basics</b>	<b>13</b>
2.1	Introduction	13
2.2	Number systems – binary, hex, and decimal	17
2.2.1	Bits, dibits, nibbles, bytes, words and long words	17
2.2.2	Conversion from binary to hexadecimal	18
2.2.3	ASCII, hexadecimal and BCD	18
2.2.4	Converting from ASCII to BCD and HEX	19
2.3	Gates – AND, OR, XOR and NOT gates	20
2.3.1	AND gates – physical and virtual	20
2.3.2	OR gates – physical and virtual	21
2.3.3	XOR gates – physical and virtual	21
2.3.4	NOT gates – physical and virtual	22
2.4	Accumulators, A, B and D	22
2.4.1	Addressing modes	22
2.4.2	Loading, storing and transferring accumulators	23
2.4.3	Add, subtract, compare, increment and decrement A and B	24
2.4.4	AND, OR and test bits	24
2.4.5	Arithmetic and logical shifting and rotating	24
2.4.6	Data and bit testing	24
2.5	Registers – X, Y, the stack and ports	25
2.5.1	The X and Y registers	25
2.5.2	The stack	28
2.5.3	Ports A, B, C, D and E	27
2.6	Communications synchronous and asynchronous	28



2.6.1	Asynchronous character communications	28
2.6.2	Synchronous packet communications	29
2.6.3	Specifying a system – asynchronous vs synchronous	30
2.7	Power systems	30
2.7.1	Hardware vs software	31
2.7.2	Hardware reset design	32
2.7.3	COP watchdog (Woof)	33
2.7.4	Power failure and brownout protection	34
2.8	Crystals and oscillator	34
2.8.1	Values vs baud rate	35
2.8.2	EMC and PCB crystal clock design	35
2.9	Conclusion	37
<b>3</b>	<b>Microcontroller programming</b>	<b>39</b>
3.1	Introduction to programming the microcontroller	39
3.2	Programming structure and specifications	41
3.2.1	Programming structures	42
3.2.2	Inputs	42
3.2.3	Manipulation of data	43
3.2.4	Outputs	44
3.2.4	Flow charts	44
3.2.6	Loading a program into an evaluation module (EVM)	48
3.2.7	Setting up the EVM	49
3.3	Addressing modes	49
3.4	Load, stores and transfers	50
3.5	Arithmetic operations	51
3.6	Logical operations	52
3.7	Shifts and rotates	53
3.8	Index registers and the stack	54
3.9	Condition code register	57
3.10	Branches, jumps, interrupts and calls	57
3.11	BASIC and C++	58
3.11.1	BASIC	58
3.11.2	Using C++ in embedded programming	58
3.12	Conclusion	58
<b>4</b>	<b>Microcontroller memory</b>	<b>60</b>
4.1	Introduction to memory	60
4.2	User RAM	61
4.2.1	Microcontroller internal RAM	62

4.2.2	External RAM	62
4.3	BUFFALO routines, memory map and vectors	64
4.3.1	BUFFALO as a development tool	64
4.3.2	BUFFALO utility subroutines	65
4.3.3	BUFFALO memory map	65
4.3.4	BUFFALO interrupt pseudo-vectors	66
4.4	Interrupts, vectors and pseudo-vectors	66
4.4.1	Software vs hardware interrupts	67
4.4.2	Maskable vs non-maskable interrupts	68
4.5	Control registers	70
4.5.1	Memory mapped I/O	70
4.5.2	Accessing and using control registers	70
4.6	EEPROM	71
4.6.1	Clearing the EEPROM example	72
4.6.2	Writing to the EEPROM example	73
4.7	Conclusion	74
<b>5</b>	<b>Microcontroller inputs and outputs</b>	<b>76</b>
5.1	Introduction to inputs and outputs	76
5.2	Single ended vs differential inputs	77
5.2.1	Single ended analog circuits	77
5.2.2	Single ended digital circuits	78
5.2.3	Differential analog circuits	79
5.2.4	Differential digital circuits	79
5.3	Digital inputs	80
5.3.1	Switch sensing and de-bounce	80
5.3.2	Normally open (NO) and normally closed (NC) switches	81
5.3.3	Electronic switches	81
5.4	Digital outputs	82
5.4.1	Digital control	82
5.4.2	Back EMF causes and solutions	83
5.5	Analog inputs	84
5.5.1	Voltage, current and resistive measurement	84
5.5.2	Analog and digital filtering and amplification	85
5.5.3	Nyquist and the sample rate	86
5.5.4	Resolution management	87
5.6	Digital control of analog devices	87
5.6.1	Basic stepper motors	87
5.6.2	Stepper motor control and communication	88
5.7	Keypad interfacing	88
5.7.1	Connecting the keypad to the evaluation modules (EVM)	88
5.7.2	Reading the keypad in software	89

5.8	LCD interfacing	91
5.8.1	LCD software setup	92
5.8.2	Writing to the LCD	94
5.9	Conclusion	95
<hr/>		
<b>6</b>	<b>Data communications</b>	<b>96</b>
<hr/>		
6.1	Introduction to data communication	96
6.2	Basics of serial data communication	97
6.2.1	History of serial data communications	97
6.2.2	Three parts of data communications	98
6.3	Open system interconnection model	100
6.3.1	Application layer	101
6.3.2	Session, presentation, transport and network layers	102
6.3.3	Datalink layer	102
6.3.4	Physical layer	102
6.3.5	Protocols and the three layer model	103
6.4	Modes of communications	103
6.4.1	Simplex	103
6.4.2	Half-duplex	103
6.4.3	Full-duplex	104
6.4.4	The master slave bus	104
6.4.5	The CSMA/CD bus	105
6.4.6	The token bus system	106
6.4.7	Timed systems	106
6.5	RS-232	107
6.5.1	Introduction to RS-232	107
6.5.2	Function of the lines	108
6.5.3	RS-232 installation and troubleshooting	108
6.6	RS-485	111
6.6.1	Introduction to RS-485	111
6.6.2	RS-485 vs RS-422	112
6.6.3	RS-485 installation and troubleshooting	113
6.7	Fiber optic cables	114
6.8	Fieldbus protocols used in controllers	115
6.9	Conclusion	116
<hr/>		
<b>7</b>	<b>Noise reduction</b>	<b>118</b>
<hr/>		
7.1	Introduction to noise reduction	118
7.1.1	The decibel	119
7.1.2	Signal to noise ratio	119
7.1.3	Sources of noise – internal vs external	119
7.1.4	Single ended or grounded circuits	121

7.1.5	Single ended measurement of grounded sources	121
7.1.6	Single ended grounded equipment	122
7.1.7	Differential noise circuits	123
7.1.8	Differential test equipment	124
7.1.9	Common mode noise problems	125
7.1.10	Low impedance drops as noise sources	126
7.1.11	Types of externally induced noise	127
7.2	Conductive coupled noise	127
7.2.1	Conductive noise from external equipment	128
7.2.2	Conductive noise from transmission lines	128
7.3	Capacitive coupled noise	129
7.3.1	Capacitive noise from adjacent equipment	129
7.3.2	Capacitive noise from communication lines	130
7.4	Magnetically coupled noise	130
7.4.1	Magnetically induced noise from adjacent cables	131
7.4.2	Magnetically induced noise from adjacent equipment	132
7.5	EMC and noise reduction in PCB design	132
7.5.1	Placement of analog, digital and power supply circuits	133
7.5.2	Digital circuit decoupling	134
7.5.3	Ground planes	135
7.5.4	1D and 3D Faraday shields	135
7.6	Conclusion	136
<b>8</b>	<b>EMC grounding solutions</b>	<b>137</b>
8.1	Introduction to EMC grounding solutions	137
8.2	EMC grounding	138
8.2.1	Ground specifications	138
8.2.2	Types of earth grounds	141
8.3	EMC grounding on a PCB	143
8.3.1	PCB design recommendations	143
8.3.2	Track placement	147
8.3.3	Faraday boxes	148
8.4	Protecting a PCB from lightning	150
8.4.1	Placement of protection on the PCB	151
8.4.2	The GDT, MOV and transorb	152
8.5	Microcontroller equipment ground	153
8.6	Enclosure or safety ground	154
8.6.1	Spiked earth grounds	154
8.6.2	Cable trench grounds	155
8.6.3	Tower lightning protection	156
8.7	Conclusion	157

---

<b>9</b>	<b>Installation and troubleshooting</b>	<b>159</b>
9.1	Introduction to installation and troubleshooting	159
9.2	Connections – screw, crimp and solder	160
9.2.1	Screw connectors	162
9.2.2	Crimp connectors	163
9.2.3	Soldering connections	164
9.2.4	Connector problems and solutions	164
9.3	Cable runs and trays	166
9.3.1	Metal vs plastic runs and trays	166
9.3.2	Vertical runs and trays	167
9.3.3	Horizontal runs and trays	168
9.4	Cable ties and mounting	169
9.5	Cooling, heating and air conditioning	170
9.6	Wire management in a cable run	171
9.7	Conduit installation	172
9.8	Troubleshooting techniques	173
9.9	Safety considerations	174
9.10	Conclusion	175
<b>10</b>	<b>End notes</b>	<b>176</b>
10.1	Conclusion	176
10.2	CPU design and functions	176
10.3	Assembly language programming	177
10.4	Memory	178
10.5	Inputs and outputs	178
10.6	Data communication	179
10.7	Noise reduction	180
10.8	Grounding solutions	181
10.9	Installation techniques	181
10.10	Final words	182
	<b>Practicals</b>	<b>183</b>
	Practical 1: Setting up the 68HC11 emulator board	183
	Practical 2: Activating LEDs on the EVM	191
	Practical 3: Reading switches on the EVM	197
	Practical 4: Sending characters to an LCD display	204
	Practical 5: Reading keypad input	212

---

Practical 6: Using the PAT software	221
Practical 7: Viewing character data transmission	226
Practical 8: Troubleshooting a data communication system	233
Practical 9: Troubleshooting a protocol problem	236
Bibliography	242
Index	243

# 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

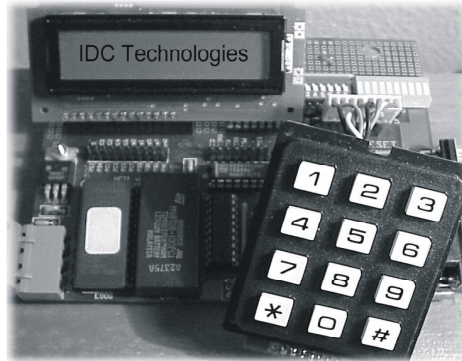
## 1.1 Microcontroller introduction

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.

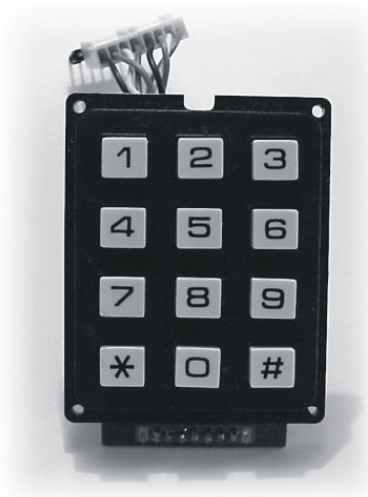
Embedded controllers are intelligent electronic devices used to control and monitor devices connected to the real world. This can be a microwave oven, programmable logic controller (PLC), distributed control system (DCS) or a smart sensor. These devices are used in almost every walk of life today. Most 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*

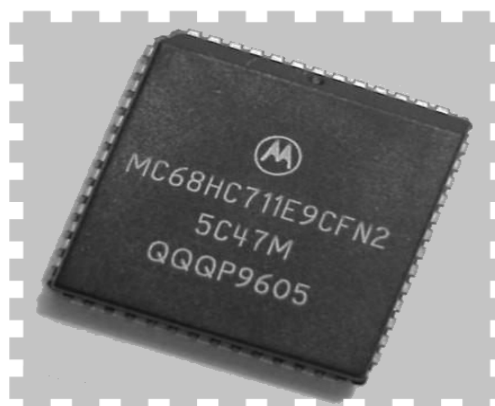
Electronic equipment is becoming more and more susceptible to noise and other outside influences that can cause catastrophic problems. To be able to troubleshoot and ultimately repair the embedded controller it is not only necessary to understand the inter-workings of the embedded controller but also 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.

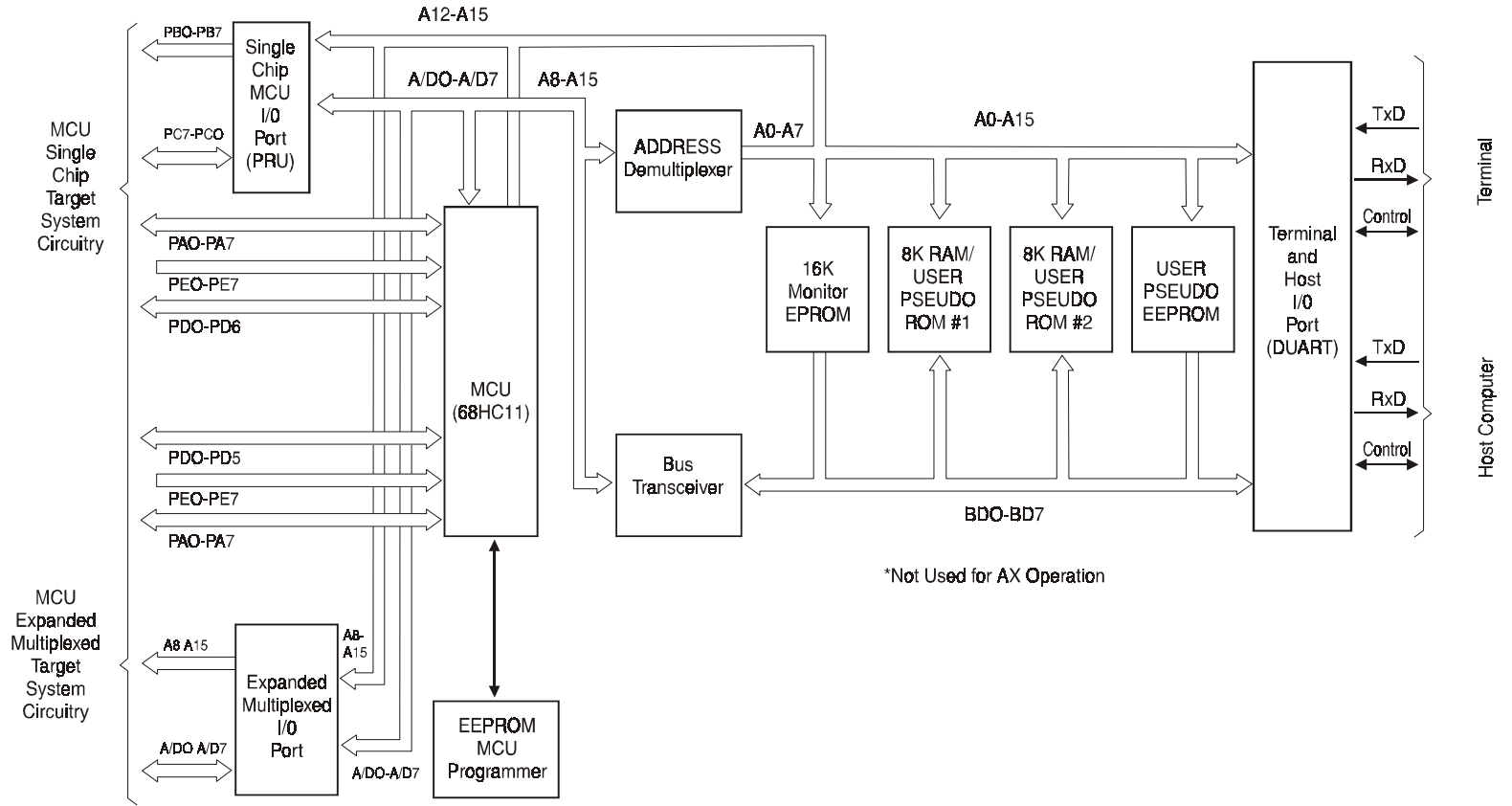


**Figure 1.3**  
*68HC11 CPU*

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



**Figure 1.4**  
Block diagram of a microcontroller

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.

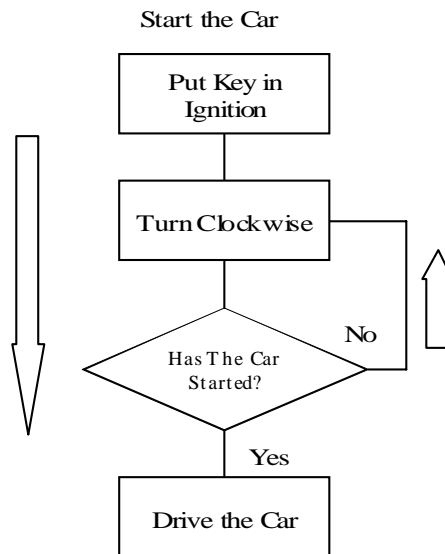
Programming is often compared to painting a picture. One difference though is that in art it is often unclear when the painting is finished. In programming the program is done when it does what it was designed to do. This can be defined and specified before the program is written. Strangely enough, this 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 is 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.



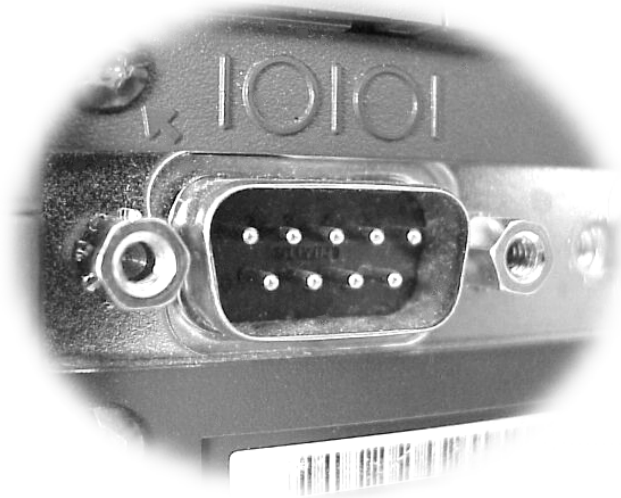
**Figure 1.7**  
*Typical inputs and outputs*

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 than 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.

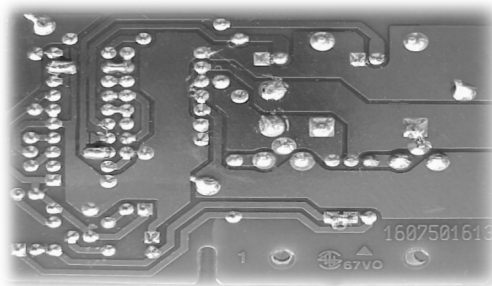


**Figure 1.9**  
*USB connector*

## 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.

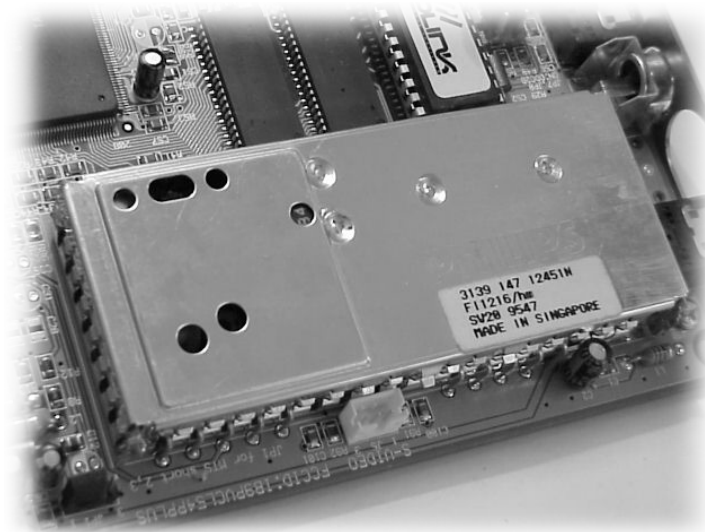
We find that the reduction of noise can be as easy as either moving the offending transmitting wire away from the victim wire or moving the victim wire away from the broadcasting wire. In the past, noise reduction, troubleshooting and repair was done by using oscilloscopes and filters. Since the advent 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*



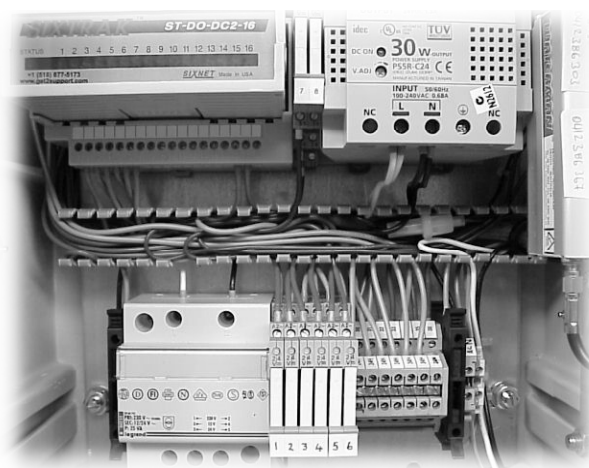
Each of these areas has gone through substantial changes of late and will continue to evolve over the years as noise reduction requirements change. The need for increased noise reduction from a PCB/EMI radiation point of view is universally expected to increase in the future. Proper placement of components 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.

## Microcontroller basics

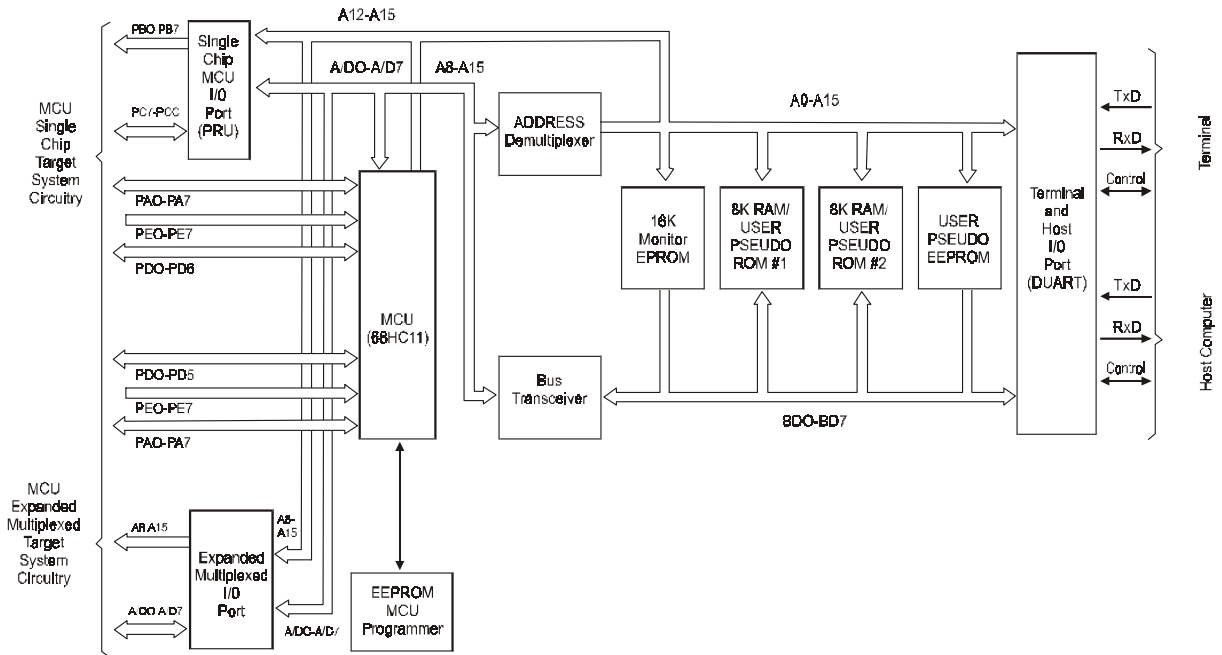
### Objectives

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

- Describe the basic building blocks and functions of microcontrollers
- Explain what numbering systems are used in microcontrollers
- Describe gates and their function in a microcontroller
- Describe accumulators and their function
- Describe the X, Y and stack registers
- Explain the difference between asynchronous and synchronous communications
- Explain why power systems are important to microcontrollers

### 2.1 Introduction

This chapter gives an introduction to the microcontroller, the main component of all embedded controller systems. The microcontroller is the most powerful chip in the arsenal of the electronic designer. At the beginning of the project the designer often starts with microcontroller selection. Everything that flows on in the design will depend on the intelligence, functionality and availability of the microcontroller. Some experienced engineers may find this chapter a little too basic and the absolute beginner may find this chapter a bit advanced but it is important to have an understanding of the basics of microcontrollers.



**Figure 2.1**  
Block diagram of the microcontroller

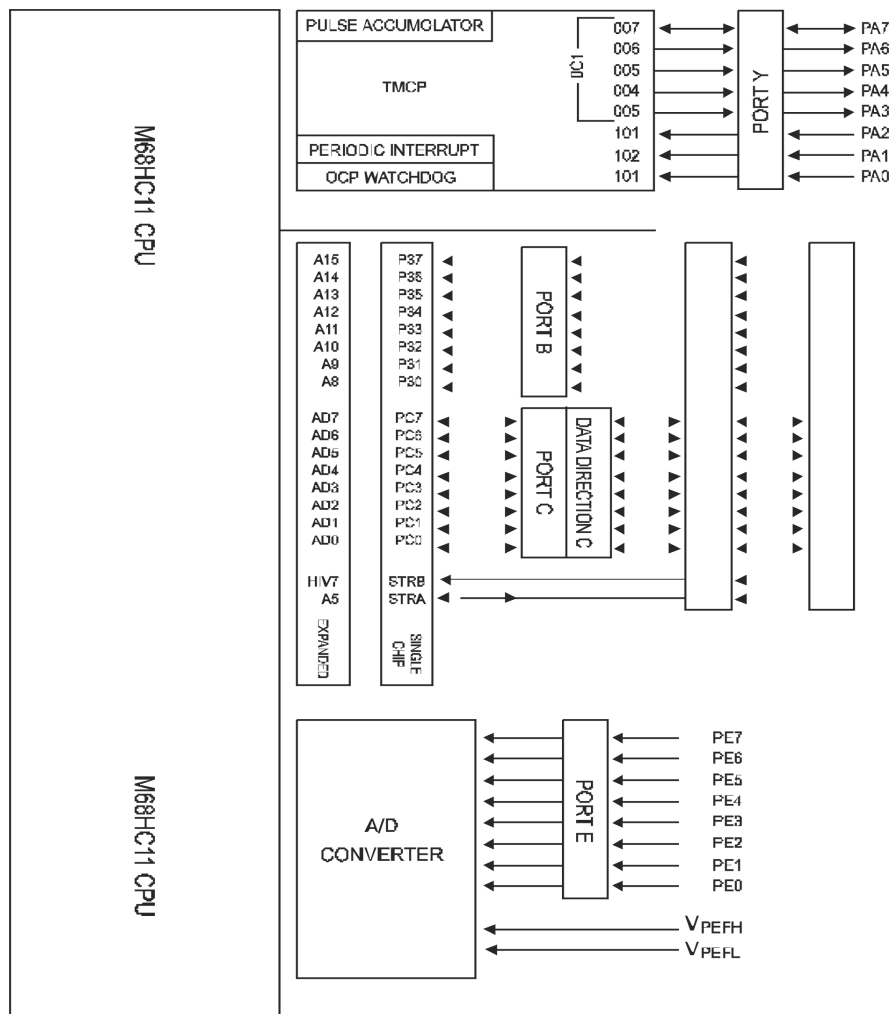
Microcontrollers in embedded controllers are sequentially programmable devices. This means that they execute one function at a time. The program is held in memory like RAM (random access memory) in the form of an eight-bit (1 byte) binary number. We usually think of the byte in hexadecimal form. An example of this would be 46h. The little 'h' after 46 tell us that the 4-6 is a hexadecimal number and not forty-six. The engineer could use little h, \$, H or 0X to indicate a hex number (\$A3F1). Sometimes we need to convert a decimal number into an ASCII code number. To do this, first the decimal number 3 would be converted into 03h. Then we would add 30h to 03h and get the ASCII 33h.

The instructions that are used to tell the microcontroller what to do are included in an instruction set. This instruction set changes from microcontroller to microcontroller as different manufactures and even different chip sets within a manufacturer have different instruction sets. The good thing is that most instruction sets are similar. This makes it easy to change from one microcontroller to another for the experienced designer. The instructions that are used to program the microcontroller can be grouped into different sections. These sections make it easier to understand the ultimate function of the instructions.

Typical instruction groups would be:

- Logical operations
- Loads and stores
- Arithmetic functions
- Shifts and rotates
- Data testing and bit manipulation
- Stack and index registers
- Condition code registers
- Branches
- Jumps calls and interrupt handling

The microcontroller holds program data in temporary storage locations within the microcontroller called accumulators or registers. The accumulators are locations that can be added to and therefore accumulate data. Most microcontrollers have either 8-bit or 16-bit accumulators. The 68HC11 has two 8-bit accumulators that can be accessed as one 16-bit accumulator (A (8-bit) + B (8-bit) = D (16-bit)). If A has \$30 and B has \$03 then D has \$3003. Accumulator D is actually accumulator A and accumulator B. If you change D you will be changing A and B accumulators. The registers such as X and Y are temporary storage areas that are used for holding 16-bit data like addresses and pointers (such as \$DE00). The X and Y registers can hold addresses, other register values and I/O port locations. Pointers are addresses that specify the locations of something in the microcontroller or program. The program might ask a register for the address that ‘points’ to the location of port D. The register might then come back with an address of \$1008. This would be the location of port D. The program might then ask the microcontroller to place something in port D (\$1008).

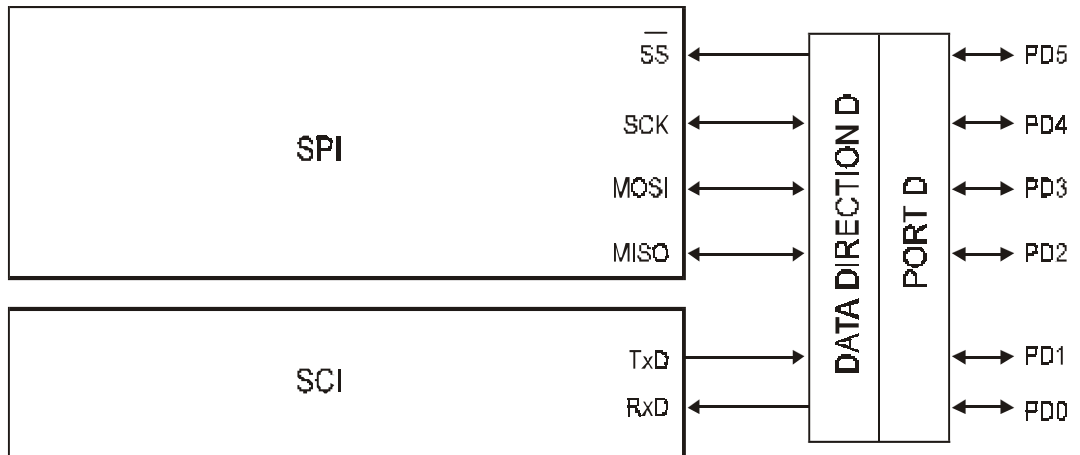


**Figure 2.2**  
Block diagram of the A, B, C, D and E ports

The microcontroller not only can transfer information to locations within itself but also is able to send data to the outside world through data ports. The HC11 has five data ports (A, B, C, D (not to be confused with accumulator D) and E). These ports are used for

digital inputs, digital outputs and analog inputs. Some of the ports are fixed as inputs or outputs, while others are definable by the programmer. An example of this is port A. It has three dedicated inputs, four dedicated outputs and one definable input or output line. Another example could be port B. All eight lines on port B are outputs and they can't be anything else.

The HC11 has two communication modes that share one port. One mode is asynchronous and the other is synchronous. Either the asynchronous (SCI) or the synchronous (SPI) modes can use two lines of port D as a transmit and receive lines. It is not possible to send or receive Asynchronous and Synchronous data at the same time as they use the same two lines for TX and RX. The other lines on port D are used for configuration of the transmit and receive system. The communication system is used to talk to either a PC or other controllers on some type of physical system such as RS-232, RS-485 or even fiber optic cables.



**Figure 2.3**  
*Block diagram of the communications ports (SCI and SPI)*

Microcontrollers must get power from somewhere and usually this is from mains supply, although this may change in the future. Mains power has many problems such as spikes, brownouts and blackouts. In the past it has been somewhat reliable, but this seems to be changing with brownouts and total loss of mains happening more and more often. Solar power and battery power are on the horizon for power systems, but this doesn't solve all of the problems of mains power. Brownouts and blackouts can and often happen on solar and battery systems. The two systems will increasingly be used together, one as backup to the other. When these problems do happen, it is important that the microcontroller handle the problem invisibly. Both good PCB design and power system design are used to reduce the potential cause and effect of spikes, brownouts and blackouts. Microcontrollers are often used to control mains/solar power systems.

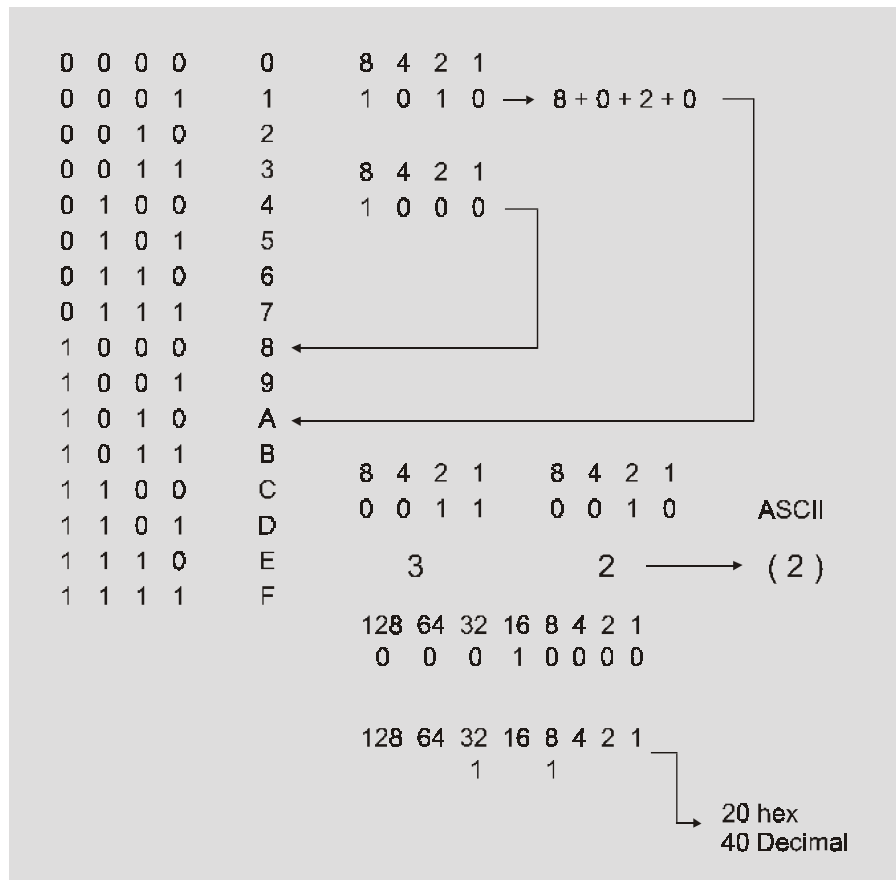
Another part of having a quality microcontroller system is consistent clocking. One of the greatest advances in electronics in the last twenty years has been in the area of clocking. Placement of the oscillator on the PCB is very important from both a functional point of view and also from an EMC view. If the crystal is too far away from the microcontroller it may cause excessive noise or it may not work at all. If it is too close to certain chips it will fail or at the least be unreliable. In the design of controllers it is important to choose the right oscillator, as this has a direct bearing on the speeds of the data communication.

## 2.2 Number systems – binary, hex, and decimal

Since data within the microcontroller is held as numbers it is important to understand some basics about number systems. Most people find working with numbers to be boring or a bit too hard, but everything in this book is based on numbers.

### 2.2.1 Bits, dibits, nibbles, bytes, words and long words

A bit is a one or a zero. This seems obvious enough, but it is important to remember that the one and zero are real voltages on the printed circuit board (PCB). A one is usually a +5 volt level with a zero being a ground or 0 volt level. Often people that are new to digital electronics think that a one is on and a zero is off. This is not necessarily true. In fact often a zero is on and a one is off. It is therefore not a good idea to think in on and off terms when it comes to microcontrollers. It is better to think in ones or zeros or voltage levels.



**Table 2.1**  
*Binary, nibble, hex and decimal conversion*

A dibit is two bits in a row, such as 10 or 11 or 00 or 01. Notice that there are only four possible combinations with a dibit. A nibble is four bits and starts at 0000 and goes to 1111. The decimal equivalent of the nibble is 0 to 15. The hex equivalent is 0 to F (see Table 2.1). Notice that the decimal values of each place are 8 then 4 then 2 then 1 as read from left to right. This is the same as 1000 then 100 then 10 then 1 in decimal. A byte is

two nibbles and therefore 8 bits. The byte is usually written as \$3F or 34h or 34hex or 0X34 depending on the school you went to. This book will use the \$ sign as a representation of hex because that's the way it is represented in most assembly language programming. The decimal range for a byte is 0 to 256. The hex equivalent is 00 to FF (see Table 2.1). All instructions in an 8-bit microcontroller are usually given in hex bytes. Good microcontroller engineers think in hex. A digital word is 16 bits or two bytes. This is used mostly in registers within the microcontroller. Addresses in an 8-bit data bus by 16-bit address bus microcontroller are written as a 16-bit word.

A bus is a group of parallel lines on a PCB or in a microcontroller chip that carry the 1s and 0s (voltages) to a common point.

There are three types of buses on a microcontroller:

- **Address bus:** This bus holds the 16-bit address of the port or register
- **Data bus:** This bus holds the 8-bit data from the port or register
- **Control bus:** This bus holds the bits that control chips and devices

### 2.2.2 Conversion from binary to hexadecimal

Often in the program it is important to convert from binary to hexadecimal. An example of this would be when writing to a register. If you wanted to set bit 0 in a register to a 1 you would send the byte \$01. This seems pretty straightforward but what if you wanted to turn on bit 7, 5 and 2. A 10100100 in binary (\$A4 in hex) would be sent. To convert from binary to hex you first split the 8 bits into two sections of binary nibbles. Then convert the left or most significant binary nibble into hex. Then do the same for the least significant binary nibble on the right. The two hex nibbles are then put next to each other and the conversion is done.

### 2.2.3 ASCII, hexadecimal and BCD

Morse Code			
A	· · -	J	· - - -
B	- · · ·	K	- · ·
C	- · - ·	L	· - · ·
D	- · ·	M	- -
E	·	N	- ·
F	· · - ·	O	- - -
G	- - ·	P	· - - ·
H	· · · ·	Q	- - - -
I	· ·	R	· - - ·
S	·	T	-
U	· · -	V	· · · -
W	· - -	X	- · - ·
Y	- · - -	Z	- - · ·
Period	· - - - -		- - - - -

**Table 2.2**  
*Morse code*

In the early days of data communication Morse code was used as the language to send data. One problem with Morse code was that the number of characters was limited and the other is that there is a different number of bits for various characters. When microcontrollers were developed, it was necessary to develop a code that was flexible and yet had the same number of bits for every character. ASCII was developed and is the most common code used in electronics today. Often people get confused because the binary and hex can represent ASCII. Remember all hex characters can be converted into



binary and all binary numbers can be converted to hex. Another confusion with ASCII is there are two versions. There is the basic 7-bit version and the 8-bit extended version. It is easy to remember that the 7-bit set is just a sub-set of the 8 version. Because the 7-bit set has most of the characters that we use, often just the 7-bit section is used instead of the complete 8-bit ASCII character set. A good way of thinking of this is that you only use a small portion of the total English words available on a day to day basis. This begs the question, what do we do with the left over bit (bit 7)(remember it is bit 0 through bit 7)? Usually we just set it to 0 and ignore it.

		MSB								
		HEX	0	1	2	3	4	5	6	7
LSB	HEX	BIN	000	001	010	011	100	101	110	111
	0	0000	(NUL)	(DLE)	Space	0	@	P	'	p
	1	0001	(SOH)	(DC1)	!	1	A	Q	a	q
	2	0010	(STX)	(DC2)	"	2	B	R	b	r
	3	0011	(ETX)	(DC3)	#	3	C	S	c	s
	4	0100	(EOT)	(DC4)	\$	4	D	T	d	t
	5	0101	(ENQ)	(NAK)	%	5	E	U	e	u
	6	0110	(ACK)	(SYN)	&	6	F	V	f	v
	7	0111	(BEL)	(ETB)	'	7	G	W	g	w
	8	1000	(BS)	(CAN)	(	8	H	X	h	x
	9	1001	(HT)	(EM)	)	9	I	Y	I	y
	A	1010	(LF)	(SUB)	*	:	J	Z	j	z
	B	1011	(VT)	(ESC)	+	;	K	[	k	{
	C	1100	(FF)	(FS)	,	<	L	\	l	
	D	1101	(CR)	(GS)	-	=	M	]	m	}
	E	1110	(SO)	(BS)	.	>	N	^	n	~
F	1111	(SI)	(US)	/	?	O	_	o	DEL	

**Table 2.3**  
ASCII table

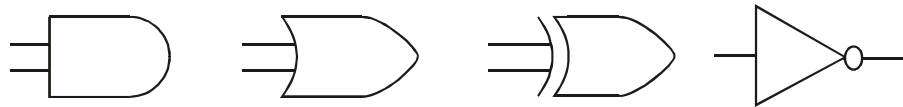
## 2.2.4 Converting from ASCII to BCD and hex

Binary coded decimal (BCD) is a sub-set of hex from \$0 to \$9. BCD is used to do arithmetic functions within programs. It is not very obvious to the most casual observer what the result of adding \$33 and \$31 in ASCII is in decimal. Note that in the ASCII table (Table 2.3) that \$33 is 3 and \$31 is 1 decimal. It would be easy for us to add them, but how does the microcontroller do it? The microcontroller converts the numbers into BCD and then adds them. It first removes or subtracts \$30 from each character. This leaves us with \$03 and \$01 BCD. Remember that BCD is the sub-set of hex from \$0 to \$9. The addition of \$03 and \$01 is \$04. It then adds \$30 to the result and it gets \$34 or 4 decimal in ASCII.

## 2.3 Gates – AND, OR, XOR and NOT gates

One of the most powerful functions of the microcontroller is its ability to replace hardware with software functions. A good example of this is the hardware gates, AND, OR, XOR and NOT gates. Because hardware takes up space on the PCBs, they increase the cost of the product. It can be a large benefit to replace hardware with software functions. Once the software is written the continued product cost is minimal and can be considered almost nil.

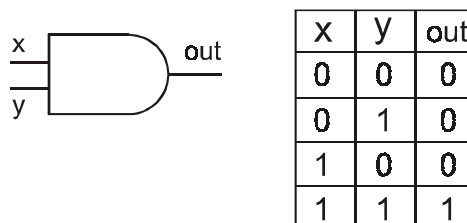
So how does the microcontroller replace a physical gate with a virtual gate? The method is simple; the microcontroller reads the information from a port or address location and then performs the gate function in the microcontroller. The result of the operation is then sent to a port or placed in another address location. This makes the microcontroller very flexible and powerful. When using hardware gate functions, the output of the gate is usually located on the same physical chip as the inputs. With microcontrollers, the output can be anywhere on the system. The result can even be stored and used later in another operation.



**Figure 2.4**  
*AND, OR, XOR and NOT gates*

### 2.3.1 AND gates – physical and virtual

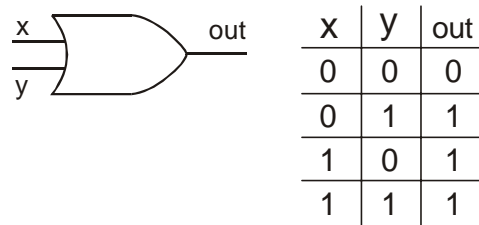
The AND gate is a device with two or more inputs, where the result is an AND function of the inputs. Figure 2.5 shows that only when the two inputs (x and y) are ones will the output be a one. In a physical gate, chips would be used to make this happen, but in a microcontroller the AND instruction performs this operation. If  $x = 1$  and  $y = 0$  or if  $x = 0$  and  $y = 1$  then the output is 0. If  $x$  and  $y$  are 0 then the output is 0. And gates are often used for what is called masking. Masking is seeing the bits you want and ignoring or disregarding the rest. Masking was used in early computer punch cards. Certain holes would be masked so that only, say, the cities in one particular state would be collected or the cities with the same name could be masked and the others disregarded. When a binary number on a port is masked using the AND function, only the bits we are interested in come through and the rest are disregarded. In Figure 2.5 it can be seen that the bits on the left are the ones we are interested in and the bits on the right are ignored. This masking function is often used to view the condition (1 or 0) of a bit or bits in a register, port or address location.



**Figure 2.5**  
*AND gate truth table and masking*

### 2.3.2 OR gates – physical and virtual

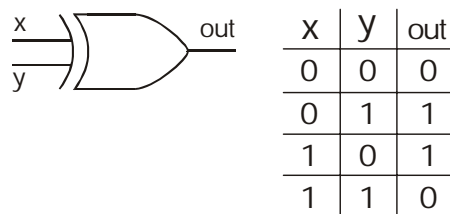
The OR gate function is used in both physical and microcontroller operations to see if any of the inputs are a 1. As seen in Figure 2.6 the output of the gate is a 1 if any or both inputs are a 1. If both inputs are 0s then the output is a 0. This gate is used to allow two devices to turn on the same thing. An example of this would be two switches, one at each end of an assembly line that can turn on the lights in the area. Combining the OR gate with the AND gate it would be possible to turn on the lights when either (OR) switch is thrown and if it is dark enough in the area (AND). In microcontrollers this is used when two inputs, either from outside locations such a port or from inside the controller such as the state of a memory bit, control an output.



**Figure 2.6**  
*OR gate truth table*

### 2.3.3 XOR gates – physical and virtual

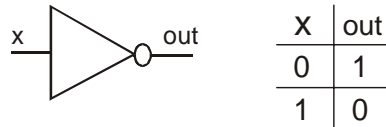
XOR gates are exclusive OR gates. The exclusive OR gate defines that the output is a 1 only if the inputs are different (i.e. one input is a 1 and the other input is a 0). Any single 1 creates a 1 on the output, but if the input has two 0s or two 1s then the output is 0. This gate works similar to the example of the OR gate but with one important difference. In the OR gate example if the lights were on, both switches would have to be placed in the off position for the lights to be turned off. The last person out at night would have to walk down to the end of the assembly line and turn all the switches off. Remember in a normal OR gate both inputs have to be 0 for the output to be 0 and if both inputs are 1s then the output is a 1. With the XOR gate system if one switch is turned on (whether that is a 1 or 0 doesn't matter) and then the other switch is turned on, the light will go off. In this way the last person out can turn either switch and the light will go out. This system is great for emergency switches, because either switch would turn off the machinery. In a software-controlled microcontroller system this function cannot only be used on hardware but also within the program.



**Figure 2.7**  
*XOR gate truth table*

### 2.3.4 NOT gates – physical and virtual

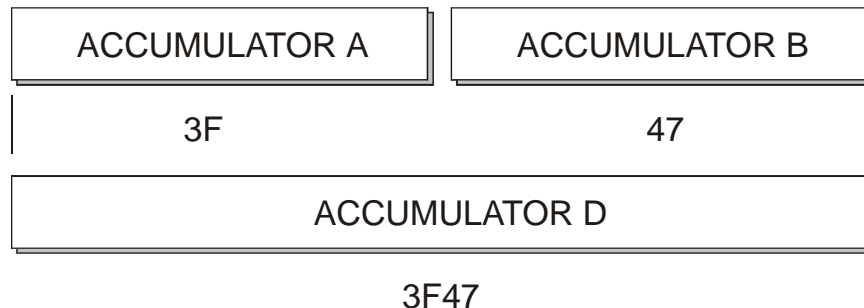
The NOT gate is used to reverse the input and place it on the output. This gate is often used in conjunction with the AND, OR and XOR gates. The NOT gate is placed on the input or output of the other gates to inverse the function of the gate. The software equivalent of the NOT gate is the *one's compliment*. This function inverts every bit. All 1s become 0s and all 0s become 1s. This instruction is useful in certain arithmetic functions. One example of the use of the NOT gate would be if the output of a function was a 1 and it was represented as ON, but the outside function was looking for a 0 as ON. The NOT gate would be put in between to change the 1 to a 0.



**Figure 2.8**  
*NOT gate truth table*

## 2.4 Accumulators, A, B and D

When writing a program it is necessary to have some temporary place to hold the data while the program is doing something else. The accumulators as mentioned before are 8-bit registers that temporarily hold one byte of data and have the ability to accumulate data. This means that you can add or subtract bits from this register. For example, if the programmer wanted to save some data before sending it to a port, accumulator A or B could be loaded with the data. The instruction for loading accumulator A is LDAA. That is **LoaD** Accumulator **A**. To send the data to an accumulator the programmer might use STAA, that is **STore** Accumulator **A**.



**Figure 2.9**  
*Diagram of the A, B and D accumulators*

### 2.4.1 Addressing modes

The addressing mode defines how the instructions will access the memory. There are six types of modes or addressing.

Immediate

This mode of addressing takes the data after the instruction and places it in the location as defined by the instruction. An example of this could be LDAA #\$4F. The instruction

loads \$4F in accumulator A immediately. The LDAA instruction defines where the data is stored.

#### Direct

This mode uses the first 256 bytes of RAM in memory. This address range starts at \$0000 and ends at \$00FF. An example of this could be STAA \$2A. This example would store the data that is in accumulator A directly into address \$002A. The \$00 part of the address is implied.

#### Extended

The extended mode can access an address anywhere in the total range of addresses from \$0000 to \$FFFF. An example of extended mode is LDAA \$DE00. This loads accumulator A with the data located in address \$DE00. This addressing extends to the whole addressing range of the microcontroller. Remember that the data held in an address is one byte.

#### Indexed

The indexed mode of addressing uses the index registers X or Y to hold or point to the address that is used in the instruction. This addressing mode can use offsets as shown in the example

```
#LDX #START      This points to the address of START
STAA $01,X       This stores what is A at X (START) address plus 1
```

If START is defined as address \$DF00 then the data in A will be placed in address \$DF01.

#### Relative

Relative addressing is used in branching and is limited to +127 and -128 addresses from the branch. This mode is relative to the address location of the branch. An example of relative addressing is BNE START, where START must be within +127 addresses of the branch or -128 addresses of the branch.

#### Inherent

The inherent mode of addressing is used by instructions that hold the address within the instruction. An example of this addressing mode is RTS. This instruction is return to subroutine. The programmer doesn't need to tell the instruction where to return it is inherent.

## 2.4.2 Loading, storing and transferring accumulators

An example of this loading and storing data in an accumulator is...

```
LDAA #$39
STAA $1008
```

In this example the number 9 in ASCII (\$39) is loaded in to accumulator A. The data (\$39) would then be stored in address \$1008. Port D is address \$1008 in an HC11 microcontroller. Data that is being held in an accumulator can be transferred from one accumulator to another. This would be done with the commands, TAB or TBA. That is transfer accumulator A to accumulator B (TAB) and transfer accumulator B to accumulator A (TBA). Accumulator D is a combination of A and B, with A being the most significant byte and B being the least significant byte.

### 2.4.3 Add, subtract, compare, increment and decrement A and B

Besides being loaded, stored and transferred the data in the accumulators can also be added, subtracted and tested. The addition, subtraction, compare, increment or decrement instructions can be from one accumulator to another, from the X or Y registers to an accumulator or from a memory location to an index register. Accumulator A can be added to B with the command ABA, which is Add accumulator A to accumulator B. The accumulators can also be added to the X and Y registers. The problem with this addition is that the X and Y registers are 16-bit register and the accumulators are 8 bit. This means that the 8 bits in the accumulator will be added to the least significant byte in the X or Y register. The instruction for this addition is ABX or ABY.

Comparing of the accumulators to each other, X or Y registers or a memory location is used to check if the data in the accumulator is the same or different from what was expected. This is used extensively to stay in or exit a loop in a program. Common instructions used in program loops are CMPA and CMPB, although CBA (compare A to B) is also used. It is also common sometimes to compare accumulator D to the data in a memory location with the instruction CPD.

The increment and decrement instructions are used by the program to increase or decrease the contents of the accumulators or index registers. This can be used to count down or up within a loop.

### 2.4.4 AND, OR and test bits

If the programmer wants to AND or OR bits in a particular memory address, the instructions ANDA or ORA could be used. This is a logical AND and is the same as the physically ANDing the bits. ANDing and ORing are used for bit masking as described in Section 2.2. Using the instructions ORAA or ORAB the programmer can do the 'normal' type of ORing process. By using the EORA or EORB the programmer can do an exclusive OR function.

If the programmer wishes to just test one bit then the instructions BITA or BITB could be used. This instruction only tests the bits defined in the data following the instruction. An example of this might be...

LDAA \$1008	This loads accumulator A with the value in port D (one byte)
BITA #\$10	This checks to see if bit 4 is a 1 (00010000)

**Note:** This instruction does not change the data value in A. Whereas the instruction ANDA changes the value of accumulator A.

### 2.4.5 Arithmetic and logical shifting and rotating

Shifting of bits in an accumulator or memory area is used typically in arithmetic functions. The difference between a shift right and a rotate right is that in a rotate right the most significant bit (bit 7) is shifted to bit 0. The data goes around and around in a circle. In a shift right bit 7 either stays the same (arithmetic shift) or is moved into the carry bit of the condition code register (logical shift). There is no difference between an arithmetic shift left and a logical shift left.

### 2.4.6 Data and bit testing

As mentioned in 2.4.4 the instructions BITA or BITB are used to test one or more bits in an accumulator. Once the bits have been tested the usual next step is to branch if a condition is true, false, high, low, equal or not equal. Some of the branching instructions that are used for this are BEQ or BNE. These are **B**ranch if **E**qual or **B**ranch if **N**ot

Equal. Another pair that is common is BHI and BLO. These of course are branch if higher and branch if lower. Two others are BHS and BLS. These instructions are branch if higher or the same and branch if lower or the same. These instructions could be used in the following program.

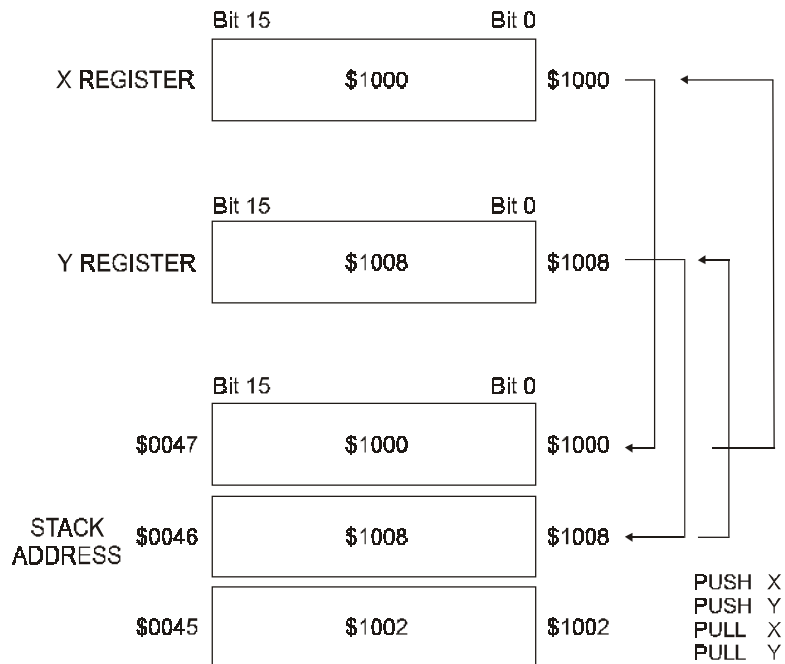
```

START      LDAA      $1008      This loads A with the data in port D.
BITA      #$31        This checks if A contains the bits (00110001)
BNE       START      If it does not then the program loops to START.
STAA     LCD         If it does then it stores A in the LCD screen.
RTS
    
```

## 2.5 Registers – X, Y, the stack and ports

### 2.5.1 The X and Y registers

Since the HC11 microcontroller has a 16-bit wide address bus the 16-bit register is a convenient place to temporarily hold addresses and other data needed by the microcontroller. Eight-bit registers are usually used for data or values, whereas 16-bit registers are usually used to hold addresses. Two index registers, X and Y are used in 68HC11 microcontrollers. These registers are used by the microcontroller to point to the addresses of ports and other registers within the microcontroller. One advantage of using the X and Y pointing feature is that the addresses held by these X or Y registers can be manipulated within the program. For example, the address of a RAM memory location like \$CF00 could be placed in the X register. The RAM locations of \$CF00 through \$CF09 could hold the numbers \$30 through \$39. These numbers are the ASCII numbers 0 to 9. The X register in the program could then be incremented by one. The information retrieved from the RAM would be 0 to 9. This would be done with the following example.



**Figure 2.10**  
Diagram of the X, Y registers and stack

	LDX	\$CF00	This points to address \$CF00 (loads X)
XAMPLE	LDAA	\$00,X	This loads the data in \$CF00 (offset of 0)
	STAA	\$1008	This stores the data to address \$1008
	INX		This increments the <u>address</u> pointed to by X
	BRA	XAMPLE	This branches back to the beginning

In this example if the ASCII numbers 30 to 39 were in addresses \$CF00 to \$CF09, then the ASCII numbers 30 to 39, would be placed in port D (\$1008).

Data direction registers are locations in the microcontroller where the programmer places data to define the direction of the lines of the port. The registers hold binary information bits 0 through 7. For example, a register may hold the bits 10001000, and we would read the register as \$88. The data direction register of port D is located at address (\$1009). If we place a 1 in bit 0 at address \$1009, this would define bit 0 of port D (\$1008) as an output, whereas if we place a 0 at bit 0 of \$1009, bit 0 of port D would be an input.

XAMPLE	LDX	\$1009	Point to data direction register for port D
	LDAA	#\$01	Load A with 00000001
	STAA	\$00,X	Store A (\$01) in \$1009 (bit 0 output)

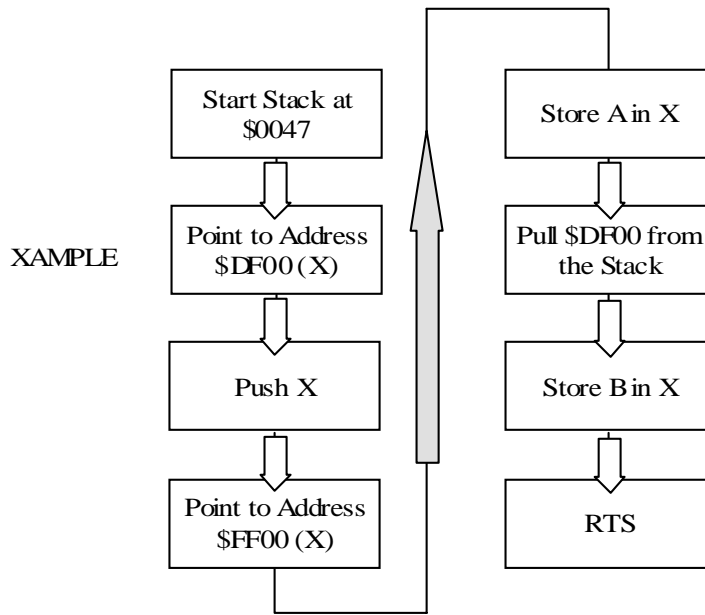
Bit 0 in Port D would then be configured as an output. We could then send a 1 or 0 to \$1008 and the output of port D would have that value of bit 0. If we put a 1 in bit 0 there would be 5 volts on port D bit 0 (PD0), or if we put a 0 in bit 0, there would be 0 volts on the PD0 pin.

## 2.5.2 The stack

The stack is a defined 16-bit register located somewhere in the microcontroller system. The programmer usually sets up the location of the stack at the beginning of the program with the instruction LDS. An example of this would be... LDS \$50. This places the bottom of the stack at address \$0047 (see Figure 2.10). The stack in a microcontroller can be compared to a stack of plates in a kitchen cupboard. The last plate put on the stack is the first plate off. Often the program will push the address of something on the stack and then go off and do something else. Later the program will come back and pull the address off the stack. For every push instruction used in a subroutine there must be a pull instruction. Other wise the stack will get full and the program will stop. An example of the correct way to use the stack is...

	LDS	\$47	This defines the bottom of the stack at \$0047
XAMPLE	LDX	\$DF00	This points to address \$DF00
	PSHX		This puts bytes \$00 at \$0047 and \$DF at \$0046
	LDX	\$FF00	This points to \$FF00
	STAA	\$00,X	This stores the data in \$FF00 in A
	PULX		This pulls the address \$0047 from the stack
	STAB	\$00,X	This stores the data at \$DF00 in B
	RTS		



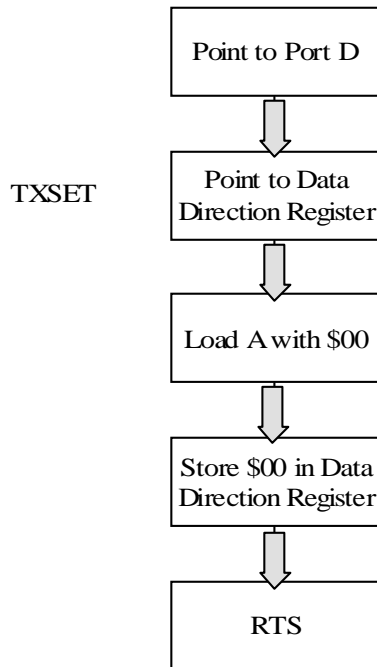


**Figure 2.11**  
Flow chart for push and pull

### 2.5.3 Ports A, B, C, D and E

There are five input and output ports on the 68HC11, A B C, D and E. As mentioned before these ports are used to send information to other chips and devices connected to the microcontroller. Some of the ports can be defined by the programmer as inputs or outputs. Ports A, B, C and D are 8-bit digital I/O, whereas port E is an 8-bit analog input port. Port A is often used for keypad or timer inputs. Ports B and C are often used as address and data buses for external chips when used in expanded mode. Port D is used for serial data communications and port E is used to measure voltages on external analog sensors. An example of using the data direction register is as follows

PORTD	EQU	\$1008	Port D address
DDRD	EQU	\$1009	Data direction control register
TXSET	LDX	#PORTD	
	LDY	#DDRD	
	LDAA	#\$00	
	STAA	\$00,Y	
	RTS		



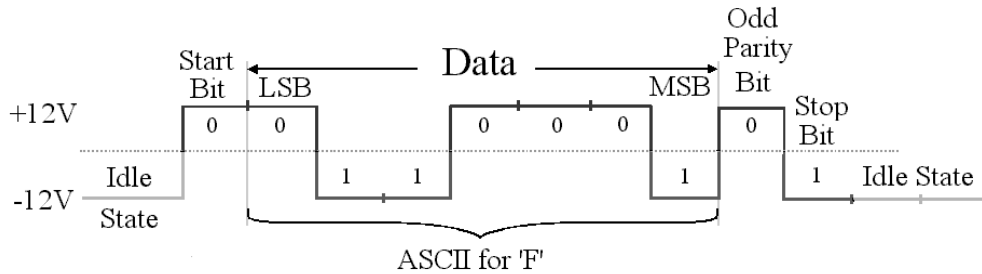
**Figure 2.12**  
*Flow chart showing a setup on port D*

## 2.6 Communications synchronous and asynchronous

The 68HC11 has two communications ports. These ports can be used with RS-232, RS-485 or any other type of physical serial communication standard. Serial communication is used because a parallel communication system would use too many wires and be very limited in overall distance. In the future we may see a movement to parallel communications using fiber optic cables. This is because it is possible to send many different colors of light down a single fiber optic cable at the same time and fiber optic communications is immune to external noise. Using fiber optics it would be possible to send parallel communications tens if not thousands of kilometers and at the speed of light. For the moment we are limited to serial communications. Within serial communications there are two basic transmission modes, asynchronous and synchronous.

### 2.6.1 Asynchronous character communications

Asynchronous communication was developed on teletypes and is still being used today because it is simple and available. Often asynchronous is used because it is available on all computers and is cheap (less than \$5US per port). It is also used because it is simple and easy for the programmer to develop an asynchronous communication system.



**Figure 2.13**  
*Asynchronous character*

Asynchronous communication is a serial communications system where the amount of time between the characters is not defined. One or many characters can be sent at a time. Because it is not known how many characters are going to be sent, each character must be able to stand-alone within the communication system. To make this happen a start and stop bit is added to each character. The start bit is always a 0 and the stop bit is always a 1. Most asynchronous systems use the 1 as an idle state on the communication line. Therefore the start bit (0) differentiates between idle (1) and the character. The stop bit (1) is placed at the end of the character to provide some time between characters. (This was more important on teletypes.) Another part of asynchronous communications is that often the start bit is seen as a voltage level and not as a rising edge. This helps to make asynchronous systems less susceptible to noise, but having said that it should be noted that asynchronous communication is very susceptible to noise because each character must synchronize within the first bit. At 115 k this is less than .001 millisecond. Noise can cause phantom characters that can be misinterpreted by the receiver.

## 2.6.2 Synchronous packet communications

Synchronous communications is a relatively recent development in communications. Because synchronous systems usually run very fast bit rates, timing is very important. Synchronous communication does not have start or stop bits like asynchronous communications. The synchronous characters are usually packaged with start and stop characters and sent as a packet. Ethernet for example sends seven bytes of clocking and then one start byte at the beginning of every packet. At the end of the packet four cyclic redundancy characters are sent as stop characters. Synchronous systems use rising or low going edge triggering to synchronize the receiver. As mentioned above edge triggering is more susceptible to noise but the extra clocking negates this problem. In fact synchronous systems are overall less susceptible to noise and usually have a better data to overhead ratio than asynchronous systems. Overheads are non-data bytes or extra characters sent with the packet. These characters can be clock bytes, addressing characters or error checking information.

Preamble	Start Delimiter	Destination Address	Source Address	Length	Data	CRC
7 Bytes	1 Byte	2 or 6 Bytes	2 or 6 Bytes	2 Bytes	64 - 1500 Bytes	4 Bytes

**Figure 2.14**  
*Synchronous Ethernet packet*

### 2.6.3 Specifying a system – asynchronous vs synchronous

When specifying a system it is often a choice between asynchronous and synchronous.

Typically the following physical systems are:

- RS-232           Asynchronous
- RS-422          Asynchronous or synchronous
- RS-485          Asynchronous or synchronous
- USB             Synchronous
- Firewire        Synchronous
- Ethernet        Synchronous

Asynchronous systems are used when speed is not an issue (Max 115 kbps) and the distances are short (50 meters). Synchronous systems are used when speed and distance are important. Synchronous systems can run at tens of gigabytes speeds and using fiber optic kilometers of distance.

## 2.7 Power systems

From the early days of microcontroller systems they have had the reputation as unreliable. And as with anything, once it is labeled unreliable the reputation sticks. It will take many years, if not decades before microcontrollers will be totally trusted (some say never).

The reasons for this apprehension are:

- People distrust things when they don't know how they work
- Programming mistake
- Hardware failure
- Unknown failure

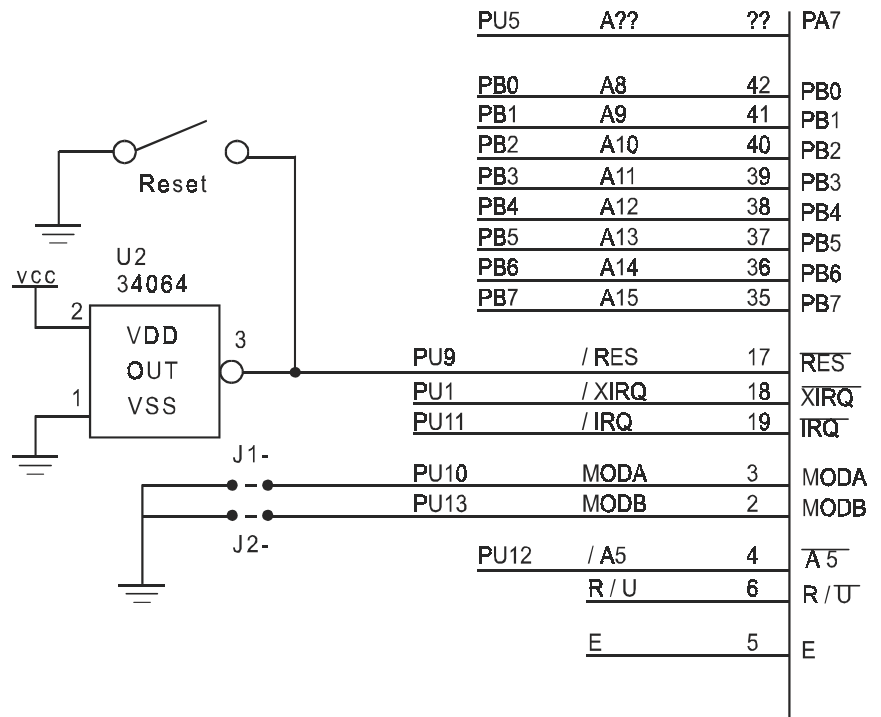


**Figure 2.15**  
*Un-interruptible power supply*

The first three are bad enough but at least we know what is causing the problem, but the last one ‘unknown failure’ is not acceptable. These unknown failures are very common in microcontroller systems. The technician or engineer shows up on site and finds that the equipment has either failed or locked up. The troubleshooter turns the power off and then back on and the equipment starts to work again. The logbook shows ‘unknown failure’. The cause of the lockup is not exactly known but the troubleshooter has a fair idea. There may have been a high voltage spike or power failure that has caused the microcontroller to go off into never-never land. This is a place in the microcontroller’s memory from where it cannot return. Resetting the equipment is the only solution to this problem.

### 2.7.1 Hardware vs software

There are two types of resets on microcontrollers, hardware and software. The hardware reset is a physical line that is toggled usually from high to low by way of a switch. This resets the microcontroller completely from the beginning. The advantage of the hardware reset is that it resets all the chips in the device and clears everything. The disadvantage is that the hardware reset must be done manually. There is a semi-hard reset that can be done by hardware chips on the microcontroller. This semi-hardware reset is not a true hardware reset because it does not turn the power to the chip off and on. The semi-hardware reset toggles the hardware-reset line going into the microcontroller.



**Figure 2.16**  
Reset circuit on an HC11

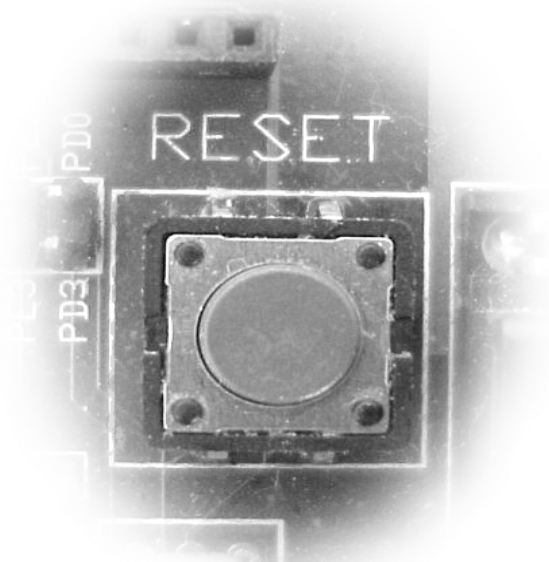
The software reset stops the program and moves the program pointer to the address defined by the manufacturer as the beginning of the program. This is usually \$FFFE. The address located in the data at \$FFFE and \$FFFF is where the program pointer would jump to. If the bytes \$C0 and \$00 were in \$FFFE and \$FFFF respectively, then the program on reset would jump to address \$C000.

## 2.7.2 Hardware reset design

Hardware reset design can take four forms and all four can be used in the same system:

- Power on/off switch
- Reset momentary switch
- Power failure and brownout protection (hardware or software)
- External watchdog timer

The power on/off system is self-explanatory, but is the best way of resetting the equipment. This is because it clears and resets all of the chips. The problem with this type of reset is that the troubleshooter may have to travel long distances to arrive at the site, only to turn the power off and back on again. This can cost the company thousands of dollars per trip. The momentary reset switch, although not as good a reset as using the power switch is often sufficient to reset the equipment. It has the advantage that the power stays stable during the reset. It has the disadvantage that only the microcontroller is reset and the other chips in the system are left in their existing state. The memory may be corrupted and the reset switch may not clear the memory like the power on/off hardware reset.



**Figure 2.17**  
*Momentary reset switch on an EVM*

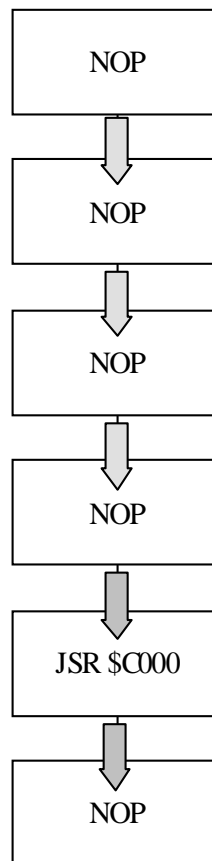
When a reset is performed it is necessary for the programmer to check the memory to see if it has been corrupted. The programmer develops a memory-checking program that makes sure that the memory is error free. Using an error correction system like arithmetic checksum or cyclic redundancy (CRC) on the memory from time to time the program can verify that the memory is accurate. It is possible to verify the memory during a brownout or as the power is being turned off. When the microcontroller notices that the power is dropping it jumps to a memory check program and runs a checksum or CRC on the memory. It then places the result in a memory location before the power is completely off. When the power is restored the program rechecks the memory and compares the checksum or CRC. If they are exactly the same then the memory is left as is. If they are different and therefore assumed corrupted, the program will erase the memory.

### 2.7.3 COP watchdog (Woof)

When a spike of high voltage or static is introduced into a microcontroller, the program counter can become garbled. The program counter points to the next instruction in the program. If the program counter becomes mixed up, the program may start to do strange things. The problem with microcontrollers is that it is impossible to stop this from happening.

But having said that, there are things that can be done to reduce the possibility of this become a problem:

- Reduce the possibility of endless loops in the program
- Put no operation instructions (NOP) in any memory that is not being used
- Put jumps to the beginning of the program every so often in unused memory
- Use the on board computer operating properly (COP) function on board the microcontroller
- Use external hardware watchdog circuits to reset the microcontroller
- Design circuits with high voltage and spike protection on all external wires
- Design in hardware brownout and power failure protection
- Use un-interruptible power systems for the microcontroller



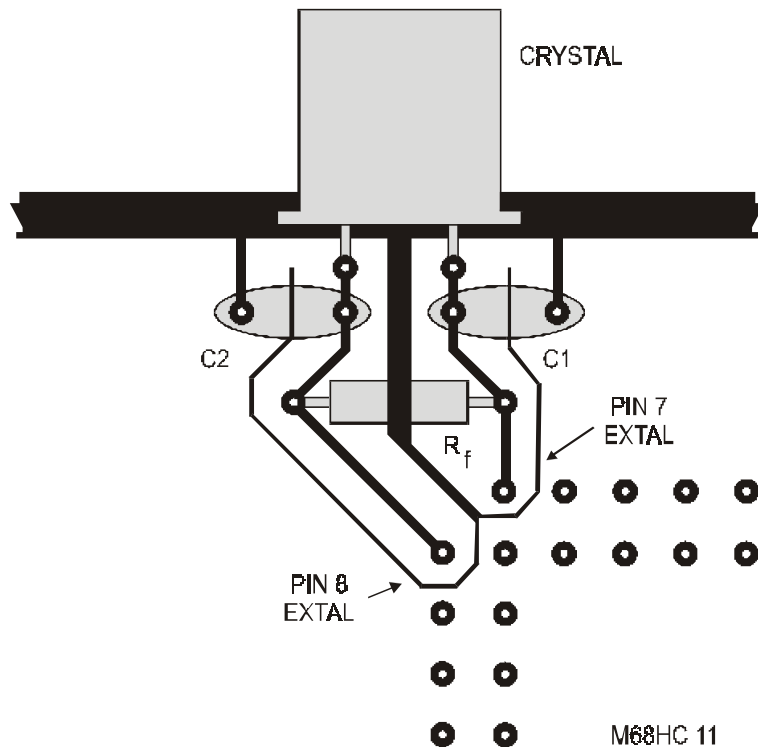
**Figure 2.18**  
*Flow chart for empty memory locations*

### 2.7.4 Power failure and brownout protection

If the input power to the microcontroller becomes unstable the RAM, EEPROM or program counter within the microcontroller could be corrupted. To reduce the possibility of this problem manufacturers of microcontrollers usually suggest the use of external hardware circuitry that resets the chip in a controlled manner. It can be seen in figure 2.16 that a power control chip feeds the microcontroller-reset circuitry. Whenever the voltage drops below legal limits the microcontroller is reset. The problem with these circuits is that if the power is going up and down quickly, the microcontroller will continue to reset. Battery backed RAM and even battery-powered microcontrollers are becoming more and more popular. In this way if the power fluctuates the voltage on the chips remains stable. Some companies are developing completely battery-powered equipment as a way of reducing spiking. This is possible because of the low power drain of recent chips. A small battery can run some devices for up to 5 or even 10 years without changing the batteries.

## 2.8 Crystals and oscillator

Crystals are used to stabilize the clocking in microcontroller oscillators. The selection of the frequency of the crystal determines the speed of the microcontroller and also the baud rate of the communications. The crystal doesn't determine the exact baud rate, but rather a range of baud rates. Placement of the crystal and its support parts is important in PCB design to reduce noise. Crystals are extremely noisy devices and when the lines are long from the crystal to the microcontroller, noise can be transmitted to other chips. To reduce noise on a crystal it is important to place the crystal as close to the microcontroller as possible. It also helps to put a one-dimensional Faraday shield around the crystal if possible.



**Figure 2.19**  
*Layout of a crystal as per the Motorola 68HC11 reference manual*



## 2.8.1 Values vs baud rate

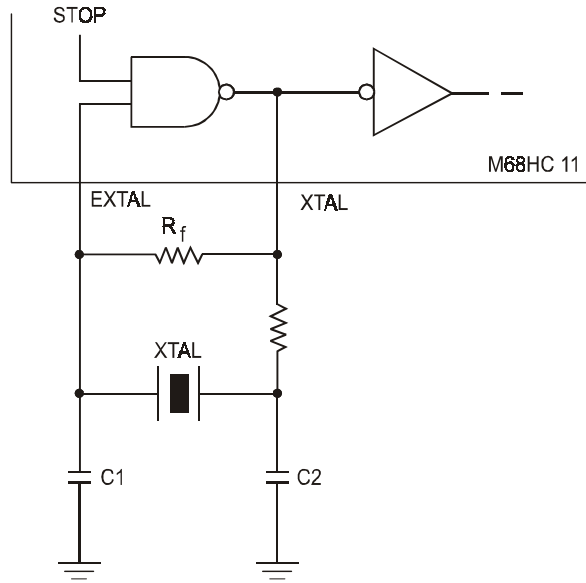
Below are some examples of prescale baud rates and baud rate selection. Prescaling the baud rate determines the highest Baud rate that can be achieved by setting two bits in the baud setup register \$102B. Bits 4 and 5 in the \$102B register define the prescale and bits 0, 1 and 2 in the same register define the actual baud rate. Which prescale is used is a function of the frequency of the crystal.

SCP1	SCP0	SCR2	SCR1	SCR0	Crystal Frequency				
					2 <sup>23</sup> Hz	8 MHz	4.9152 MHz	4 MHz	3.6864 MHz
					Baud Rates				
0	0	0	0	0	<b>131.072K Baud</b>	<b>125.00K Baud</b>	76.80K Baud	62.50K Baud	57.60KBaud
0	0	0	0	1	65.536K Baud	62.50K Baud	38.40K Baud	31.25K Baud	28.80KBaud
0	0	0	1	0	<b>32.768K Baud</b>	31.25K Baud	19.20K Baud	15.625K Baud	14.40KBaud
0	0	0	1	1	16.384K Baud	15.625K Baud	<b>9600 Baud</b>	7812.5 Baud	7200 Baud
0	0	1	0	0	<b>8192 Baud</b>	7812.5 Baud	4800 Baud	3906 Baud	3600 Baud
0	0	1	0	1	4096 Baud	3906 Baud	<b>2400 Baud</b>	1953 Baud	1800 Baud
0	0	1	1	0	2048 Baud	1953 Baud	<b>1200 Baud</b>	977 Baud	900 Baud
0	0	1	1	1	1024 Baud	977 Baud	600 Baud	488 Baud	450 Baud
0	1	0	0	0	43.691K Baud	41.666K Baud	25.60K Baud	20.833K Baud	19.20KBaud
0	1	0	0	1	21.845K Baud	20.833K Baud	12.80K Baud	10.417K Baud	<b>9600 Baud</b>
0	1	0	1	0	10.923K Baud	10.417K Baud	6400 Baud	5208 Baud	4800 Baud
0	1	0	1	1	5461 Baud	5208 Baud	3200 Baud	2604 Baud	<b>2400 Baud</b>
0	1	1	0	0	2731 Baud	2604 Baud	1600 Baud	1302 Baud	<b>1200 Baud</b>
0	1	1	0	1	1365 Baud	1302 Baud	800 Baud	651 Baud	600 Baud
0	1	1	1	0	683 Baud	651 Baud	400 Baud	326 Baud	<b>300 Baud</b>
0	1	1	1	1	341 Baud	326 Baud	200 Baud	163 Baud	150 Baud
1	0	0	0	0	<b>32.768K Baud</b>	31.250K Baud	19.20K Baud	15.625K Baud	14.40KBaud
1	0	0	0	1	16.384K Baud	15.625K Baud	<b>9600 Baud</b>	7812.5 Baud	7200 Baud
1	0	0	1	0	<b>8192 Baud</b>	7812.5 Baud	4800 Baud	3906 Baud	3600 Baud
1	0	0	1	1	4096 Baud	3906 Baud	<b>2400 Baud</b>	1953 Baud	1800 Baud
1	0	1	0	0	2048 Baud	1953 Baud	<b>1200 Baud</b>	977 Baud	900 Baud
1	0	1	0	1	1024 Baud	977 Baud	600 Baud	488 Baud	450 Baud
1	0	1	1	0	512 Baud	488 Baud	<b>300 Baud</b>	244 Baud	225 Baud
1	0	1	1	1	256 Baud	244 Baud	150 Baud	122 Baud	<b>112.5 Baud</b>
1	1	0	0	0	10.082K Baud	<b>9600 (+ 0.16%)</b>	5908 Baud	4800 (+ 0.16%)	4431 Baud
1	1	0	0	1	5041 Baud	4800 Baud	2954 Baud	<b>2400 Baud</b>	2215 Baud
1	1	0	1	0	2521 Baud	<b>2400 Baud</b>	1477 Baud	<b>1200 Baud</b>	1108 Baud
1	1	0	1	1	1260 Baud	<b>1200 Baud</b>	738 Baud	600 Baud	554 Baud
1	1	1	0	0	630 Baud	600 Baud	369 Baud	<b>300 Baud</b>	277 Baud
1	1	1	0	1	315 Baud	<b>300 Baud</b>	185 Baud	150 Baud	138 Baud
1	1	1	1	0	158 Baud	150 Baud	92 Baud	75 Baud	69 Baud
1	1	1	1	1	79 Baud	75 Baud	46 Baud	38 Baud	35 Baud
					2.1 MHz	2 MHz	1.2288 MHz	1 MHz	921.6 kHz
Bus Frequency (E clock)									

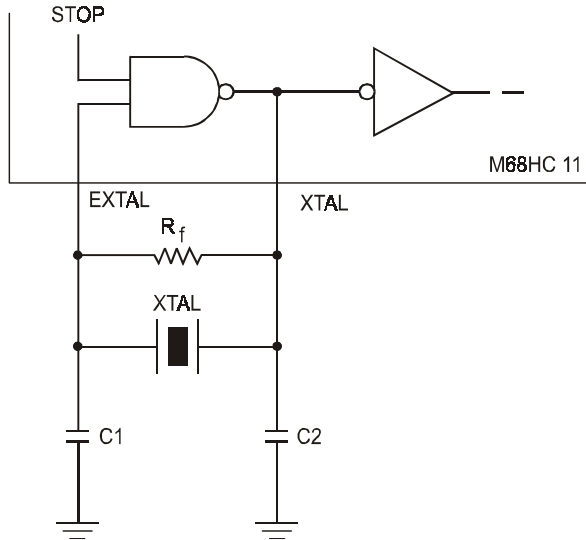
**Table 2.4**  
Baud rates (as per the Motorola 68HC11 reference manual)

## 2.8.2 EMC and PCB crystal clock design

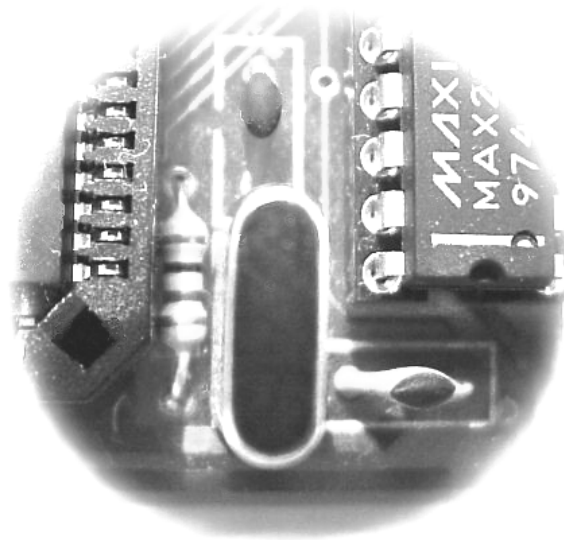
There are very specific design rules laid out by the manufacturer. It is important to check the manufacturer's microcontroller reference manual for the suggested way to connect the crystal on the PCB. As far as the HC11 is concerned the following drawings show the suggested method for connecting the crystal. Notice that the high frequency connection has the resistor closer to the crystal. There are two resistors for the low frequency crystal.



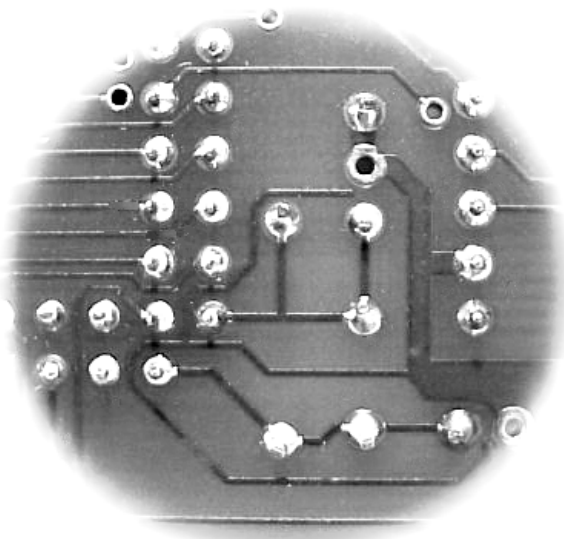
**Figure 2.20**  
*Low frequency crystal connection*



**Figure 2.21**  
*High frequency crystal connection*



**Figure 2.22**  
*Crystal top view*



**Figure 2.23**  
*Crystal bottom view*

## 2.9 Conclusion

This chapter gives an introduction to microcontrollers. Although the reader may never program or design a microcontroller system it is still important to understand the basics of microcontrollers. The starting place for a good awareness of microcontroller systems is with numbers. The microcontroller is, after all in its basic form, a counting device. Even the basic binary ones and zeros have become part of our global culture. Once the

numbering systems of microcontrollers are understood then it is a quick jump to ASCII code. This is the code used by most communications systems to transfer data to and from the microcontroller. ASCII is an English language code that converts hex bytes into letters and numbers that the user or designer can understand. Besides communication it is also used for LCD displays.

The AND, OR, EOR and NOT gates are building blocks for comparing or changing bits in a register. For example the AND gate can be used to mask out bits in a register. These gates can be used in hardware or in software programs to control outputs as defined by the inputs. The result of the gate is often placed in an accumulator in the microcontroller. The accumulators are temporary 8-bit storage places in the microcontroller where data is placed before being used by the program. The 8-bit A and B accumulators are combined to form the 16-bit D accumulator. The instructions LDAA or LDAB are used to place data in these accumulators. The STAA and STAB instructions are often used to store the data in memory locations pointed to by the 16-bit X and Y registers. The 16-bit X and Y registers can hold any data that is needed by the program, but their most common use is pointing to RAM, EEPROM and EPROM memory locations in the microcontroller. These registers are often stored in a first in last out memory location called the stack.

One of the most powerful functions of the microcontroller is its ability to communicate with the outside world. Some microcontrollers have on board communication ports. The HC11 has both a synchronous and asynchronous communication ports. The asynchronous port is often used for the RS-232 or RS-485 voltage standard. Asynchronous has been the most common type of communication in the past, but it is quickly being replaced by synchronous packet systems.

One of the most common problems with microcontrollers is spiking due to high voltage or static. This causes skepticism in the reliability of microcontroller systems. It is therefore very important for the microcontroller designer to do everything possible to make the design as stable as possible and with a high resistance to spiking. Watchdog systems, good software design and reset circuits are a few of the ways to make the microcontroller a more stable platform.

## Microcontroller programming

### Objectives

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

- Describe the basic programming structures of microcontrollers
- Explain flow charts and how they are used in programming microcontrollers
- Describe how to load a program into an EVM
- Describe the different modes of addressing in assembly language
- Describe how data is loaded and stored in the memory of the microcontroller
- Explain how arithmetic functions are done within the program of a microcontroller
- Explain how index registers point to addresses within the program
- Describe how branches and jumps are used to move around the program

### 3.1 Introduction to programming the microcontroller

This book is not intended to be a manual on programming, but when working with microcontrollers it is helpful to understand the basics of how the microcontroller works from a programming point of view. This chapter is concerned with a general overview of programming methods, functions and specific assembly language instructions. At the end of this chapter it briefly mentions BASIC and C++ high level programming languages. But it is not within the scope of this chapter to get into any detail on BASIC, C++ or other high level programming language.

Programming is the method of telling the microcontroller how to handle inputs, manipulate data and control outputs. Programmers write the instructions or code that specifically tells the microcontroller how to function. Even before computers were invented people were wondering if it would be possible to write a program that could program itself. This day is getting closer and closer. With the advent of language recognition and high level languages it is possibly only a matter of a decade or so before this happens. But for the moment we still need to rely on programmers to write the

programs. Most programmers find out very early on in their work that the microcontroller is a very stupid device. It only does what you tell it to do and even then it can get confused. This is because it does what **we tell it to do** and not **what we want it to do**. Often a straightforward subroutine can cause hours if not days of trouble. I have written a bit of code and for no apparent reason it does not work. I then duplicated the code exactly and verify that it is the same and now it works for some reason.

Four of the most common problems in software design are:

- Lack of planning
- The user doesn't know what he/she wants or what can be done
- The programmer doesn't follow the specifications of the user
- Lack of planning

Lack of planning is mentioned twice because it is the most common problem.



**Figure 3.1**  
*Lack of planning*

Once the planning is done and the code written, the program is tested. This testing is done on a microcontroller evaluation module. The program is loaded into the module and the program is run. If it works the way the programmer wants, the programmer continues to write more code. The programmer will do this over and over until the program is finished. Often programming is compared with building a house. The builder puts up a board, test it to see if it fits and if it does goes on to the next board. And just like a house the builder should start with extensive planning. Only the foolish builder would assume what the user wants and starts building without having detailed plans. As they say, planning and preparation are everything. A lot of programmers believe it or not, just start coding and then wonder why the project gets bogged down.

The traditional method of program planning is done using the flow chart. One of the differences between the university-trained programmer and the self-trained programmer is that the universities spend time (although not enough in my opinion) teaching planning and structure. There are good and cheap programs available to help the programmer create flow charts. Often when I am having a problem coding a subroutine I find it helps

to get out the flow chart program and do a detailed analysis of the subroutine. This has never failed to help me find the solution.

## 3.2 Programming structure and specifications

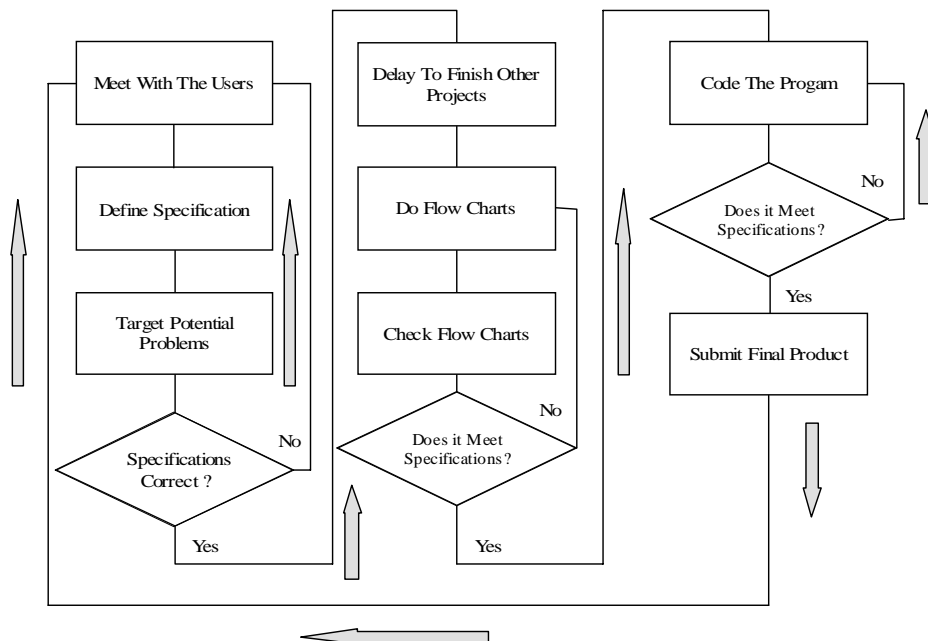
When starting a programming project the programmer will often sit down with the user and work out the specific functions of the project. This sounds easy but it is often difficult.

There are three problems that can cause friction between the user and the programmer.

- Lack of knowledge of embedded systems by the user
- The language barrier between programmer and user
- The artistic nature of programming

Very often the user does not know what the electronic device or program is capable of doing. Once the user gets a taste of what the program and device can do, they often want more and more functions. The user may start off with a simple project, thinking it is going to be hard, and the programmer will tell the user that it is going to be easy. The user will then decide to add more features. This leads to the never-ending project.

Programming can be compared to painting a picture. The user starts with the idea of a simple portrait and soon the painting is of the whole family. To reduce this problem it is best to have a third person or group that can interface between the user and the programmer. This person or group can collate the specifications as defined by the user and discusses them with the programmer. The end result should be that the supplier gives the user what is specified, no more or less.



**Figure 3.2**  
*Planning the project*

The programmers often live in a world of their own. The language they speak is programming language. What is simple for the programmer is often seen as hard to do by

the user and what is hard for the programmer to do is often seen as easy by the user. One of the most difficult questions to answer for the programmer is how long is it going to take to write the program. Adding more programmers to a problem or program doesn't mean it will get done quicker. Adding programmers to a project usually means it is going to take longer. When the programmer starts using words like, subroutine, non-maskable interrupts and vectors most users give up. The interface person or group is useful, when it is necessary to explain to the user what the programmer is saying.

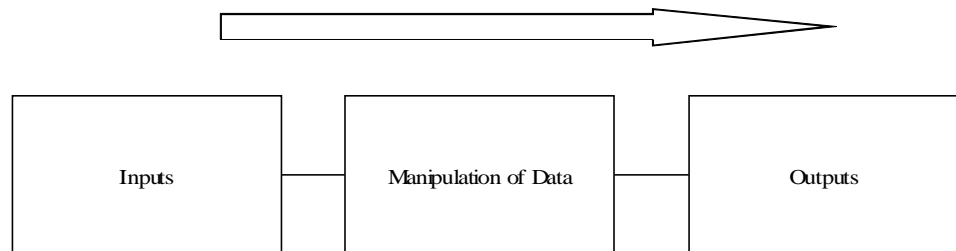
The nature of programming is such that every programmer approaches a program differently. There is no correct way to write a program. One programmer may take two weeks to write a 50-line subroutine while another may take one week to write a 200-line subroutine that does exactly the same thing. Which is better is hard to say. On the surface the 200-line subroutine looks better because it is cheaper, but the short program might fit in the microcontroller's RAM whereas the 200 line would not. This means more hardware, which is costly in the long run. The artistic nature of programming should be counter balanced by good planning. The interface manager or group should know enough about programming to help the programmer plan the project.

### 3.2.1 Programming structures

After the specifications and preliminary planning are complete, the next step is to develop the flow charts. Strangely some electronic designers will spend days or weeks designing the schematic for a circuit, but when it comes to writing a program the programmer will often skip the flow-charting step. In its simplest form, a program is a collection of subroutines. Each of these subroutines can be written as a flow chart. The subroutines can often be divided into smaller subroutines. Before writing any code, the program should be developed using flow charts.

There are three basic parts to any program:

- Inputs
- Manipulation of data
- Outputs



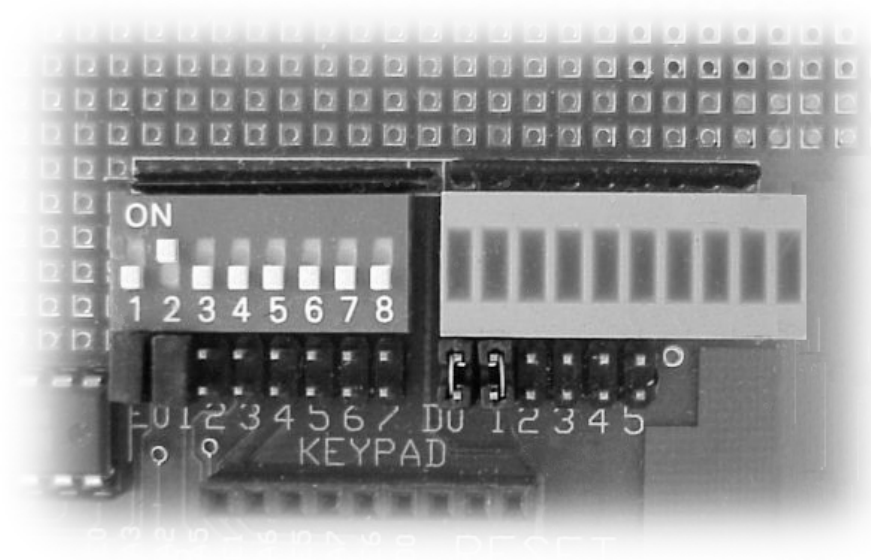
**Figure 3.3**  
*Three basic parts of a program*

### 3.2.2 Inputs

The inputs can be digital inputs or analog inputs. Digital inputs could take the form of keypad switches, magnetic switches, toggle switches or even motion sensors. Analog inputs could include DC voltage, mains current, or resistance measurement. These inputs must be dealt with in some manner. What kind of inputs and how often they are going to be read, and the priority must be specified during the planning phase. The inputs can be categorized in the following way...



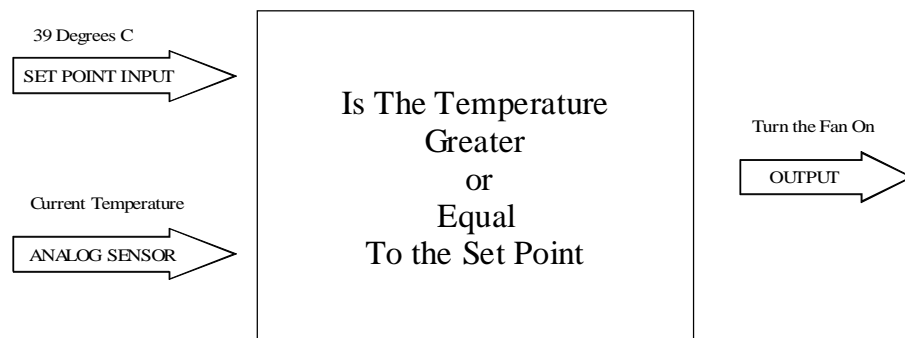
High priority	Read often and may need to interrupt the microcontroller
Fast acting	Read quickly (as in samples per second)
Electronic switches	Need power either mains or battery
Mechanical switches	Need debounce and are normally open or closed
Analog inputs	Must be sampled at some rate, level and resolution



**Figure 3.4**  
*EVM inputs and outputs*

### 3.2.3 Manipulation of data

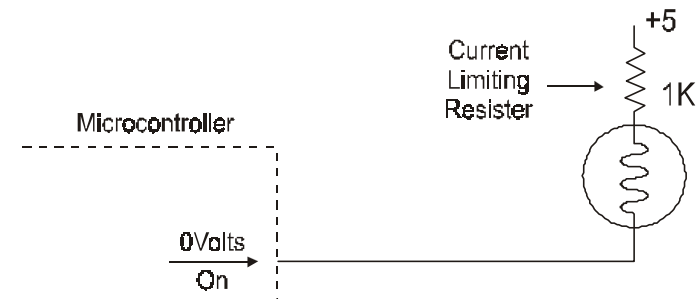
The manipulation of the data can take on many forms. It can be a menu system defined by the keypad input or it could be the timing calculation of multiple switch closures. The timing of events is so common that many microcontrollers, including the HC11 have timers on board. Analog inputs are sampled and the values used to develop responses. These analog inputs are also used in conjunction with digital inputs to formulate outputs.



**Figure 3.5**  
*Manipulation of the data*

### 3.2.4 Outputs

The outputs from microcontrollers are usually digital outputs. These in turn drive electromagnetic or solid state relays. The relays are then used to turn on motors, lights or any electrical device. The output from the microcontroller is a 1 or a 0 in the form of a voltage. Either one of these states can indicate an 'on' situation. The binary 1 is usually represented by +5 volts, whereas a binary 0 is 0 volts. In most systems a 0 indicates that the device is on. This is because TTL (transistor–transistor logic) uses less power when supplying 0 volts as an output. A typical circuit would supply a constant + 5 volt to a device that is to be controlled. Then when the device is to be turned on, the program tells the output driver to supply 0 volts to the device. The device would then turn on.

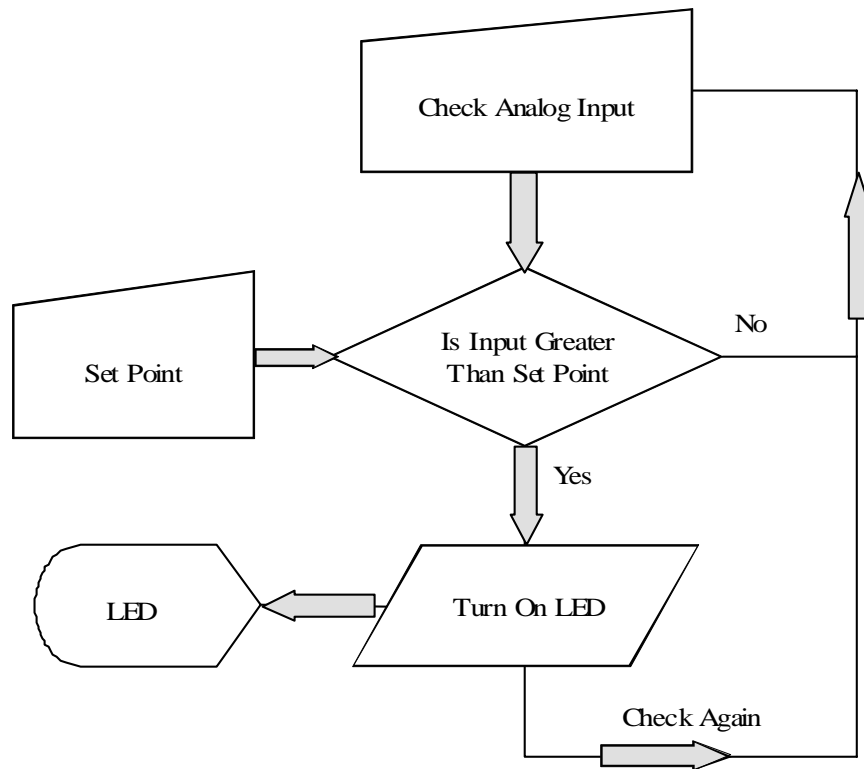


**Figure 3.6**  
*Digital outputs*

Once the inputs, manipulation of data and outputs have been specified, the programmer can then go on to the next part of the preparation phase which is flow charting the program.

### 3.2.4 Flow charts

Flow charts are used to give the programmer an easy way of seeing the overall program while also seeing the more minute detail of every subroutine. A program is a group of major subroutines. Each of these subroutines is then made up of more detailed subroutines. Once the specifications for the inputs, data manipulation and outputs are done the overall program subroutines are charted. Then each major subroutine is broken down into small subroutines and these are charted. These sub-subroutines should be charted completely, down to the last branch.

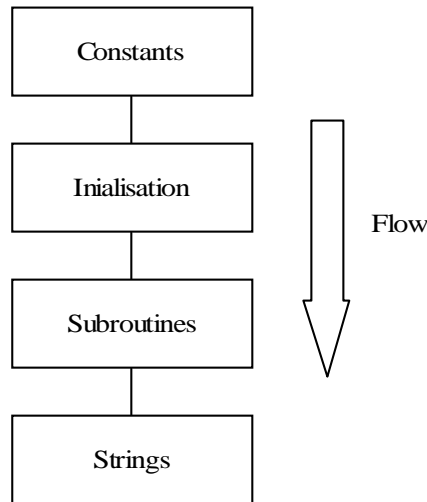


**Figure 3.7**  
Flow chart example

**NOTE:** One major mistake programmers make is that they often use the data from one subroutine directly into another. An example of this could be... The result of one subroutine could be placed in the index register X. Then another subroutine could use the value in the index register X as a variable. It is better to take the result and place it in a memory location, then when it is needed it can be retrieved from there. All subroutines should be completely independent and stand-alone. No subroutine should directly depend on the output of another.

The overall structure of a program is usually done in this manner:

- Constants
- Initialization
- Subroutines
- Strings



**Figure 3.8**  
*Program flow*

The constants are a list of names of memory locations, register locations and subroutines. This could include RAM, EEPROM and microcontroller registers. An example of this could be...

```

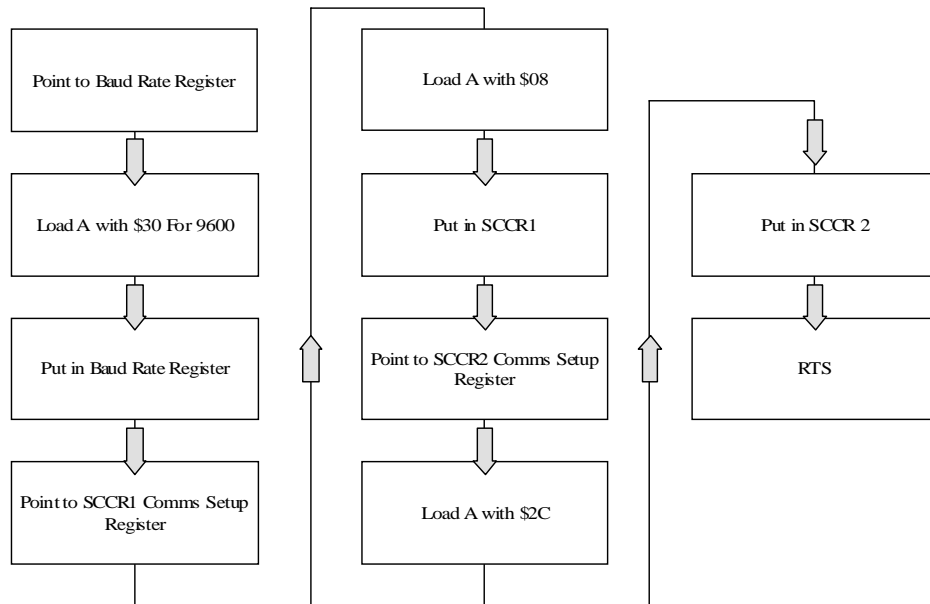
ORG    $C000          Starting Address Of Program in External RAM
LDS    #50            Initialize System Stack in Internal RAM
*****
KEY_B   EQU    $1000   Key Press (Port A)
PACTL  EQU    $26     Pulse Accumulators Control Register Offset
TCTL1   EQU    $20     Timer Control Register
PORTD   EQU    $1008   Port D Address
DDRD    EQU    $1009   Data Direction Control Register
SPCR    EQU    $1028   SPI Control Register
BAUDR_B EQU    $102B   Baud Rate Setup Register
SCCR1   EQU    $102C   SCI Control Register 1
  
```

In the previous example the address location of \$1000 is named KEY\_B and the program can at any time go to address \$1000 by invoking the name KEY\_B.

Next is the program could initialize the inputs and outputs. Next is an example of communication port initialization.

```

COMSET  LDX      #BAUDR_B  Point To Baud Rate Setup Register
        LDAA     #$30      Load For 9600 Bps
        STAA     $00,X     Store $30 in Setup Register
        LDX      #SCCR1    Points To SCCR Comms Setup
                               Register 1
        LDAA     #$08      Load 08 in Accumulator A
        STAA     $00,X     Store 08 In Comms Setup Register 1
        LDX      #SCCR2    Point To TX Set Register 2
        LDAA     #$2C      Load 2C in Accumulator A
        STA      $00,X     Store It In Comms Setup Register 2
        RTS
  
```

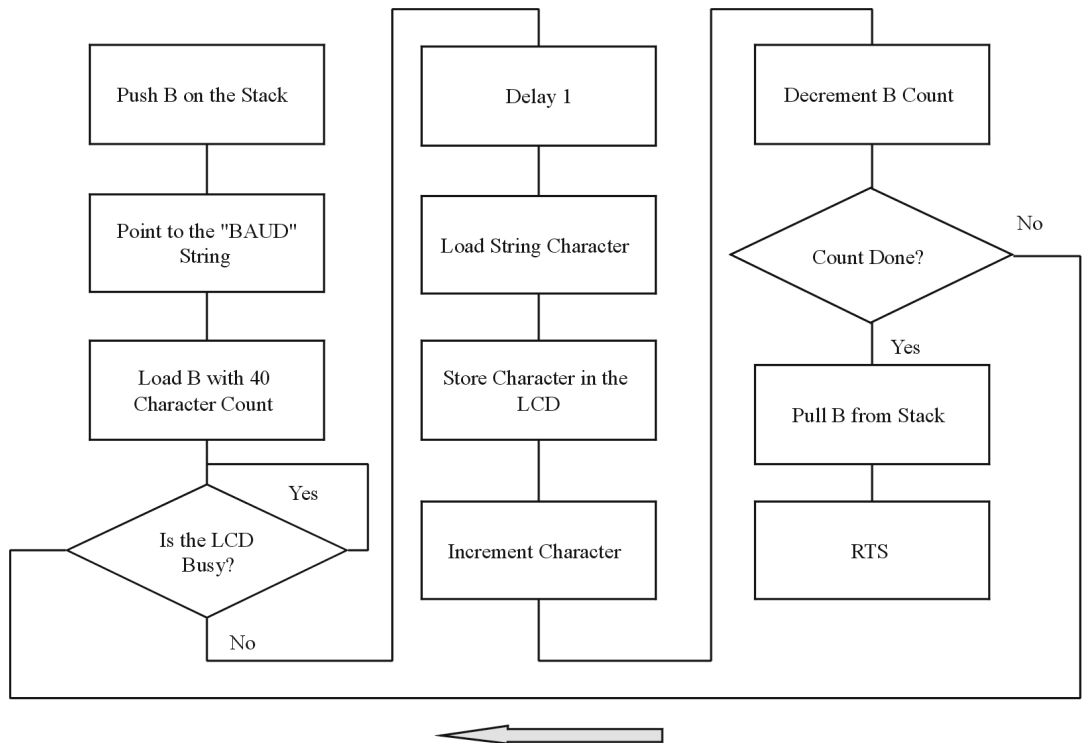


**Figure 3.9**  
Communication initialization flow chart

Once the inputs and outputs are initialized the programmer will then write the code for the subroutines. Below is an example of a display subroutine.

```

*****
*           Display Subroutine
*****
DISPLAY    PSHB           Push Accumulator B Onto The Stack
           LDX  #STRING1  '   BAUD   '
           LDAB #$28      Load B With 40 Count
LOOP       JSR  LCDBUSY   Check If The LCD is Busy
           JSR  DELAY     LCD   Delay (Slows the Change Down)
           LDAA $00,X     Loads First Byte From Mess
           STAA $8001     Display Character On The LCD
           INX           Increment Character
           DECB         Decrement B (40 Count)
           BNE  LOOP     If Not 40 Characters Then Loop
           PULB         Pull Accumulator B From the Stack
           RTS          Return To Subroutine
  
```



**Figure 3.10**  
Display flow chart

The last section of the program is the string storage area. This is where the strings are held. When the program wants to send a standard string, it goes down to this area and picks up the string. It does this by pointing to the name of the string and then incrementing the characters one at a time. An example of this is...

```

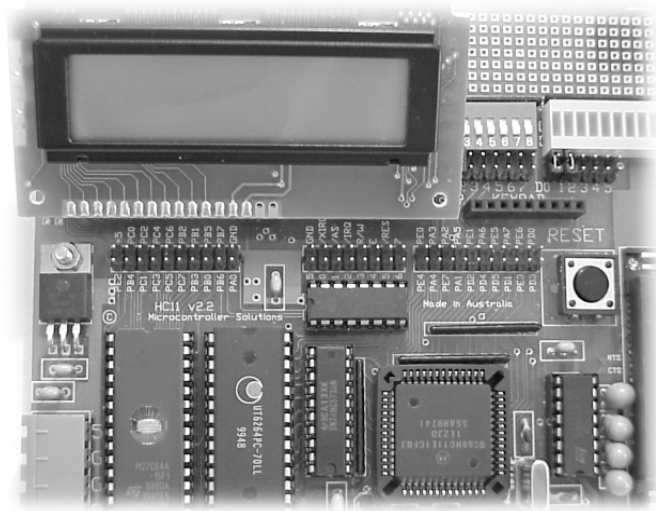
*****
*           Strings To Be Sent
*****
STRING1      FCC      '      BAUD          '
STRING2      FCC      '      DISPLAY       '
STRING3      FCC      '      LIVE LIST HERE '
STRING4      FCC      '      <<-S/N        '
STRING5      FCC      '      <<-MODEL       '
STRING6      FCC      '      NEXT ADDRESS  '
STRING7      FCC      '      << YES NO >>  '
  
```

In this example each of the strings are enclosed between ' '. Each string, including spaces, is 16 characters long. This is because the display only can show 16 characters in one line at a time. Notice that the display subroutine has a count of 40. This is because the LCD display RAM holds 40 characters for each line, but only displays the first 16 of them. The programmer may then want to load the second line with another forty characters.

### 3.2.6 Loading a program into an evaluation module (EVM)

There are many different types and makes of microcontroller evaluation modules, although the purpose is basically the same. The evaluation modules are used to test the

program on a real microcontroller. Often an EPROM with a BUFFALO (c) program is used to help in talking to the microcontroller. This BUFFALO program has data communications and I/O functions that let the programmer easily access the microcontroller.



**Figure 3.11**  
*EVM board*

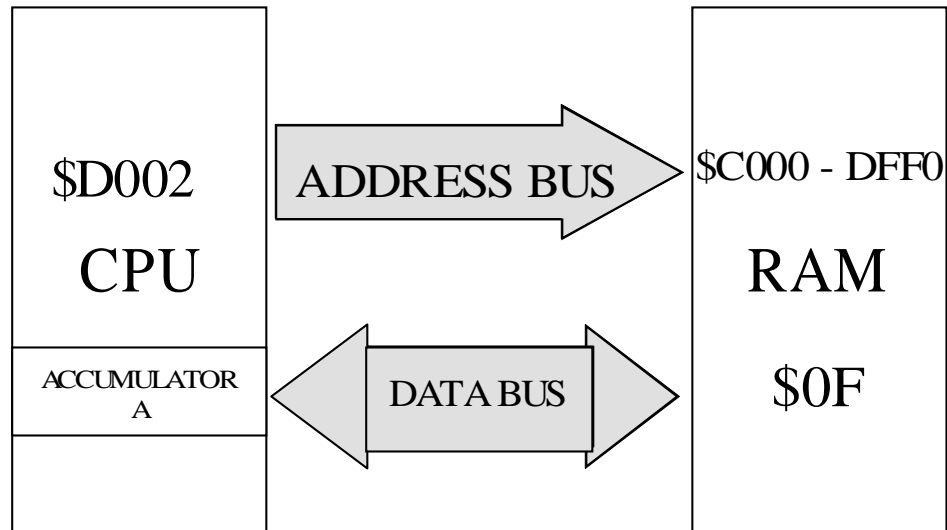
### 3.2.7 Stetting up the EVM

- Verify that the display and keyboard are installed correctly
- Plug in the power supply to the board
- Plug in the RS-232 cable from the computer
- Apply power
- Verify that the power LED illuminates
- Run the ASM11 program
- Click on the DOS icon to bring up ProComm
- Push the RESET button on the EVM
- Notice that the BUFFALO displays a line on the screen
- Press Enter on the computer keyboard
- Type MD 0000 (this is a test)
- The EVM will send the data to the computer screen
- Type LOAD T then enter
- Press the Page Up key and then the 7 key
- Type ASM11 program.asm then enter
- When the program has been loaded type G C000
- The program should now run

## 3.3 Addressing modes

Addressing modes often confuse the new programmer. To this end the following section will endeavor to make addressing modes as clear as possible. There are six different ways the CPU in the microcontroller can handle an address. It is important at this point to remember how the microcontroller gets information. When the CPU wants some data it

places the address where that data is located on the address bus. In the HC11 this is a 16-bit address. A typical address could be 1101 0000 0000 0010, that is \$D002. Once this address is placed on the address bus, the device that is located at the address \$1002 places the data in that address (such as \$0F) on the 8-bit data bus. The CPU then reads the data bus and puts the data wherever the instruction says. An example of this could be...



**Figure 3.12**  
*Loading data from an address in RAM*

LDAA                                  \$D002

This loads the data in the RAM at address \$D002 into Accumulator A

There are six ways the CPU can handle the addressing...

Immediate                          LDAA #MESSAGE1

Loads the 16 bit address defined at Message1 following the instruction LDAA

Extended                          LDAA \$D002

Loads the 16-bit address following the instruction LDAA as per Figure 3.12.

Direct                                LDAA \$02

Loads the low byte of the 16-bit address that is located at \$00xx. All addresses here start with \$00.

Indexed                            LDAA \$02,X

Loads the address relative to the index register. The above example shows an address of X plus 2. If X = \$1004 then the address would be \$1006 and \$1007 (16-bit address)

Inherent                            INCA

Load the address that is inherent to that instruction. Some other instructions that use inherent addressing are INCB, INX and INY.

Relative                            RTS

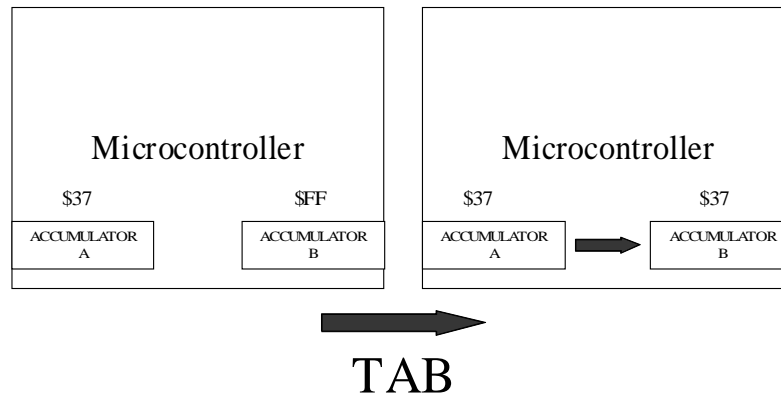
Loads the address that is relative to that specific instruction.

### 3.4 Load, stores and transfers

Instructions like LDAA, LDAB, STAA, STAB, PSHX, PSHY, TAB, TAP and TPA, PSHA and PULA are considered load, store and transfer instructions. These instructions load data from somewhere, store data somewhere or transfer data somewhere in the



microcontroller. Usually the data is in the form of 8-bit data but it can be 16-bit such as in the instruction PSHX. An example of this would be a microcontroller pushing an address (16 bits) pointed to by the X register on to the stack. The address would then be pulled from the stack later and used somewhere else in the program. The transfer instructions transfer the data (8-bit) from one accumulator to another. The instruction TAB transfers the data in accumulator A to accumulator B. The data in accumulator B is lost but the data in accumulator A doesn't change.



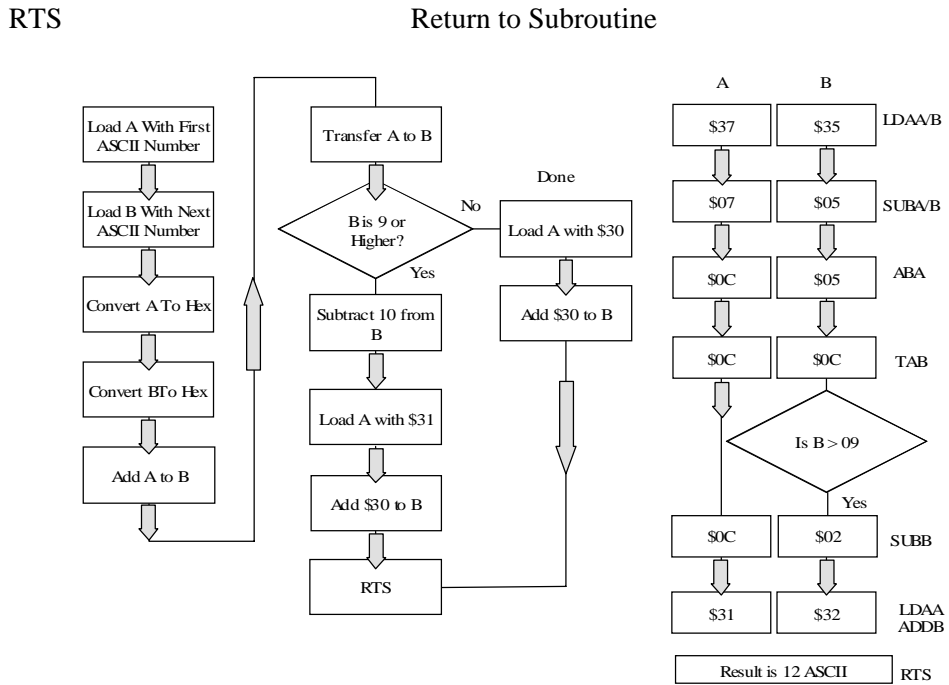
**Figure 3.13**  
*Transferring accumulator A to accumulator B*

### 3.5 Arithmetic operations

The arithmetic functions are used to do mathematics within the microcontroller. This mathematics is usually done in either binary coded decimal or hex. It is not very easy to add, subtract, multiply or divide in ASCII. Adding \$37 (7) and \$35 (5) just doesn't work. So how would we add two ASCII characters?

To add them, they first need to be converted to hex by subtracting \$30 from both characters then adding the \$07 and 05. The sum would then be \$0C hex. Converting the result back to ASCII would be more difficult. Since the sum of two characters is never higher than 18 decimal (\$39 + \$39) it is easy to subtract \$0A (10 in decimal) from \$0C and then convert the result back into ASCII by adding \$30 to the result. A program example of this would be...

ADD	LDAA	#\$37	Loads Accumulator A with \$37 (7)
	LDAB	#\$35	Loads Accumulator B with \$37 (7)
	SUBA	#\$30	Subtract \$30 from \$37 (\$07)
	SUBB	#\$30	Subtract \$30 from \$35 (\$05)
	ABA		Add B to A (\$0C)
	TAB		Transfers A to B
	CMPB	#\$09	Check if B higher than 9 decimal
	BLS	DONE	If 9 or lower then go to DONE
SUBB	#\$0A		IF B is 10 or higher subtract 10 decimal
	LDAA	#\$31	Loads Accumulator A with \$31 (1)
	ADDB	#\$30	Add \$30 to B
	RTS		Return to Subroutine
DONE	LDAA	#\$30	Loads Accumulator with \$30 (0)
ADDB	#\$30		Add \$30 to B



**Figure 3.14**  
Flow chart to add two numbers

This is by no means the only, or even the best way of adding two ASCII numbers. Some of the other instructions used in arithmetic functions are...

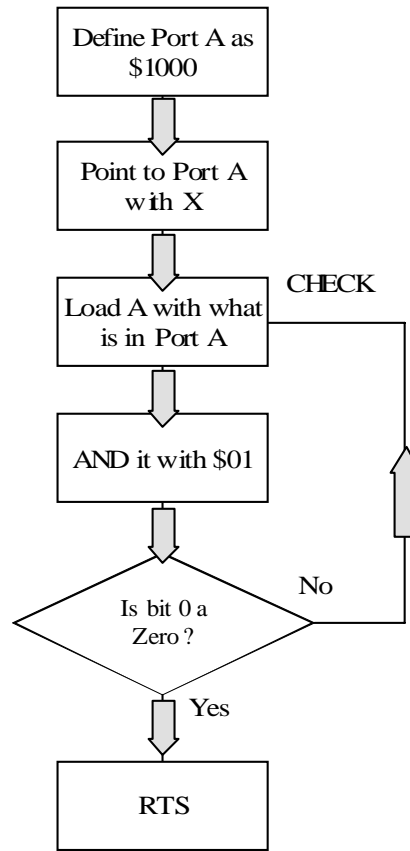
- ABA, ABX and ABY
- ADDA, ADDB and ADDD
- CMPA, CMPB and CPD
- DECA, DECB and DEC
- INCA, INCB and INC
- SUBA, SUBB and SUBD
- TSTA, TSTB and TST

### 3.6 Logical operations

The logical functions used in programming are used to check if certain bits are present after an operation. If the programmer wanted to check the condition of a certain digital input the bit that corresponds to that digital input would be checked. There are many ways of doing this and the following example is just one.

```

PORTA EQU      $1000      Define port A
*****
LDX            PORTA      Point to Port A
CHECK LDAA     $00,X      Load A with bits in Port A
AND            #$01       Mask out BIT 0
CMPA          #$00       Is bit 0 a 0
BNE           CHECK      If bit is 0 then go to CHECK
RTS            RTS        If not Return to program
    
```



**Figure 3.15**  
Logical AND flow chart

In the above example the highest 7 bits are AND with 0. This means the output of this operation will always be 0000 000X with X being the state of bit 0 of Port A. The original contents of accumulator A are replaced with the new value. Again this is by no means the only or even the best way of checking the bits of a port.

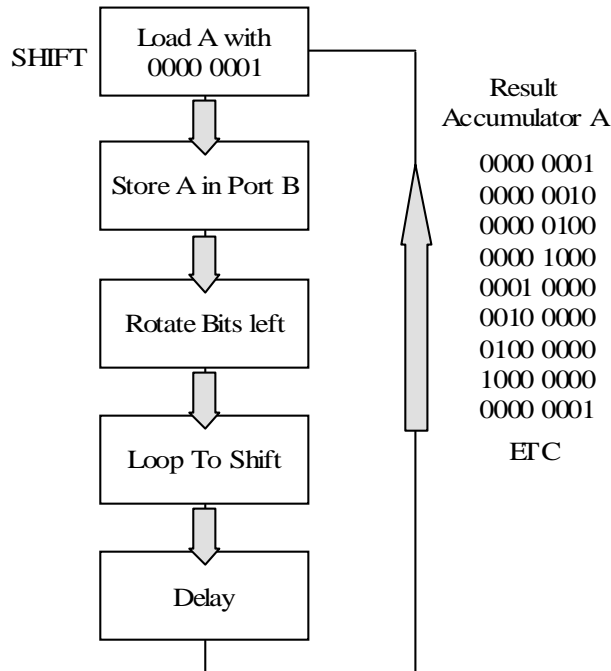
Some of the instructions used in logical functions are:

- ANDA and ANDB
- BITA and BITB
- COMA, COMB and COM
- EORA and EORB
- ORAA and ORAB

## 3.7 Shifts and rotates

Shifting and rotating bits are useful functions for bit manipulation. It could be used to sequence bits in a stepper motor or lights. An example of this function is..

SHIFT	LDAA	#\$01	Load A with 0000 0001
	STAA	\$1004	Store 0000 0001 in Port B
	ROLA		Rotate the bit left one position
	JSR	DELAY	Jump to Delay
	BRA	#SHIFT	Loop to SHIFT



**Figure 3.16**  
*Rotating bits flow chart*

If the output of port B went to a line of LEDs then the LEDs would sequence one at a time. Again this is by no means the only way of rotating one bit of a port.

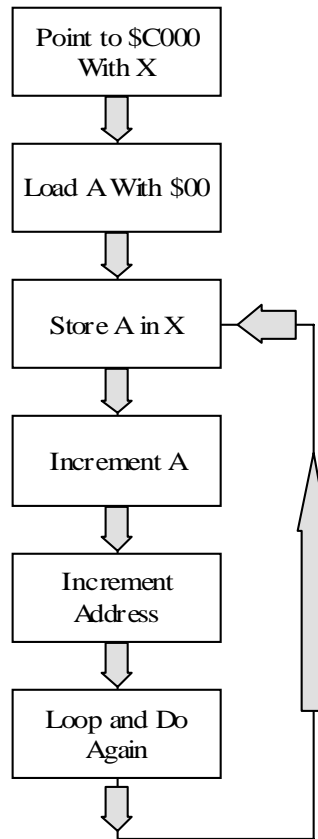
Some of the instructions used in shift and rotate functions are:

- ASLA, ASLB, ASLD and ASL
- ASRA, ASRB, ASRD and ASR
- LSRA, LSRB, LSRD and LSR
- ROLA, ROLB and ROL
- RORA, RORB and ROR

### 3.8 Index registers and the stack

Index registers and the stack are 16-bit registers that often hold pointers for programs. A pointer is an address that is used or pointed to by the program. If the programmer wishes to use an address over and over, that address might be placed in an index register (X or Y). The program could then use that address later in the program. This function is often used when the address needs to be incremented or decremented as shown in the following example.

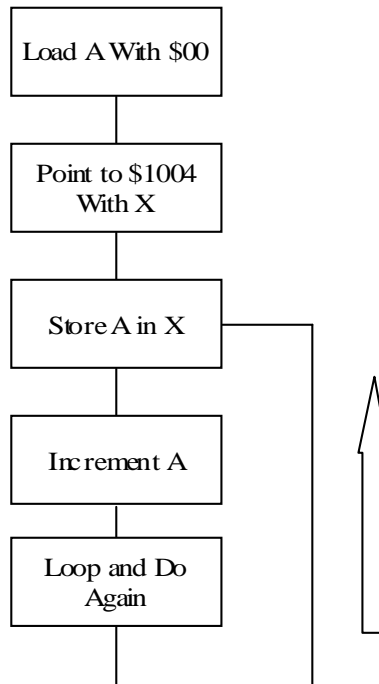
	LDX	\$C000	Loads X with \$C000
	LDAA	#\$00	Loads A with \$00
INDEX	STAA	\$00,X	Stores \$00 in \$C000
	INCA		Adds one to A (\$01)
	INX		Increments the pointer (\$C001)
	BRA	#INDEX	Loops and stores the incremented A in X



**Figure 3.17**  
*Flow chart showing incrementing the address*

Notice that the loop starts after the X register has been initialized and accumulator A has been loaded. If the label INDEX was placed before the LDX instruction then the same data would be put in the same address every time. If the INDEX label was placed before the LDAA instruction then the value \$00 would be placed in the incremented address pointed to by the X register. To increase the value of one address LDX and LDAA could be reversed and the INDEX label placed in front of the LDX. The instruction INX would not be needed as shown in figure 3.18.

	LDAA	#\$00	Loads A with \$00
INDEX	LDX	\$1004	Loads X with \$1004
	STAA	\$00,X	Stores \$00 in \$C000
	INCA		Adds one to A (\$01)
	BRA	#INDEX	Loop to Index



**Figure 3.18**  
*Flow chart showing incrementing the count in one address*

In the above example the output of port B would count up in a hexadecimal format forever. Again this is by no means the only way of incrementing the value in a port.

The stack is a user-defined RAM location that is used by the program to hold data and addresses to be used later on in the program. The stack is a first-in/last-out temporary memory block. The stack is usually specified at the beginning of the program and the address defined is the bottom of the stack. The stack is often compared to a pile of plates in a cupboard. The plate on the bottom of the pile is the first plate put in the cupboard.

Often data or an address is placed on the stack to free up the A or B accumulators or the X or Y registers in a subroutine. The subroutine usually pushes a value on the stack and then later on pulls the value off the stack. It is possible to use values on the stack by using an offset instruction. This pulls the value off the stack in the same way one might pull a plate from the middle of a pile of plates. Some programming books suggest that the programmer push A, B, X and Y on the stack at the beginning of every subroutine. And of course the A, B, X, and Y registers will need to be pulled from the stack at the end of every subroutine. This practice is rarely done, as it is a waste of memory and instructions in the program. It is only necessary if the programmer shares data between subroutines. Often a program will stop after a few loops because the stack gets full. This happens because something is being pushed on the stack, but was never pulled off. When the stack gets full the program stops.

Some of the instructions used in index register and stack functions are:

- ABX and ABY
- CPX and CPY
- DEX, DEY and DES
- INX, INY and INS
- LDX, LDY and LDS

- PSHX, PSHY and PULX, PULY
- STX, STY and STS
- TSX, TSY and TXS, TYS
- XGDX and XGDY

### 3.9 Condition code register

It is said that the difference between the amateur and professional assembly language programmer is that the professional programmer pushes the use of the condition code register to its limits. The 8-bit condition code register is a function of the results of implementing instructions. Whenever an instruction is executed the condition code register is usually affected. For example when the instruction **CMPA** is executed, the bits in the condition code are changed as defined below.

- N** Set if the most significant bit of the result of the compare is set (1), it is cleared otherwise (0).
- Z** Set if the result of the compare is \$00, otherwise it is cleared.
- V** Set if the result of the compare is a 2's compliment overflow, otherwise cleared.
- C** Set if the most significant bit of the result of the compare was a borrow, otherwise cleared.

The condition code can be somewhat manipulated manually by the programmer using instructions like:

- CLC, CLI and CLV
- SEC, SEI and SEV
- TAP and TPA

### 3.10 Branches, jumps, interrupts and calls

Branches, jumps, interrupts and calls are used by the programmer to move around within the program. This is very useful and a very powerful part of coding a program. The branches and jumps can be divided into two basic types, conditional and unconditional jumps. Conditional branches and jumps depend on the result of some instruction or condition of the condition code register. If the results of an instruction are higher, lower, equal or not equal to the defined value, the instruction branches the program to some other location. Unconditional branches or jumps are used when it is important to move to another part of the program, no matter what. The most common unconditional jumps are the instructions **JMP** and **RTS**. These instructions are used to jump to a subroutine and then return when the subroutine is finished. If the program is written properly, every subroutine will finish with an **RTS** instruction. One of the advantages of the **RTS** over the **BRA** instruction is that the **RTS** addressing mode is relative. This means that the programmer does not have to keep up with where the subroutine needs to return after a **JMP**.

Some of the jump, branch and call instructions are:

- BCC and BCS
- BEQ and BNE
- BHI and BLO
- BMI and BPL
- BVC and BVS
- BRCLR and BRSET

- BRN, BRA and JMP
- JSR, BSR and RTS
- RTI, SWI and WAI
- NOP, STOP and TEST

## 3.11 BASIC and C++

Often the experienced programmer will use the same subroutines over and over and will even keep a list of subroutines like initialization, display and print to be used later. Using basic, Pascal and C++ is similar to having a set of pre-written subroutines. The good part of using these higher level programs is that it makes the programming faster and easier. C++ is especially good because it allows the programmer to write the program using a high level language and also manipulate the microcontroller at the bit level. The problem with using a high level program to program the microcontroller is the large memory that these languages need and the lack of control that assembly language allows.

### 3.11.1 BASIC

BASIC takes many forms, from the old GW BASIC to Visual BASIC. BASIC is built around simple and obvious commands like IF, Then and Goto. If the basic programmer wants to do something based on something else then the statement below might be used.

```
10 If X = 31 then goto 50
```

This says that, if the value of X is 31, then jump to line number 50. Again the programmer uses a number of basic subroutines to develop the program. Once this has been done the programmer uses an interpreter such as basic11.arc to convert the basic program into a machine level program. The machine level program is then loaded into the microcontroller similar to the way an assembly program is loaded.

### 3.11.2 Using C++ in embedded programming

One of the situations with assembly languages is that they are not portable. This means that the assembly program from one family of chips will not work directly on the family of another. C++ high level programming language is portable. A C++ program written for a Motorola chip should be convertible to an AMD chip. Unfortunately the biggest problem with using C++ is the size of the program. Often a C++ program will be 4 to 5 times larger than an assembly language program.

C++ cross compilers are used to convert the C++ program into BIN files. These binary files are then linked together to make a complete program.

## 3.12 Conclusion

Programming is becoming easier and easier but also very memory hungry. As microcontrollers become more powerful and have larger amounts of memory on board, programming will become easier and simpler. For the moment the most efficient way is using assembly language programming. This requires the programmer to have exhaustive knowledge of both programming and the hardware sides of the device.

As we have seen, assembly language instructions are very versatile and allow the programmer to tweak the program in very small ways. Instructions like LDAA and BRA are easy to understand and use. Often these instructions are paired and compliment each other. A good example of this is the JMP and RTS instructions. The JMP jumps to a subroutine and then RTS returns from that subroutine. The results of an instruction can



affect the condition code register and conversely the values in the condition code register affect the results of other instructions.

It is true that most programmers use the higher programming languages of BASIC or C++ to program microcontrollers. Most programmers find these higher languages easier and more straightforward. They are also more convertible. This means that if for some reason the microcontroller chip needs to be changed to another type, the program can be easily converted to the other chip.

---

# 4

---

## Microcontroller memory

### Objectives

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

- Describe the different types of memory used in microcontrollers
- Explain the function of RAM and how it is used in microcontrollers
- Explain the memory map of the HC11 microcontroller
- Describe the BUFFALO and its purpose in software development
- Describe vectors, interrupts and pseudo-vectors
- Explain the control registers used in the HC11 microcontroller
- Explain how to store data in an EEPROM

### 4.1 Introduction to memory

There are many types of memory used in microcontrollers, RAM, ROM, EPROM and EEPROM. Each type of memory has its uses and may be included in the system as internal or external to the microcontroller. It is not unusual to see internal and external memory used in a microcontroller at the same time. The memory included in the microcontroller is often limited and therefore it is often necessary to use both internal and external memory. Also sometimes the size of the program determines that external memory must be used. Although assembly language is very compact, subroutines such as error checking and data base management take up so much room that an external RAM chip may be required. EEPROM space is also limited on a microcontroller and therefore external EEPROM is often used.



**Figure 4.1**  
*Ram chip with battery attached*

The following sections define the different types of memory and when they should be used. Often different types of memory lend themselves to different uses. This is often very subjective and up to the programmer and hardware engineer to determine what type of memory to use and when. As we saw in Chapter 3, assembly instructions are used to move around within memory locations in the microcontroller system. Another method of accessing memory locations is vectoring. Vectoring is a way of jumping or branching from one location in memory to another by way of an interrupt. There are two types of interrupts, hardware or software. Often the vectors are held in RAM but can be placed just about anywhere in the memory map.

## 4.2 User RAM

Random access memory (RAM) is used in a microcontroller to temporarily hold data that is used in the program. The RAM can be either volatile or non-volatile. Volatile RAM is RAM that empties when the power is turned off. Non-volatile RAM keeps its memory even when the power is off. This is usually accomplished by way of battery back-up. The battery could be located on the PCB or on top of the RAM chip itself. Just because non-volatile RAM holds its memory when the power is off does not mean that the memory will never disappear. A common mistake that is made by hardware and software engineers is thinking that the RAM memory will last forever. Often either outside static, high voltage or the program itself can corrupt the RAM. This can happen by doing something as simple as turning the power to the device on and off. The program can cause the RAM to become corrupted by writing over the data area of the RAM. This can happen if the microcontroller becomes spiked due to high voltage spikes or power surges.

In a microcontroller system, there are three possible RAM locations:

- In the microcontroller itself
- In external chips
- In external devices like LCD displays

### 4.2.1 Microcontroller internal RAM

The RAM on the microcontroller is usually very limited. Often this RAM will be measured in bytes instead of kbytes or Mbytes. The 68HC11 microcontroller has between 192 bytes and 1 Kbytes of RAM with an average of about 512 bytes. Initially the RAM is located from \$0000 to \$01FF but it can be relocated anywhere in the 64K area of the microcontroller. This is done by writing to the INIT register. This must be done within the first 64 cycles of the microcontroller after reset. The location of the relocated RAM must be at a 4K page within the 64K memory, although most programmers leave the microcontroller RAM at its default. This puts the RAM at \$0000 to \$01FF with the internal registers at \$1000 to \$103F. After 64 cycles of the clock after reset the INIT register is write-protected and cannot be changed. Even 512 bytes are not enough RAM for most programs. The programmer can expand the RAM using external RAM in the expanded mode. The expanded mode allows the programmer to boot off of an external chip like an EPROM.

Part Number	EPROM	ROM	EEPROM	RAM	CONFIG'	Comments
1C68HC11A8	-	-	512	256	\$0F	Family Built Around This Device
1C68HC11A1	-	-	512	256	\$0D	'A8 with ROM Disabled
1C68HC11A0	-	-	-	256	\$0C	'A8 with ROM and EEPROM Disabled
1C68HC811A8	-	-	8K+512	256	\$0F	EEPROM Emulator for 'A8
1C68HC11E9	-	12K	512	512	\$0F	Four Input Capture/Bigger RAM 12K ROM
1C68HC11E1	-	-	512	512	\$0D	'E9 with ROM Disabled
1C68HC11E0	-	-	-	512	\$0C	'E9 with ROM EEPROM Disabled
1C68HC811E2	-	-	2KF	256	\$FF	No ROM Part for Expanded Systems
1C68HC711E9	12K	-	512	512	\$0F	One-Time Programmable Version of 'E9
1C68HC11D3	-	4K	-	192	N/A	Low-Cost 40-Pin Version
1C68HC711E9	4K	-	-	192	N/A	One-Time Programmable Version of 'D3
1C68HC11F1	-	-	512	1K	\$FF	High-Performance Nonmultiplexed 68-Pin
1C68HC11K4	-	24K	640	768	\$FF	>1 Meg memory space, PWM, Cs, 84-Pin
1C68HC711K4	24K	-	640	768	\$FF	One-Time Programmable Version of 'K4
1C68HC11L6	-	16K	512	512	\$0F	Like 'E9 with more ROM and more I/O, 64/6
1C68HC711L6	16K	-	512	512	\$0F	One-Time Programmable Version of 'L4

**Table 4.1**  
M68HC11 family members (Courtesy of Motorola)

### 4.2.2 External RAM

External RAM memory is one or more IC chips that are used to hold large amounts of data. Some adventurous (stupid) programmers store the program in this external RAM. This is not safe. The RAM can be easily corrupted as mentioned above. The external RAM can be located anywhere in the 64k-addressing scheme as long as it doesn't conflict with the internal RAM, ROM, EEPROM or registers of the microcontroller. But it should not conflict with any other external memory. It is possible to use external EEPROM instead of RAM, but this is not as straightforward as it sounds. Writing to RAM is very simple but writing to an EEPROM is not.

An example of writing data to a RAM location is...

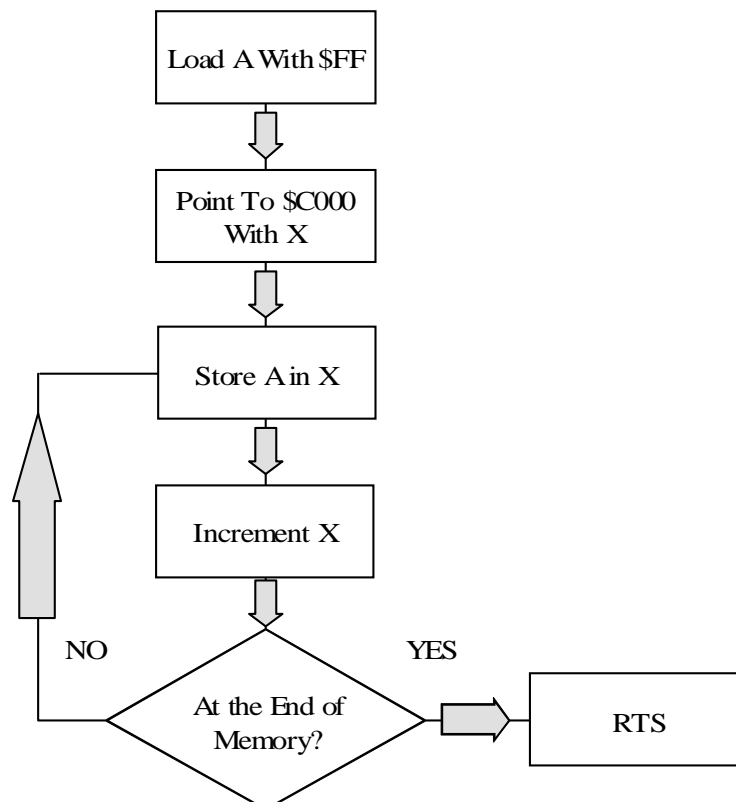
RAM	LDAA #30	Loads accumulator A with \$30
STAA	\$C00A	Stores \$30 in RAM at address \$C00A
RTS		Return to subroutines

Because RAM can get corrupted easily it is important to clear the RAM at start up and to check every so often that the data in the RAM is not corrupted.

There are three rules for guaranteed quality RAM:

- NEVER put the program in RAM, only temporary data. It is tempting to place the program in RAM, but as mentioned before, RAM is very susceptible to static and noise
- Clear the RAM at the beginning of the program. This is easy and only takes a few instructions. The following example clears 4K of RAM starting at \$C000 by placing \$FF in each memory location

CLEAR	LDAA	#\$FF	Load A with \$FF
	LDX	\$C000	Point to RAM
LOOP	STAA	\$00,X	Store \$FF in RAM
	INX		Increment the RAM location
	CPX	#\$CFFF	Is the Clear at the end of the RAM
	BNE	LOOP	If not at the end of RAM then LOOP
	RTS		Return to subroutine



**Figure 4.2**  
Flow chart for clearing ram memory

- If data is really important, write it to EEPROM and compare the data in RAM with the EEPROM. (NASA writes to three places and votes on the correct data.)

## 4.3 BUFFALO routines, memory map and vectors

### 4.3.1 BUFFALO as a development tool

Bit user's fast friendly aid to logical operation or BUFFALO is a common utility for development on microcontrollers. It contains many subroutines that can be used by the programmer during development of an embedded controller system. The most common subroutine is the one that lets the EVM communicate with the outside world using the SCI port. This port is part of the HC11 microcontroller and most EVM manufacturers use this port to connect to the RS-232 voltage standard. The following is a list of some of the BUFFALO subroutines.

### 4.3.2 BUFFALO utility subroutines

<b>\$FF7C</b>	WARMST	Go to > prompt
<b>\$FF7F</b>	BPCLR	Clear Breakpoint Table
<b>\$FF82</b>	RPRINT	Display user's registers
<b>\$FF85</b>	HEXBIN	Convert ASCII character in A to a 4 bit binary number
<b>\$FF88</b>	BUFFAR	Read 4 bit hex number from input buffer to SHFTREG.
<b>\$FF8B</b>	TERMAR	Read 4 digit hex number from serial port to SHFTREG.
<b>\$FF8E</b>	CHGBYT	Write value from SHFTREG+1 to memory location
<b>\$FF91</b>	READBU	Read next character from input buffer.
<b>\$FP94</b>	INCBUF	Increment input buffer pointer.
<b>\$FF97</b>	DECBUF	Decrement input buffer pointer.
<b>\$FF9A</b>	WSKIP	Read input buffer until non-whitespace character.
<b>\$FF9D</b>	CHKABR	Monitor input for Ctrl-X, Delete, and Ctrl-W.
<b>\$FFA0</b>	UPCASE	Convert character in A register to upper case.
<b>\$FFA3</b>	WCHEK	Return with Z flag set if char in A register is whitespace. BUFFALO considers space, comma, and tab to be whitespace.
<b>\$FFA6</b>	DCHEK	Return with Z flag set if character in A register is a delimiter
<b>\$FFA9</b>	INIT	Initialize I/O devices.
<b>\$FFAC</b>	INPUT	Read from serial port
<b>\$FFAF</b>	OUTPUT	Write to serial port
<b>\$FFB2</b>	OUTLHL	Convert left nibble of A register to ASCII and print it
<b>\$FFB5</b>	OUTRHL	Convert right nibble of A register to ASCII and print it.
<b>\$FFB8</b>	OUTA	Print character in A register.
<b>\$FFBB</b>	OUT1BY	Print one hex byte at memory location pointed to by X register.
<b>\$FFBE</b>	OUTIBS	Same as OUT18 Y, but print a space as well.
<b>\$FFC1</b>	OUT2BS	Same as OUT1BY, but print two bytes followed by a space.
<b>\$FFC4</b>	OUTCRL	Print carriage return followed by linefeed.
<b>\$FFC7</b>	OUTSTR	Print the string pointed to by X register. \$04 marks the end of the string.

<b>\$FFCA</b>	OUTSTO	Same as OUTSTR, but without the leading CR/LF.
<b>\$FFCD</b>	INCHAR	Read a character and echo it. Return character in A
<b>\$FFDO</b>	VECINIT	This routine loops until a character is received. Initialize the pseudo-vector table.

### 4.3.3 BUFFALO memory map



**Figure 4.3**  
*BUFFALO EPROM on EVM Board*

The following memory map shows the location of various stacks, variables, vectors and registers associated with the microcontroller.

\$0000-\$0047	User RAM and User Stack
\$0048-\$0065	BUFFALO Stack
\$0066-\$00C3	BUFFALO Variables
\$00C4-\$00FF	Interrupt Pseudo-Vectors
\$0100-\$01FF	User RAM
\$1000-\$103F	HC11 Control Registers
\$8000-\$8001	LCD Data and Control Registers
\$B600-\$B7FF	EEPROM
\$C000-\$DFFF	Memory Socket
\$E000-\$FFFF	Memory Socket Containing BUFFALO EPROM

### 4.3.4 BUFFALO interrupt pseudo-vectors

A pseudo-vector is a vector defined by the BUFFALO (in this case) to point to a specific address. The programmer can define the interrupt vectors as needed by the program. The BUFFALO points all of the interrupt vectors to a set of pseudo-vectors in RAM. The programmer is free to change them. Each pseudo-vector contains 3 bytes of data, one instruction and a two-byte address such as (\$7E) followed by the address of the service routine.

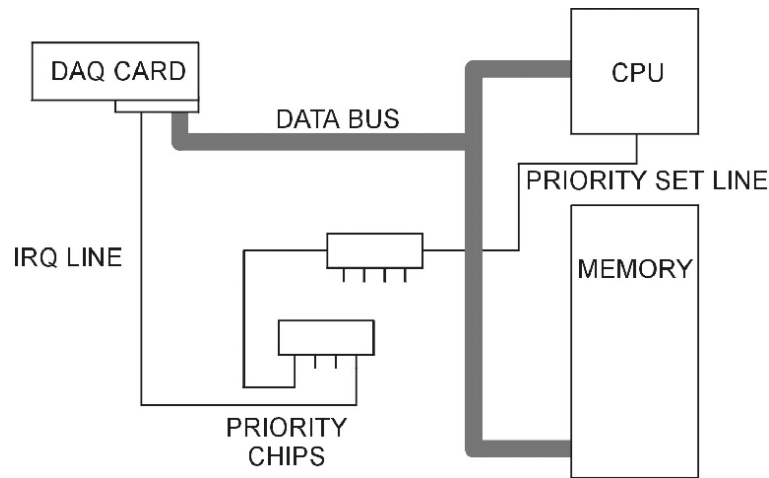
SCI	\$00C4
SPI	\$00C7
Pulse Accumulator Input	\$00CA
Pulse Accumulator Overflow	\$00CD
Timer Overflow	\$00D0
Output Compare 5	\$00D3
Output Compare 4	\$00D6
Output Compare 3	\$00D9
Output Compare 2	\$00DC
Output Compare 1	\$00DF
Input Compare 3	\$00E2
Input Compare 2	\$00E5
Input Compare 1	\$00E8
Real Time Interrupt	\$00EB
IRQ	\$00EE
XIRQ	\$00F1
SWI	\$00F4
Illegal Instruction	\$00F7
COP	\$00FA
Clock Monitor	\$00FD

## 4.4 Interrupts, vectors and pseudo-vectors

A vector contains the start address of a reset or a subroutine. The pseudo-vector address is a specific address that contains the 3 bytes of information about the vector. The first byte is usually a JMP or BRA type of instruction with the next two bytes the address of the subroutine. The chip manufacturer defines the vectors while the programmer or BUFFALO defines the pseudo-vectors. An example of vectoring could be...

If the IRQ hardware line goes low (0 volts), the microcontroller puts the IRQ's vector contents \$00EE (FFF2 and FFF3 as defined by Motorola) in the program counter of the microcontroller. The microcontroller then executes the user-defined 3-byte instruction starting at \$00EE. The instruction located at \$00EE could be \$7E (JMP) \$C000. This address, \$C000 would be the address of the start of the subroutine that services the IRQ.





**Figure 4.4**  
*IRQ*

#### 4.4.1 Software vs hardware interrupts

Hardware interrupts and software interrupts are the two ways the programmer has of telling the microcontroller that something has happened in the outside world or in the program. An interrupt can be compared to a telephone call. When the phone rings there is an interrupt. Plugging in the phone is enabling the interrupt. Needing a phone card to use the phone is like having a maskable interrupt. Only people with the phone card can use the phone. Telephones that work without a phone card are like a non-maskable interrupt. Anyone can use the telephone.

The programmer can program the microcontroller to ignore certain interrupts by either not enabling the interrupt or using a maskable interrupt. To enable or mask an interrupt the programmer would use an enable or mask bit in the interrupts setup register.

Vector	Address	Source	Vector	Pseudo-vector
\$FFF0	F1	RTI	\$00EB	\$7E \$C500
\$FFF2	F3	IRQ	\$00EE	\$7E \$C15A
\$FFF4	FS	XIRQ	\$00F1	\$7E \$C19B
\$FFF6	F7	SW1	\$00F4	\$7E \$C367
\$FFF8	P9	IOT	\$00F7	\$7E \$C256
\$FFFA	FB	COP	\$00FA	\$7E \$E000
\$FFFC	FD	CMF	\$00FD	\$7E \$E000
\$FFFE	FF	RESET	\$E000	

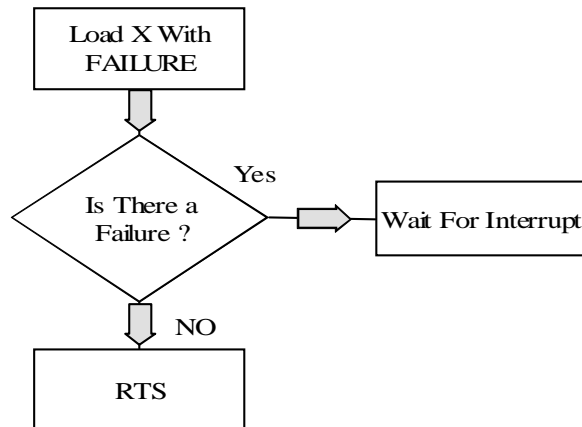
**Table 4.2**  
*Vector assignments (Courtesy Microcontroller Technology Spasov)*

Hardware interrupts by definition require a hardware line to initiate the interrupt. This hardware line could be attached to a card, module or device. An example of this could be a non-maskable hardware interrupt line from a data acquisition module (XIRQ). When

the data acquisition module needs to send data it would place a low (0 volts) on the IRQ line into the microcontroller. The microcontroller would then activate the vector located at \$FFF2 (see table 4.2). The vector would then run the pseudo-vector as explained in 4.3.4

A program can activate its own software interrupts. The programmer may want to jump to a pseudo-vector based on an error or something that has happened within the program. A software interrupt as the part of a subroutine might look like the following.

ERROR	LDX	FAILURE	Points to failure address
	CPX	#\$01	Check if a failure occurred (\$01)
	BNE	END	If not return to subroutine
	SWI		Software Interrupt
END	RTS		Return to subroutine after interrupt



**Figure 4.5**

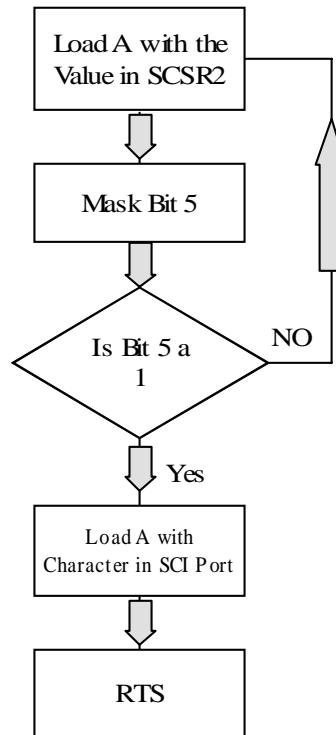
*Flow chart of wait for software interrupt*

#### 4.4.2 Maskable vs non-maskable interrupts

As mentioned before it may be necessary for the programmer to mask off an interrupt. Masking an interrupt can be compared to placing a mask over your face. People will only see those parts of your face that are not masked. When an interrupt is activated the microcontroller will only see the interrupt if the bits that are not masked are enabled. An example of the maskable interrupt could be the SCI (asynchronous communications) interrupt. The SCI interrupt is a maskable interrupt that activates when a character is ready to be received or transmitted by the microcontroller. The following example shows how the maskable interrupt might work. The receive interrupt could be masked by bit 5 (RDRF receive ready) of the SCSR2 (\$102E) register. Bit 5 is set to a 1 if a character has been received is ready to be collected. A 0 here masks the interrupt and does not let the program collect the character from the SCI port. This register could be read by the program as follows...

COMMS	LDAA	\$102E	Load A with SCSR2 Register
	BITA	#\$20	Check if bit 5 is a 1 (mask)
	BNE		COMMS If bit 5 is a 0 Then Loop
	LDAA	\$1008	Load A with the Character in SCI Port
	RTS		Return to Subroutine

**Note:** The BITA instruction does not change the value in accumulator A.



**Figure 4.6**  
Flow chart of masking a register

SCI Serial System
Interrupt Request (IRQ)
External Interrupt (XIRQ)
SPI Transfer Complete
RTI Interrupt
Timer Input Capture 1
Timer Input Capture 2
Timer Input Capture 3
Timer Output Compare 1
Timer Output Compare 2
Timer Output Compare 3
Timer Output Compare 4
Timer Input 4 / Output Compare 5
Timer Overflow
Pulse Accumulated Overflow
Pulse Accumulated Input Edge

**Table 4.3**  
Maskable interrupts (The M68HC11 microcontroller Kheir)

Non maskable interrupts are interrupts that cannot be masked. They are defined by the microcontroller.

Reset
Clock Monitor
COP Watchdog
Illegal Opcode
Software Interrupt (SWI)

**Table 4.4**

*Non-maskable interrupts (The M68HC11 microcontroller Kheir)*

## 4.5 Control registers

### 4.5.1 Memory mapped I/O

A microcontroller would not be of much use if it couldn't communicate with the outside world. One of the main differences between a CPU and a microcontroller is that a microcontroller has on board inputs and outputs. An I/O is a line or collection of lines (bus) that can be defined as either inputs or outputs. When we discuss I/O we usually mean digital inputs or outputs. Analog inputs or outputs sometimes use the same lines as digital inputs. The way the lines are configured is defined or controlled by the data direction control registers. By setting a certain bit or bits in the data direction registers (a 1 is setting a 0 is clearing), the programmer is able to control one or more I/O lines.

Following is a list of I/O ports and their respective data control registers

I/O	Address	Notes	Register	Address
Port A	\$1000	Fixed ex Bit 3 and 7	PACTL	\$1026
Port B	\$1004	Fixed	None	None
Port C	\$1003	I/O	DDRC	\$1007
Port D	\$1008	I/O	DDRD	\$1009
Port E	\$100A	Digital or Analog Input	None	None

**Table 4.5**

*I/O ports*

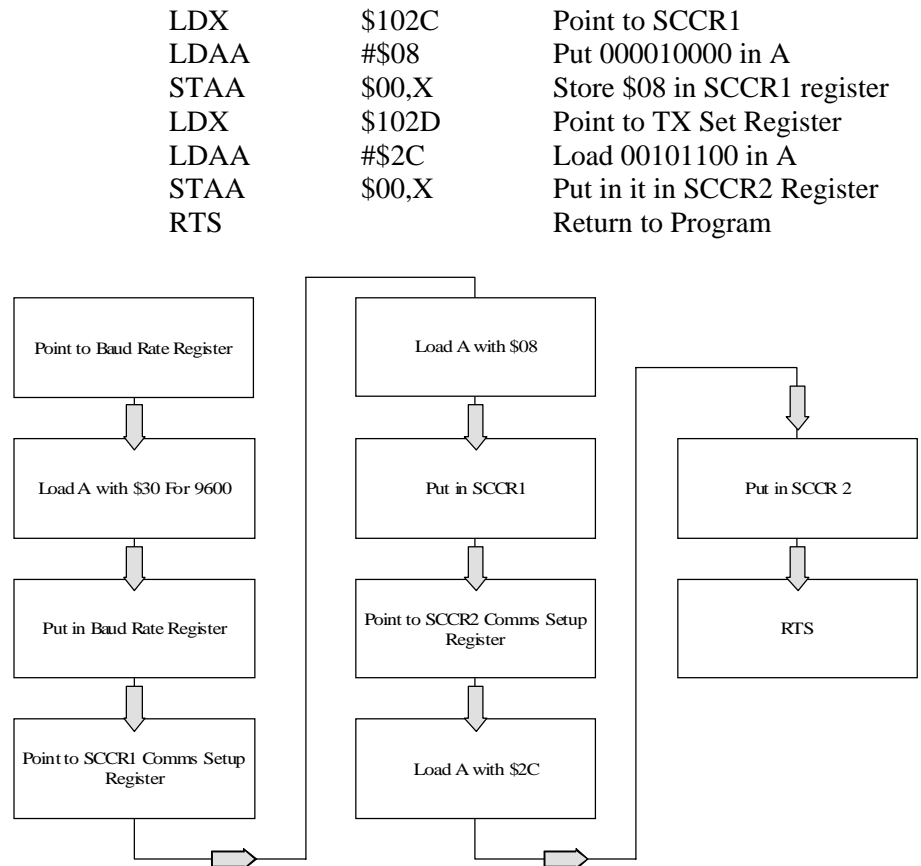
### 4.5.2 Accessing and using control registers

Control registers are used for many purposes:

- Data direction
- Enabling interrupts
- Masking
- Baud rates
- System configuration
- Input and output compare

A typical example of using a control register would be...

```
COMSET    LDX    $102B    Point to baud Rate Register
          LDAA   #$30    Load A for 9600 baud
          STAA  $00,X    Store $30 in baud Register
```



**Figure 4.7**  
*Control register setup flow chart*

In the above example the value \$30 is stored in the register that defines the baud rate of the microcontroller. When \$30 is placed in the baud rate register, the communications is setup as 9600 baud. When \$08 (00001000) is stored in the SCCR1 (\$102C) register, it sets the SCCR1 setup register for 8 data bits, no parity and 1 stop bit. Next the value \$2C (00101100) is stored in the second SCI setup register (SCCR2 at address \$102D). Bit 5 enables the interrupts for the communications port. The port does not have to poll the external device for information. Bit 3 is used to enable the transmitter and bit 2 is used to enable the receiver. When receiving or transmitting data to or from port D, the program uses the SCSR2 (\$102E) to view the status of the communication port. It is often necessary to retrieve the status of the communications port before any character is sent and when a character has been received.

## 4.6 EEPROM

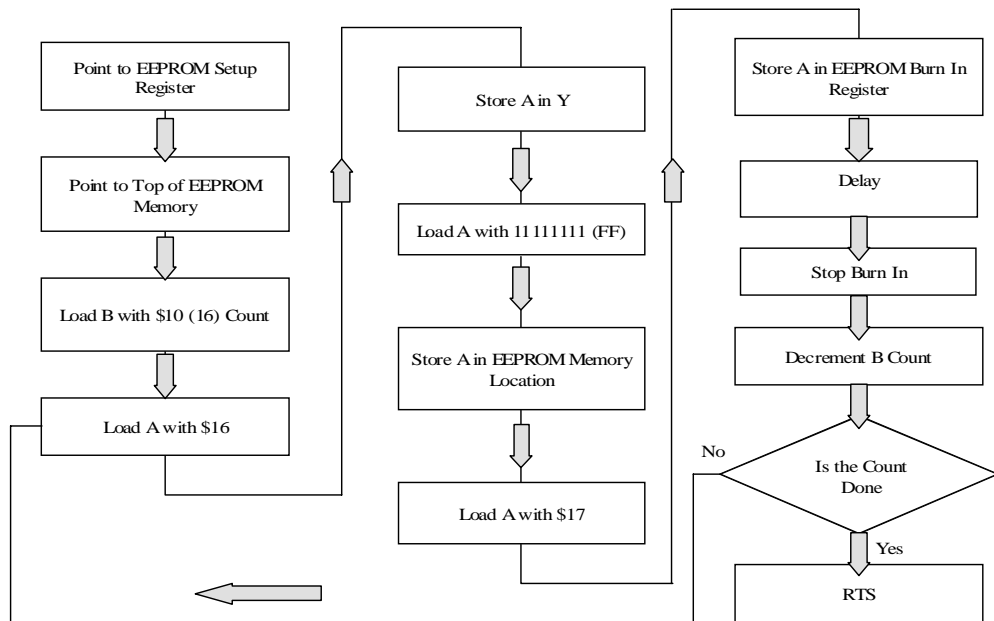
Electrically erasable programmable read only memory devices (EEPROM) are becoming very popular in microcontroller systems. Notice that this is a contradiction. How can a device be both erasable and read only? But it is. The reason for this is that once the device has been programmed the device acts very much like a ROM (read only memory). The EEPROM is very stable and not easily spiked or corrupted by high voltage or static. The down side of the EEPROM is that it is difficult to program. The EEPROM programming method is not as straightforward as RAM. In RAM the microcontroller simply puts the

data in the memory location. Any data that is there already is written over. The write-enable line for the RAM chip is automatically set by the store instruction.

Programming an EEPROM is a two-step function. First the memory has to be cleared and then the data is written. One problem is that the EEPROM writing system only changes the 1s to 0s in the memory. This means that if there already is a 0 in a memory location and a 1 is put in that location the 0 will not change to a 1. This could cause problems with wrong data. All ones must be placed in the EEPROM memory before any data can be written. Another situation with EEPROMs is that once the data has been sent to the memory location a high voltage 12 V must be placed for a short time on the chip. The voltage is placed on the chip by sending a byte of data to the EEPROM setup register. After a short delay the burn in voltage is turned off. This has to happen every time a byte is saved to an EEPROM memory location.

#### 4.6.1 Clearing the EEPROM example

AREA	EQU	\$B600	EEPROM memory location
EESSET	EQU	\$103B	EEPROM setup register
Writes \$FF from \$B600 to \$B6FF EEPROM Memory Locations – Pre-Clear			
EECLR	LDX	#AREA	Point to EEPROM setup register
	LDY	#EESSET	Point to EEPROM memory location
	LDAB	#\$10	Loads B with 16 byte count
LOOP	LDAA	#\$16	Load A with 00010110
	STAA	\$00,Y	Store \$16 in EEPROM (setup for write)
	LDA	#\$FF	Load A with 11111111
	STAA	\$00,X	Store \$FF in EEPROM memory location
	LDAA	#\$17	Load A with 00010111
	STAA	EESSET	Activate high voltage for burn in
	JSR	DELAY	Delay 10 ms for burn in
	CLR	\$00,Y	High voltage off and sets EEPROM to read
	DECB		Decrement the 16 count
	CMPB	#\$00	Check if done
	BNE	LOOP	If not done the loop
	RTS		Return to subroutine



**Figure 4.8**  
Clearing the EEPROM flow chart

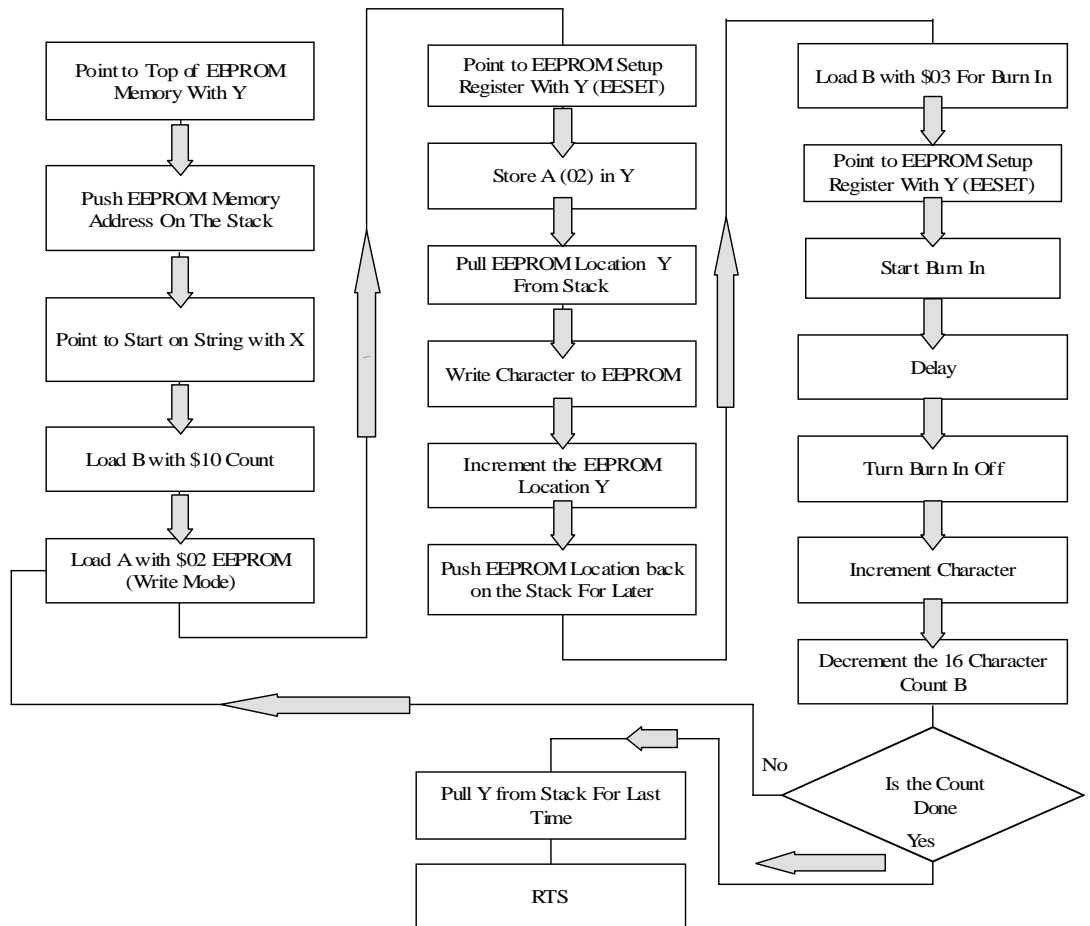
## 4.6.2 Writing to the EEPROM example

Writes 16 bytes of data at STRING1 to the EEPROM

**Note:** Setting bit 1 (\$02) of the PPROG (\$103B) register enables the EEPROM as being able to be written to and setting bit 0 also (\$03) enables the burn in mode.

EERITE	LDY	#AREA	Point to EEPROM area for data
		PSHY	Push area pointer on the stack
		LDX	#STRING1 Point to string of data to be stored in EEPROM
LOOP		LDAB	#\$10 Load count with 16 characters
		LDAA	#\$02 Load 00000010 for EEPROM write mode
		LDY	#EESET Point to EEPROM memory location
		STAA	\$00,Y Store \$02 in setup register (write mode)
		PULY	Get the EEPROM area from stack
		STA	\$00,Y Store character in EEPROM memory
		INY	Increment the EEPROM area
		PSHY	Push EEPROM area onto the stack for later
		LDB	#\$03 Put high voltage for burn in
		LDY	#EESET Point to EEPROM memory location
		STAA	\$00,Y Burn in data
		JSR	DELAY Delay 10 ms
		CLR	EESET High voltage off
		INX	Increment character
		DECB	Decrement the character count
		CMPB	#\$00 Check if done
		BNE	LOOP If not done loop
		PULY	Pull EEPROM area for last time.
		RTS	Return to subroutine

Notice in the above example that the Y register is used for both the EEPROM area location and EEPROM setup register. The stack is used to hold the string area pointer when Y is being used to hold the EEPROM setup register. Also note that it is important to pull the EEPROM area location from the stack when finished.

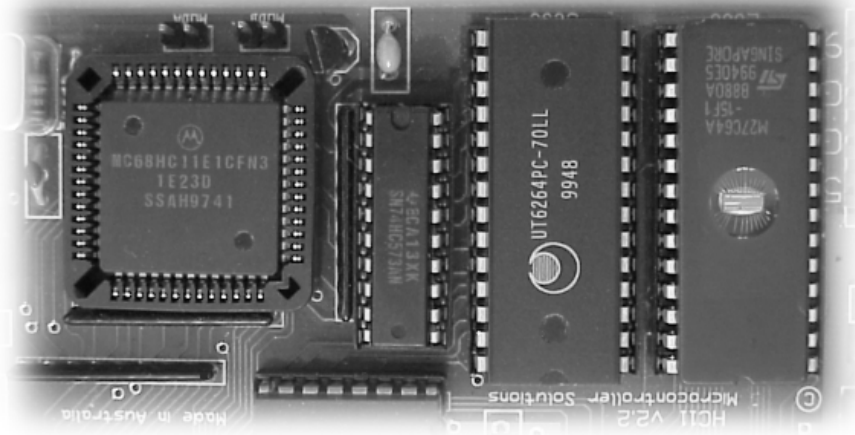


**Figure 4.9**  
*EEPROM write flow chart*

## 4.7 Conclusion

In the early days of electronics, memory chips were very expensive and it took a lot of chips to make up a small amount of memory. Now the prices have fallen dramatically (in fact if you check the magazines, they usually say CALL for prices). Soon gigabytes of memory will be available on a single chip. Microcontroller systems have yet to take full advantage of this explosion in memory. Most microcontroller systems still talk about kbytes of memory at best. The 68HC11 microcontroller has a maximum of 1 K of RAM and 512 bytes of EEPROM. Because of this small amount of on board RAM and EEPROM most designers have to include external chips such as battery backed RAM, EPROM and/or EEPROMs in their circuits.





**Figure 4.10**  
*Microcontroller, RAM and EPROM*

Memory within the microcontroller system can be accessed by instructions within the program. Depending on the type of memory and its location in the Memory Map the programmer may use RAM for one type of data or an EEPROM for another type. The programmer often uses vectoring to move around within the memory map of the microcontroller system. The vectoring system is partly defined by the manufacturer of the chips and partly defined by the programmer. User definable pseudo-vectoring allows the programmer to have an interrupt vector to almost anywhere in the total memory map of the microcontroller system. These interrupt vectors can be masked or non-maskable. The maskable interrupts let the programmer set conditions for certain interrupts. Whereas the non-maskable interrupts are defined by the chip manufacturer and never change. This has the advantage that the programmer can depend on them to work the same way every time without worrying about configuration.

The maskable interrupts are configured using control registers. These control registers are bit oriented and every bit may have a specifically defined purpose. The bits are either set or cleared in the program by the programmer to configure the register. Besides being used for maskable interrupts the control registers are often used for control of the I/O ports within the microcontroller. These ports can be digital outputs, digital inputs or analog inputs. The control registers often define the direction of the I/O lines on the ports. The registers can also define conditions for memory like the EEPROM. Setting one bit on a condition control register could tell the EEPROM that it is read mode or write mode.

It is hoped that in the near future the huge amounts of memory that are being seen in the personal computer industry will be included in microcontroller systems. Until that day we will still need the programmers' skill in manipulating the code to optimize the volume of the program. External chips take up room, power and add substantially to the cost of the microcontroller device

## Microcontroller inputs and outputs

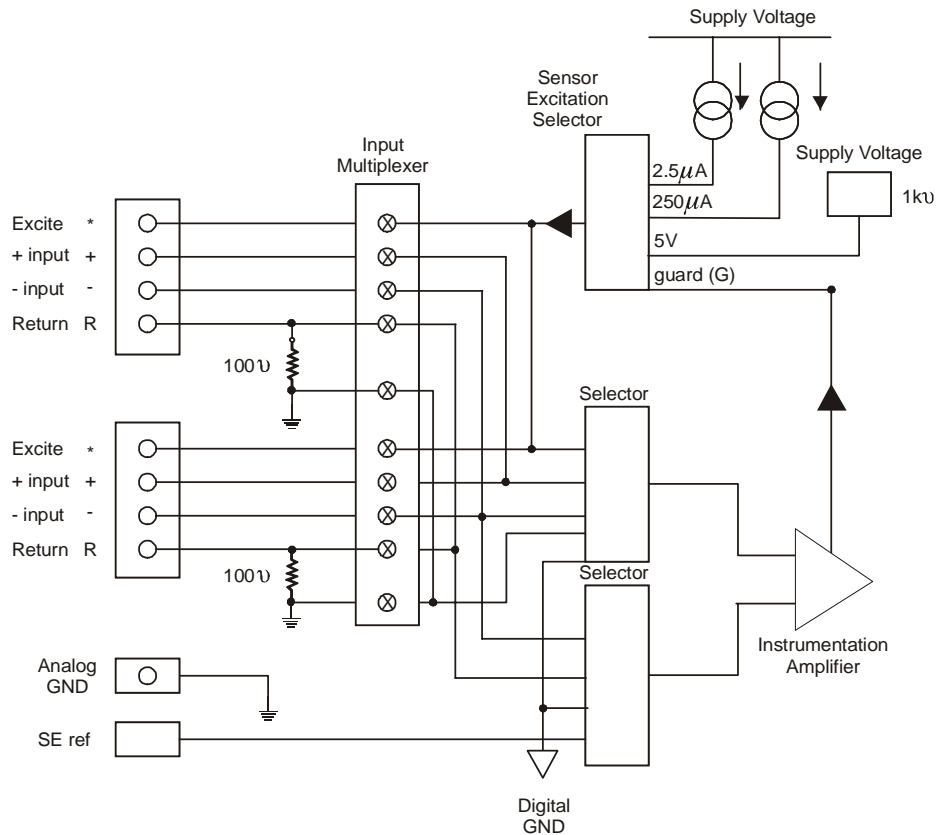
### Objectives

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

- Describe the differences between single ended and differential inputs and outputs
- Explain how digital inputs and outputs are used in a microcontroller
- Explain how analog inputs are sampled by an A to D converter
- Describe the Nyquist sample rate and how it relates to sample frequencies
- Describe the digital control of analog devices
- Describe how to interface a keypad to a microcontroller
- Explain how to interface an LCD screen to a microcontroller

### 5.1 Introduction to inputs and outputs

The ultimate goal of most controllers is to take in data, do something with that data and then output some signal that ultimately controls a device. To accomplish this it is necessary to use digital and analog inputs and outputs. Because controllers are getting smaller, cheaper and easier to use, the amount of data acquisition and control in industry is expanding tremendously. It is therefore important to understand the different types of inputs and outputs and how they interact with each other. This chapter will discuss single ended and differential digital and analog inputs and outputs and how they relate to microcontroller systems.



**Figure 5.1**  
*Digital inputs and outputs*

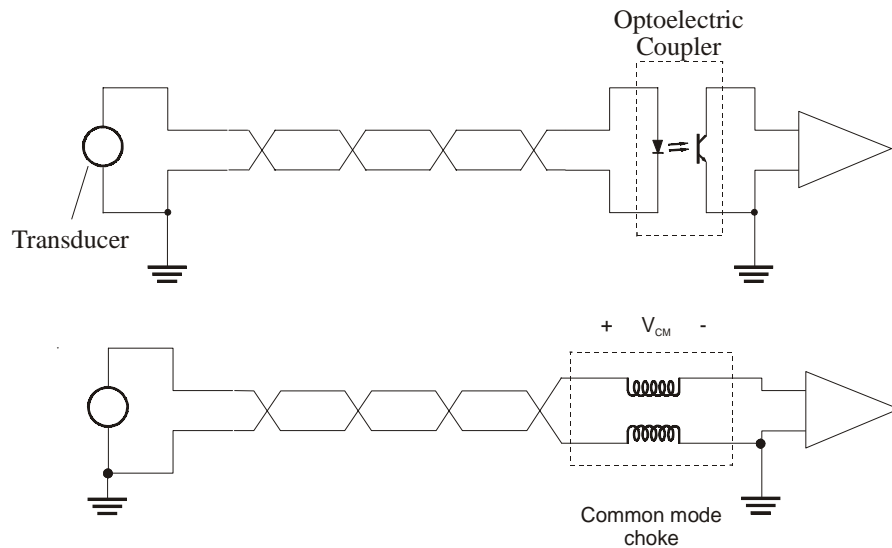
## 5.2 Single ended vs differential inputs

There are two different types of input circuits used in data acquisition systems. One is single ended and the other is differential. Each of these circuits has their uses and their good and bad points. Different types of circuits could or should be used depending on different situations. With these circuits nothing is absolute and both types could be used in the same circuit. But having said that, the user and engineer probably would find that in some situations one is greatly preferred over the other.

### 5.2.1 Single ended analog circuits

Single ended analog circuits are used to input analog values of voltage, current or resistance into a microcontroller. One side of the connection is commoned together with the common side of the other circuits. This common line is usually placed at ground level, either at the sensors or back at the equipment. In most single ended systems the common lines are connected together out in the field. There is a version of this called pseudo-differential where the two wires from the sensor are brought back to the equipment and one is connected to a common there. Pseudo-differential is electrically and functionally single ended. The advantage of single ended analog circuits is that only one input channel is needed for each analog input. This reduces the number of channels and cost of an installation. The disadvantage of single ended analog input systems is that they are very susceptible to noise. Any change in the potential voltage difference in the ground level will change the output of the sensor as seen by the equipment. This noise is very difficult

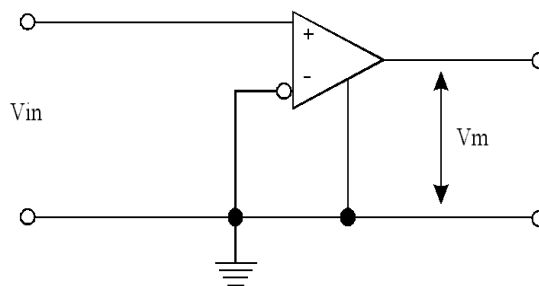
to remove because the noise is ground based. Often the best way to reduce noise induced on single ended systems is to use a transformer or choke based isolator.



**Figure 5.2**  
*Single ended analog circuit*

## 5.2.2 Single ended digital circuits

Single ended digital circuits are typically used to connect switches together. These switches can be either mechanical or electronic. Again the common lines of the switches are either connected together in the field or they are connected together back at the equipment. The advantage of single ended inputs is the same as analog inputs, only one channel per input. One difference between analog inputs and digital inputs is that digital inputs are more resistant to noise. This does not mean that digital inputs are completely immune to noise. Ground based noise can affect digital inputs, it just takes a stronger noise signal. To make the digital input more resistant to ground induced noise the switches are often connected to the equipment via opto-couplers. These opto-couplers isolate the ground from the equipment and therefore reduce the amount of noise. The opto-isolator is a voltage saturation device. This means that for it to work the voltages must be typically either +5 volts or ground. Small voltage changes are ignored by the opto-isolator.

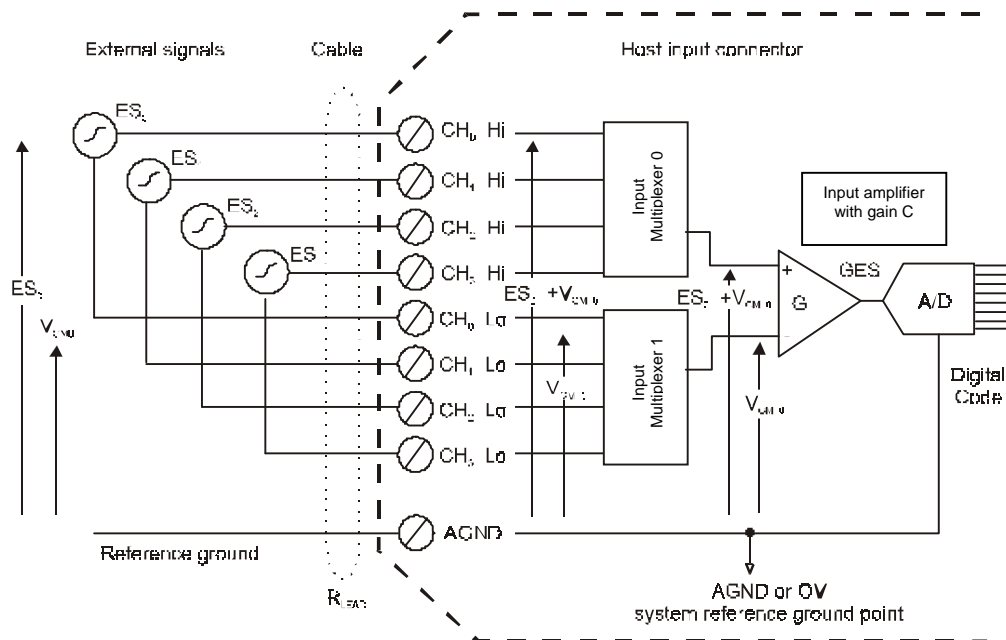


**Figure 5.3**  
*Single ended digital circuit*

### 5.2.3 Differential analog circuits

Differential analog systems are connected by installing each wire from the sensor into a different channel on the equipment. Differential analog circuits take twice as many channels as single ended analog circuits. Because the two lines are referenced to each other and not to ground, differential circuits are much more resistant to noise than single ended circuits. The differential nature of the circuit gives a measure of isolation for each line to ground. The value of this measurement is defined as common mode resistance ratio. The CMRR value is defined by the manufacturer in db. A reasonable CMRR level would be around 90 db. Common mode voltages can be adjusted by using external resistors. The manufacturers usually define the value of the resistors, but often they are around the 100 k $\Omega$  area. Both resistors must be equal.

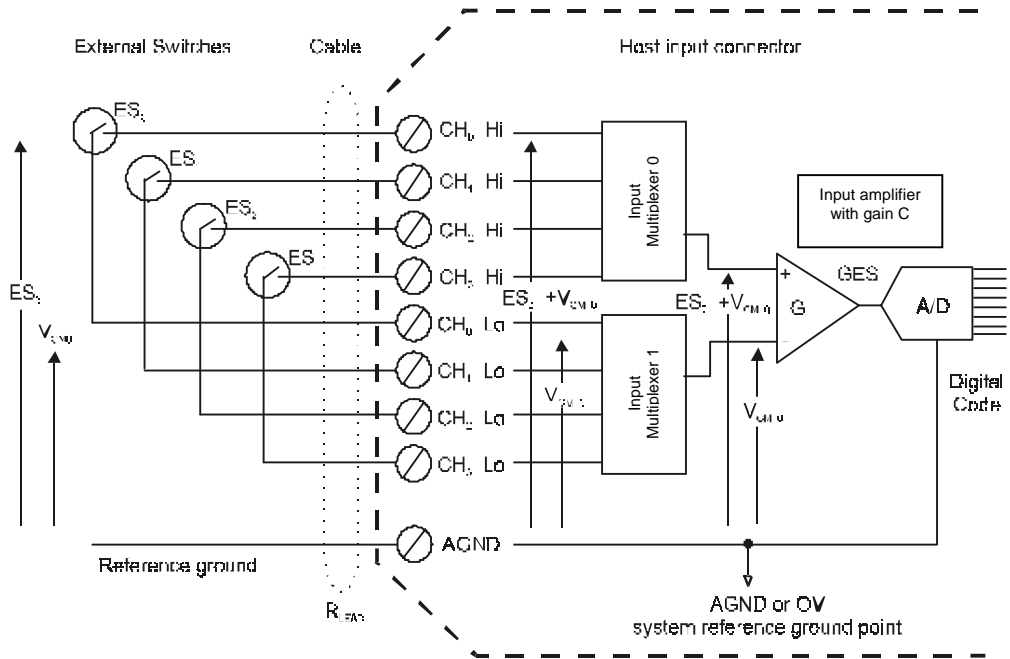
**Hint:** if using an oscilloscope to look at common mode voltages, make sure the input to the oscilloscope is differential and not single ended. Most oscilloscopes are single ended.



**Figure 5.4**  
*Differential analog circuit*

### 5.2.4 Differential digital circuits

Digital differential inputs are often switch inputs and are used where high noise can be a factor. These inputs also often use opto-isolators to reduce noise and provide protection against high voltages. When switches are located outside and have long leads they are very susceptible to lightning and static voltages. The use of a differential two-channel input and the saturation of the opto-isolator greatly reduce the chance of noise or high voltage. Noise created by high voltages can cause incorrect inputs. But no equipment is completely immune to noise or lightning, no matter how well protected. The disadvantage of differential digital circuits is the high cost due to the use of two channels and the extra equipment needed for the opto-isolation. The advantage is of course their high noise immunity.

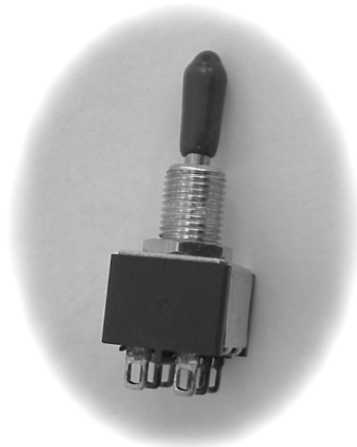


**Figure 5.5**  
*Differential digital circuit*

## 5.3 Digital inputs

### 5.3.1 Switch sensing and de-bounce

Digital inputs are usually switch closures and most of the time these are mechanical as opposed to electronic switches, although this may change in the future. For the moment we get most of our information about the outside world using mechanical switches. One of the problems with mechanical switches is de-bounce. When a mechanical switch closes the metal parts compress and then relax. At this point the switch opens and closes very quickly. The problem is that the microcontroller can read the switch so fast that the microcontroller sees the switch open and close during the bouncing of the metal parts. The microcontroller would then see multiple switch closures. To adjust for this bouncing, the designer would put in some type of de-bounce. In the early days of microcontrollers, engineers used transistors and chips to delay the reading of the switch during the bouncing. Now most de-bounce is done in software. The programmer writes a subroutine that reads the input then delays for a period of time and then re-reads the switch. If the programmer had just put in a delay and didn't re-read the switch, then the microcontroller might see false switch closures.



**Figure 5.6**  
*Mechanical switch*

### 5.3.2 Normally open (NO) and normally closed (NC) switches

Switches are known in the industry as either normally open or normally closed. A mechanical switch is considered normally open if the contacts are open at rest. It is considered normally closed if the contacts are closed at rest. At rest can sometimes be subjective and different manufacturers sometimes mark their switches differently. For example, is a magnetic switch at rest when the magnet is next to the switch? Or is it at rest when the magnet is away from the switch? Security systems normally use contacts or switches that are normally closed when the magnet is next to the switch. This is done so that if the bad guys cut the wires the circuit will be open and the alarm will go off. Thieves of course know this, and often short out the switches in hope of disabling the system. A good alarm system would then use a combination of both NC and NO switches. Most switch sensor systems connect one side of the switch to a common ground wire and the other to the equipment. In this way only one more wire than the number of switches comes out of the equipment. I.e... 5 switches – 6 wires.



**Figure 5.7**  
*Normally open and normally closed switches*

### 5.3.3 Electronic switches

Electronic switches use either chips or transistors to indicate a change in state. These circuits are often magnetic, inductive, infrared, or radio frequency devices. Because they are electronic, they are more complicated and require power from mains or batteries. Another problem with electronic switches is that they are highly susceptible to electromagnetic fields (radio frequencies), to the point that a strong enough field could activate, deactivate or even destroy the switch. Use of mobile phones and other

transmitters should be kept well away from all electronic switches. These complications are negated by the fact that electronic switches can recognize situations that mechanical switches cannot. For example microwave sensors can see through materials, infrared devices can see in the dark and inductive devices can sense magnetic fields.

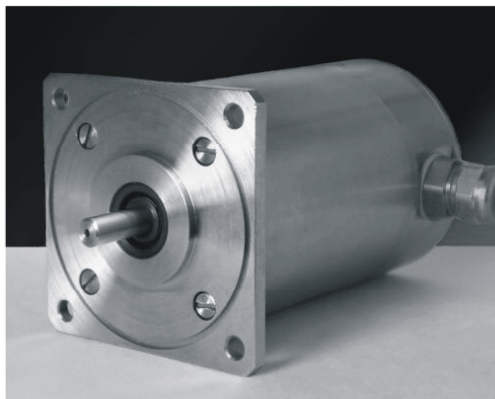


**Figure 5.8**  
*Electronic switch*

## **5.4 Digital outputs**

### **5.4.1 Digital control**

Outputs from a microcontroller can control any type of electrical or electronic equipment. Most control of devices today is done using digital control. In the Eighties digital to analog control was seen as the way to control analog devices. The logic was that the microcontroller was digital and the world was analog and therefore we would need lots of digital to analog control circuits. Before computers, nearly all electrical devices were digitally controlled. Switches and relays were used to control all equipment. When computers were developed it was realized that it would be easy to continue to control things digitally. Even today there is very little analog control compared to digital control. Remarkably we now control some analog functions using digital control. For example a stepper motor is a digital device, even though it moves in an analog manner.



**Figure 5.9**  
*Stepper motor (Courtesy of [www.Phytron.com](http://www.Phytron.com))*



Microcontrollers use address decoders and hardware drivers to connect to electromechanical and solid-state relays. Often successive relays are used to increase the current and/or voltage output. The address decoder is used by the microcontroller to connect to the correct device. For example the programmer may want to turn off an output. To get to that output the programmer would do something like this.

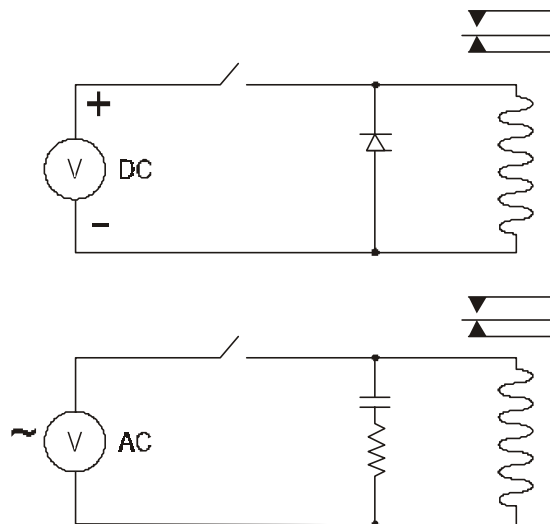
LDX	\$1008	Point to the address of port D with X
LDAA	#\$01	Load the data 0000 0001 into A temporarily
STAA	\$00,X	Store the data in bit 0 of port D

Notice that a 1 turns bit 0 off. This is very common as a 0 is often **ON** in digital outputs. The address decoder sees \$1008 on the bus and knows this is port D. It then views the value \$01 on the data bus and puts it in the port D output register. This may be the last driver in the output chain or the designer may have put another driver in the circuit like a ULN 2003. The ULN 2003 driver is a very common relay driver chip.

## 5.4.2 Back EMF causes and solutions

The main problem with using relays to control devices is back EMF (electro magnetic force). Back EMF happens when the relay is turned off and the magnetic field collapses across the coils of wire in the relay. The collapsing field induces a large voltage in the relay. Lenz's law states that the voltage produced by a coil is determined by the size of the field, the number of coils and the speed at which the field cut across the coils of wire. Long lengths of the wire connected to the relay can also add to the back EMF voltage. This happens because the field around the long wire collapses and cuts across the wire. This adds to the back EMF voltage.

Solid-state relays are not relays at all. They are really transistors controlling devices. So called solid-state relays are often thought of as a good replacement for relays, but they have problems. The back EMF created by coils or long wires can easily destroy a solid-state relay. They are very sensitive to high voltages and static. If a relay, close to the microcontroller is used to drive another relay, both relays have 10k of voltage protection across their contacts. The solid state relay doesn't usually have this protection. They do not create back EMF, but they are very susceptible to back EMF.



**Figure 5.10**  
*Snubber networks on DC and AC circuits*

All relays should have snubber networks across their coils to eliminate any back EMF problem. These snubber networks consist of either a diode, for DC relays or a capacitor and resistor for AC circuits. A typical diode could be a 1N4004 1/4 to 3 amp for the DC relay. For the AC relay a 0.1 uFd AC capacitor with a 100 Ω 1/4 amp resistor would be typical. These values may be different for your system and it is best to check with your manufacturer for the proper values.

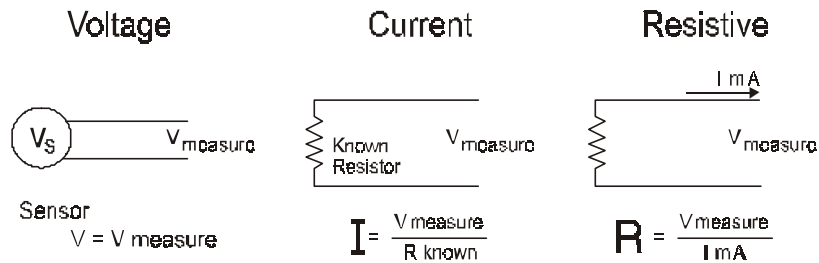
## 5.5 Analog inputs

### 5.5.1 Voltage, current and resistive measurement

Typically data acquisition systems use analog inputs to measure voltage directly from a sensor. If the user wants to measure current or resistance then a different and indirect method is used. Current is indirectly measured by finding the voltage drop across a precision resistor. Using Ohm’s law, the microcontroller calculates the current. To find the resistance, a known current is passed through the circuit and is divided into the voltage that is measured across the resistor.

Voltages often don’t come in the level we would like and therefore they need to be adjusted. For example, a measurement system is designed as a 0 to +10 volt system, but the voltage on the system is 0 to +12 volt. This means that a voltage divider network would be used so that the voltage would be dropped to a more reasonable level.

AC and DC currents cannot be measured directly. To measure DC currents we would place a low value precision resistor in line with the circuit. This resistor must be low enough not to reduce the current used by the circuit. If the circuit is AC then measurement is done by measuring the voltage drop across a coil of wire looped around the current carrying wire. Using Ohm’s law, if we know the impedance of the coil and the measured voltage, we can calculate the current. For measuring resistance we place a known current through the resistor and then measure the voltage drop across the resistor. Again using Ohm’s law if we know the current and the measured voltage then we can calculate the resistance. This known current is often 10 mA.



**Figure 5.11**  
Voltage, current and resistive measurement

We can see by the following paragraph that whether we are measuring voltage, current or resistance, in the end we have a voltage. The microcontroller samples the voltage on a regular basis and a digital value in the form of ones and zeros is developed. The number of ones and zeros used to define the number is called the resolution. For example...

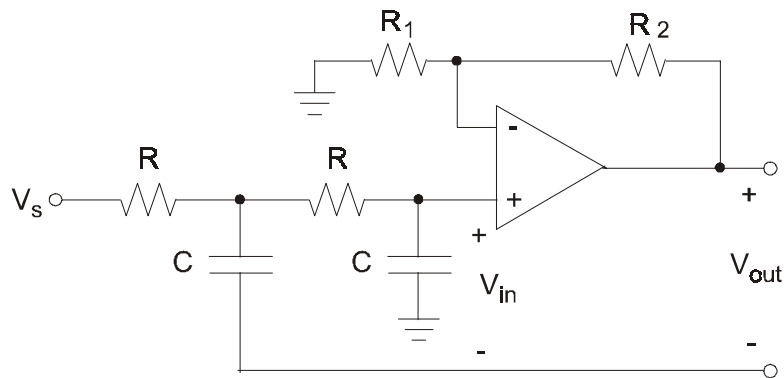
Measured value	+5 volts from a 0 to +10 volt level
Sample rate	1 thousand samples per second – (1 sample every $10^{-3}$ second)
Resolution	12 bits – 4096 (0–2.44 millivolts per bit)

Digital value 1000 0000 0001 – binary, \$800 – hex (2049 decimal)

Question... does  $2049 \times 2.44$  equal 5? No, digital sampling rarely comes out even. The calculated value is 4.99956.

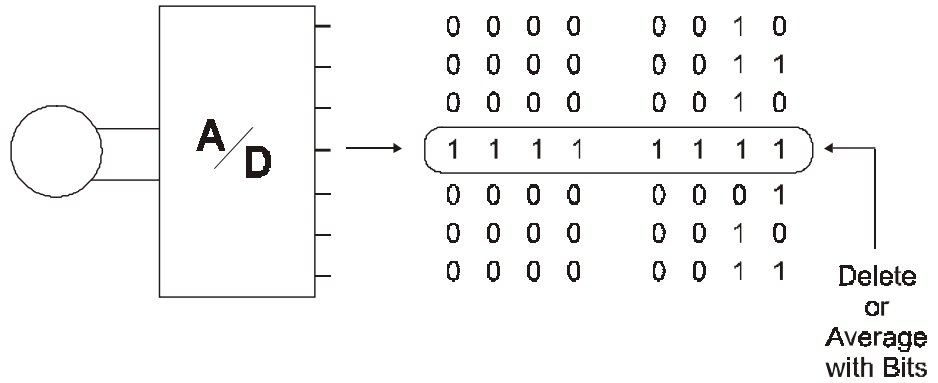
## 5.5.2 Analog and digital filtering and amplification

When working with analog values the designer can choose between analog or digital filtering. Often when a noise problem is diagnosed the engineer or technician thinks of filtering first. It has been found that on new installations it may be better to look at amplification first. One of the most common mistakes made in analog systems is improper location of the amplifier. An example of this problem could be an analog system where a sensor was being read by a microcontroller over a long distance with injected noise. If the sensor had an output of 0 to 1 volt then the signal needs to be amplified by ten somewhere in the system for a full scale of 0 to +10 volts. If the signal is amplified at the microcontroller end then the signal to noise ratio would not be good, but if the amplifier were placed at the sensor end, then the signal to noise ratio would be much better.



**Figure 5.12**  
*Analog filter schematic*

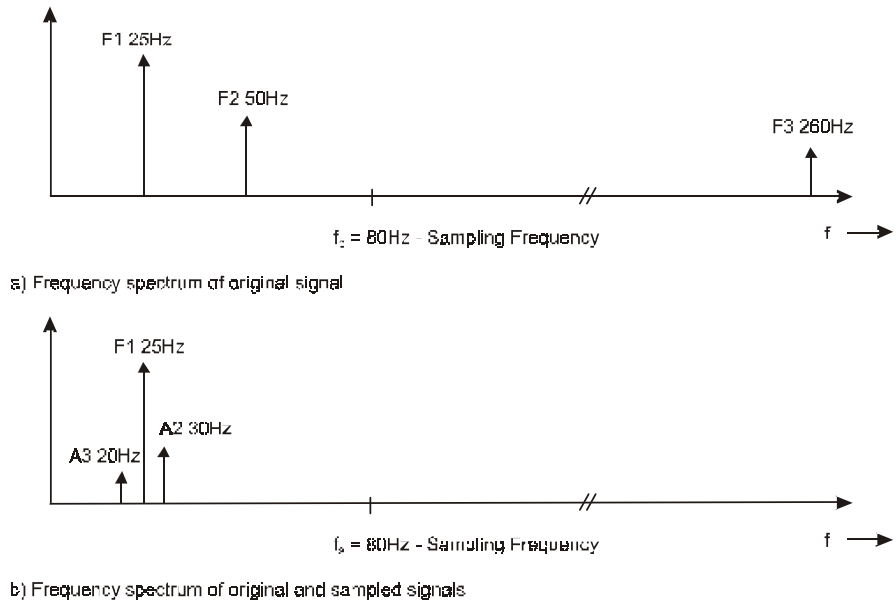
If filters are to be placed in the system, the designer needs to determine whether to use analog or digital filtering. The type of noise that is being reduced will determine which filter to use. Analog filters are good at removing frequency-related noise whereas digital filters are better at removing voltage noise. This is not to say that analog filters can't remove voltage noise or that digital filters can't reduce frequency information. Designers usually use the filter that works best for their circuit. For example if there were a spike of voltage noise induced into a circuit, it would be possible to build an analog filter to filter it. But the filter would also attenuate the signal. Whereas a digital filter could easily filter the spikes by removing the odd sample that has the spike without attenuating the signal.



**Figure 5.13**  
Digital filtering

### 5.5.3 Nyquist and the sample rate

When measuring an analog input it is necessary to know how fast to sample the values. The sample rate is a subjective measurement and therefore there is no perfect value. If the value is sampled too fast the microcontroller will receive too many samples. If the microcontroller samples too slow alias frequencies could be created. Alias frequencies are a function of the Nyquist rate. Harold Nyquist developed the rule that the sample rate of any frequency must be at least twice the highest frequency being sampled. If the sample rate is less than twice the highest frequency being sampled, then aliases, or fake frequencies, are created. Sampling of frequencies is similar to the mixing of frequencies. When frequencies are mixed, the sum and the difference of the frequencies are created. If a frequency of 1 kHz is sampled at 1.5 kHz then a 0.5 kHz alias frequency would be created. Most of the time, actual sample rates range between 5 and 20 times the highest possible frequency being sampled.



**Figure 5.14**  
Nyquist example

## 5.5.4 Resolution management

When measuring an analog input value it is necessary to match the resolution of the microcontroller, the amount of information needed and the ability of the sensor to supply the resolution. The resolution of the microcontroller defines the number of bits that will be associated with the analog value. The most common resolution values are 8, 12 and 16. At +10 volts the following values would be produced.

8-bit resolution	256 possible values	39.06 millivolts per division
12-bit resolution	4096 possible values	2.44 millivolts per division
16-bit resolution	65536 possible values	152.59 microvolts per division

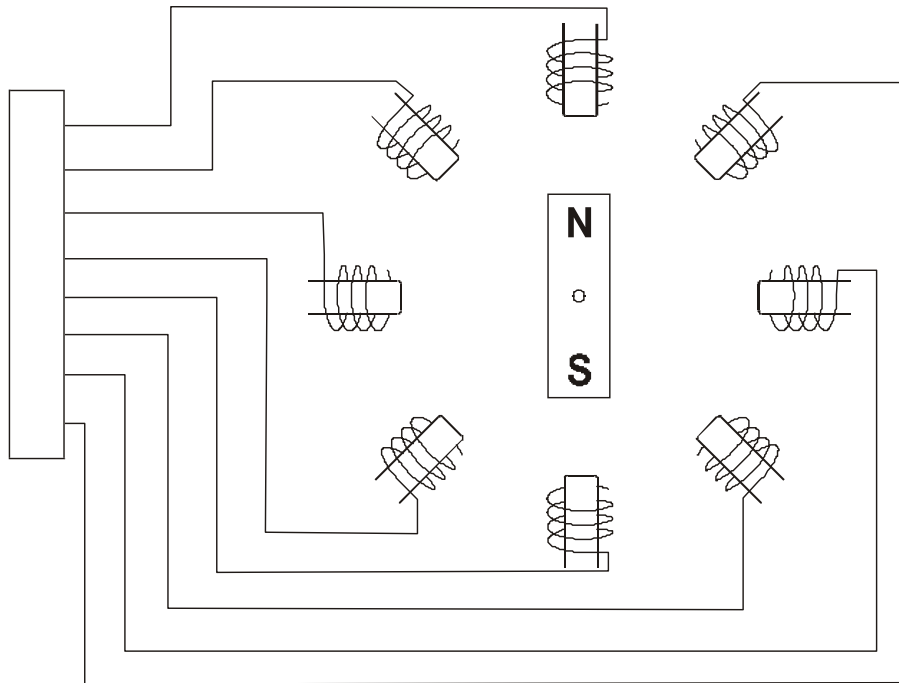
The amount of bits sampled relates to between 3 to 6 times the amount of bytes needed to store it. For example with 12-bit resolution in some devices it can take 6 k bytes of memory to hold 2 thousand samples. The matching of the sample rate of the microcontroller, the value of the highest frequency, the number of samples needed and the ability of the sensor to give that resolution will determine the ideal resolution for that system.

## 5.6 Digital control of analog devices

It was thought in the past that creating a voltage or current from a string of digital numbers would be the way we would control analog devices. It is now becoming more popular to control analog devices by turning the analog value on and off very quickly with high-speed digital control. This is made possible because of high quality timing circuits used in digital electronics today. The down side of controlling analog devices this way is that the high speed switching can be complicated and transmit a huge amount of noise. It works by reading the change in input voltage or current and then turning off or on the input to the device. An example of this could be the brightness control of an LED. The input current to the LED would be monitored and the input turned off when the current value reached a certain level. This level would determine the brightness of the LED.

### 5.6.1 Basic stepper motors

Stepper motors are another example of digital control of an analog device. Stepper motors are named because of their ability to step whereas traditional motors were either rotating or not. Turning on and off electro-magnets placed around the center rotor creates the rotation of the motor (the central magnet around the shaft can be permanent or electronic). The speed, direction and torque of the rotation are determined by how fast the outer electromagnets are turned on and off.



**Figure 5.15**  
*Stepper motor diagram*

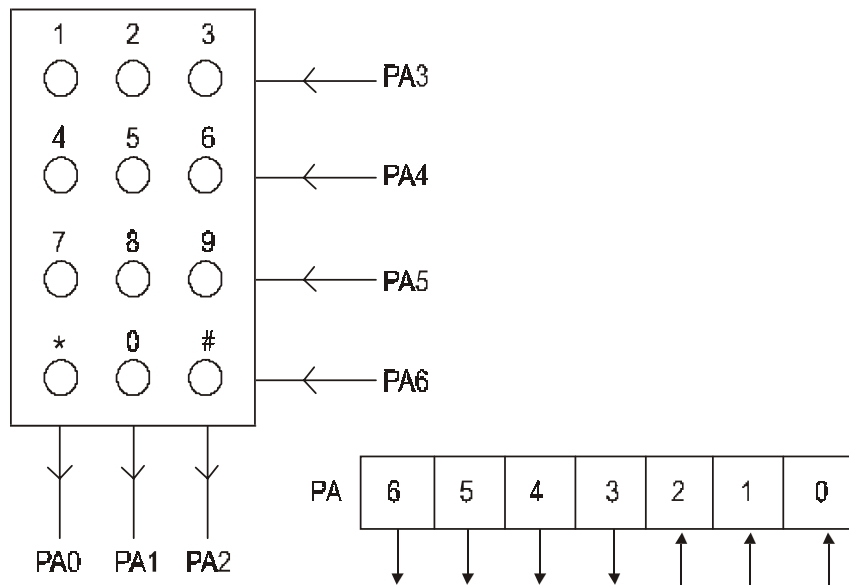
## 5.6.2 Stepper motor control and communication

Due to the high torque of DC stepper motors and their ability to change direction very quickly, they are often used in printers, plotters and robotic equipment. Often in a robot, each stepper motor would have its own dedicated microcontroller. These microcontrollers would then talk to each other over some serial bus system. The speed of the communications then becomes critical to the smoothness of the robot's movements. Using fuzzy logic and learning programs each of the stepper motors could anticipate the next function and thus could reduce the amount of data communications.

## 5.7 Keypad interfacing

### 5.7.1 Connecting the keypad to the evaluation modules (EVM)

There are three types of keypads that are connected to EVMs. They are the 12 key number keypad and the 16 key hex keypad and the full alphanumeric keyboard. The most popular is the 12 key number keypad. The 12 keypad has all 10 numbers and (0 to 9) and the # and \* keys. This keypad is laid out in a four by three method. Four inputs are on the rows of the keys and the three outputs are on the columns of the keys. This fits very well with the four output pins and the three inputs in port A on the 68HC11 microcontroller.

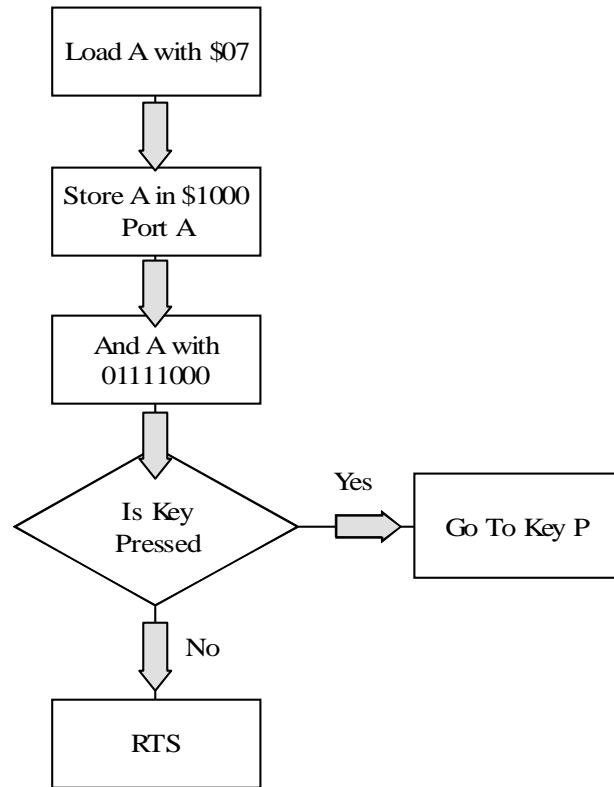


**Figure 5.16**  
Keypad diagram and wiring

## 5.7.2 Reading the keypad in software

Reading the keypad is done by first checking if any key has been pressed. One of the problems with this type of keypad is that there is no interrupt request pin that says that a key has been pressed. So to see if a key has been pressed a 0 is placed on all the inputs and then the outputs are read to see if any 0 is present. If a 0 is read on any column then it is assumed that a key has been pressed.

LDAA #\$07	Load A with 00000111
STAA \$1000	Store it in Port A
LDAA \$1000	Read port A
ANDA #\$78	Mask for output bits from keypad
CMPA #\$00	Check to see if key is pressed
BNE KEYP	If a key has been pressed then go to KEYP
RTS	If no key has been pressed then return



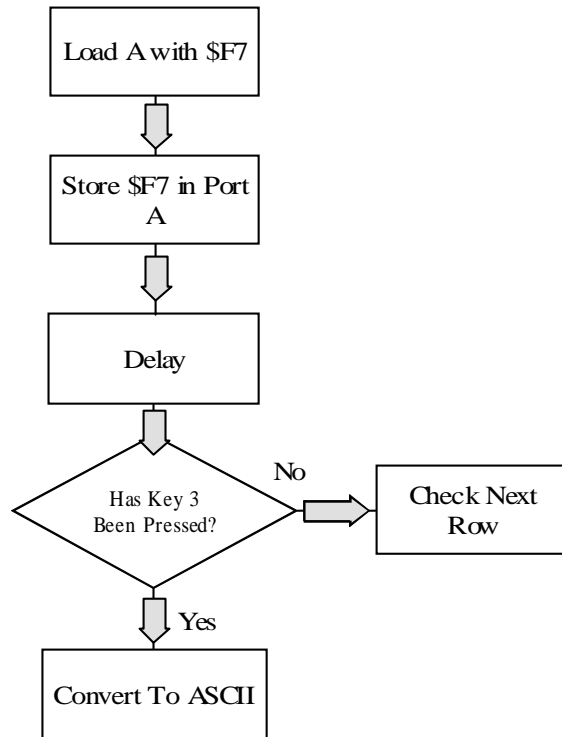
**Figure 5.17**  
*Key pressed flow chart*

The subroutine to check the keypad would have to be run quite often in the program. The keypad is only checked when this subroutine is activated. There is no delay between the storing of # $\$07$  in the accumulator A and the reading of accumulator A because we are not concerned about de-bounce at this point. De-bounce is done later when we check to see what key has been pressed. Any false key presses will be picked up later.

Once it is determined that a key has been pressed, the program would need to determine which key had been pressed. To do this, first a 0 is placed on one of the input lines. The output lines are then read. If the key is found the program branches off to the program. If not the next row receives a 0 and the column is checked. This continues until all columns are checked and the key that has been pressed is found. When the key has been found the program then converts the key press into either an ASCII or hexadecimal number.

LDAA	# $\$F7$	This load A with 1111 0111
STAA	$\$1000$	This places a 0 in bit 3 which could be row 1
JSR	DELAY	Go to keypad delay then come back here
CMPA	# $\$FE$	This checks to see if bit 0 is a 0 (column 3)
BEQ	CONVERT	If it is a 3 (row 1 column 3 is the 3 key)
BRA	NEXTC	Branch to next row check

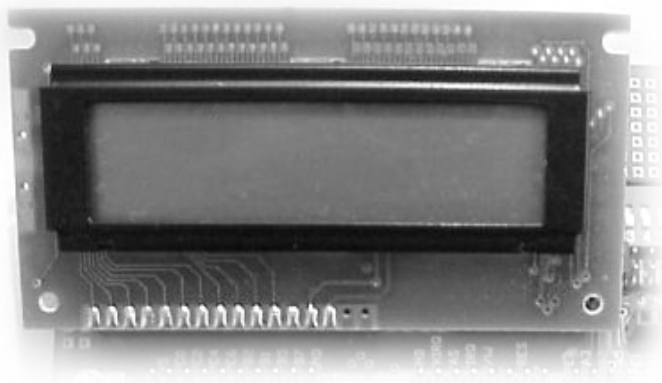




**Figure 5.18**  
*Key press check flow chart*

Notice that this subroutine only checks one key in one row. With a 12 key keypad there are three keys in 4 different rows that need to be checked. This complete subroutine to check all 12 keys would be very inefficient if written like this. The programmer could write a program that places a 0 in the first row, then checks the three columns one at a time. Then the 0 would be rotated to the next row.

## 5.8 LCD interfacing

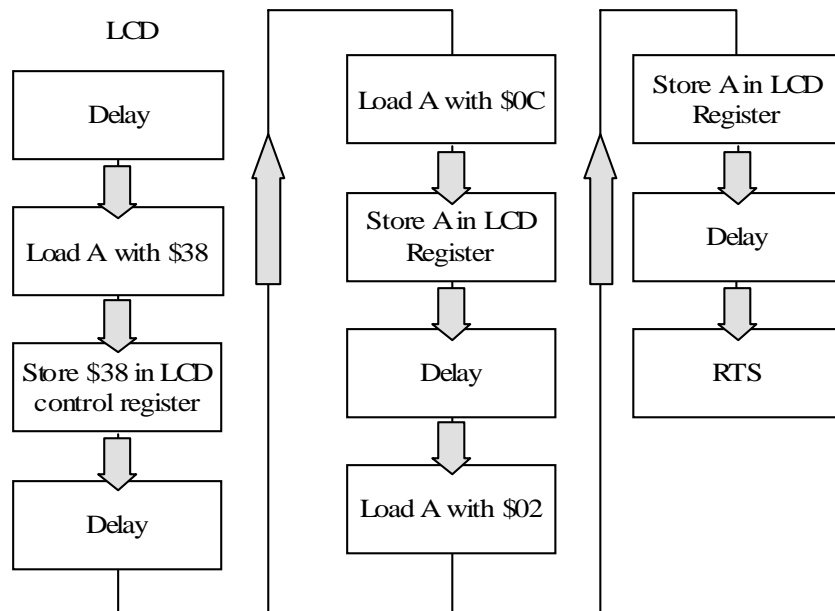


**Figure 5.19**  
*Typical liquid crystal display*

The LCD display is a very popular device for displaying information from a microcontroller. It is easy to use and is very powerful. The displays often have microcontrollers incorporated within the display itself. The most common LCD is the 16 character by two-line display. This is a misnomer because the display actually has forty characters across on each line, but only 16 can be seen. This is handy because the programmer can hide characters and then shift them over as needed. The display also has a little on-board RAM that can be accessed by the programmer. Newcomers to programming often find it difficult to get the LCD display to work correctly. This is usually because of the LCD setup sequence. The problem happens when the programmer tries to use the register setup right at the beginning. Strangely the first setup sequence on a LCD is a timed setup, from then on the programmer can use the register to determine if the LCD is ready for another character. The following example is used as a timing reset for the LCD. This should be used once and from then on a simple register check can be used.

### 5.8.1 LCD software setup

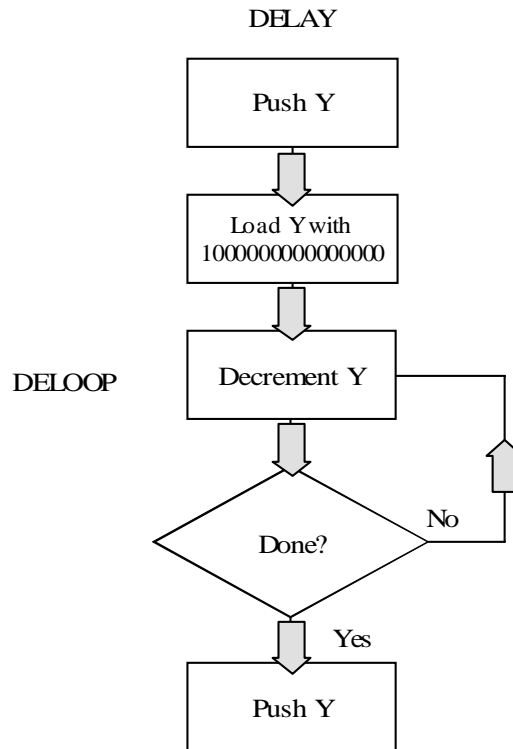
LCD	JSR	DELAY	Go to delay for LCD
	LDAA	#\$38	Loads A with first control byte
	STAA	\$8000	Store it in the LCD register
	JSR	DELAY	Go to delay for LCD
	LDAA	#\$0C	Load A with 0C for reset
	STAA	\$8000	Store in LCD control register
	JSR	DELAY	Go to delay for LCD
	LDAA	#\$02	Load 02 for reset
	STAA	\$8000	Store it in the LCD register
	RTS		Return



**Figure 5.20**  
LCD setup flow chart

A delay for the LCD might look like this...

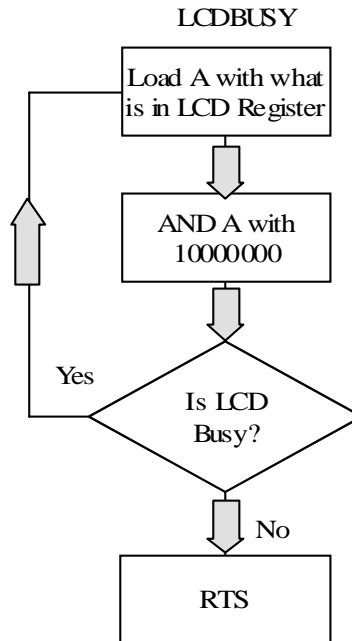
DELAY	PSHY		Push Y on stack
	LDY	#\$1000	Load Y with delay time
DELOOP	DEY		Decrement Y
	BNE	DELOOP	Check If Time = 0 if not 0 then do again
	PULY		Pull Y from stack
	RTS		



**Figure 5.21**  
Delay for LCD flow chart

A check of the LCD register to see if it is busy could look like this.

LCDBUSY	LDAA	\$8000	Get LCD register data
	ANDA	#\$80	Check if bit 7 is a 1
	CMPA	#\$80	Compare A for bit 7
	BEQ	LCDBUSY	If it is loop to LCDBUSY
	RTS		Return if not busy

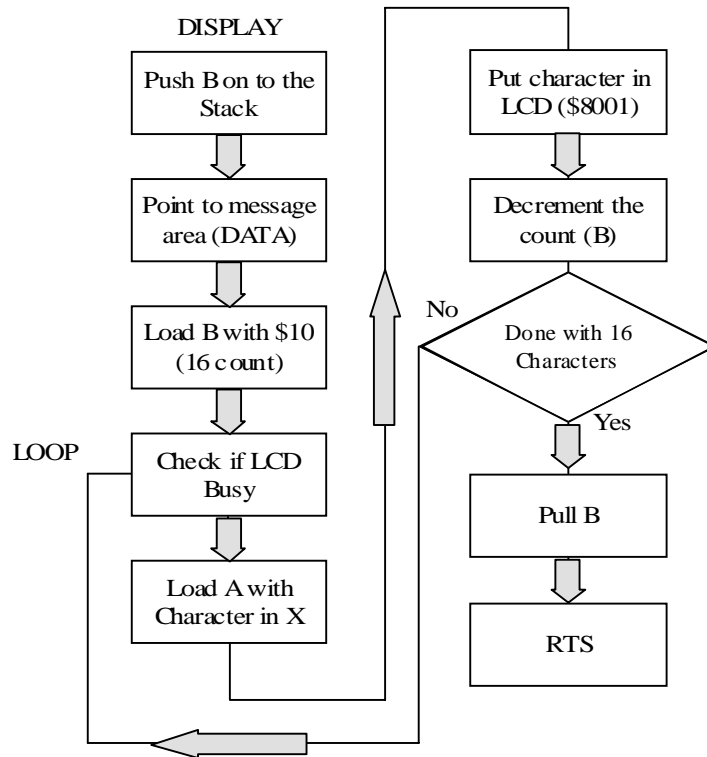


**Figure 5.22**  
Check if LCD is busy flow chart

## 5.8.2 Writing to the LCD

To write to the LCD, the programmer might place the data to be sent in a message area. When the data was to be sent, the data would then be reclaimed and sent to the LCD.

DISPLAY	PSHB		Push B on the stack
	LDX	#DATA	Points to message to be displayed
	LDAB	#\$10	Load B counter with 16 characters
LOOP	JSR	LCDBUSY	LCD Check (see above)
LDAA	\$00,X		Loads A with first byte from DATA
	STAA	\$8001	Store the character in LCD
	INX		Increment character
	DECB		Decrement B counter
	BNE	LOOP	If not done displaying then loop
END	PULB		Pull B from the stack
	RTS		Return
DATA	FCC	' — HELLO — '	This is the 16 characters message



**Figure 5.23**  
*LCD display flow chart*

## 5.9 Conclusion

Without digital and analog inputs and outputs the microcontroller has little use. By receiving data in the form of digital 1s and 0s the microcontroller is able to read switches in the real world. These switches tell the microcontroller that some event has occurred. This event could be just about anything and as far as the microcontroller is concerned it is just an event. It is up to the software to interpret that event into a situation that the user can understand. Analog inputs are voltage measurements of inputs from sensors. These analog input values from the sensors are sampled at some resolution and then stored in a database. The database is used by the upper level software and is shown to the user as graphs, charts and on alphanumeric displays. The digital outputs are used to turn things on and off. Before computers, almost all control was done using on or off signals. This tradition has continued with microcontrollers. In the early days of digital electronics it was assumed that in the future most control would be digital to analog. Instead digital control continues today and probably will be the main method for controlling devices in the future.

Keypad and LCD display systems are the two examples of interfacing to a microcontroller. The keypad used in this example was a 12 key unit with 0 to 9 and # and \*. The # and \* are used in programs for functions like redisplay menu and return to top of program. It is not uncommon to connect a 16 key keypad and even full size keyboards to microcontrollers. These input devices let the user input data and make decisions for the microcontroller. LCD displays are used to show information to the user in an easy and clear manner. There are some little tricks to interfacing the LCD to the microcontroller but once the programmer becomes familiar with the LCD they become easy to interface.

---

# 6

---

## Data communications

### Objectives

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

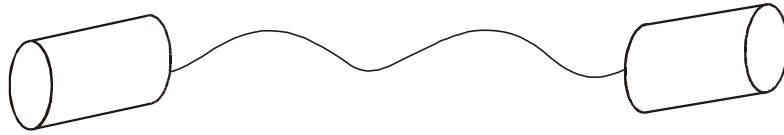
- Explain the three basic parts of a data communication system
- Describe the open system interconnection model
- Describe the three modes of communication
- Describe the four types of bus systems
- Describe the difference between RS-232 and RS-485
- Explain serial communications between microcontrollers
- Explain fieldbus systems and how they relate to microcontrollers

### 6.1 Introduction to data communication

In troubleshooting and design of microcontroller equipment it is important to understand some of the basics and the main functions of data communications systems. Data communication systems have become a required part of the overall controller system. In some ways data communications has changed little in the last thirty years. We are still moving serial data with changing voltages or currents on copper wires. In other ways it has changed a lot, such as the advent of fiber optics and infrared. Thirty years from now we will probably be doing data communications completely different than we do it today.

At the end of the day data communications can be thought of as two tin cans and a piece of string. To the programmer the data communications system may seem invisible. The programmer creates the packet and then sends it to the data port. What happens to the data after that is not usually the programmer's concern? In the past speed, distance and noise have limited serial data communications. This is slowly changing due to better electronics and the industry wide move to fiber optics as a transmission medium. Speeds of 100 Mbs and higher are now common and soon gigabyte speeds will become standard. With the old RS-232 voltage standard, we were limited to distances of 50 meters or so, but with the advent of the RS-485 voltage standard two kilometers were possible. Now

with fiber optic data communications, tens and even hundreds of kilometers are possible without repeaters. Because noise does not affect fiber optic cable, we now have systems that are ideal for sending high-speed long distance digital information.



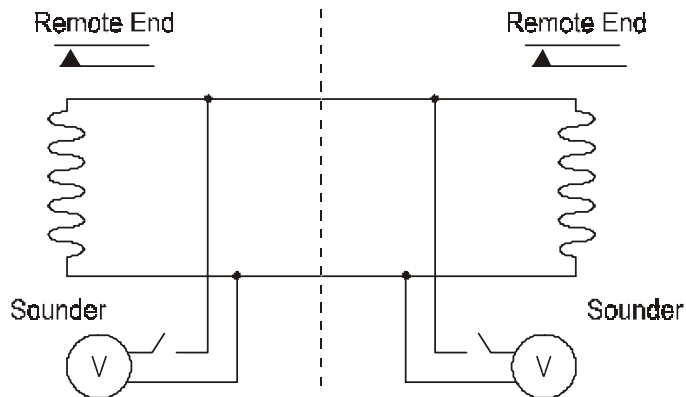
**Figure 6.1**  
*Two tin cans and a piece of string*

## 6.2 Basics of serial data communication

### 6.2.1 History of serial data communications

It is not the purpose of this book to spend a lot of time on the history of data communications, but it is important to understand today's data communication systems. And sometimes it makes it easier to understand the systems of today by knowing where they came from.

The first electric data communication system was the telegraph system. It had all the components that we have in modern data communication systems. The key transmitted the data, the sounder (a relay connected to a metal tin) was the receiver and the wire that connected the two together carried the data. The language that was used to send the data was Morse code. The protocol (rules for sending the data) was similar to the one used in Ethernet. The problem with the telegraph system was the speed and the human component. People can send and receive only so fast using a key and if no one was there to receive the data, it wasn't received.



**Figure 6.2**  
*Telegraph system*

To overcome the human and speed problem, devices were invented such as the teletypewriter and then the teletype. One problem with using a machine to send data is that the old Morse code didn't work very well. Morse code has a different number of bits for different characters. A machine requires the same number of 'bits' for each character. The Baudot code was developed for teletypes because it had five bits (and a shift function) for each character. When computers were invented in the sixties and seventies it was found

that the Baudot code was too limiting because it only had 62 characters and therefore a new code was needed. ASCII became the dominant code for computers. It has up to 256 characters with a 128 basic set and another 128 extended set. The need of connecting computers together with some type of data communication system led to the development of systems like Ethernet and then the World Wide Web.

**6.2.2 Three parts of data communications**

There are three basic parts to every data communication system. They are the code, the voltage standard and the protocol:

- Codes Morse – Baudot – Hex – ASCII
- voltage standards RS-232 – RS-422 – RS-485 – Ethernet
- Protocols Modbus – Profibus – DeviceNet – Ethernet

Notice that Ethernet is both a voltage standard and protocol. It is fairly rare that a protocol defines its own voltage standard. The protocol may or may not define a code and voltage standard. For example Modbus defines two versions (Modbus A and Modbus B) where Modbus A uses ASCII and Modbus B uses hex as a code. Neither Modbus A nor B defines a particular voltage standard. This means that a design of a Modbus system can use any voltage standard. Profibus FMS for example defines RS-485 as the voltage standard and ASCII as the language.

6.2.2.1 Code standards

The most common code used in data acquisition systems in the eighties was hexadecimal. Because data rates increased and the users needed to send more data, designers changed to the ASCII code in the nineties. These higher speeds negated the fact that ASCII uses twice as many bits as hex for the same character. The advantage of ASCII is that more letters or characters can be sent than when using the hexadecimal code. For example ASCII has all the letters, numbers and punctuation that might be needed to send the serial number of a device. The problem with ASCII is that it is an English code and the world speaks hundreds of languages. With the advent of the WWW it is becoming increasingly important to have a code that all people can use. To this end the ITU universal code was developed and probably will become the excepted standard code in the future. The universal code has 65536 characters.

	HEX	0	1	2	3	4	5	6	7
<b>HEX</b>	<b>BIN</b>	<b>000</b>	<b>001</b>	<b>010</b>	<b>011</b>	<b>100</b>	<b>101</b>	<b>110</b>	<b>111</b>
0	0000	(NUL)	(DLE)	Space	0	@	P	`	p
1	0001	(SOH)	(DC1)	!	1	A	Q	a	q
2	0010	(STX)	(DC2)	"	2	B	R	b	r
3	0011	(ETX)	(DC3)	#	3	C	S	c	s
4	0100	(EOT)	(DC4)	\$	4	D	T	d	t
5	0101	(ENQ)	(NAK)	%	5	E	U	e	u
6	0110	(ACK)	(SYN)	&	6	F	V	f	v
7	0111	(BEL)	(ETB)	'	7	G	Q	g	w
8	1000	(BS)	(CAN)	(	8	H	X	h	x
9	1001	(HT)	(EM)	)	9	I	Y	i	y
A	1010	(LF)	(SUB)	*	:	J	Z	j	z
B	1011	(VT)	(ESC)	+	;	K	[	k	{
C	1100	(FF)	(FS)	,	<	L	\	l	
D	1101	(CR)	(GS)	.	=	M	]	m	}
E	1110	(SO)	(RS)	.	>	N	^	n	~
F	1111	(SI)	(US)	/	?	O	_	o	DEL

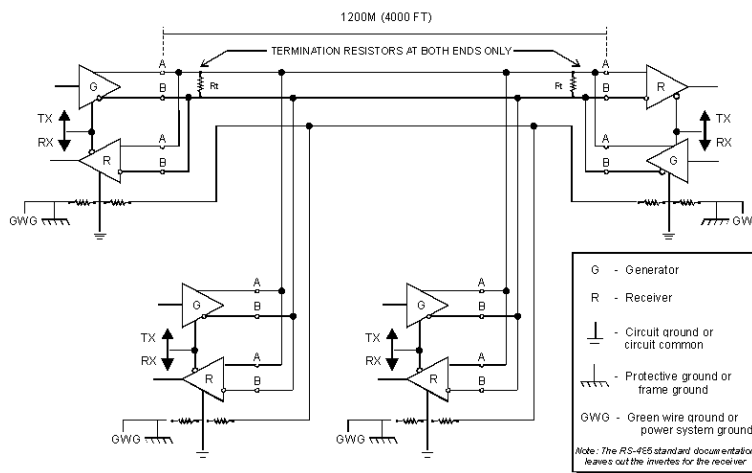
**Table 6.1**  
*The ASCII table*



### 6.2.2.2 Voltage standards

Voltage standards changed little until the microchip revolution. When the transistorized differential amplifier was developed, the range and speed of data communication increased tremendously. RS-232 uses a single ended grounded method that is very susceptible to noise and this limits its speed and distance. RS-422 and RS-485 use a differential system that is more resistant to noise and therefore can send data faster and further.

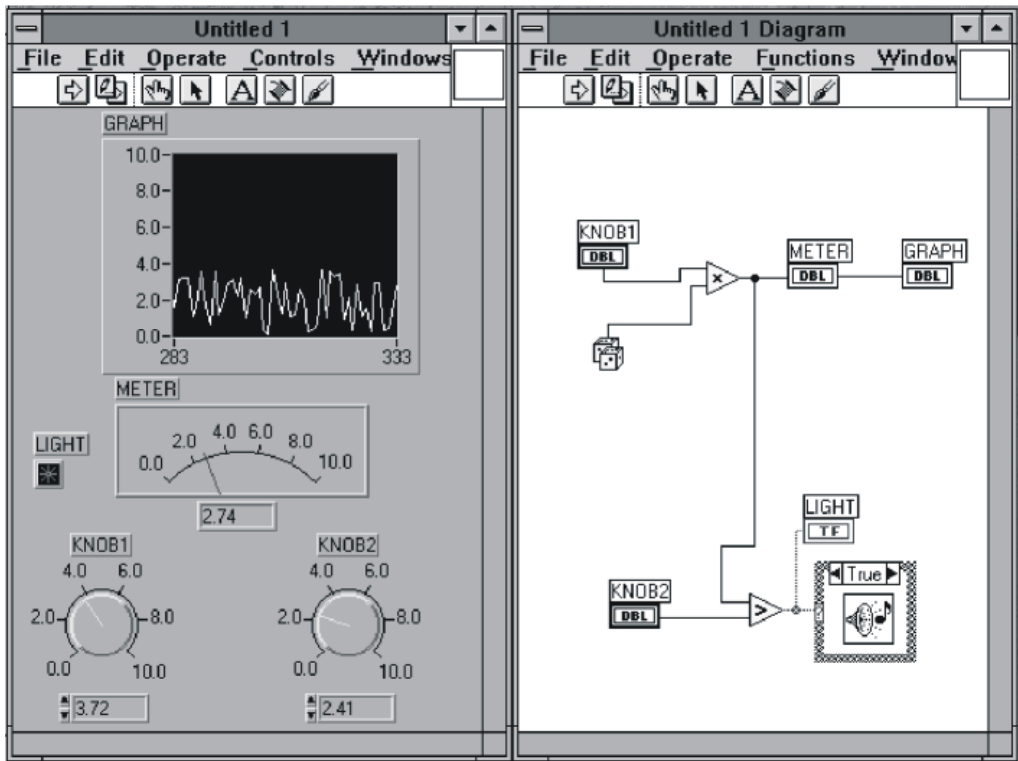
Voltage standards can be divided up into two types, voltage standards like ITU, IEEE and EIA and semi-standards like RS-232 and RS-485. A few protocols have developed their own voltage protocols but most use a standard or semi-standard voltage standard. Often the question is asked ‘why we don’t use standard voltage systems and instead use semi-voltage standards like RS-485?’ This is usually because true standards are often either very limiting or have very little relation to real life. A good example of this is the EIA-422 standard. The true textbook RS-422 voltage standard is almost unrecognizable and unusable when compared to RS-422.



**Figure 6.3**  
*RS-485 system*

### 6.2.2.3 Protocols

Protocols have gone a long way towards standardizing data communication systems, but having said this, there are hundreds of protocols available. There are those in industry that would like to see one protocol for all data communications. This would be like having one language for the whole world. The alternative is to have a universal translator that can change one protocol to another. There are software programs that do this now. For example the program Citect has multiple device drivers that allow many different protocols to be used on one system. The protocols are all translated at the software level.

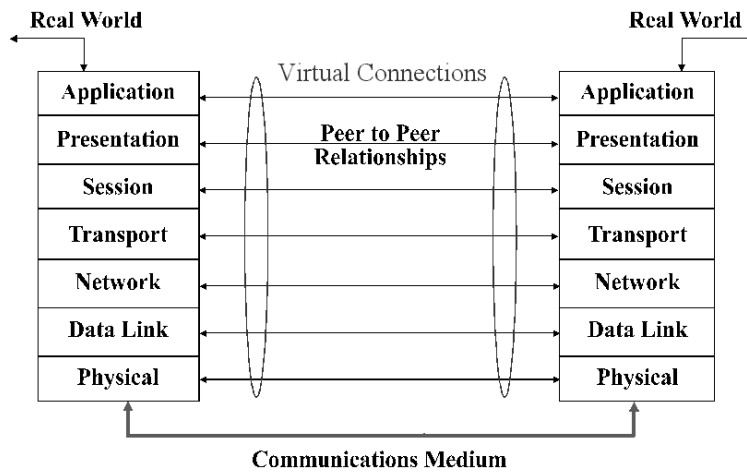


**Figure 6.4**  
Citect software system

In the area of data acquisition and control systems protocols can be divided into two camps, open standard protocols and proprietary protocols. An example of an open non-proprietary standard protocol could be Profibus or Foundation Fieldbus. These standards do not belong to any manufacturer but are open standards that anyone can use. Protocols like Allen Bradleys Data Highway Plus or Seimens SP-5 are proprietary protocols. These protocols are owned and maintained by their manufactures and are part of a complete system that they sell. The obvious problem with using these proprietary protocols is that the customer is locked into the complete system and one manufacturer. With the open standard protocols the user can buy the hardware from any manufacturer and even have multiple protocols running in the same system. The open protocols (and therefore the controllers) can be tied together with an open proprietary software package like Citect, Wonder Ware or Intelution. These software programs use the application and datalink layers of the OSI model.

### 6.3 Open system interconnection model

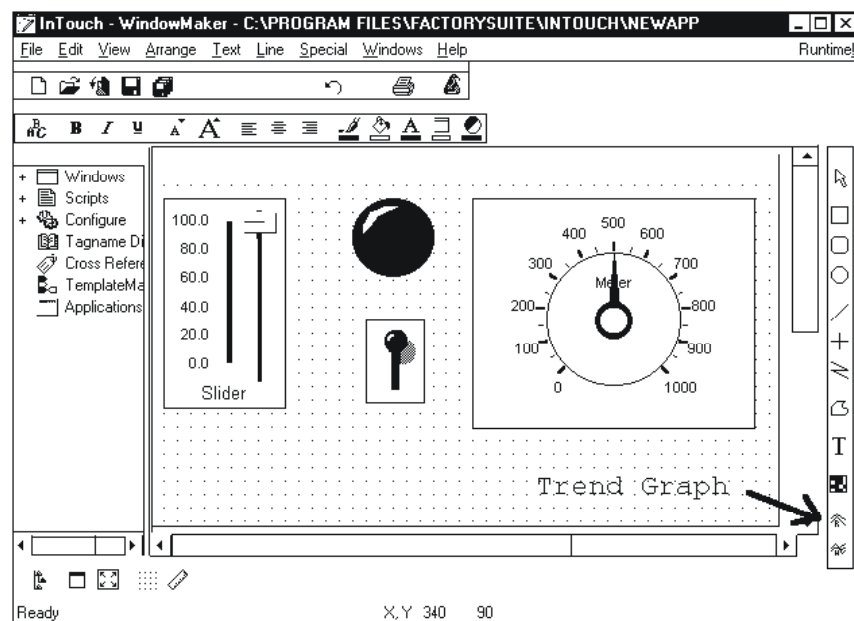
The OSI model is the International Standards Interconnection model. This model describes a working data communication system in the form of different layers. The model helps us understand the overall functions of each part of a data communication system and how they interact with each other. In Figure 6.5 the application layer is at the top with the physical layer at the bottom. From the datalink layer up the first five layers are software. The physical layer and half of the datalink layer are hardware. The datalink layer usually is a combination of both software (device drivers) and physical hardware (the communication port in the controller).



**Figure 6.5**  
*OSI model*

### 6.3.1 Application layer

The application layer is the upper layer software application that could include information that would be used to develop software programs like Citect and Wonder Ware. The application layer has physical communications with the layers below it while at the same time it has a logical communication with the application layer on the receiving device. This means that the application layer talks to the layer below it as though it was talking to the application layer on the other end. In the same way that a caller talks to the person on the other end of the telephone line as though the telephone didn't exist. We talk to the person not the telephone.



**Figure 6.6**  
*Wonder Ware example*

### **6.3.2 Session, presentation, transport and network layers**

All layers in the OSI model have their own functions, although in some systems the layers are combined. Often in controller communications some layers are not used.

The reason for this is as follows:

- The session layer is not used because this function is usually contained in the datalink layer (session overheads)
- The presentation layer is not used because encryption and compression are rarely needed
- The transport layer is not usually used because microcontrollers rarely communicate with other networks. They usually stay within one network and if they do connect to other networks they would use another protocol like Ethernet
- The network layer is not used because since the data stays in one network it does not need an intra-netting or inter-netting addressing system like IP or IPX. Again if the data is going between different networks, Ethernet or some other protocol is usually used

### **6.3.3 Datalink layer**

The bulk of what makes up a protocol lives within the datalink layer of the seven-layer model. The datalink layer is divided into separate layers, the logical link control layer and the media access control layer.

The logical link control layer is the software (or firmware) side of the datalink layer. This layer is where the program that creates the communications in the controller resides. The logical link control layer may consist of a low level device driver in a computer or a subroutine in the microcontroller that interfaces the application layer to the media access layer in either a card or module.

The media access layer is the hardware layer in the controller or card in the computer that accesses the physical media. This could be a UART (universal asynchronous receiver transmitter) within the controller or an RS-232 card in the computer. In synchronous systems a USART (universal synchronous asynchronous receiver transmitter) could be used. These chip sets vary greatly and depending on the protocol and physical layer used the designer might use a synchronous or asynchronous system.

### **6.3.4 Physical layer**

The physical layer of the OSI model can consist of cards, modules and the wire of the system. The voltage standard resides within the physical layer although the physical system could be either a current or light transmission system. The physical layer can include driver chips, repeaters, connectors and wire. The physical system determines the speed of the data, the distance and the number of devices that can be connected together. The RS and EIA-232, 422 and 485 standards are combinations of components made up of the media access layer and the physical layer with emphasis on the physical layer. Often a software programmer might see the physical layer as just a pipe or two tin cans and a piece of string. They write the program without regard to what kind of physical layer is going to be attached. They just dump the data into the media access layer via the logical link layer and let the hardware engineer deal with it.

### 6.3.5 Protocols and the three layer model

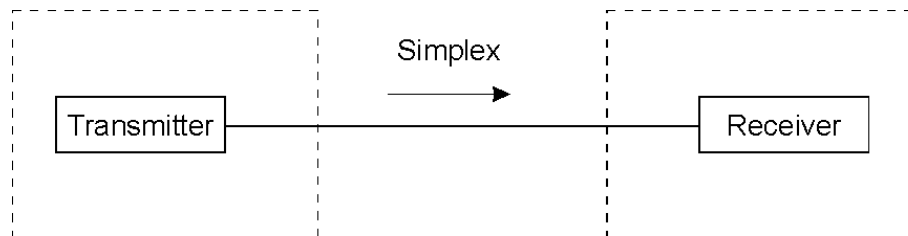
As we can see from the previous pages the seven-layer model has become a three-layer model with the application layer, datalink layer and the physical layer as the main components. The protocol of a control system is mostly a datalink commodity with access to the upper application layer and lower physical layer through something called 'service access points'. These service access points are usually a small software interface (like a device driver) between the different layers of the OSI model. Some protocols like Ethernet or Profibus include a physical layer in the protocol. Whereas others like Modbus provide only a datalink and a bit of application layer. Neither of these examples includes a true application layer. This would be too limiting for the designer. It would be ridiculous to have a system where only one application is allowed to access Ethernet, Profibus or Modbus. Consequently all open protocols have a datalink layer, some have a dedicated physical layer but none have a dedicated application layer.

## 6.4 Modes of communications

Data communications in general has three modes, simplex, half-duplex and full-duplex.

### 6.4.1 Simplex

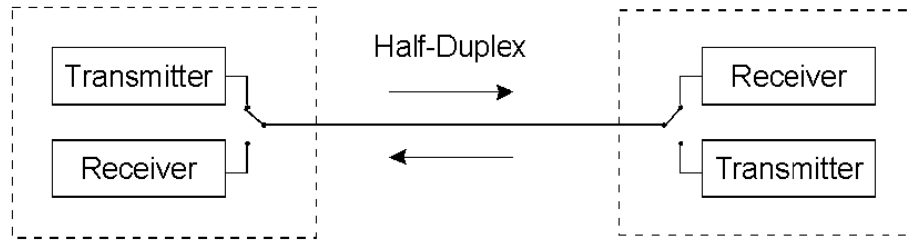
Simplex is used when it is not necessary to get any information back from the receiver. The transmitter sends out data, but nothing can come back. It is a one way communications system. Simplex is rarely used because we almost always need information back from the receiver.



**Figure 6.7**  
*Simplex mode*

### 6.4.2 Half-duplex

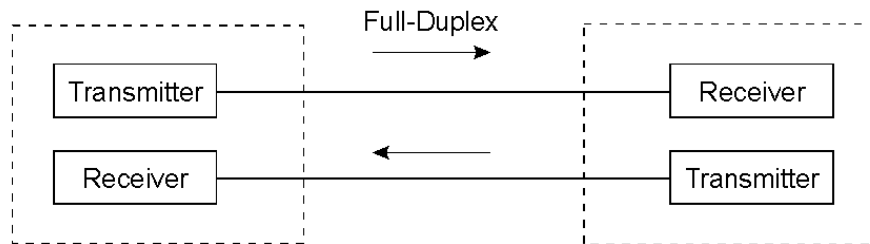
Half-duplex is the most common communication mode in use today. The transmitter sends data to the receiver and then the receiver answers back with an acknowledgment or other data. This method often uses a two wire multidrop system like RS-485 but can be used with three wire (RS-232) or four wire (RS-422) point-to-point systems. Ethernet and fieldbus systems also use half-duplex. It is used in almost all master/slave, CSMA and Tokenbus systems. This half-duplex system is simple and easy to control. Because in a half-duplex system only one device is supposed to be on the line at a time, the chances of two or more devices conflicting with each other are minimized.



**Figure 6.8**  
*Half-duplex mode*

### 6.4.3 Full-duplex

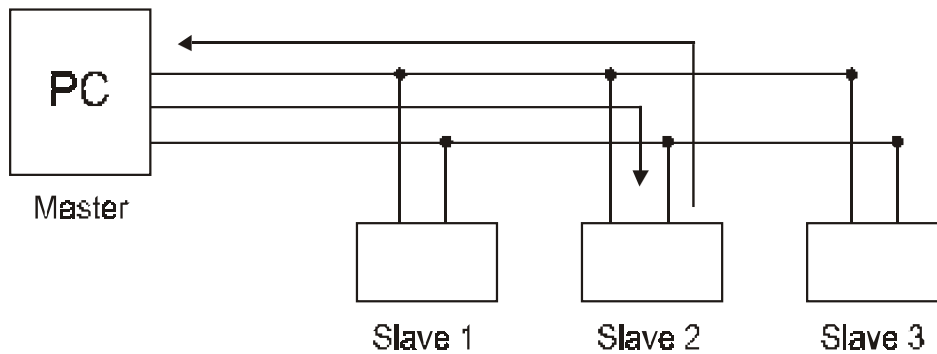
Full-duplex is rarely used by communication systems because of the difficulty in controlling the conversation. In true full-duplex communications both the transmitter and receiver would be able to talk at the same time. This is fine if only two devices are connected together, but when three or more are multidropped full-duplex becomes very difficult, if not impossible. The good thing is that with very fast half-duplex systems like Ethernet, from the user point of view, the system appears to be working in full-duplex.



**Figure 6.9**  
*Full-duplex mode*

### 6.4.4 The master slave bus

The master slave bus system is a very common method of doing multi-drop communication. Master/slave protocol rules state that the master can talk to one or more slave but the slaves can only talk back to the master. The slaves are not allowed to talk to each other. This communication is typically done half-duplex. The master/slave communication system uses a polling method. The master sends out a poll to the slave and then the slave responds with either data or an acknowledgment. The master polls all of the slaves one at a time in a round robin method. The poll goes on forever or until the user stops it to talk directly to a slave via the software. The system is very deterministic because the programmer or the user can determine which device talks and when.

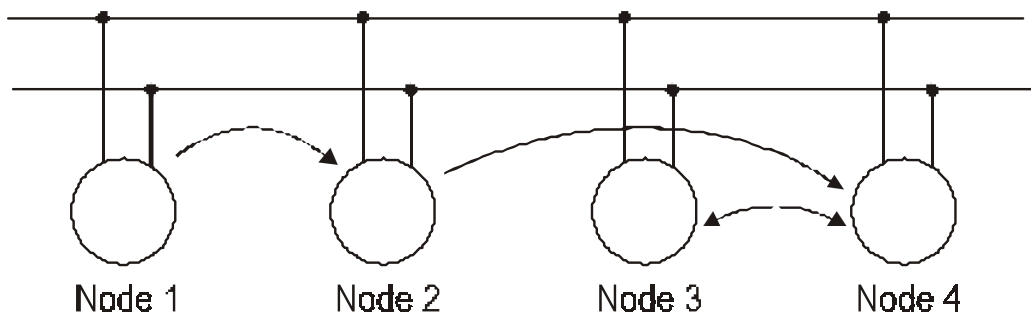


**Figure 6.10**  
*Master slave diagram*

The master/slave system is very popular because of its simplicity and determinism. Programmers like the master slave system because it is easy to write and control. It fits nicely within sequential programming methods. Fieldbus users like the master/slave system because it is easy to understand and all the devices on the system are continually checked. The users also have a lot of control over what device communicates and when. The down side of the master/slave system is the time it takes to do a complete poll. If a device is polled and then goes into an alarm situation, the master and therefore the user will not hear about it until the master gets around to it on the next poll. A typical example of the master/slave system is Modbus.

#### 6.4.5 The CSMA/CD bus

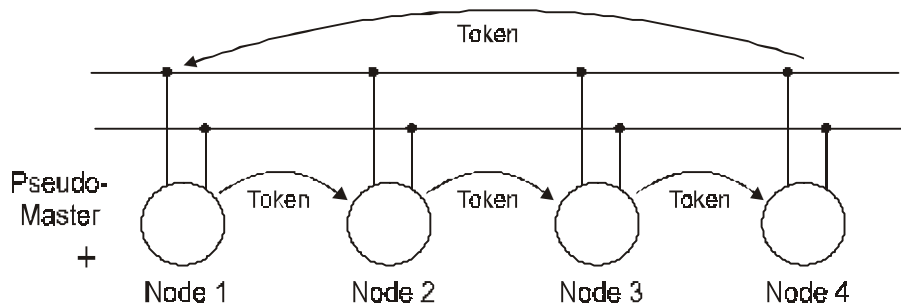
CSMA/CD stands for carrier sense multiple access with collision detection. Carrier sense means that the devices (nodes) that are connected on the multidrop system listen to the line and if another device is transmitting it has to wait. The multiple access means that any node can talk as long as no one else is on the line. There is no master or slaves in this system, only nodes. The collision detection is the method the system uses to recover when two or more devices try to access the line at the same time. The advantage of the CSMA/CD system is that any node can talk whenever it wants. At low traffic level this system works very well. The most common CSMA/CD system is the Ethernet system. The problem with CSMA/CD systems is when the traffic reaches critical levels (30%) the system stops running and must be reset. This can be catastrophic in an industrial controller system.



**Figure 6.11**  
*Ethernet (CSMA/CD)*

### 6.4.6 The token bus system

Token bus is a very popular communication system that connects multiple nodes together on one system. The main rule of a token bus system is that any node can talk to any other node as long as it has the token (a short packet that everyone sees as the token). The token bus system must have a pseudo-master. One of the functions of the pseudo-master is to create the token and send it to the first node. That node then can talk to any other node while it has the token. When the node is done sending data or its time is up, it must send the token to the next. The token is passed around the system from one node to another until it returns to the pseudo-master. Token bus is very popular because it has most of the advantages of both the master/slave and the CSMA/CD systems. It is deterministic and yet any node can talk to any other node. The biggest disadvantage of this system is that a node has to have the token to talk and must wait until it comes around. Often in a token bus system more than one node can be the pseudo-master, but only one device can act as the pseudo-master at a time. This means that if the pseudo-master is disconnected, another node can take over control. Profibus is one of the most popular non-proprietary open token bus communication protocols on the market today.

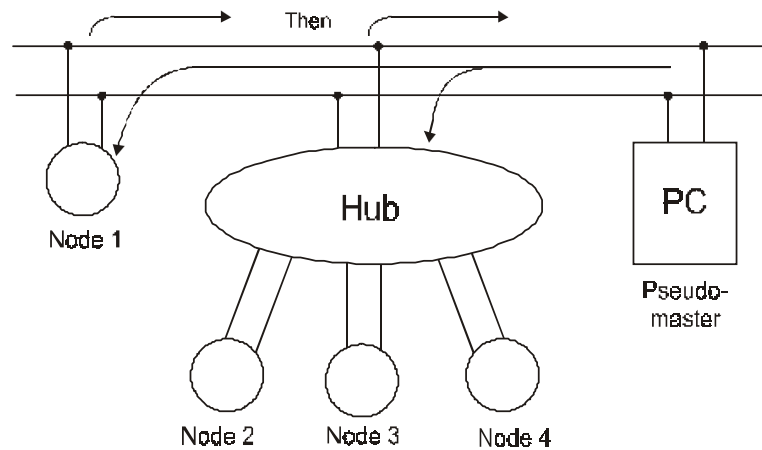


**Figure 6.12**  
*Profibus (token bus)*

### 6.4.7 Timed systems

Timed communication systems are the fastest data transfer systems on the market at the moment. With a timed system the link active scheduler (LAS) creates a packet where each node is allocated a portion of time for its information. This is sent to the nodes and then the nodes respond in time to the LAS. The LAS determines who talks and how long. For example the USB communication system on your computer may only need data from the keyboard every 100 milliseconds or so but on the other hand it may need information from the CD player more often. The LAS would configure the system so that the CD player talks more often and the keyboard less often. The good thing about timed system is that it is very fast and the data to overhead ratio in the packet is extremely good. One problem with timed systems is that they rely on a master LAS and if the master dies the system stops.





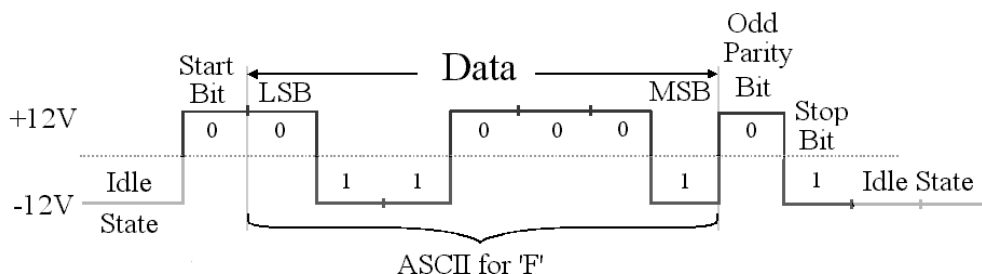
**Figure 6.13**  
*USB (timed)*

## 6.5 RS-232

### 6.5.1 Introduction to RS-232

RS-232 is often used to communicate from the PC to a controller. It is an easy and simple asynchronous communication system. One of the confusion factors with RS-232 is that there are two versions, EIA and RS. The system was developed in the sixties and submitted to IEEE by Bell Labs. Unfortunately it was not accepted and from then on was known as RS-232 (Recommended Standard 232). The 232 is just a number IEEE given to it during the acceptance process.

The voltage levels of RS-232 (we will be discussing RS and not EIA here) are well known as +25 volts to -25 volts with an undetermined area of +3 volts to -3 volts. This means that if a voltage between +3 volts and -3 volts is received the UART (universal asynchronous receiver transmitter) will not be able to determine if it is a 1 or 0. The idle voltage of RS-232 is a minus voltage and is usually represented by a 1, although different systems represent it differently. A positive voltage usually is seen as a zero. RS-232 uses a three-wire transmission method where the transmission and receiver lines are referenced to a common ground. This makes it very susceptible to noise from the ground or from outside signals. Usually we do not run RS-232 more than 50 feet, but this varies depending on the speed of the transmission and the amount of noise in the area.

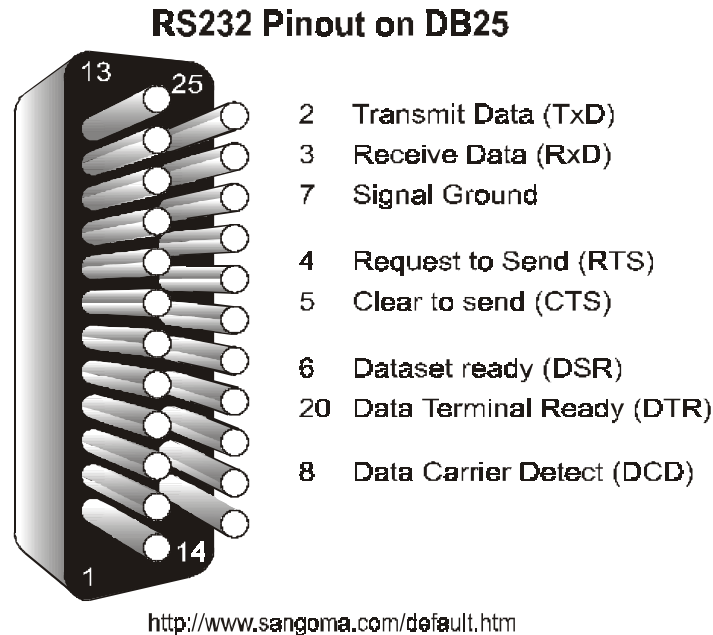


**Figure 6.14**  
*RS-232 character transfer*

A common confusion factor in RS-232 is the use of the 9 lines. RS-232 is not a protocol it is a voltage standard. RS-232 tells what the lines are, but not how to use them. There are as many ways to use the lines, as there are line combinations. There are three main types of lines in RS-232.

### 6.5.2 Function of the lines

- Indicator lines (RI, CD, DTR and DSR)
- Control lines (RTS and CTS)
- Transmission lines (TX, RX and C)



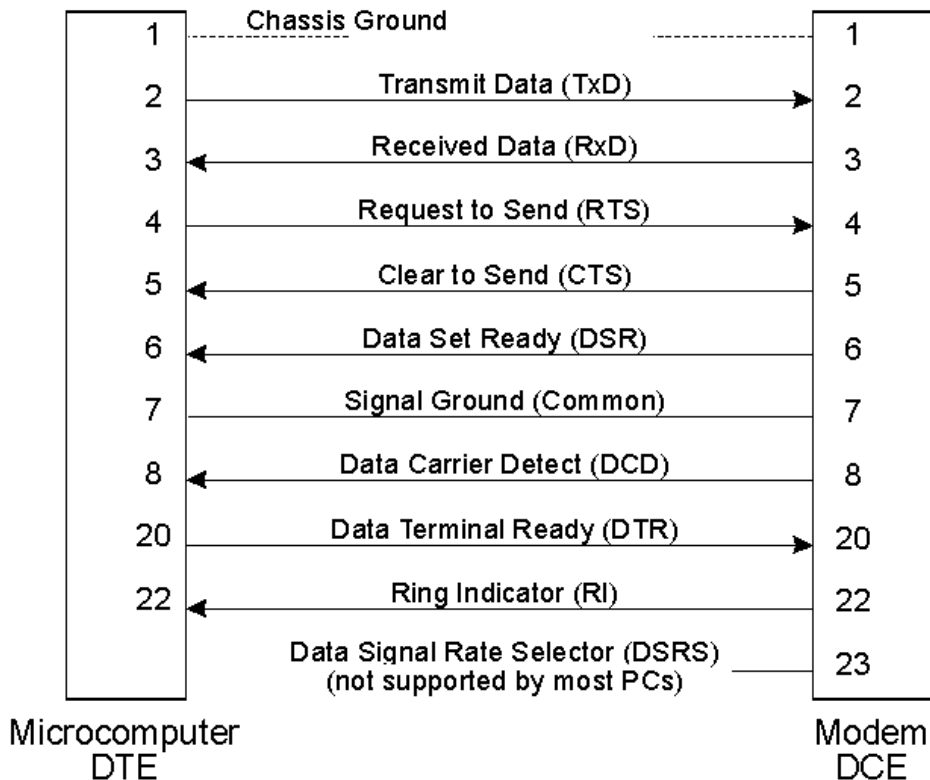
**Figure 6.15**  
*RS-232 pin outs*

It is important to remember that in RS-232 the lines are one-way lines. There are no bi-directional lines in RS-232. The ring indicator line only goes from the DCE to the DTE. The DTE is the data terminal equipment. The key word here is terminal. It comes from the Latin for the end of something, like train terminal. The terminal in this case is the computer, controller or PLC to name a few. The DCE is the data communication equipment. This originally was the equipment in the telephone exchange where the modems were located. That is why all modems are configured as DCEs.

### 6.5.3 RS-232 installation and troubleshooting

Since RS-232 is a point-to-point system installation is straightforward and simple. All RS-232 equipment use either db9 or db25 connectors. These connectors are used because they are cheap and allow multiple insertions and disconnections. None of the 232 standards define what device uses a male or female, but traditionally the male pin connector is used on DTE (terminal) and the female socket connector is used on DTE (modem) equipment. This is only traditional and it may vary on different equipment. It is often asked why use a 25-pin connector when only 9 pins are needed on RS-232? This

was done because RS-232 was used before computers and therefore used hardware control (RTS / CTS). It was originally thought that in the future we might need more hardware control lines, hence the need for more pins.



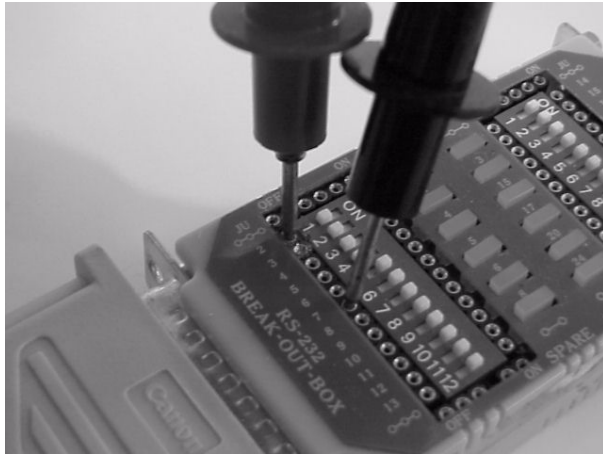
**Figure 6.16**  
RS-232 lines

When doing an initial installation of an RS-232 connection it is important to note the following specifications:

- Is one device a DTE and the other a DCE?
- What sex and size of connectors are located at each end?
- What speed is the communications?
- What is the distance between the equipment?

To determine whether the devices are DTE or DCE, connect a breakout box at one end and note the TX light (pin 2 or 3) on the box. If it is on pin 2, the device is probably a DTE. If it is on pin 3 it is probably a DCE. Another clue could be the sex of the connector, male DTE or female DCE, but not always.

The speed and distance of the equipment will determine if it is possible to make the connection at all. Most engineers try to stay less than 50 feet or about 16 meters at 115 kbs. This is a very subjective measurement and will depend on the cable, voltage of the transmitter and the amount and noise in the environment. The transmitter voltage can be measured at each end when the cable has been installed. A voltage of at least  $\pm 5$  volts should be measured at each end on both the TX and RX lines.



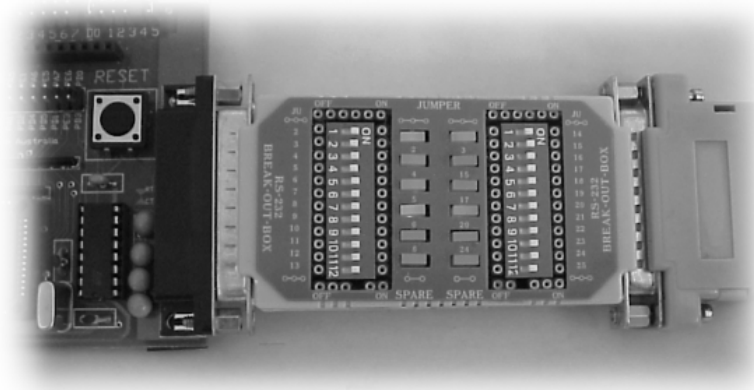
**Figure 6.17**  
*Measuring the voltage on RS-232*

Once it has been determined that the communication is connected as DTE to DCE and that the distance and speed are not going to be a problem, the cable can be connected at each end. The breakout box can still be left connected with the cable and both pin 2 and 3 lights on the breakout box should now be on.

The color of the light depends on the breakout box. Some breakout boxes use red for a one and others use green for a one.

If only one light is on then that may mean that a wire is broken or there is a DTE to DTE connection. A clue that a DTE to DTE connection has been made would be that the light on pin 3 would be off and the one on pin 2 would be on. To correct this problem first check the wires for continuity, then turn pins 2 and 3 off on the breakout box and use jumper wires to swap them. If the TX and RX lights come on, a null modem cable or box will need to be built and inserted in-line with the cable.

Once the cable has been connected correctly, try the communication and see if it works. It might be prudent at this point to see if data is being sent, by looking at pin 2 on the breakout box. Be careful here because it is possible that the data is being transmitted so fast that the light on the breakout box doesn't have time to change. If it is possible, lower the speed of the communication at both ends to something like 1200 bps. If one device is transmitting but the other receiver is not responding then the next thing to look for is what type of control the devices are using. The equipment manual may define whether hardware or software control is being used. Both ends should be set up for hardware control, software control or none.



**Figure 6.18**  
RS-232 breakout box

If pin 2 and pin 3 lights are on, one end is transmitting and the control is correct, then the only thing left is the protocol. Either a hardware or software protocol analyzer will be needed to troubleshoot the communications between the devices. On new installations one common problem is mismatched baud rates. The protocol analyzer will tell exactly what the baud rates are for each device. Another thing to look for with the analyzer is the timing. Often the transmitter waits some time before expecting a proper response from the receiver. If the receiver takes too long to respond or the response is incorrect, the transmitter will 'time out'. This is usually denoted as a 'communications error.'

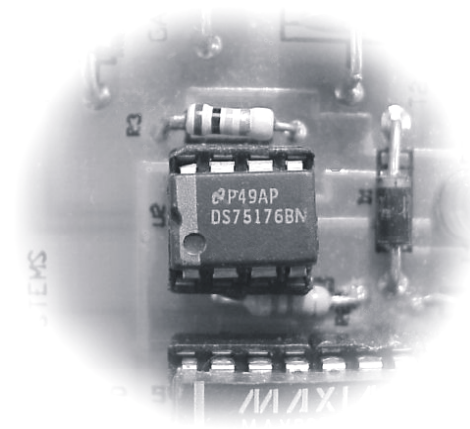
## 6.6 RS-485

### 6.6.1 Introduction to RS-485

RS-485 is the most common voltage standard in use today for multidrop communication systems. This is because of the following:

- It is very resistant to noise
- It can send data at high speeds (up to 10 Mbps)
- It can be run for long distances (5 km at 1200 bps, 1 km at 9600 bps)
- It is easy and cheap to use.

The RS-485 chips are differential chips. This means that the two TX and RX wires are referenced to each other. A one is when one of the lines is +5 volts and the other is ground. A zero is when the lines reverse and the line that was +5 volts is now ground (0 volts) and the line that was ground is now +5 volts. In working systems the voltages are usually somewhere around  $\pm 2$  volts with reference to each other. The indeterminate voltage levels are  $\pm 200$  mV. Up to 32 devices (TX/RX) can be connected on one system without a repeater. Some systems allow the connection of five legs with four repeaters and get 160 devices on one system. There are chips now being manufactured that can connect up to 256 devices on one pair of wires with no repeaters.



**Figure 6.19**  
*RS-485 chip*

Resistors are sometimes used on RS-485 systems to reduce noise, common mode voltages and reflections. Using bias resistors of values from 1k $\Omega$  to 4k $\Omega$  can sometimes reduce noise. These resistors are connected from each line to +5 volts. Higher voltages should not be used because anything over +12 volts will cause the system to fail. Unfortunately sometimes these resistors can increase the noise on the system by allowing a better path for noise from the ground. It is best not to use bias resistors unless required by the manufacturer.

Common mode voltage resistors usually have a value between 100 k and 200 k ohms. The values will depend on the induced voltages on the lines. They should be equal and as high as possible and placed on both lines and connected to ground. The common mode voltages should be kept less than +7 volts, measured from each line to ground. Again, sometimes these resistors can increase the noise on the system by allowing a better path for noise from the ground. It is best not to use common mode resistors unless required by the manufacturer or as needed.

The termination resistor value depends on the manufacturer and will be between 60 and 220 ohms. This resistor is placed between the lines and reduces reflections on short and high-speed lines. If the lines are more than 100 meters long and speeds are 9600 or less the termination resistor usually becomes redundant, but having said that, you should always follow the manufacturers' recommendations.

### 6.6.2 RS-485 vs RS-422

In practice RS-485 and RS-422 are very similar to each other. In practice manufacturers often use the same chips for both. The main working difference is that RS-485 is used for 2 wire multidrop half-duplex systems and RS-422 is 4-wire point to point full-duplex systems. Manufacturers often use a chip like the 75154 with two RS-485 drivers on board as an RS-422 driver. One driver is used as a transmitter and the other is dedicated as a receiver. Because the RS-485 chips have three states, TX, RX and high impedance, the driver that is used as a transmitter can be set to high impedance mode when the driver is not transmitting data. This is often done using the RTS line from the RS-232 port. When the RTS goes high (+ voltage) the transmitter is effectively turned off by putting the transmitter in the high impedance mode. The receiver is left on all the time, so data can be received when it comes in. This method can reduce noise on the line by having a minimum of devices on the line at a time.

### 6.6.3 RS-485 installation and troubleshooting

Installation rules for RS-485 vary per manufacturer and since there are no standard connectors for RS-485 systems, it is difficult to define a standard installation procedure. Even so, most manufacturer procedures are similar. The most common type of connector used on most RS-485 systems is either a one part or two part screw connector. The preferred connector is the 2-part screw connector with the sliding box under the screw (phoenix type). Other connectors use a screw on top of a folding tab. Manufacturers sometimes use the db9 connector instead of a screw connector to save money. Unfortunately the db9 connector has problems when used for multidrop connections. The problem is that the db9 connectors are designed so that only one wire can be inserted per pin. RS-485 multidrop systems require the connection of two wires so that the wire can continue down the line to the next device. This is a simple matter with screw connectors, but it is not so easy with a db9 connector. With a screw connector the two wires are twisted together and inserted in the connector under the screw. The screw is then tightened down and the connection is made. With the db9 connector the two wires must be soldered together with a third wire. The third wire is then soldered to the single pin on the connector.

**Note:** When using screw connectors the wires should NOT be soldered together. Either the wires should be just twisted together or a special crimp ferrule should be used to connect the wires before they are inserted in the screw connector.



**Figure 6.20**  
*Bad RS-485 connection*

Serious trouble with RS-485 systems is rare (that is one reason it is used). Four possible problems can arise in the installation process:

- The wires are reversed (black to white and white to black)
- Loose connections due to improper installation. (See Figure 6.20)
- Excessive electrical or electronic noise in the environment
- Reflection of the signal

To make sure the wires are not reversed, check that the same color is connected to the same pin on all connectors. Check the manufacturer's manual for proper wire color codes.

Verifying that the installers are informed of the proper installation procedures can reduce loose connections. If the installers are provided with adjustable torque screwdrivers then the chances of loose or over tightened screw connections can be minimized.

Excessive noise is often due to the close proximity of power cables or signal wires. Another possible noise problem could be caused by an incorrectly installed grounding system for the cable shield. Installation standards should be followed when the RS-485 pairs are installed close to other wires and cables. Some manufacturers suggest biasing resistors to limit noise on the line while others dissuade the use of bias resistors completely. Again the procedure is to follow the manufacturer's recommendations. Having said that, I have found that biasing resistors are of minimal value, and that there are much better methods of reducing noise in an RS-485 system. Often common mode voltage resistors are used to reduce noise. The value of the common mode voltage resistors is often from 100 k to 200 k ohms.

**Note:** When using bias resistors neither the A- nor the B+ line on the RS-485 system should ever be raised higher than +7 volts or lower than -5 volts. Most RS-485 driver chips will fail if this happens.

A termination resistor of 60 to 220 ohms is used to reduce reflections on the line. This is more important at higher speeds and shorter distances. Again it is important to follow the manufacturer's recommendations for the value or whether it is needed at all.

## 6.7 Fiber optic cables

Fiber optic communication systems are becoming more popular every day. This is because as the quantity of electronic equipment increases the amount of radiated noise increases. Therefore it makes sense to move to noise free fiber optic systems. Communication on a fiber optic cable is done by turning on and off the light created by a laser diode or LED. The light is sent down a glass fiber cable to a light sensitive receiver at the other end. The receiver turns the pulses of light back into ones and zeros. This serial communication can be done in either an asynchronous or synchronous manner. Because the fiber optic cable is simply a pipe for the data, any protocol can be used to send the data. RS-232 to fiber optic cable adapters are becoming very popular.

If an RS-232 system is too noisy, the person responsible could put together an RS-232 to fiber optic system with the following:

- Two RS-232 to fiber optic converters
- A length of fiber optic cable
- Four hot melt connectors.

A hot-melt connection unit is used to melt the connectors onto the end of the fiber optic cable. The converter is then plugged in to the RS-232 port on each end. The cable is then connected to the converter and the whole system becomes an invisible pipeline to the transmitter and receiver. The reason there are four connectors needed is because there is actually two fiber optic cables needed. One cable is for data in one direction and the other cable is for the other direction.





**Figure 6.21**  
*Fiber optic cable with hot melt connectors*

The most common type of fiber optic cable used to connect industrial devices is the zip cable. The zip cable is a fiber optic cable with two fibers along side of each other. They are loosely attached to each other. By pulling on both cables at the end, the two cables can come apart very easily. The two cables are then hot melted into the connectors. Once the connectors have cooled they are plugged into the controller.

## 6.8 Fieldbus protocols used in controllers

Fieldbus protocols have changed over the last decade from proprietary protocols such as the Allen Bradleys Data High Plus system to open non-proprietary protocols like Profibus. This change means that the user is no longer dependent on one manufacturer.

Some other open protocols used to communicate to controllers are:

- |                         |              |                            |
|-------------------------|--------------|----------------------------|
| • Modbus                | Master/slave | No physical layer          |
| • DNP3                  | Master/slave | RS-232 or FSK              |
| • CANBUS                | CSMA / BA    | No physical layer (RS-485) |
| • Profibus (FMS and DP) | Token bus    | RS-485                     |
| • Profibus (PA version) | Token bus    | 1158-2                     |
| • Interbus S            | Timed        | RS-485                     |
| • Foundation Fieldbus   | Timed        | 1158-2                     |

Request Message

Address	Function Code	Initial Coil Offset		Number of Points		CRC
		Hi	Lo	Hi	Lo	
01	01	00	0A	00	02	9D C9

Response Frame

Address	Function Code	Byte Count	Coil Data	CRC
01	01	01	03	11 89

**Figure 6.22**  
*Typical Modbus packet*

All of these protocols have their good and bad points and none of them would be considered by the author as perfect. The selection by the user or designer really is done on a case by case basis. For example if the system was going to be on wire and radio then probably DNP3 would be the choice, but if only wire was going to be used and the system was small probably Canbus or Modbus might be used. Larger industrial systems would probably use Profibus or Foundation Fieldbus. The fastest data transfer system is the Interbus S system.

## 6.9 Conclusion

Data communications is a very important and a useful part of microcontroller systems. Controllers that communicate can reduce the largest cost in a factory, the labor. Therefore, the real power in microcontrollers is in connecting them together. We have seen in this chapter that the code, voltage standard and the protocols are the three basic parts of any data communications system. We also saw that the OSI seven-layer model is important in helping us understand the different parts of a working communication system. It also was shown that the seven-layer model really doesn't work for microcontrollers and that most often we use a simplified version of the three-layer model. The application layer, datalink layer and the physical layers are the typical layers used by most control systems.

Getting a little more specific, concerning voltage standards, we saw how the modes of communications define the way we set up the overall system. With simplex, half-duplex and full-duplex defining the main way we communicate. Using one of the four basic types of fieldbus systems, master/slave, CSMA/CD, token bus and timed, modern industrial data communication systems can communicate. It was also shown that different communication systems have their good and bad points depending on needs of the designer and/or user.

RS-232 and RS-485 were defined and shown as two very different but equally useful voltage standards. RS-232 is used mainly when connecting only two controllers, but RS-

485 is used when connecting multiple devices together. RS-232 is usually used when an engineer or technician wants to program or communicate with a microcontroller directly. We found that RS-232 is used mostly when the communication lines are short and the data speeds are slow. Whereas RS-485 was seen as typically a multidrop, high speed and long distance voltage standard. It was also noted that the RS-485 voltage standard is very immune to noise compared to RS-232. Fiber optics was also discussed and seen as the communication standard of the future. Signal wire systems are quickly being replaced by fiber optic cable systems.

Lastly we saw that there are a many fieldbus systems developed lately for microcontroller communications. The movement to open fieldbuses is unstoppable and will no doubt continue in the future. We saw that different open standard communication fieldbuses are used for different situations.

At the moment serial data communication is used extensively as the only system to connect devices at some distance. On the printed circuit board of course most communications is done in parallel. These wires are short and protected from outside noise. But in the not too distant future data communication may move to parallel communication over medium to long distances using multiple frequency light on fiber optic cables. In these systems the data will be sent with 8, 16 or 32 different colors of light at a time. It is theoretically possible to send millions of colors at a time, at least for short (2 km) distances. The future of data communications is in radio and light using parallel communications.

## Noise reduction

### Objectives

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

- Define noise and how it affects a microcontroller circuit
- Explain the decibel and the signal to noise ratio
- Explain the difference between single ended and differential circuits in relation to noise reduction
- Describe common mode voltages
- Describe the three ways to couple noise from one circuit to another
- Describe the four ways to reduce noise on a PCB
- Explain the function of a Faraday shield and its use

### 7.1 Introduction to noise reduction

Noise is fact of life. Everyone that has children knows that noise is part of being a parent. And like the noise that children bring into our lives, electronic noise is always present. Electronic noise cannot be totally eliminated; it can only be reduced, hopefully to the point that it does not cause us any problems with our signal. This chapter explains electrical noise theory and practical ways to reduce it in our circuits.

This includes:

- What is electronic noise?
- Sources of electronic noise
- Grounding on printed circuit boards
- Reduction of electronic noise

**Definition of Electronic Noise:**

*'One man's noise is another man's signal.'*

**Steve MacKay**

To understand electronic noise we must first look at the origins of electronics. In the seventeen and eighteen hundreds a few physicists started out doing simple experiments with permanent magnets, coils of wire and glass jars. These founding fathers of electronics used common materials to demonstrate the physical properties of inductance, capacitance and magnetism. Little did they know that those parlor tricks of filling jars with electricity, moving magnetized needles with current through a coil of wire and creating sparks in adjacent wires, were the building blocks of modern electronics. When Marconi developed the first spark gap transmitter, he created not only radio but also what we know today as wide band noise.

Noise can be loosely defined as any electric or electronic signal that interferes with *my* signal. It may even be that some of my own signals are interfering with themselves. Noise is usually separated into two types, spikes or frequencies. Spikes are transient and frequencies are usually constant. All electrical noise is created by the changing power of a circuit. Rarely is DC voltage a noise component. Noise is a combination of both voltage and current. And since noise is a relative manifestation, electronic equipment can be more or less susceptible to noise. Electronic noise can be considered both a transmit and receive problem. Many arguments have been born out of the disagreement over whether the noise source is too strong or the noise receiver too sensitive. It is best therefore to design equipment that produces as little electronic noise internally and filters external electronic noises as much as possible. Maintenance of the noise that is left in our circuit is then a prime concern. The susceptibility of our equipment to this unavoidable electronic noise that is left in our circuit is the true test of good electronic design.

### **7.1.1 The decibel**

To define how much electronic noise we are creating or want to eliminate, we must first give it a value of  $-20$  dB. This value is known as the decibel. The decibel is a value that represents the power of noise produced or received by a circuit. It is a unit based on the ratio between the power into and the power out of the circuit.

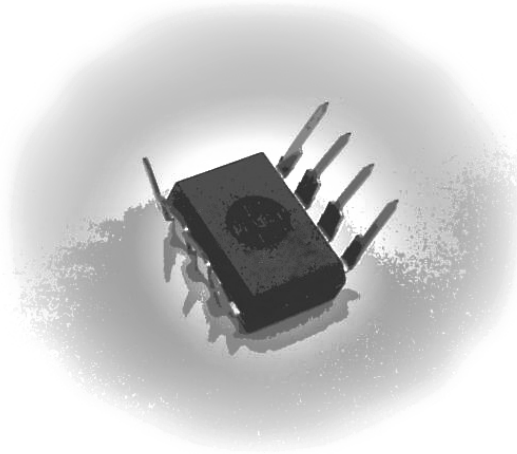
### **7.1.2 Signal to noise ratio**

The signal to noise ratio is the ratio of noise to our signal. This ratio is used to show the relationship of some external noise to our known good signal. The signal to noise ratio is measured in decibels or dB. The signal to noise ratio may be defined within some bandwidth and/or center frequency. This is because the signal to noise ratios may change outside of a bandwidth, or away from a particular center frequency. It is easy to see how that if the voltage on the line is  $+5$  volts and there is also  $+0.05$  volts of noise on the line then the signal to noise ratio is 100:1. This would be a value of minus 100 dB.

### **7.1.3 Sources of noise – internal vs external**

Electronic noise source can be divided up into two areas, internal created and externally received. Internal noise sources are electronic noise sources that originate within the same electronic device that is experiencing the noise. This noise usually is discovered when the equipment is first turned on. It is rare that a piece of equipment develops an internal noise problem after being in use for some time. Internal noise can develop when a component

within the equipment becomes defective. This problem is unusual on newer digital equipment, as digital chips either work or they don't.



**Figure 7.1**  
*Dead chip*

External noise sources usually are transmitted to our circuit by way of some wire. The many wires that connect different equipment can be thought of as radio antennae. The closer the wires, and the longer they lay next to each other, the greater the noise is transmitted to our circuit. It is important then in noise reduction to keep and or move the wires that are transmitting noise away from the wires that are connected to our equipment.

The ratio of noise transmitted from a wire to our circuit is determined by the following:

- The size and type of wire used in both circuits
- The strength of the transmitted signal
- The susceptibility of the receiver to the noise
- The distance between the wires
- The length they run parallel
- The type of circuits (single-ended vs differential)

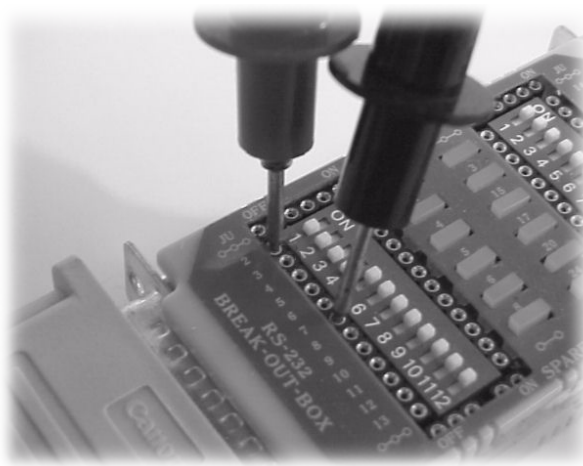


**Figure 7.2**  
*Wires in parallel*

### 7.1.4 Single ended or grounded circuits

Every piece of electronic equipment is a noise-receiving device. Even a resistor sitting on a table is receiving electronic noise. It is important then in the development of a noise reduction system to understand how noise is received. There are basically two different types of noise receiver circuits. There is the differential or floating receiver and then there is the single ended or grounded receiver circuit. Each of these types of circuits has a different reaction to electronic noise. The single ended circuit is easier to use, but is more susceptible to noise. The differential receiver is less susceptible to noise, but it has common mode voltage problems.

By definition, a voltage is a measurement of the potential difference between two points. Grounded signal sources or receivers have one of their signal lines connected to the system ground as shown in Figure 7.3. This is theoretically shown as earth potential, although the system ground is not necessarily at earth potential. The voltage from the signal receiver is the potential difference between the system ground and the signal.

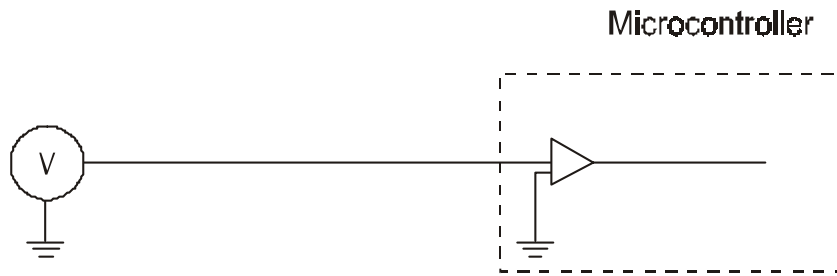


**Figure 7.3**  
*Voltage measurement on a circuit*

On a PCB the inputs and outputs on the chips are connected with one wire. The return path is usually common to a ground plane. Figure 7.4 shows a typical grounded circuit in the form of a digital gate. The outputs of single ended circuits are always referenced to the common side of the circuit. If the input of the circuit is raised or lowered the output will rise or decrease in the same manner. If more than one circuit is tied to the same common ground then noise from one circuit will flow to the other. An example of a single ended circuit would be RS-232 communications systems. On data acquisition systems it is common to see grounded sensors and common analog inputs.

### 7.1.5 Single ended measurement of grounded sources

Single ended measurements of grounded sources are circuits where each end of the circuit is referenced to the same ground or common. Most equipment is designed with the output of one circuit connected to the input of another circuit and a common ground. In this type of circuit, if the ground reference of either the source or receiver changes then the output voltage will also change.



**Figure 7.4**  
*Single ended to single ended*

### 7.1.6 Single ended grounded equipment

If one of the leads of a piece of test equipment has a low resistance to ground then that piece of equipment is a single ended measurement device. Usually any piece of test equipment that plugs directly into the mains power source can be considered a grounded piece of test equipment.

Some of the usual types of single ended test equipment are:

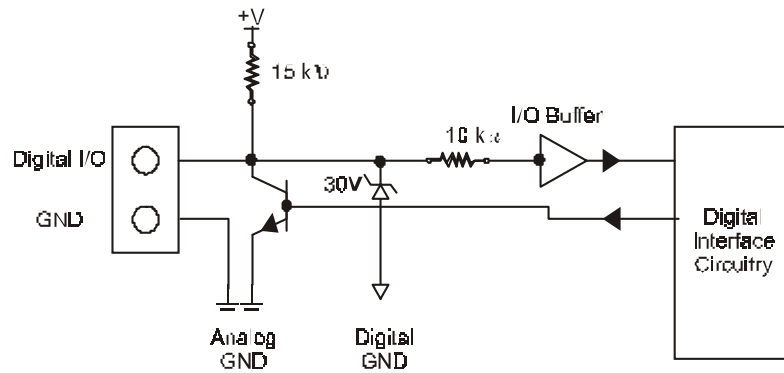
- Bench top digital voltmeter
- Bench top oscilloscope
- Bench top function generator
- Bench top spectrum analyzer

Measuring a signal with one side grounded can cause incorrect reading, because the ground potential on the test equipment may be different to the ground potential on the equipment. This ground may or may not be earth ground and indeed it is possible that there are multiple types of grounds on the same printed circuit board.

Some of these grounds are called:

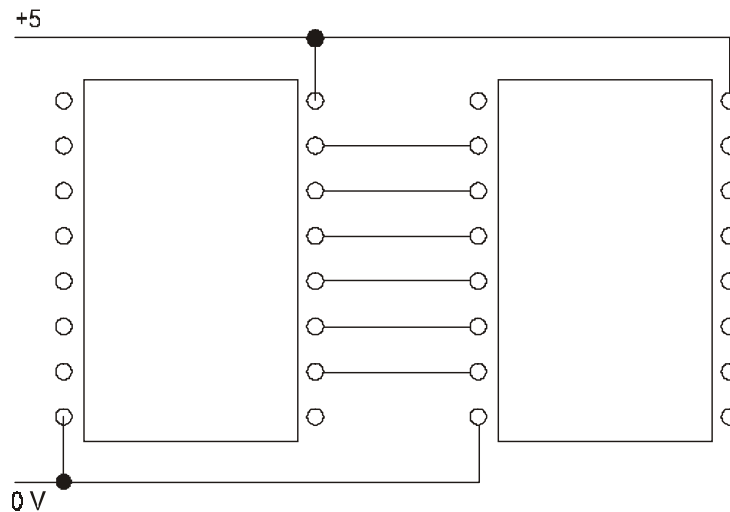
- |                  |   |
|------------------|---|
| • Earth ground   | Earth ground  |
| • Ground         | Typically earth ground                                    |
| • Shield ground  | Typically earth ground                                    |
| • Common         | A common connection point                                 |
| • AGN            | Analog ground   |
| • Digital ground | +5 volt power supply ground                               |
| • VSS            | +5 volt power supply ground                               |
| • Signal ground  | Sensor or communication common                            |
| • VCC            | Not +5 volt supply ground<br>(might be +/-12 volt ground) |





**Figure 7.5**  
*Single ended input measurement*

A single ended measurement of floating sources is when a floating or differential source signal is measured with reference to a grounded measurement circuit. The source can be any type of circuit that is not grounded. The receiver can be any type of circuit that has one of its inputs referenced to ground or a common. (See Figure 7.6.) As can be seen from Figure 7.6 there cannot be any common mode voltages with this type of circuit because one of the outputs from the floating source is referenced to ground. And if the ground potential changes, the voltage of the output signal will change in reference to the ground.



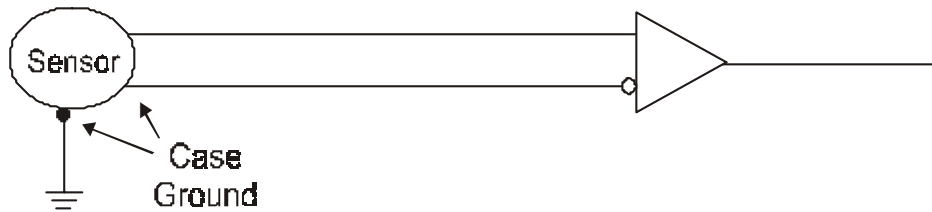
**Figure 7.6**  
*Single ended circuit*

### 7.1.7 Differential noise circuits

A differential input amplifier is an amplifier with two inputs, usually one is labeled with a plus and the other is labeled with a minus. (See Figure 7.7.) The output of the amplifier is the difference between the inputs. The outputs are not referenced to ground or common, only to the difference of the inputs. The amplifier is usually extremely high input impedance. This means that the receiver will not load down the circuit when connected to the transmitter. Differential circuits also typically have a high common mode rejection ratio. This makes the amplifier very resistant to common mode voltages and noise.

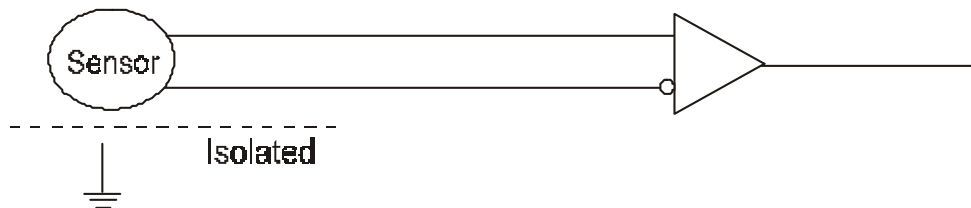
Floating or ungrounded signal sources, as shown in figure 7.8 do not have either of their signal sources directly connected to the system ground. This means that the signal source is not referenced to an absolute reference. The potential difference between the signal input lines vary according to the output line from the transmitter, not according to the ground potential. Both of the lines have a voltage potential to ground, but this is not part of the measurement equation.

A typical example of measuring a grounded signal source with a differential measurement circuit is shown in Figure 7.7. This system places the output of the source on the positive input of the differential amplifier while connecting the ground or common of the source to the negative input of the differential amplifier. Using a battery-operated voltmeter to measure an RS-232 transmission line is a typical example of differential measurement of a grounded source.



**Figure 7.7**  
*Ground signal source to differential amplifier*

The following Figure 7.8 shows a typical example of a differential measurement of a floating source. The source is called a floating source because neither of its outputs are directly referenced or connected to ground or common. This type of system is considered the least noise inducing method of measurement. Because neither circuit is connected to ground, there is little chance of ground induced noise. An example of this type of measurement and source system is an RS-485 communication system.



**Figure 7.8**  
*Floating signal source to differential amplifier*

### 7.1.8 Differential test equipment

It is important in developing an electronic noise reduction system to understand which test equipment is single ended and which is differential. The following is a list of single ended and differential test equipment. To be sure which type your equipment is it is best to test it by measuring the resistance of each test lead to earth ground.

Any piece of test equipment that is battery operated. Such as:

- Hand-held digital voltmeter
- Battery powered oscilloscope
- Portable spectrum analyzer

### 7.1.9 Common mode noise problems

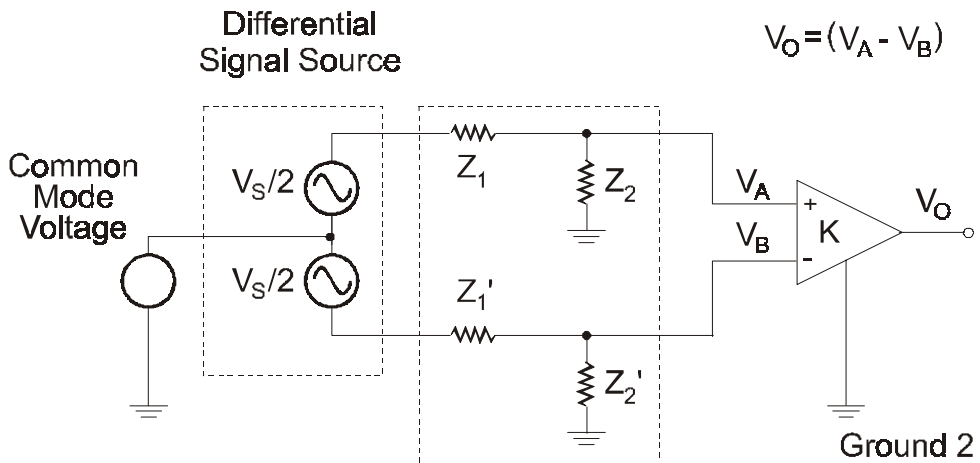
In an ideal world the input to a floating receiver would only receive the difference between the two inputs. Unfortunately everything on and above the earth is referenced to ground. And because the ground level can change it is possible for the input voltages on a circuit to reach a point where the circuit fails. The equal voltages, with reference to ground, on the inputs into a differential amplifier are the common mode voltages. A diagram of this effect is shown in Figure 7.10. To calculate the common mode voltage, use the following formula;

$$V(\text{common mode}) = (V_a + V_b)/2$$

Where

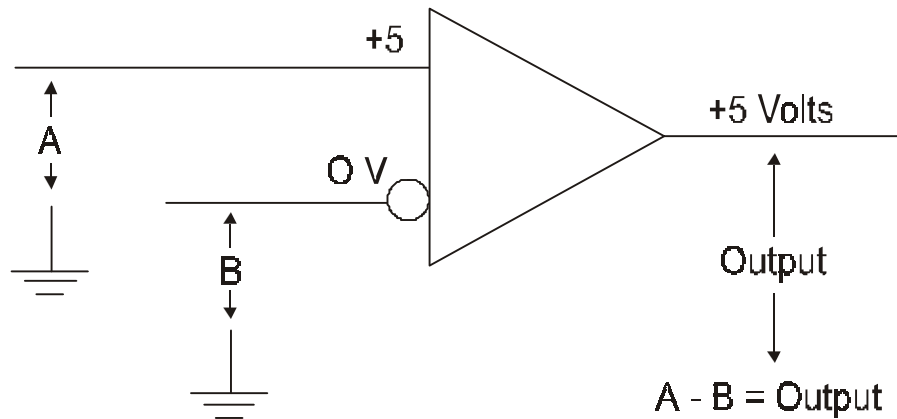
$V_a$  = voltage at the non-inverting terminal of the measurement system with respect to the instrumentation amplifier ground

$V_b$  = voltage at the inverting terminal of the measurement system with respect to the instrumentation amplifier ground



**Figure 7.9**  
Common mode voltage

Manufacturers supply common mode voltage limits with the specifications of their products. These values are the upper and lower voltage limits that the product can reliably reject before it fails. A common example would be an upper limit of +12 and a lower limit of -7 volts. This means that the voltage on each line of the device with reference to ground can not exceed +12 volts or be lower than -7 volts. If these levels are exceeded the device will malfunction and possibly be damaged.



**Figure 7.10**  
*Common mode and the differential amplifier*

### Common mode rejection ratio

In an ideal world with ideal differential amplifiers the differential measurement device would reject any common mode voltages. Unfortunately this does not happen. Every differential input device does reject some common mode voltages. This rejection ratio is expressed in the formula as

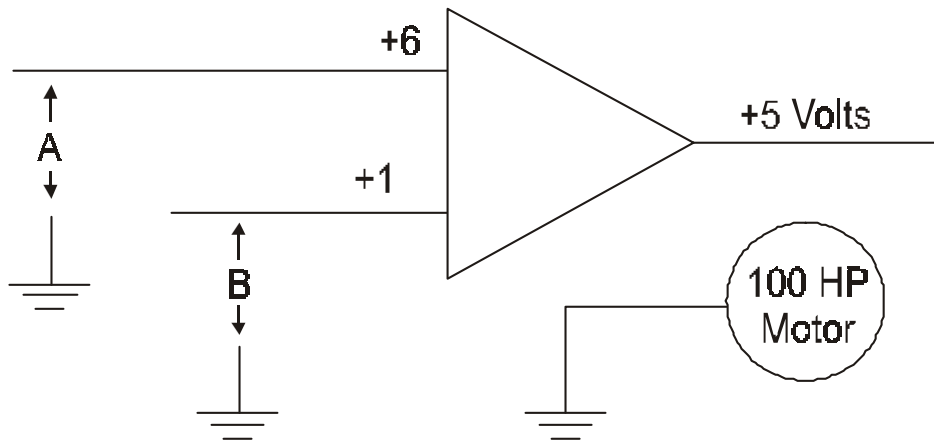
Formulae

$$\text{Common mode voltage} = \frac{(V_a + V_b)}{2}$$

$$\text{Common mode rejection ratio} = 20 \log_{10} \left( \frac{\text{CMV}}{V_{\text{out}}} \right)$$

### 7.1.10 Low impedance drops as noise sources

Whenever a system is developed where there are multiple receivers on a common wire, see Figure 7.11, a noise source can be developed due to low impedance on one of the nodes. On a rare occasion a chip will develop low impedance on its output. This will cause the chip to draw a large amount of current and the chip will get hot. Low impedance can be thought of as a low resistance on the line. This can drag the correct signal down and also allow injected noise from the faulty node. A node is defined as any receiver and/or transmitter that are connected to a system. This low impedance can be caused either through faulty design or a failure of parts within the node.



**Figure 7.11**  
*Common mode voltage with noise*

### 7.1.11 Types of externally induced noise

There are three basic types of external noise sources that can induce noise into electronic systems. They are:

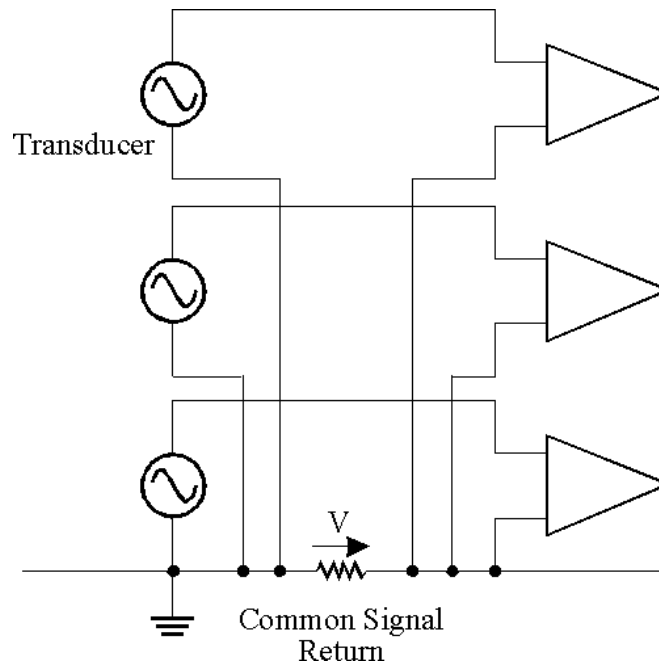
- Conductive coupled noise
- Capacitive coupled noise
- Magnetically coupled noise

Conductive coupled noise is transmitted to the equipment by way of direct connection, where as the other two, capacitive and magnetic coupled noise are induced into the circuit by way of close but not direct contact. It is unfortunate that each of these types of noise coupling has the same effect on electronic circuits. Any of these noise insertion methods can combine to create havoc in a circuit. A full and complete understanding of each of the noise sources is an important part of developing a noise reduction system.

When one signal from a circuit is induced into another circuit it is sometimes called cross talk. Cross talk can happen on cables, on printed circuit boards or even within a single device such as a digital chip. One of the most common ways cross talk is produced is with conductive coupled noise.

## 7.2 Conductive coupled noise

Conductive coupled noise is created when part of an electronic circuit is electrically connected to another. Which circuits are affected depends on the noise produced by each circuit and the susceptibility of each circuit. Either a common impedance ground, a common supply voltage or common signal input can generate conductive coupled noise. In Figure 7.12 each circuit has a common impedance ground. The current from one circuit can combine with the current from a second circuit and therefore conduct noise into the second circuit's amplifier. This type of noise coupling is common in poorly designed circuits and systems where large current or voltage devices share a common ground or power supply with a low current or voltage device.



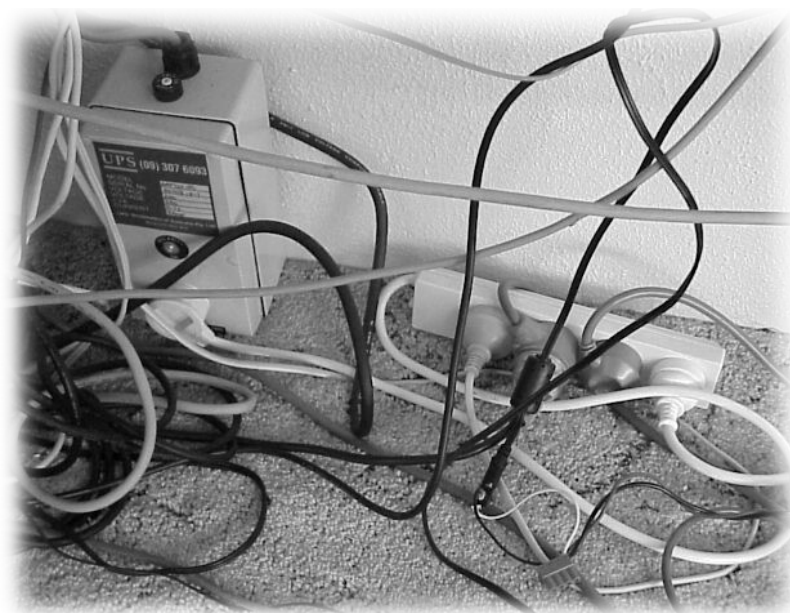
**Figure 7.12**  
*Conductive noise*

### 7.2.1 **Conductive noise from external equipment**

The amount of noise caused by a common impedance ground can depend on the relative power levels of each of the circuits or the power of the external noise. In one instance a noise voltage of minus 30 dB may not cause any problems, but at other times a noise level of minus 70 dB below the signal level may cause all kinds of problems. The choice is then whether it is appropriate to reduce the amount of noise from the offending equipment or reduce the sensitivity of the receiver.

### 7.2.2 **Conductive noise from transmission lines**

The earth ground that connects our equipment to the power point is a perfect example of a common impedance ground. If some high powered device like a motor is connected to that same mains power ground then there is potential for common induced noise. Noise can be introduced to our equipment through that ground when the motor is turned on. Noise can also be created by the power station itself. These can include over-voltage, under-voltage, switching at the power station, lightning, brownouts or even noise induced from a high power user. This noise cannot only come from the groundside of the system but also come from the active or neutral side of the transmission line. Most of the time this type of interference is only temporary, but it is possible that noise induced from transmission lines could cause ongoing problems or even destroy unprotected equipment.



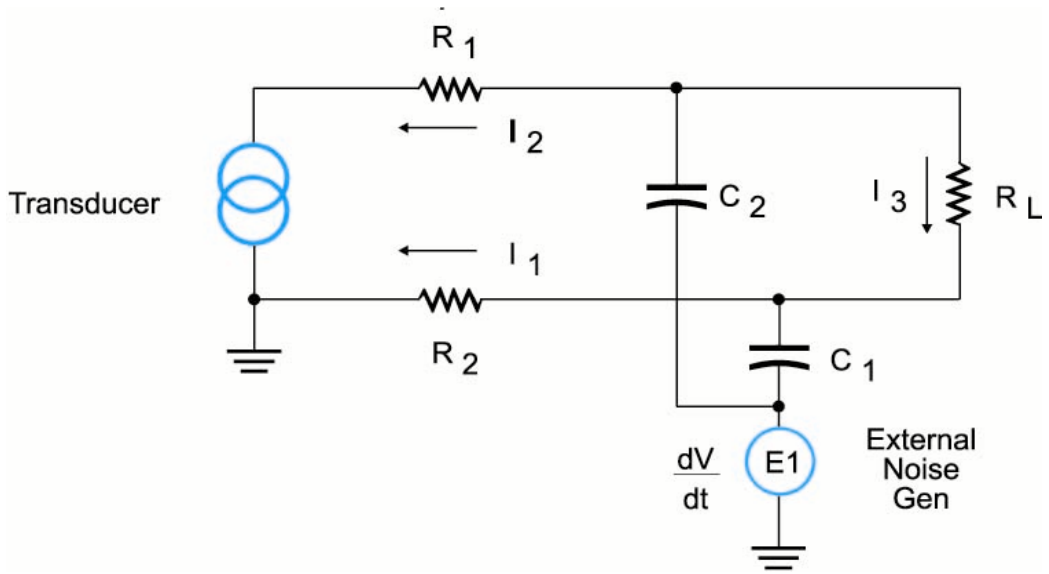
**Figure 7.13**  
*Power cable noise*

## **7.3 Capacitive coupled noise**

A capacitor is defined as two plates of conductive material separated by a dielectric. A dielectric can be any type of insulating material such as air, paper, glass, mica, ceramic, silicon or plastic. The amount of capacitance depends on the insulating material, the closeness of the plates and the size of the plates. The frequency of the noise is also a factor. Because capacitors like to pass higher frequencies, the higher the frequencies of the offending signal the more extreme the problem becomes. The closeness of the wires and how long the wires run next to each other increases the amount of induced noise.

### **7.3.1 Capacitive noise from adjacent equipment**

Electronic or electrical noise can be transmitted from one device to another because of this capacitive effect. AC voltage noise is the main type of noise transmitted. As shown in Figure 7.14 the stray capacitance caused by the close proximity of one circuit to another can induce a noise level into the second circuit. This induction is a function of the closeness of the two pieces of equipment, the frequency of the offending equipment and the types of enclosures that surround the equipment. Plastic boxes or enclosures give no protection from noise.



**Figure 7.14**  
*Capacitive noise coupling*

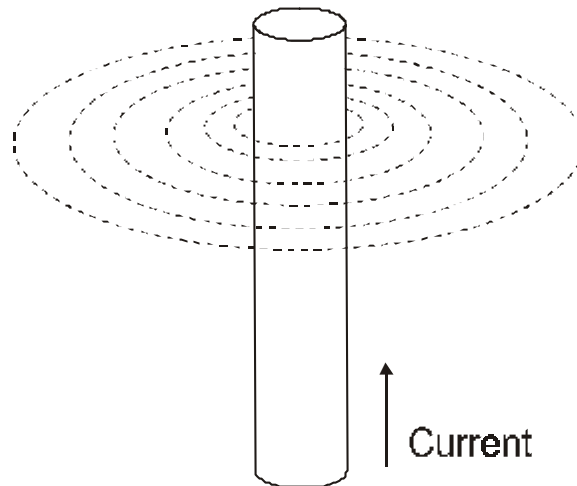
### 7.3.2 Capacitive noise from communication lines

The most common type of delivery of capacitive coupled noise is through communication lines. All types of wire transmitted communication systems have the ability to induce or be induced with capacitive noise. This problem exists because the wires for the communication lines are invariably placed adjacent to other cables. These cables can run next to each other for miles. This length can cause a large plate size between the wires. The insulation and air between the cables become the dielectric. This closeness also increases the capacitance between the cables. Manufactures of cables supply information about the capacitance of their cables. This capacitance is usually expressed in pica farads per foot or nanometers per meter. It is not uncommon to run many different types of communication, control and power lines in one cable. This can lead to multiple sources of capacitive coupled noise within a single cable.

## 7.4 Magnetically coupled noise

In the 1819 Hans Christian Oersted a Danish physicist discovered that when a current was passed through a wire it created a magnetic field around the wire. Inversely, it was found that a magnet passed close to a wire would cause a current to flow in the wire. These two discoveries explain the cause and effect of magnetically induced noise. When the current flows in the wire a magnetic field is produced. This magnetic field is made up of what is called lines of force. Theses lines of force rotate in a clockwise direction when viewed from the positive pole (see Figure 7.15). The strength of the magnetic field is proportional to the amount of current flowing through the wire.





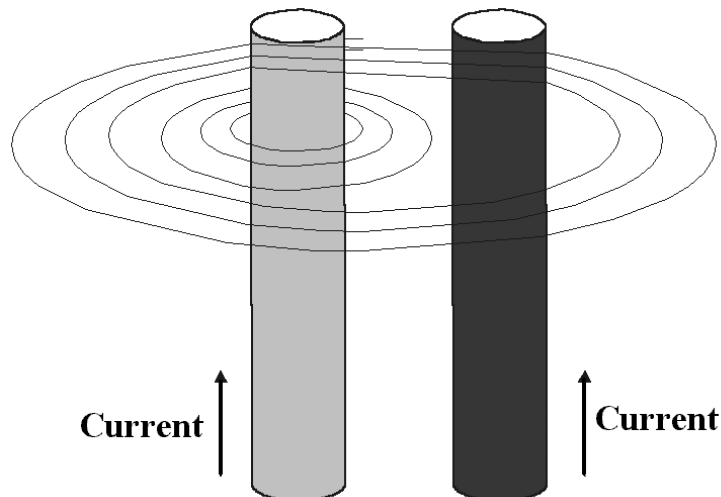
**Figure 7.15**  
*A magnetic field created*

### 7.4.1 Magnetically induced noise from adjacent cables

When AC is applied to a wire, current can be induced to flow in adjacent wires. As the magnetic lines of force cut across the adjacent wire they induce a current and voltage into that wire (see Figure 7.16). This magnetically coupled noise is produced when the magnetic field created by one device induces an unwanted current or voltage in an adjacent device.

There are three conditions that relate to the ability of a magnetically coupled noise source to induce noise into an adjacent wire:

- The relative location of the wires to each other
- The power of the signal in each wire
- The frequency of the noise source



**Figure 7.16**  
*Magnetic coupling from adjacent wires*

Magnetic coupling occurs best when the source and receiver wires are close together, and are run in parallel. When the wires are close and are run in parallel the two wires act like a transformer. The expanding magnetic field cuts across the receiving wire and induces a current flowing in the same direction as the source. The receiving wire then transfers this current to the equipment. The closer the wires, the stronger the magnetic field will be. The longer the wires run in parallel, the more chances the magnetic field has to enter the receiving wire.

### **7.4.2 Magnetically induced noise from adjacent equipment**

Strong current from relays, AC motors, coils, transformers, solenoids, or power lines can induce magnetically coupled noise into a circuit. Whether or not the voltage/high current equipment can induce magnetically coupled noise depends on the closeness of the equipment and the current draw of the equipment. An AC water pump may be meters away from a receiver device, but because it is giving out large amounts of electromagnetic noise, it could affect the receiver circuit. High-tension lines give off tremendous amounts of electromagnetic noise and sometimes even enough to light a fluorescent tube.



**Figure 7.17**  
*High tension lines*

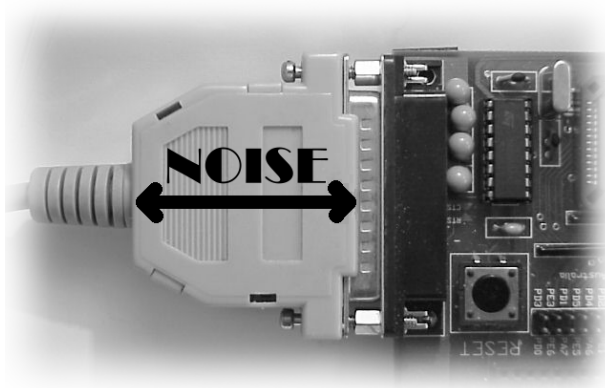
## **7.5 EMC and noise reduction in PCB design**

Noise reduction in PCB design has become an extremely important issue as of late. This is because of the huge amounts of digital electronic equipment in use today. There are two main factors that have contributed to the increase in noise problems in equipment. One has been the advent of electronic equipment that is very sensitive to noise. And the other is that devices are now running higher clock speeds. Higher clock speeds mean higher frequencies and therefore more radio frequency noise. A 700 MHz CPU can become a 700 MHz radio transmitter, all it needs is an antenna. Four methods are used to reduce the ability of a circuit on a PCB to transmit noise.

They are:

- Segregate the analog, digital and power supply circuits on the PCB
- Less right angles and ground loops on the PCB
- Ground planes to absorb the noise
- One and three dimensional Faraday shields

EMC rules are now in place that provide the designer with official noise reduction standards. In the following text a summary of those standards will be discussed.



**Figure 7.18**

*Noise to and from a PCB*

### 7.5.1 Placement of analog, digital and power supply circuits

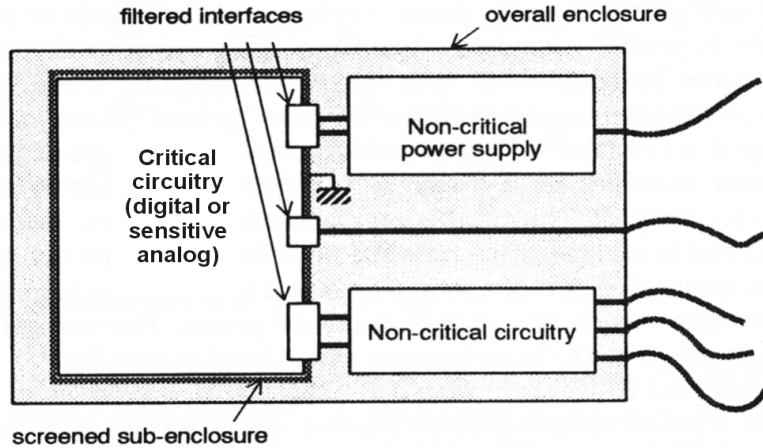
Normally in the past there were conflicts between PCBs that were a functionally layout and those that were logically layout.

In a functionally laid out PCB the parts are placed on the board in a way that optimizes the space on the board and makes it easy to connect the parts. Digital and analog parts were often placed next to each other in a structured manner. The sub-systems on the board can then interact with each other and create noise in the form of capacitive coupled, inductive coupled and a combination of both we call EMI (electro-magnetic interference).

The good thing about the functional layout was that it was easy to design and the parts could be placed very close together. The problem with this method was that sub-systems were mixed together and therefore created induced noise on the PCB. When PCBs were mostly analog and used slower clock speeds, this was not too much of a problem. With circuits becoming more susceptible to noise and running faster, this method of PCB layout is not acceptable.

In a logical layout the sub-systems are placed on the PCB as they are laid out on the schematic. When a design is done using this method the parts on the board flow from left to right. The power supplies and inputs are on the left and the outputs are on the right. This practice can mix analog systems and digital systems with each other on the PCB. The logical method has one main advantage. It is easier to troubleshoot and understand than the functional method because it follows the schematic. The problem with the logical layout is that the analog and digital sub-systems are often mixed and this can cause noise problems. This method is rarely used.

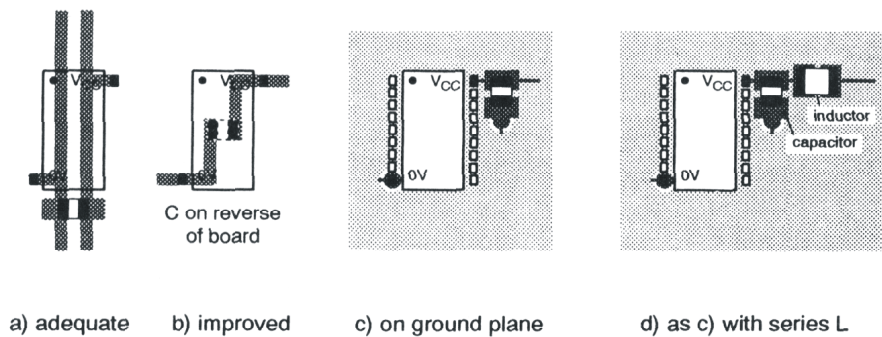
The method most commonly used by designers might be described as extended logical. That is, it is still somewhat logical but places the parts in a segregated logical manner. The board may still flow from left to right, but the power supply, analog and digital subsystems are segregated into their own physical sections of the PCB. The sections are often separated by a one-dimensional Faraday shield or a ground plane.



**Figure 7.19**  
*Proper layout of an EMC PCB (Courtesy of EMC for product designers)*

### 7.5.2 Digital circuit decoupling

The purpose of the capacitor decoupling is to reduce transient spikes of noise due to high speed switching on the PCB. The capacitor acts like a buffer to the voltage spike and keeps the supply voltage on the chip more constant. Placement of the capacitor in relation to the chip it is protecting is critical. The capacitor should be placed as close as possible to the chip. It is also best to place the capacitor an equal distance between the  $V_{cc}$  and ground supply to the chip. This is done by placing the capacitor in the middle of the chip, either on the back of the PCB or inside the chip holder. Placing the capacitor inside the socket is common when large chip sockets are used on the PCB. As more and more PCBs become surface mount the first option of placing the capacitor on the back of the PCB is becoming more common.

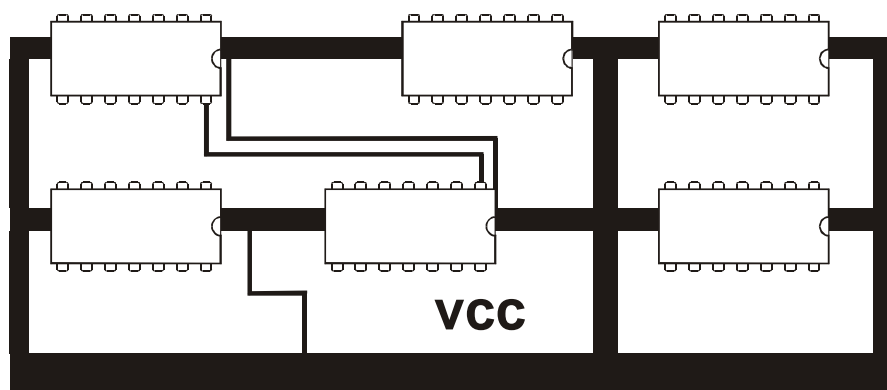


**Figure 7.20**  
*Decoupling capacitor location (Courtesy of EMC for product designers)*

### 7.5.3 Ground planes

Ground planes have changed with the new rules on EMC. In the past, designers placed a minimum amount of ground planes on the board. It was seen as useless and a waste of copper. With the new rules every square millimeter of space is either a track, space between a track or a ground plane. Often multi-layer boards have ground planes that cover the top and bottom of the PCB with the tracks in the middle. Although this is a good practice from a noise point of view, it increases the cost of the board and makes the board unrepairable. Some manufacturers even design five layer PCBs with three layers of ground plane, one on top, one on the bottom and one in the middle.

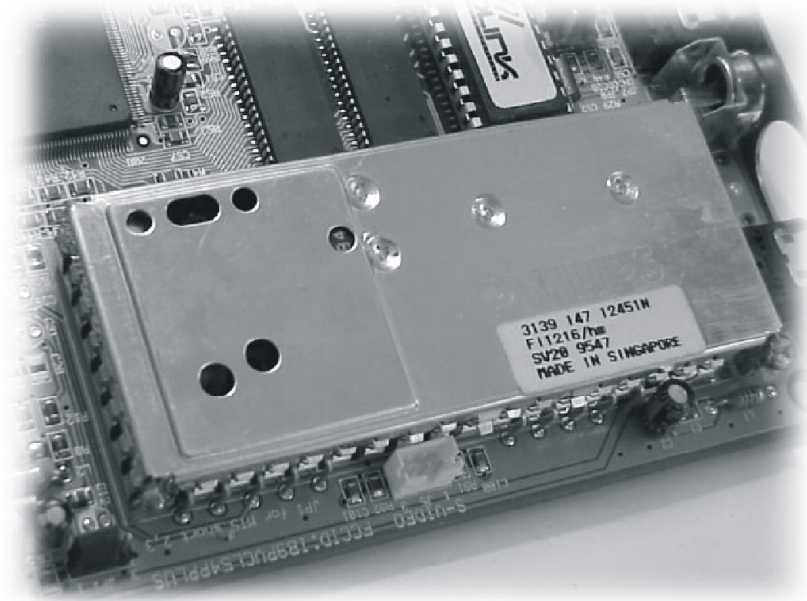
### 7.5.4 1D and 3D Faraday shields



**Figure 7.21**

*Voltage and ground tracks on a PCB (Courtesy of EMC for product designers)*

Michael Faraday (1791–1867) invented the Faraday shield. It is an electrostatic shield that is made by placing conductive material (often aluminum or copper) around some device and connecting that material to ground. The better the conductor (in other words, the less resistance it has) the better the shield will be. Three examples of a Faraday shield would be, the shield on an audio cable, the plates in a transformer and the box on an electric guitar. The braided wire around the conductors in audio cable is connected to ground and protects the signals from external noise. The plates in a transformer prevent capacitance between the primary and secondary windings. Aluminum cages or boxes are placed around sensitive and noisy systems on circuit boards.



**Figure 7.22**  
*Faraday shield on a PCB*

## 7.6 Conclusion

In this chapter we have explored the definition of electronic noise. We have stated that one man's noise is another man's signal. It is important to remember that noise is always present and that the goal is to reduce noise to the point that it does not affect our signal.

We have defined the sources of electronic noise. Electronic noise comes from one of two general sources, internal and/or external. Both internal and external sources of noise are either single ended or floating sources. We found that there are three types of noise coupling systems, conductive coupling, capacitive coupling or magnetic coupling. Each of these types of coupling has its own way of getting noise into a circuit.

Conductive coupling injects electronic noise into a circuit by way of a common ground. Capacitive coupling acts like a capacitor to put noise into a circuit. Magnetic coupling puts lines of force across the conductors of a circuit and induces noise.

There are two types of noise measuring systems, floating or differential and single ended or grounded. These two systems along with the two types of noise sources form four types of circuits, floating to ground, floating to floating, grounded to floating and grounded to ground. Also we found that test equipment was either grounded or floating.

Knowing the sources, types and measurement characteristics of electronic noise is the first step in developing an electronic noise reduction system. I hope this chapter has given you a clear understanding of where noise comes from and how it couples itself into another circuit.

# EMC grounding solutions

## Objectives

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

- Describe some common misconceptions concerning grounds
- Explain how to measure a voltage with reference to ground
- Explain how ground is used as a return path for a circuit
- Explain how ground is used as a noise insertion point for a circuit
- Describe a clean ground
- Describe how to build a spike ground
- Explain how to build a trench ground
- Explain how to build a ground plane on a PCB

## 8.1 Introduction to EMC grounding solutions

This chapter discusses basic and practical grounding techniques. As electronics becomes more and more sensitive to noise and high voltages it is increasingly more important to pay attention to grounding systems. The horror stories of electromagnetic interference are many, with some even causing loss of life. With the proliferation of electronic equipment there is bound to be clashes between different electronic systems. Mobile phones causing airlines to go off course, heart machines stopping when a computer is turned on in the room and safety interlocks failing when the maintenance person uses the hand-held radio are just a few of the ever increasing problems that can be caused by electromagnetic interference.

In the past, earth grounding was seen as the best way to reduce electronic emissions and protect existing equipment from external noise. But now, it is realized that how and where earth grounding is installed is an important factor. In fact it is often found that if the grounding is done incorrectly that even more noise can be injected into the equipment than if there was no ground at all. With poorly designed digital equipment, a common ground can be thought of as a potential noise injection system.

To develop a grounding system that will reduce rather than increase the amount of noise in the system we need to start with a few rules:

- Ground is never 0 volts
- Ground is never 0 ohms
- Ground is often a common return path for the current of a system
- Ground should be treated as a noise insertion point into our system
- A non-grounded system is the quietest from a noise point of view
- An earth grounded system is better high voltage or lightning protection than a non-grounded system



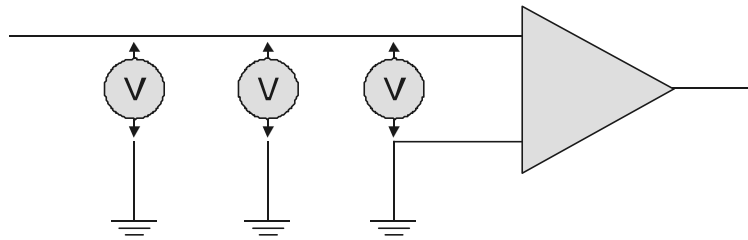
**Figure 8.1**  
*Earth ground*

## **8.2 EMC grounding**

### **8.2.1 Ground specifications**

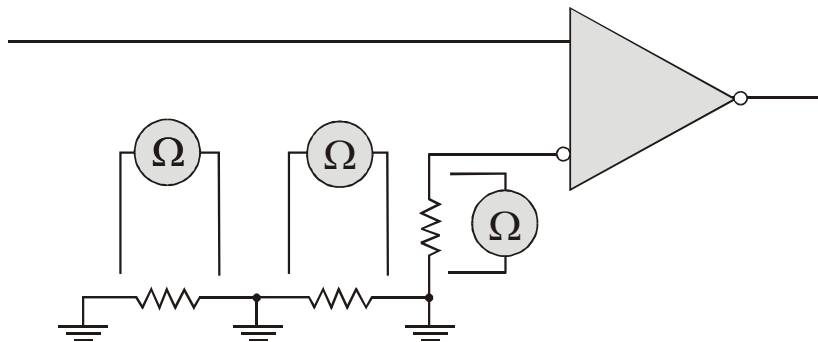
A common mistake engineers and technicians make in grounding is that they assume that ground is 0 volts. This is not true. All grounds have resistance and therefore a voltage drop between the measured ground point and any other point. Just because the technician or engineer puts the black lead of the multimeter on a point in the circuit does not make it 0 volts. The point where the black lead is placed is only a reference. Often if signal voltage is measured, with reference to a common (ground) point and then is measured with respect to earth ground there will be a different voltage. If the same point is measured to a different earth ground it will be found that the voltage is different again. This often worries technicians until someone points out that presence of voltage differences is not necessarily a problem. Noise is a ratio of the power of the signal to the power of the noise. Power is a function of voltage times the current and just voltage on a circuit without current does not usually cause a problem.





**Figure 8.2**  
*Ground is not 0 volts*

There is an old saying in electrical circles that ‘ground is green the world around’, this is not totally true. Often ground is a black wire. All grounds have resistance and are never 0 ohms. Every wire and track has some resistance and the common that returns to ground is connected through these wires. When engineers design a system they will often specify the minimum requirements for the resistance from the equipment to earth ground. This value is often in tens of ohms. They will also possibly specify the resistance of the earth ground to another earth point some distance away from the equipment’s earth ground. Isn’t earth ground 0 ohms? Well no it is not. It has a resistance to earth also. Of course the value of this resistance depends on where the earth ground is measured. When current is passed through a circuit or ground path that has resistance there is a voltage drop from one end to another. If you think you may have an earth grounding problem it might be best to call in a professional grounding contractor to measure and adjust your grounding system.



**Figure 8.3**  
*Ground is not 0 ohms*

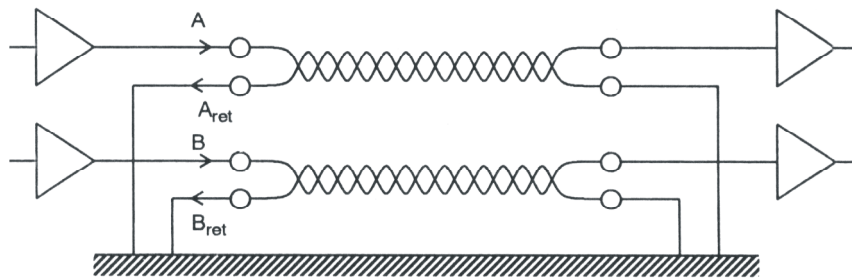
### Real world example

While troubleshooting a noise problem at a microwave sight in California, I did a visual check of the grounding system. The connections looked good and seemed to have very little corrosion. The voltage difference between the common earth point and the equipment was minimal, so I decided to check the cable to the earth ground point. The triple zero sized wire looked good until I reached the end at the actual earth grounding point. When I tugged on the cable going into the ground the wire quite easily came right out and I was holding the grounding point for the whole radio site in my hand.

It turns out that the hole where the earth ground was located was not draining. The water that was poured in every month corroded the wire until it broke off from the copper plates in the hole. To rectify this problem the grounding engineers put in a trench ground thereby replacing the pit ground. This example is a typical problem that can happen at a

site. Good planned maintenance is essential for consistently good working equipment. The cause of the above example was that the designers of the earth ground assumed that the hole would drain. One area of common presumption is the mains power system.

With mains and other power systems the grounding, or common system is the return path for the whole system. This means that in a series grounding system any common point along the way to the ground will be a different voltage when compared to any other point. The amount of current that can and does flow through the common wire can be quite large. These large currents are always looking for better or more paths to ground. If sensitive digital equipment is connected to the same ground as a high current device then it is very possible that the noise from the high current equipment will show up on the digital system. Single point parallel grounds and grid grounds are better than series grounds because there is no common return path to ground. Series single point grounding systems should be avoided at all times.



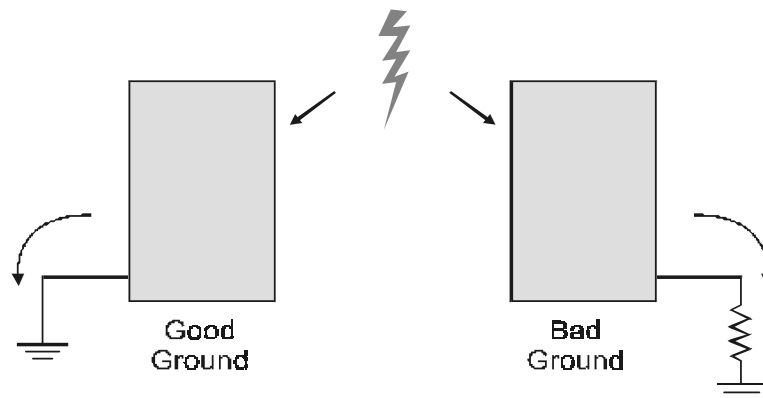
**Figure 8.4**  
*Ground as a return path*

The worst example of a ground path problem I ever saw was at a golf course in Western Australia. The ground and neutral wires corroded in the ground on the lines to the garden maintenance building. The 240-volt active line found a return path through the irrigation controllers on the fairways by way of the computer in the gardening shed. The AC voltage flowed through the computer in the building, jumped onto the common wire on the communications wire and then flowed through the lightning protection parts on the PCB in the controllers to ground. The electronic parts on the PCBs in the controllers became so hot that they literally fell out of the PCB (the PCB is upside down in the box). Luckily when the melted parts were replaced with new parts the controllers still worked, but the computer was toasted. But even there they were lucky because the computer didn't catch on fire and burn down the building. This is a pretty extreme example; most of the time return path noise causes only noise on the line.

In the past electrical engineers have looked at ground as a place to dump noise. All filters are designed to remove the unwanted noise from the circuit and put it in the ground. Unfortunately as electronic equipment has evolved it has become more susceptible to noise. This means that filters may only move the noise from one part of the circuit to another or from one device to another. Any point in the system that is connected to ground should be looked upon as a potential noise insertion point into the system. Devices that connect to a common at the bottom of the cabinet are very susceptible to this common noise. Most cabinet installations use a common ground bar at the bottom of the cabinet. The placement of equipment in the cabinet and how they are connected to the ground bar on the bottom of the cabinet can be an important factor in the reduction of noise.

From a noise point of view if two circuits or devices are completely separate from each other, the chances of them interfering with each other are minimal. When extremely noisy equipment is connected on the same mains equipment then it may be best to connect sensitive digital equipment on its own separate ground. It is becoming more and more popular to develop battery powered electronic equipment to completely isolate the electronics from high power devices. A good example of this is a microcontroller that is now in production in the UK that is completely battery operated. The manufacturer says that it will run for 5 to 10 years without replacement of the batteries. The microcontroller circuit communicates with the relays through a one-foot infrared link.

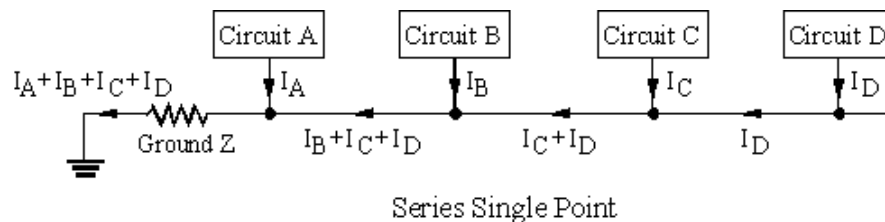
One of the situations in grounding is that when there is a possibility of high voltage entering the equipment it is better if the commons of different equipment are connected together. This is a trade off with the above noise problem. The reason for this is that if there is any resistance between close equipment when the voltage of one rises due to high voltage, the relative voltage to the other device will also rise. This can happen if some high voltage from a lightning strike makes its way to a piece of equipment in a cabinet full of different devices. A common belief is that lightning always looks for the best ground, this is not entirely true. In reality lightning will make a path for any and all grounds, even bad grounds. If two grounds are next to each other and one is a good ground and the other is a poor ground, lightning will probably take both.



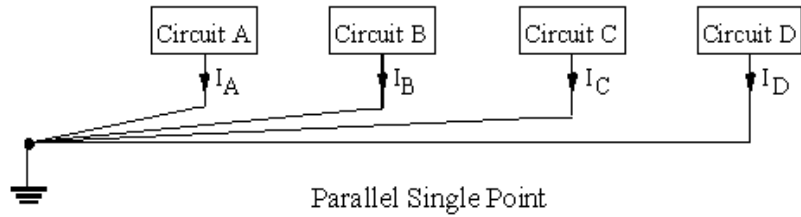
**Figure 8.5**  
*How lightning affects equipment*

## 8.2.2 Types of earth grounds

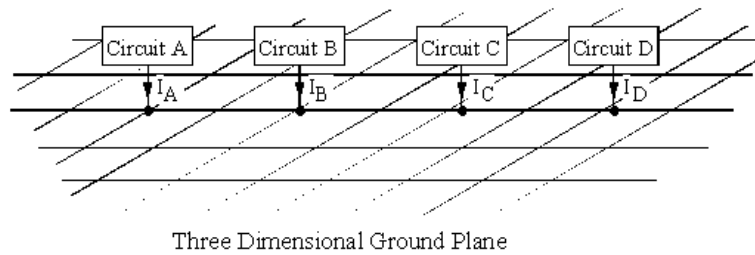
There are three basic types of earth grounds...



**Figure 8.6**  
*Series multipoint earth ground*

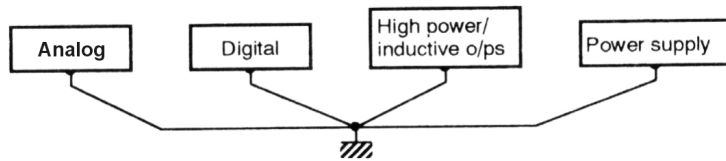


**Figure 8.7**  
*Single point parallel earth ground*

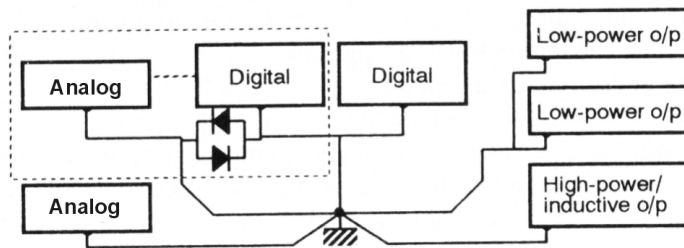


**Figure 8.8**  
*Grid earth ground*

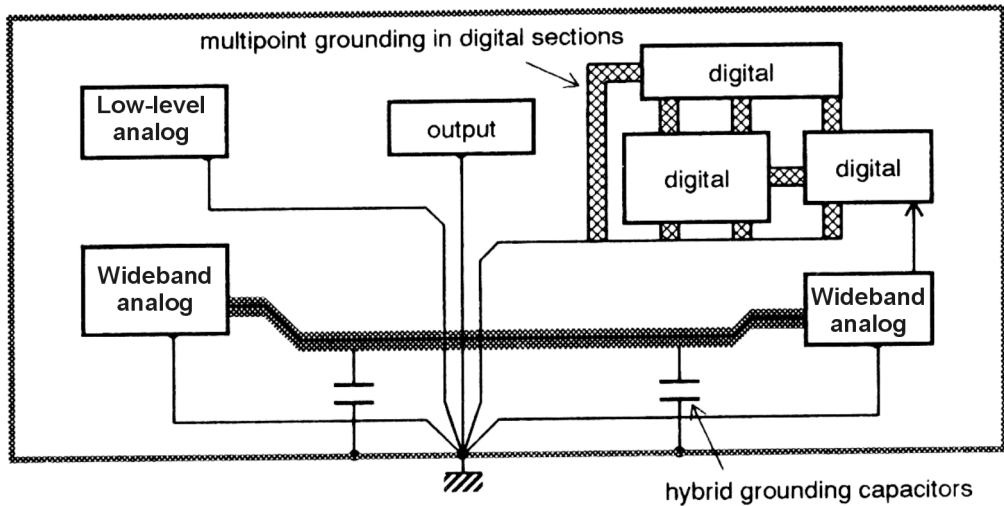
The most frequently installed earth ground is the single point parallel common ground as shown in Figure 8.7. Besides being used for earth ground the single point parallel type of ground can be used on a PCB. On a printed circuit board the single point parallel ground could be configured in three different ways...



**Figure 8.9**  
*Simple single point ground*



**Figure 8.10**  
*Modified single point ground*



**Figure 8.11**  
*Multipoint/single point ground*

## 8.3 EMC grounding on a PCB

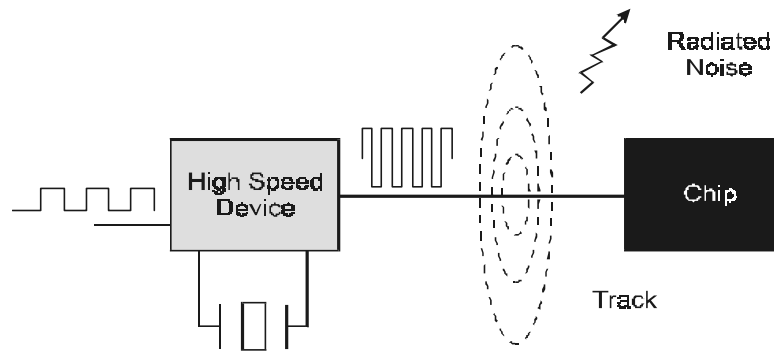
It has been found that one of the best ways of reducing noise on a printed circuit board is through good grounding and power distribution. Some of the old ways of laying out a PCB are incorrect, therefore we now layout PCBs quite differently. The designer uses correct placement of components, tracks, and Faraday boxes to create high quality and noise reduced PCBs. The biggest drawback to the placement of parts on a PCB with relation to better noise levels is that it becomes more difficult to layout the board. This means that the designer will take longer and the cost of the board and project will increase. Most countries around the world have passed laws or regulations that define the requirements for PCB manufacture in relation to EMI (electro-magnetic interference). This has forced most manufacturers to review and often redesign their products to come in line with EMI rules.

Due to recent changes in EMI regulations, laying out PCBs with respect to noise reduction it is now more defined. The following rules give us an idea of how to best design a PCB with minimum noise transmission.

### 8.3.1 PCB design recommendations

Identify the circuits that emit the most noise

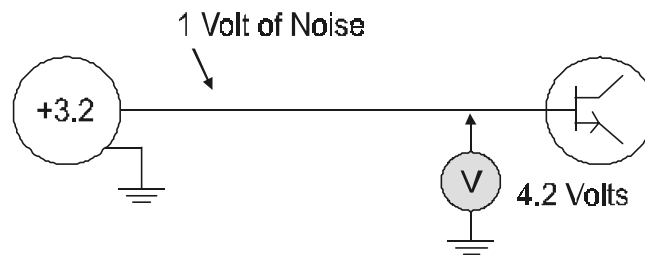
Any analog or digital high speed switching circuit can create radio emissions. These emissions can affect both circuits on the PCB or external circuits. Analog or digital high speed switching circuits should be kept well away from other sensitive circuits. The rule of thumb is to keep similar circuits next to each other and away from different circuits. If there are both analog and digital high speed switching circuits on the same PCB, then they should be separated from each other by using ground planes.



**Figure 8.12**  
*High-speed noise circuits*

Identify the circuits that are the most sensitive to emissions

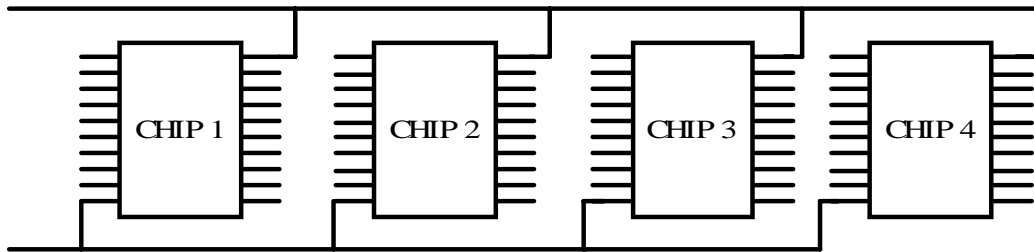
Low voltage analog circuits are more susceptible to noise than digital circuits. This is because digital circuits are saturation systems and analog circuits produce varying levels. This means that when a digital circuit is transmitting, it is only one voltage or another. This makes it less susceptible to changes in voltage than an analog voltage circuit. If the analog device is receiving some voltage and a little noise is inserted in the circuit then the output voltage of the analog device will change.



**Figure 8.13**  
*Analog circuits are very susceptible to noise*

Minimize possible ground inductance

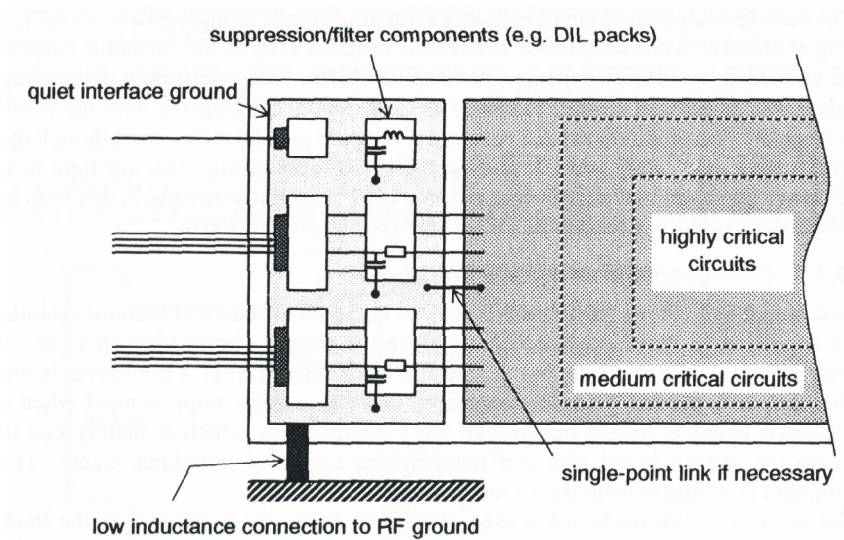
All common tracks on a PCB and external devices induce noise because of their commonality. The noise from one device is induced on the other through the wire or track that connects them. By using a parallel single point or grid type ground as opposed to series ground, the designer can reduce the noise induced from one circuit to another on the PCB. Connections from the power and ground rails to the chips should be kept as short as possible. When it is appropriate it is best to cover the board with a ground plane and leave just enough space for the components and tracks that connect them together. The comb grounding system that was popular in the past is not acceptable today, although it does make the tracking of the PCB easier.



**Figure 8.14**  
*Comb grounding system*

### Use a clean ground

A clean ground is a common that is separate from circuits and sub-systems on the PCB. This does not mean that the clean ground doesn't eventually connect to the other 'dirty' grounds. The clean ground connects the sensitive analog circuits on the PCB together and then connects at one point to the dirty ground system. Each PCB should have its own clean ground system. Each of the PCB's grounding systems are then connected together at one point. That one point could be the single earth ground point.



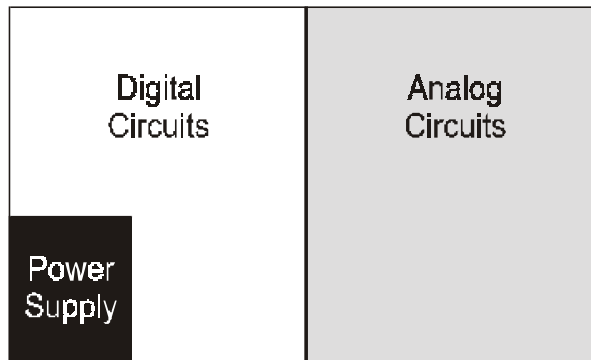
**Figure 8.15**  
*A clean ground*

### Separate different type of circuits

Analog, digital, and power supplies should be kept physically away from each other whenever possible on the PCB. This goes for complete systems in an enclosure or cabinet. Also wires from these circuits should be separated in the cable runs and conduits. Some distance should separate noise producers and noise sensitive equipment or circuits. Traditionally PCBs have been designed such that the PCB more or less matched the layout of the schematic. Usually the inputs are on the left and the outputs are on the right. From an EMI point of view, the board should be laid out such that noise producers are grouped together and separated from noise receivers. Circling each section with a ground plane physically and electrically isolates different sections from each other. Each of the ground planes should be connected together at one point, but analog and digital ground

planes should not extend over each other. If this is done, noise from the digital circuitry will be coupled to the analog circuit.

In the old days designers placed components on the PCB wherever it was convenient. The placement was usually done with the idea to reduce tracking problems. The main problem with tracking a PCB is getting all the tracks on the board into their proper place with a minimum of holes and crossovers. A crossover is when a track has to cross another track at a right angle. Since tracks cannot cross it is usually necessary to put a plated through hole in the PCB and then run a track on the other side of the board. Then another plated through hole is put in the board and the track continues. The price and complexity of the PCB increases every time a cross over is done. There are a limited number of crossovers that can be done on a board before the board becomes a mess. Multi-layer boards are used to reduce this problem. But this also increases the cost of the board.



**Figure 8.16**  
*Separate circuits on a PCB*

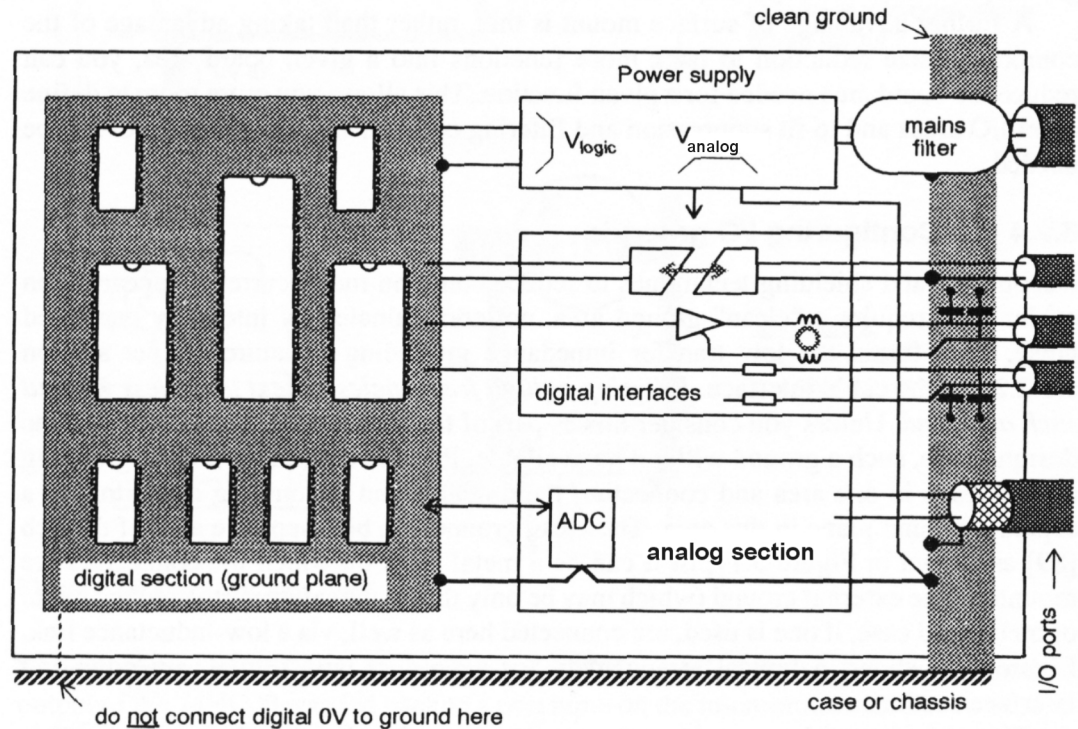
### Design proper PCB ground planes

There are many different types of PCB ground planes. The simplest is a wide track around the outside of the PCB. The most comprehensive grounding system is the compound grounding system on a multi-layer PCB. Most designers today fill every spare space on the PCB with a ground plane. This is not necessarily the best method of doing a ground plane. It must be remembered that the purpose of the ground plane is not only to connect the commons of the chips and components together but also to separate the different types of circuits.

The basic types of ground planes from worst to best are:

- The comb ground plane
- Ground track around the complete PCB
- A ground plane around each type of circuit
- One side of the PCB as a complete ground plane
- A ground plane on both sides of a multi-layer PCB
- A three-layer ground plane on a five-layer PCB

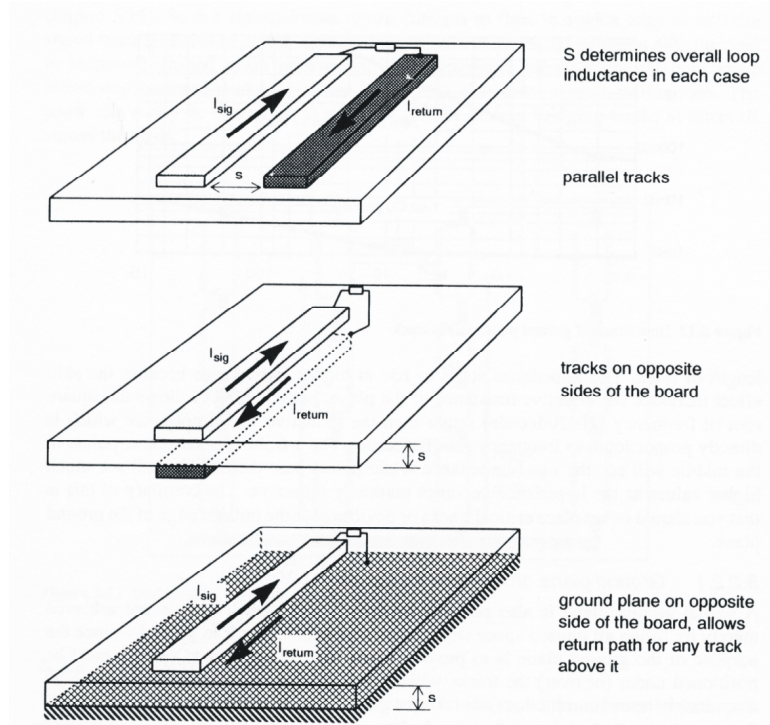




**Figure 8.17**  
Ground plane on a PCB

### 8.3.2 Track placement

When two wires are placed in close proximity and a current is applied to the wires two fields are produced. One is the electric field and the other is the magnetic field. The electric field is measured as capacitance and the magnetic field is measured as inductance. The electric field is proportional to the voltage divided by the distance between the conductors while the magnetic field is proportional to the current divided by the distance between the conductors. The further the tracks are from each other the less voltage and current is coupled from one track to another. Ideally it would be best to design the tracks on the PCB where none of the tracks are in parallel with each other. This is not possible as most digital systems use a common bus to connect the chips. So to minimize the coupling between tracks the designer can run short tracks and run the return path close and parallel to the signal wires. Short wires have less noise coupling than long tracks. And when the return path track is run in parallel and close to the signal the amount of radiated noise is reduced.



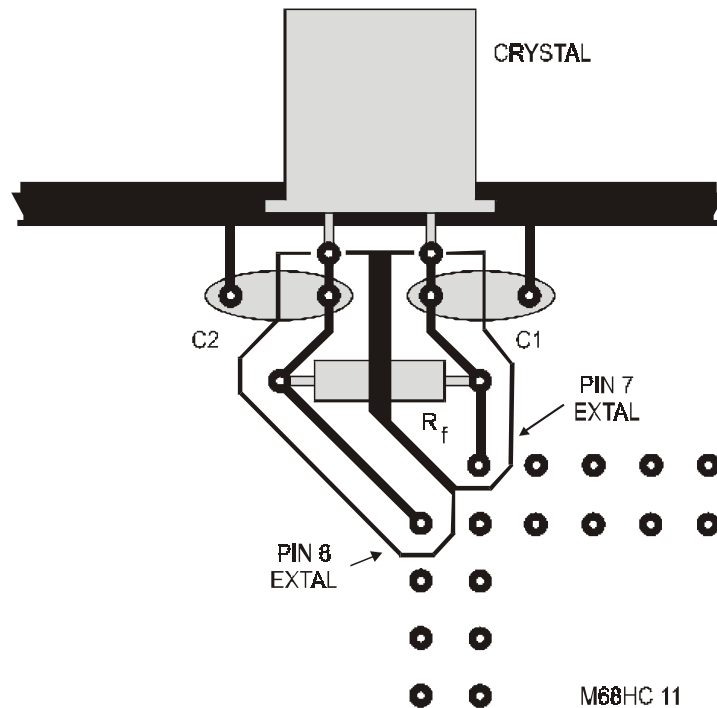
**Figure 8.18**  
*Track placement on a PCB*

### 8.3.3 Faraday boxes

A Faraday box is a metal box, track or screen surrounding an electric circuit. It can take many forms, from copper on a PCB to a complete room. The function of the Faraday shield is to protect devices from radiating or receiving electromagnetic fields. It does this in three ways. One is to reduce radiating electromagnetic noise by coupling the noise onto the Faraday shield or box instead of the circuit. Next it provides a ground path for the voltage and current created by the electromagnetic field. The third is to provide a shield against incoming electromagnetic fields. There are three methods of building a Faraday shield.

#### Floating – one dimensional non-grounded

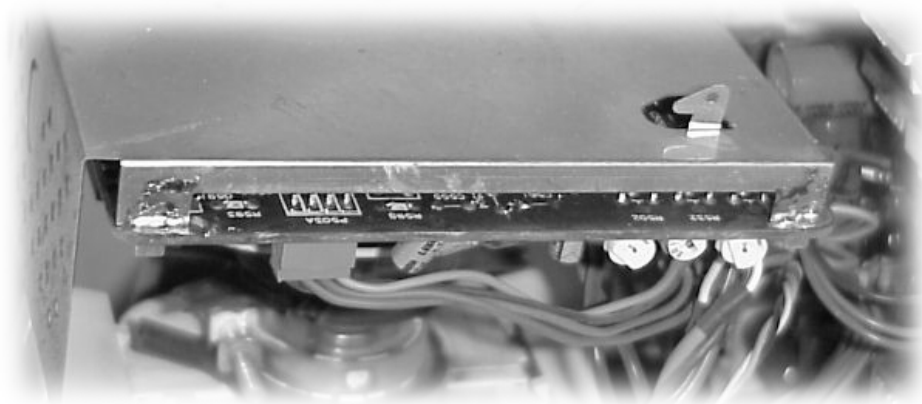
The floating non-grounded Faraday shield is a screened area or box that is not connected to anything. This type of shield is of minimal use unless the shield is made of steel or some other ferrite material to shield against magnetic radiation. The non-grounded shield is sometimes seen on PCBs as a single copper track around a crystal or sub-circuit. This is of minimal value and it would be better if the track was connected to ground.



**Figure 8.19**  
*One dimensional Faraday shield on a PCB*

#### Grounded – earth ground

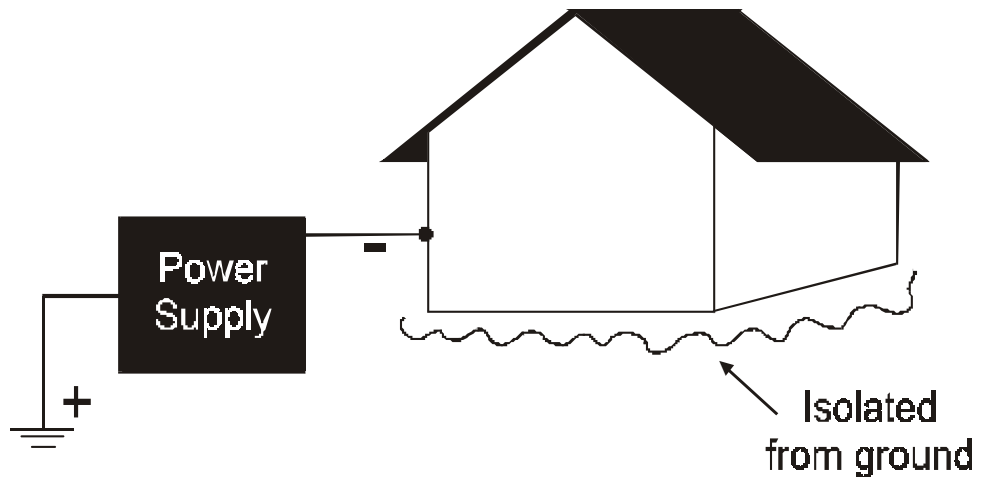
The grounded Faraday shield or box works better than the non-ground type. The ground provides a return path for the current created by the electric field created by the transmitting circuit. Because the ground provides a return path for the electric field, this type of shield works better for electric fields than magnetic fields. If the shield or box is made of a ferrite material then the grounded shield will work good for both types of fields. This ground should be a clean ground whenever possible. If the Faraday shield is in the form of a box or even a room, the shielding material should be connected to its own earth ground.



**Figure 8.20**  
*Typical grounded Faraday box*

### Negatively charged

In its best configuration, the Faraday shield should be connected to a negatively charged power supply. This effect works similar to what's commonly called the Edison effect. This effect is a function of the negatively charged electric and magnetic fields being repelled by the negatively charged shield. It also provides a return path for the electric fields. If the shield is also made of a ferrite material then the protection is provided for magnetic fields. The Faraday shield that is negatively charged and made of ferrite material is the best method for protecting against electromagnetic radiation. The negatively charged Faraday shield is made by connecting the positive lead of a power supply (a battery works best) to an independent earth ground. The negative lead is then connected to the shield. The shield must be completely isolated and insulated from normal ground or voltage. The biggest problem with negatively charged Faraday shields is corrosion due to electrolysis. To reduce the corrosion it is best to only turn on the Faraday shield when needed.



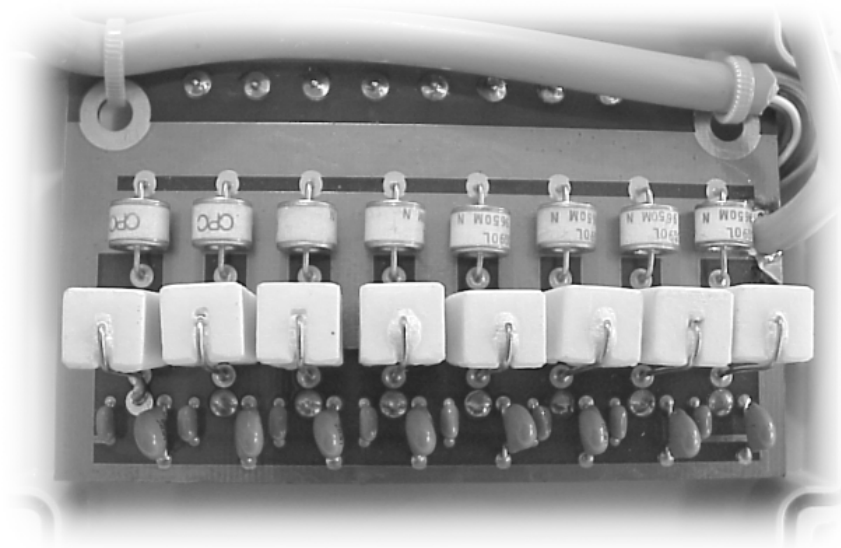
**Figure 8.21**  
*Negatively charged Faraday room*

## 8.4 Protecting a PCB from lightning

In Singapore, a city manager once asked me why we couldn't protect our electronic equipment against a 'little bit' of lightning. This is like trying to protect the equipment against a *little* atomic bomb. If God looks down and decides to take out your equipment, it is gone. There is no complete protection against lightning or other high voltages. Lightning protection is like safety in general, there is no such thing as being completely safe, there are only levels of safety. In lightning dissipation there are only levels of protection, not complete protection. Many people have been very disappointed when after spending thousands of dollars on lightning protection and the very first storm damages their equipment.

To give sensitive electronic equipment some measure of protection from lightning protection, designers do their best to give the lightning another return path. The idea is to short-circuit the lightning to earth ground instead of through the equipment. It is thought that the ideal equipment, from a lightning point of view, is the totally isolated device. The aeroplane is a perfect example of this. Thousands of aeroplanes get struck by lightning every year and only a few of them are seriously damaged. But they do get struck. Why? Everything on or near the earth is referenced to earth. One common misconception is that

lightning follows the best path to earth ground. It has been my experience that lightning follows any path to ground it wants. It will take any ground it can get and sometime for some strange reason a seemingly worst ground rather than an obviously better ground. This is why even the best-protected equipment can be damaged by high voltage or lightning. Having said that, any protection is better than none.



**Figure 8.22**  
*Typical lightning protection device*

Lightning rods were invented by Benjamin Franklin in the seventeen hundreds. The idea of the lightning rod is to collect the high voltage from the lightning and provide a safe path to earth ground through a wire instead of through the building. Ever since the first rods were installed there has been a controversy on whether the lightning rod attracts lightning. Most lightning rods are short to reduce lightning attraction. They are often placed on the sides and corners of the building because it was found that most lightning bolts strike the edge of the building and not necessarily the very top. Round balls with short spikes are just some of the many fads that have come and gone in the lightning industry over the years. Others have been, negatively charged spikes, positively charged balls, and large high ground spikes placed some distance from the building.

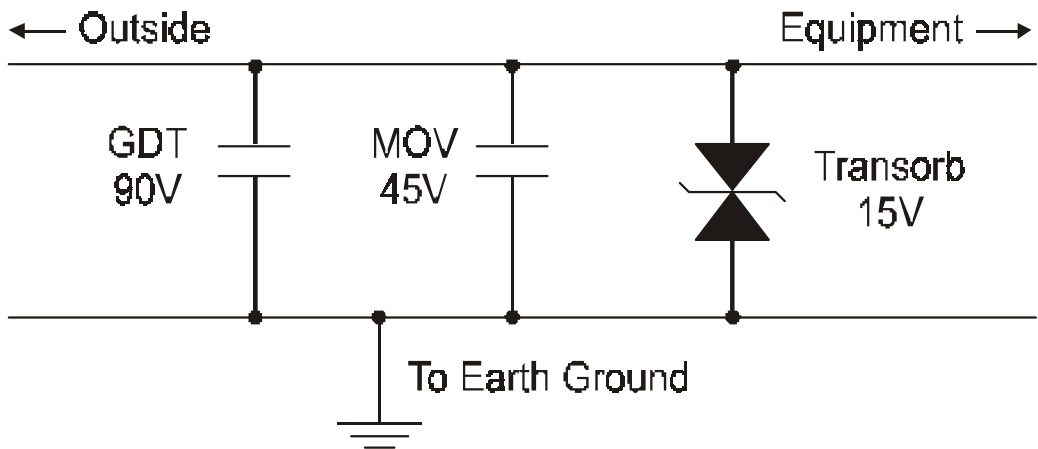
#### **8.4.1 Placement of protection on the PCB**

High voltage, such as lightning, gains entry to most equipment through the outside wiring. This is why the location of the high voltage protection is important in the reduction of lightning. On external cabinet installations the lightning protection is usually located at the bottom of the cabinet where the cables come in from the outside. On a PCB, the protection is located right next to the connector where the outside wires are installed. It is important to remove the high voltages as soon as possible before they can do any damage. The best method of protecting a cabinet from lightning would be to have a separate inspection box that contained the lightning protection. This would physically separate the lightning protection from the cabinet with the controller equipment, but this is rarely done.

### 8.4.2 The GDT, MOV and transorb

- GDT                    Gas discharge tube
- MOVs                 Metal oxide varistor
- Transorbs           Bi-directional semiconductor

GDTs, MOVs and Transorbs are three of the most common components used for lightning protection. As mentioned before, there is no perfect high voltage protection, but the use of these devices does improve the chances of the equipment surviving a relative small high voltage strike or static discharge. The functions of the GDT, MOV and Transorb is to provide a path for the high voltage to earth ground before it has a chance to get into the equipment and damage the circuitry. In Figure 8.23 it can be seen that the GDTs are on the outside of the circuit, the MOVs are in the middle and the Transorb is on the circuit side.



**Figure 8.23**  
*Placement of lightning protection on a PCB*

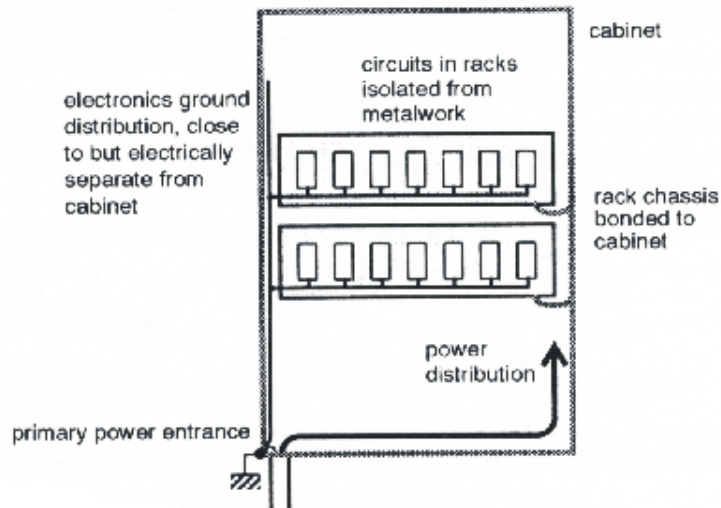
When lightning produces high voltage or static in the outside world, it travels into a device looking for a return path to earth ground. The first device that will react to incoming high voltage is the transorb. It will short out very fast when the voltage is higher than 15 volts (in this example). This is because the transorb is a silicon device similar to a zener diode. It will short out in the microsecond range. Unfortunately the transorb is very sensitive to high voltage and will be destroyed as the voltage rises. But by this time the MOV has shorted out (when the voltage goes above 45 volts) as it is a little slower reacting than the transorb. But again the MOV may not survive this ramping up of high voltage and there is a good possibility that the MOV will be destroyed. Hopefully by this time the GDT has shorted out and it is sufficiently strong to short the remaining high voltage to earth ground.

Notice that as the high voltage is coming in to the device it is shorting out in the direction of the outside world. The problem is being moved out of the device.

**Note:** One of the best things that can be done to reduce the possibility of lightning damage when a lightning storm is around is to turn off and if possible disconnect any device that is connected to mains power or a telephone line.

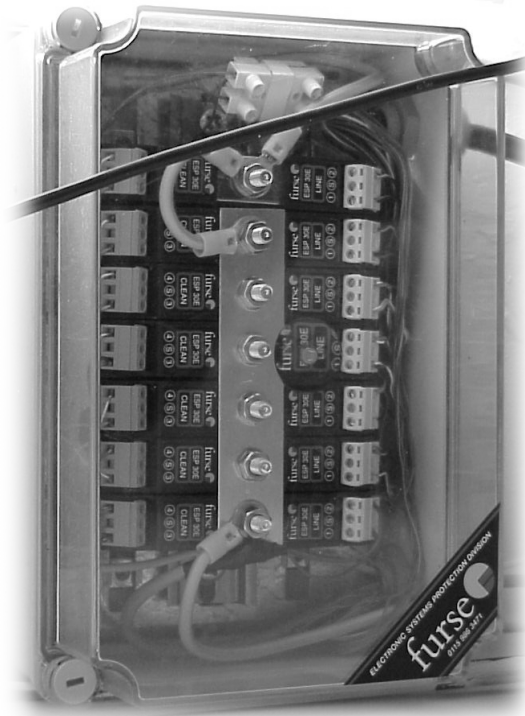
## 8.5 Microcontroller equipment ground

Whether connecting a microcontroller in a cabinet or 19-inch rack it is very important that the grounding system be designed and installed correctly. Different industries and companies have their own methods and rules for grounding systems. It is not our intent to supersede these. This is only a guide to help the reader understand the basics of grounds and the common practices used in industry for grounding microcontroller equipment.



**Figure 8.24**  
*Proper ground connection*

In Figure 8.24 it is shown the proper way to connect microcontroller equipment to ground and to safety ground. Notice that the equipment ground is connected to the safety ground at one point in the bottom left-hand side of the cabinet. Inside the cabinet the equipment has its own ground and that it is not connected to any other ground until that one earth ground point. Also note that the mains power is run on the opposite side of the cabinet. The drawing shows the power entering the cabinet on the same side of the cabinet as the earth ground. This is to keep the ground lines short from the mains power to earth and safety ground. The equipment must be completely isolated from the cabinet and any metalwork that connects to the cabinet.



**Figure 8.25**  
*The wrong way to connect ground straps in a box*

## 8.6 Enclosure or safety ground

The enclosure or safety ground is earth ground. Some type of earth stake or trench cable ground is connected to the cabinet in the field. The purpose of this ground is to provide a return path and reference for the devices in the cabinet. It is also to provide a safe environment for anyone that might come in contact with the equipment or cabinet. If the cabinet was allowed to float above ground, then it is possible for the cabinet's voltage potential to become well above earth ground level. If the potential was to get too high and a person touched the cabinet while standing on the ground, it would be possible for that person to get shocked. It is not enough to bolt the cabinet to a cement slab. It is possible, as the ground dries out that the slab could become electrically isolated from ground potential. To properly ground a cabinet the cabinet must be connected to an earth grounding system such as an earth stake or trench earth ground.

### 8.6.1 Spiked earth grounds

The earth ground spike is a copper plated steel rod about two meters long. It is pointed on one end and flat on the other. The pointed end is placed on the ground and then it is hammered into the ground until only a few inches is above ground level (see Figure 8.26). The part above ground level should be high enough to connect the green and yellow striped wire to the spike with a clamp. Only enough wire should be stripped so that only the wire fits in the connector. More wire sticking out of the connector could snag on the unwary pedestrian. If the wire is too short it may fall out of the clamp. The ground stake is inserted in the earth at a location very close to the cabinet. It is usually located behind



and on one side of the cabinet. Sometimes if the cabinet has an open or false bottom it is often inserted in the ground there and then the cabinet placed over the ground stake. Often the installers will place a concrete pad with a large oval hole in the center for the cables to come up through. On one side of the oval the stake would be inserted in the ground and then the box placed on top. At no time should the ground stake be cemented in with the pad. Ground stakes need maintenance and may have to be replaced. Local installation rules and standards must be followed.

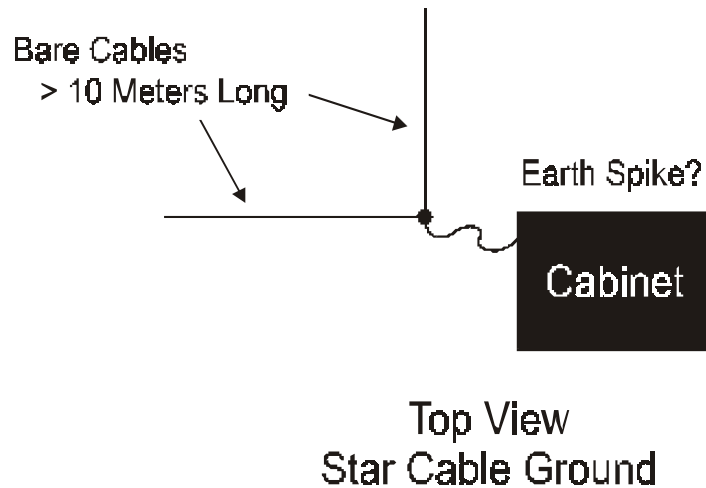


**Figure 8.26**  
*Spiked earth ground*

The connection should be checked regularly for corrosion or damage. If the earth is very dry it may be necessary to insert a plastic pipe next to the stake and fill with water during dry times of the year. The strangest stake installation I ever saw was at a park in Los Angeles. The installer placed the stake on the ground where it was to be inserted and got up on the ladder to hammer it in the ground. He hit the stake once and it went in about a foot. On the second hit the stake disappeared completely into the ground, never to be seen again. Both of us just stood there looking at the small round and now empty hole in the ground. We found out later that the park was built on an old trash dump (it's a worry). The stake is now in some empty chasm under the park. The location for the stake was changed and the next one went in correctly.

## 8.6.2 Cable trench grounds

When designing an installation it is necessary to take the natural environment into consideration, especially considering the earth ground. If the ground around the cabinets is normal soil, then an earth ground stake would be used, but if the ground is either solid rock or light dry sand then the trench ground may be best. The type and design of the trench ground depends on the ground type. The trench earth ground consists of a trench dug in the ground from one to three feet in depth. An uninsulated cable or steel copper clad rods are then laid in the trench. Vertical ground stakes are often added at the ends and middle to increase the earth ground area.



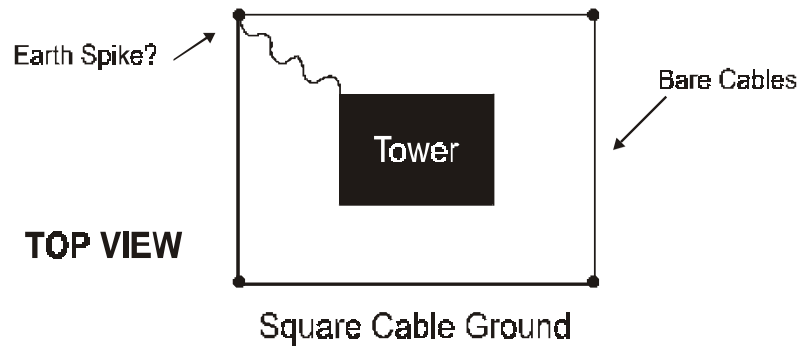
**Figure 8.27**  
*Trenched earth ground*

In very rocky ground, where there is little topsoil, it is often necessary to blast multiple trenches in the rock in a star pattern. The cable is then laid in the trenches and covered with topsoil. The soil holds the water that is poured in the trenches. It may be necessary to place water in the trenches from time to time. It is best, in this case, to leave the trench as a depression in the ground so that water collects in the trench. The cables should be checked every few months or at least bi-yearly for corrosion. The cables are then connected together at one point and this is where the site or cabinet safety ground is connected.

In dry sandy areas, like the desert, it is also often hard to obtain a good ground. In this case the multiple star (good) or square (better) cable grounds (see Figure 8.28) are used. Because the sand is dry, multiple stakes are also often incorporated into the grounding system. Water is often used to increase the quality of the ground. If the sand is near the beach or in a location where there is good soil under the sand then a well could be drilled and the steel pipe left in the ground and used as the earth ground. Because this pipe can be some distance from the site, a large uninsulated cable is placed in a trench and connected to the pipe.

### 8.6.3 Tower lightning protection

Towers use trench grounding as a method of increasing protection from lightning strikes. A square trench is dug around the bottom of the tower to a depth of approximately one or two feet. An uninsulated cable is laid in the trench and connected by welding or screw clamp the ends. It is best to put earth ground stakes in the ground at each corner of the square. If it is possible a cable or even better an aluminum tube is brought down from the lightning rod at the top edge of the tower. Often multiple lightning rods are used. The tubes are connected to the ground stakes at each corner. The tubes are stood off from the edge of the tower as it comes down to the ground.



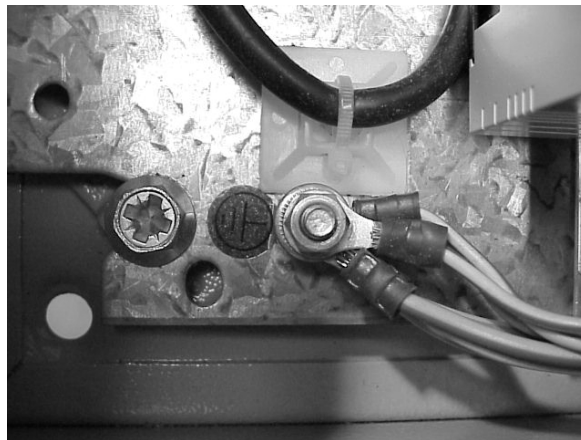
**Figure 8.28**  
*Tower lightning protection*

## 8.7 Conclusion

Even though our view of what 'ground' is has changed over the years, it is still a large part of the noise reduction in PCB, microcontrollers and all electronic equipment. The perception of what ground is and how it is used has changed due to changes in electronic equipment. As speed increases in electronics, the need for design rules to reduce radiated noise becomes more important. Also the low power of the electronics increases its susceptibility to external noise. Misconceptions about commons, ground planes and earth grounds often cause designers to create circuits that radiate excessive noise. And besides radiated noise the designer must contend with reducing the possibility of the circuit to receive noise.

Misconceptions about ground are:

- Ground is never 0 volts
- Ground is never 0 ohms
- Ground is often a return path for the current of a system
- Ground should be treated as a noise insertion point into our system
- A non-grounded system is the quietest from a noise point of view
- A grounded system has better high voltage or lightning protection than a non-grounded system



**Figure 8.29**  
*Single point parallel ground*

For the PCB designer, the placement of components and design of proper ground planes are the main technique to reduce both radiated noise and the equipment's susceptibility to noise. The segregation of the analog and digital components is the first step in reducing noise coupling. High-speed digital circuits often create excessive radiated noise while analog circuits are very receptive to noise. It is necessary then to separate these circuits physically on the PCB and surround them with a ground plane. It is also important that the ground plane of the digital and analog circuits do not run in parallel on top of each other. A Faraday shield is used to reduce noise in extreme situations. The three types, floating, grounded and negatively charged can be used to isolate circuits and equipment from radiated or induced noise.

Protection from lightning and high voltages has become very important because of the proliferation of electronic equipment. Only a few years ago, motorized clocks were controlling devices. These types of controllers are very resistive to high voltage, but now all devices are controlled by highly sensitive microcontrollers. To adjust to this increased sensitivity lightning reduction systems have changed. One way that lightning dissipation devices have changed is that in the old day's metal straps or cable was the usual way to convey the current from a strike to the earth ground system. Now it is more common to use aluminum or copper tubing. This is because of the increased surface area of the tubing as opposed to the straps or cable.

# Installation and troubleshooting

## Objectives

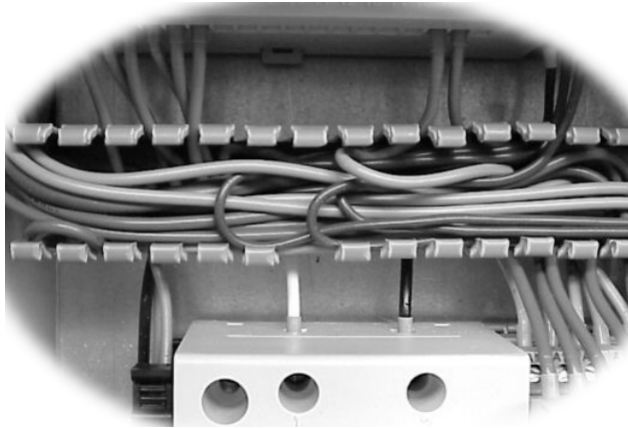
When you have completed this chapter you will be able to:

- Describe the different types of connectors used in microcontrollers
- Explain how to properly connect a wire into a screw connector
- Explain the potential installation problems with screw connectors
- Explain how to solder wires together for a good connection
- Describe two different types of cable runs
- Describe how to place different types of signal cables in a cable run
- Explain how to build a ladder type cable run
- Explain how to pull wire through a conduit

## 9.1 Introduction to installation and troubleshooting

In this chapter we will discuss proper installation and troubleshooting techniques as they apply to microcontroller systems. It is not our intention to supersede any standards, rules or regulations defined by your industry. The purpose of this chapter is to impart some common sense installation and troubleshooting techniques that specifically relate to microcontroller systems. It is well known by most experienced technicians that the installation techniques used in one segment of the electronics industry do not work, or are not the optimal installation solution, for another. An example of this would be the installation of a radio system compared to the installation of a data acquisition system. Data acquisition systems don't usually have coaxial cable or N type connectors. Having said that, there are a lot of similarities between different types of electronic systems. This is often where the installer, technician or engineer runs into trouble. They apply the techniques from one section of the industry to another and it doesn't work. Something as simple as tightening a screw connector on one type of equipment can be different from another. One of the major differences between microcontroller systems and other systems is that microcontrollers are more susceptible to noise. If the installer tries to install a

microcontroller system in the same manner as a high power motor system, there are going to be problems, guaranteed! If a data communications system is grounded in the same way as the mains power system, it will be a miracle if the communications work at all!



**Figure 9.1**  
*Equipment installation*

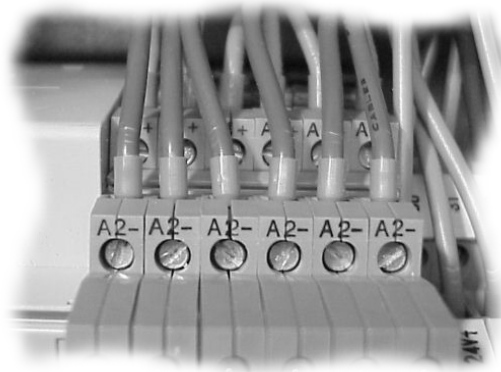
Two very common problems found in the installation of microcontroller systems are bad connections and noise from other equipment. Both of these problems can be reduced or made worse by the design or installation. The designer can use and specify quality connectors and proper installation methods. The installer can follow the defined installation procedure and make suggestions when something doesn't seem right with the installation. It is important for the technician to also be aware of the proper installation techniques when repairs need to be done. In repairing the equipment the technician usually has to remove and then reinstall the equipment. Knowledge of proper installation techniques will help the technician reinstall the equipment correctly.

## 9.2 **Connections – screw, crimp and solder**

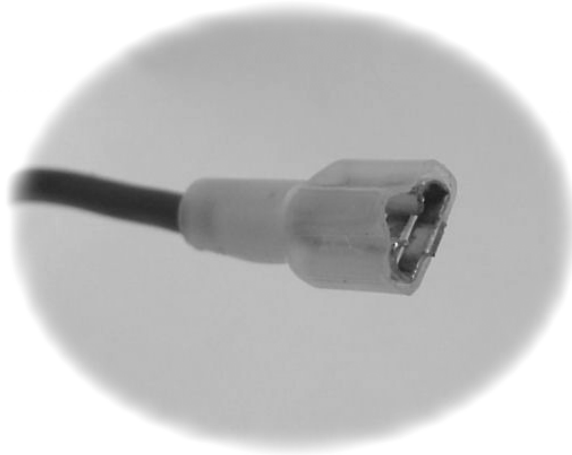
An experienced troubleshooter knows that the first test of a device that has failed is to check the power. The next check is whether the connections are correct. This usually starts with the power cord and then moves to any exposed connections. The reason this is done is that approximately 60% of problems with equipment that 'worked this morning' and 'doesn't work now' are probably due to a loose or bad connection. (Assuming that there is no visual problem, like smoke billowing from the case.)

There are three basic types of connections on microcontroller systems:

- Screw connectors
- Crimp connectors
- Soldered connections



**Figure 9.2**  
*Crimp and screw connectors*



**Figure 9.3**  
*Crimp connector*



**Figure 9.4**  
*Large solder connections*

### 9.2.1 Screw connectors

There are many different types of screw connectors, but they all have some things in common. The basic idea of all screw connectors is to pinch the wires between two pieces of metal and insulate the exposed portion of the wire from other wires or the equipment. The way the wires are pinched is the major difference between screw connectors. One type uses the moving box method and another uses a moving flap. The box method (Phoenix brand type) has a small metal box that is connected to the screw in the connector. When the screw is turned the box moves up and pinches the wires against the top of the square hole in the connector. This type of connector is considered better than the flap type, but it does have one problem. If a small solid wire is inserted in the connector, the box can act like a guillotine and cut the wire as the box is raised. To avoid this problem the correct size multicore cable should be used. To make the connection even better, a crimp ferrule unit can be attached to the wire before being inserted into the connector. It is debatable whether it is better to use the ferrule units if more than one wire is to be placed in the connector. If done properly either method will work correctly. See the following examples.



**Figure 9.5**  
*Good and bad screw connection*

The flap type screw connector moves a spring-loaded flap by turning a screw from above. The screw is not connected to the flap and therefore the flap may have to be manually pushed up if the connector has been used before. Whereas the box type of connector pinches the wires flatly, the flap type connector pinches the wires at an angle and therefore is not as reliable as the box type. Again solid wires do not work as well as multicore wires in the flap type connector. Even the crimp ferrule can be a bit of a problem with flap type connectors.

Both types of connector come as single and two part connectors. The single PCB mount is about half of the price of the two piece connector. But it is recommended that the designer use the two piece connector. The one piece is not too bad if only a few connections are to be made. If more than four wires are to be used in the connector, or if the connector is going to be connected and disconnected a number of times, then the two-piece connector is required.



## 9.2.2 Crimp connectors

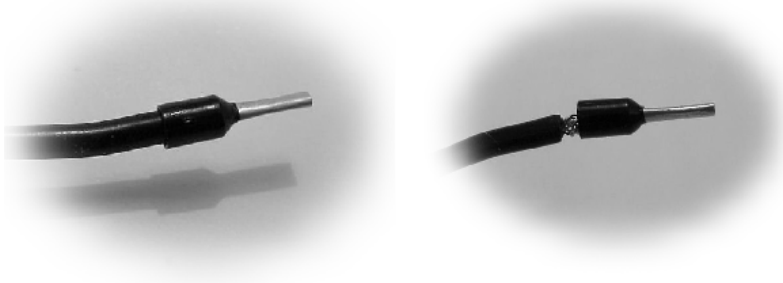
When crimping connectors most installers have problems because of the following:

- The wires are too big or small
- The wire is not stripped properly, wire too long or short
- Using the wrong crimp connector
- Using the wrong or cheap crimp tool
- Crimp too tight or too loose
- Using solid wire instead of multicore

Using crimp connectors, if done properly, is a good way of making a quality connection. The advantage of the crimp connection is that the wire makes more contact with the ferrule than it would if it was just stuck in a screw connector. Also because they are crimped inside the connector, they have less chance of touching other wires.



**Figure 9.6**  
*Ferrules for crimping*

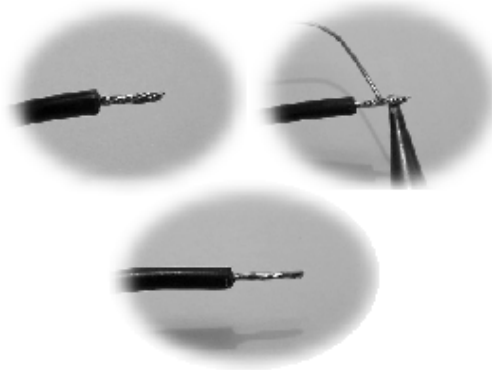


**Figure 9.7**  
*Good and a bad ferrule crimp*

### 9.2.3 Soldering connections

Most people would assume that the solder connection is the best connection, but this is not necessary the case. Often it is better to crimp the connection than solder it. The reason for this is that solder does not spring back. If the end of a wire is soldered and then put in a screw connector, over time the connection could become loose. As the connection heats up and cools down the lack of springiness of the soldered end will cause the connection to fail.

If you are going to solder wires together then the first and most important thing that should be done is to clean the wires. This is not usually a problem with new installations. The next step often causes problems. Place the proper size shrink tubing over one wire before you solder them together. Don't forget. Next the wires should always be twisted together. Always verify that there is a good mechanical connection before soldering. Next the wires should be heated, NOT the solder. Don't paint the solder on the wires. Heat the wires and then apply the solder to the wires, not the iron. Once the solder has flowed into the wires remove the iron and hold the entire connection steady until the connection has cooled. I usually count to twenty, but this varies with the size of the connection. Larger connections take longer. The shrink tubing is then slid over the connection, heated and shrunk.



**Figure 9.8**  
*Soldering the wires*

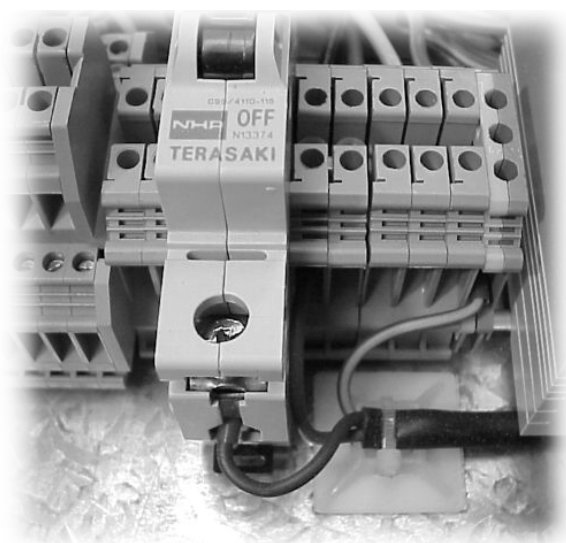
### 9.2.4 Connector problems and solutions

All types of connections whether they are screw, crimp or the solder type can come loose. This happens because of the following:

- Vibration
- Cycling from extreme cold to extreme heat to extreme cold etc
- Screw connector not installed tight enough
- Soldering the wires together on screw connector
- Wires not twisted together before insertion or soldering
- Solid instead of multicore wires were used
- Too many wires were forced into the connector
- Wires are too small
- Wire is not centered in the connector
- The connector was disconnected by pulling on the wires
- Wires were not visually and mechanically checked regularly for tightness

No connector can stand up to excessive vibration. Even the largest bridge or building will fall if given the correct vibration for long enough. Equipment should be installed in a vibration free location if feasible. If this is not possible then some type of shock absorbing material for mounting should be used. When equipment is heated and cooled repeatedly it can be looked upon as very slow vibration. As the wires expand, when heated, and contract when the wires become cool, the wires can become loose. Often this is difficult to control but if possible use air conditioning or at least a fan to cool the equipment. It would be good if the controller temperature could be tracked and if there was a problem then the connections could be checked more often.

Verify that the installer is familiar with the type of connector that is being used. Don't assume that just because the installers have decades of experience that they are familiar with the connector that is now being used. No one knows it all. Check with the manufacturer on the torque needed for tightening and use a torque-limiting screwdriver when installing the wires. As mentioned before, wires should never be tinned or soldered before put in screw connectors. Soldering the wires together is often thought of as a good practice, but when soldered wires are inserted into screw connectors they can become loose very easily. This is because the solder is not springy. It compresses and stays there. As time goes on the wires will become loose easier than if not soldered. Before any connection is made, there should be a good mechanical connection. If wires are to be soldered then they should be twisted together. If two wires are to be placed in a crimp connector then the wires should be twisted before inserted in the connector. For screw connectors the wire also should be twisted before inserted in the connector. Often if two wires are put in a screw connector without being twisted one wire will pinch and the other one will be crammed in the corner. The wire in the corner could become loose.



**Figure 9.9**  
*Proper screw and crimp connection*

Use only the proper size, type and amount of wires specified by the connector manufacturer. Most manufacturers do not recommend solid wires to be used in their connectors. This is because the solid wires do not spring back as well as multicore wires. Solid core wires are also affected more by vibrations and temperature. When too many wires are forced into a connector and it is beyond specifications it is obvious that there will be problems. Or if the wires are too small they may break easily or fall out.

It seems pretty obvious that it is not a good idea to disconnect a connector by pulling on the wires, but it is surprising how often it is done. The resolution for this problem is obvious. **DO NOT PULL ON THE WIRES** to disconnect the connector, pull on the back shell only. Also it is important to turn the power off to the equipment before disconnecting any connectors.

All installations should have some form of planed maintenance. A visual and mechanical check should be scheduled and done often. The mechanical check could consist of checking the torque of the screws on screw connectors. When first installed wires should be given a **light** tug to make sure the wires are secure. This should be done with the power off if possible.

## 9.3 Cable runs and trays

Cable runs and trays would on the surface seem to be an inconsequential and a fairly straightforward subject, but the type of tray and its placement can be the difference between a system that works and one that doesn't. The manufacturer's specifications should be followed to reduce noise and to insure a quality installation. There are many types of cable runs and ladders with plastic cable runs and steel ladders being the most common. The purpose of the cable runs is to hold the wires that connect equipment together. Some runs are enclosed while others are left open. Often the wires start at a nineteen-inch rack and are run up the wall in parallel to each other. Once up in the ceiling the wires are then bent horizontal and run across the ceiling to another wall. The wires are then bent vertical again and down the wall to the equipment on the other end and terminated at the end equipment. Sometimes the wire is split off while in the ceiling to different types of equipment.



**Figure 9.10**  
*Cables in a cable run*

### 9.3.1 Metal vs plastic runs and trays

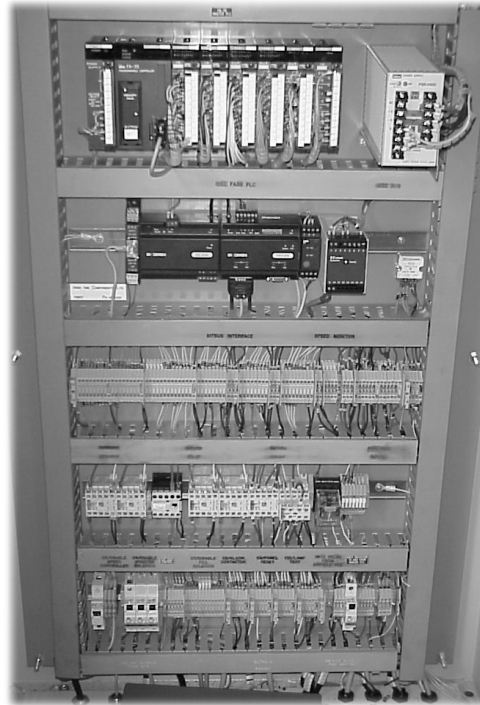
From a noise reduction point of view the completely enclosed steel cable tray is the best. The steel box that enclosed the wires is the best type of cable run to protect wires against external electromagnetic noise. It is OK to leave the top of the box off, but it does reduce the noise protection of the cable tray. Steel cable runs should be galvanized or painted as a protection against rust. Probably the most popular type of cable run is the metal ladder. The ladder cable run is cheap and allows the wires to be easily routed between cable runs. The cables are often tie wrapped to the rungs of the ladder. The main problem with the

ladder cable run is that it does not give any protection against external noise. It also may increase the noise by allowing the wires on the cable ladder to run parallel with each other.

Plastic cable runs or trays are very popular in industry due to the low cost and ease of installation. They are often used in cabinets and racks to keep the wires organized and separate from the equipment.

To reduce the possibility of noise affecting the wires in either the metal ladder or plastic cable run it is best to:

- Run like cables together
- Do not tie wrap cables together on horizontal cable ladders
- Try to keep wires from being run in parallel
- Run mains power cables in their own cable run and separate from other wires



**Figure 9.11**  
*Plastic cable runs*

### 9.3.2 Vertical runs and trays

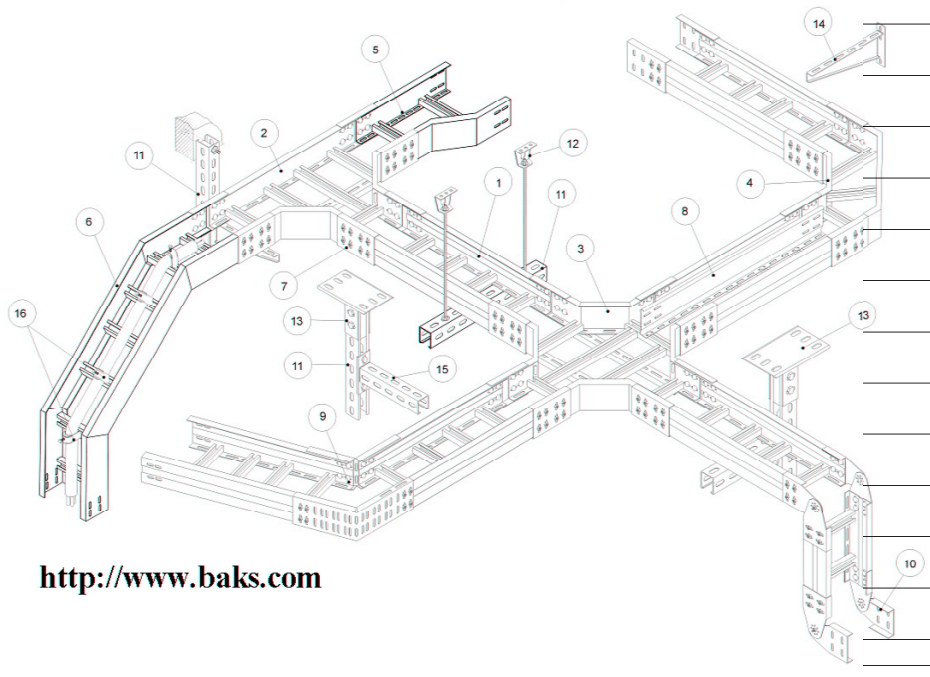
Because vertical cable runs are usually short runs, the possibility of noise affecting the wires is minimal. Therefore, cables are usually run in parallel up the wall and attached to a steel cable ladder. The ladder is mounted to the wall close to the equipment and stood off approximately 25 mm from the wall. The width of the cable run will depend on the amount of wires needed to be installed. Most of the time the wires are bungled and tie wrapped together to no more than 100 mm thick. The bungle is then tie wrapped to the rungs of the ladder in approximately one meter or less lengths.



**Figure 9.12**  
*Vertical cable run*

### **9.3.3 Horizontal runs and trays**

As mentioned before the best type of cable run is the steel box. When the steel box is used in a horizontal cable run it provides the optimal protection against noise from other cables. The chance of noise being induced from one cable to another is minimized if different types of cables are placed in their own steel box cable runs. This becomes especially important for horizontal cables because they are usually extremely long. The cables in the horizontal cable run should not be placed in parallel to each other. The cables should be placed in a random pattern in the cable run. There is no need to tie wrap cables to each other in a horizontal cable tray as long as they are not going to fall out.



<http://www.baks.com>

**Figure 9.13**  
*Horizontal cable run*

## 9.4 Cable ties and mounting

One of the most common problems that can occur with cable ties, with respect to mounting cables, is incorrect tie wrap tension. The installer either cinches the tie wrap too tight or it is left too loose. Often it is better to be too loose than too tight. If the tie wrap is too tight it can kink and damage the wire. This is especially true for fiber optic cables. Fiber optic installations should use Velcro tie wraps and which should not be pulled too tight. For normal cable installations a torque adjustable tie wrap gun is preferred. These guns come as manual, electric or pneumatically powered. The torque is set according to the type of wire used and should follow the manufacturer's recommendations. It is also important that the mounting device is secure and will not be damaged by the tie wrapping torque. A slight tug should be used to check that the wires are secure after being tie wrapped. Often tie wraps are defective and do not grip. Also when the tie wrap is cut it should be cut so close to the tie grip that none of the excess tie is exposed. If the excess tie wrap is exposed it is often very sharp and could cut someone.



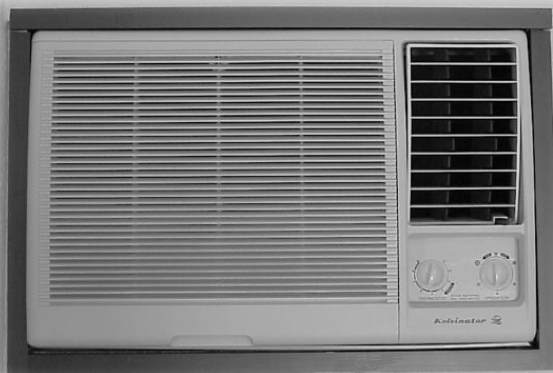
**Figure 9.14**  
*Cable tie connected correctly*

## 9.5 Cooling, heating and air conditioning

Excess heat is the most common cause of physical damage. Although technically possible, it is rare that equipment is damaged due to excessive cold. Electronic equipment generates heat and when combined with the ambient heat in an enclosed box, the equipment can become overheated. An enclosed box with electronic equipment will get too hot no matter where it is located. Unventilated equipment in the Arctic can fail. It is then often necessary to cool the equipment. This can be done with fans, air conditioning or evaporative cooling. Fans are often used when equipment is located in air-conditioned buildings or if it is not possible to air-condition the equipment enclosure.

To keep the equipment cool the following rules should be followed

- Equipment should not be placed in direct sunlight
- The temperature should be monitored if possible
- All equipment should be ventilated
- Some type of cooling should always be used
- Filters should be checked regularly



**Figure 9.15**  
*Good air-conditioning*



## 9.6 Wire management in a cable run

One problem that is often overlooked in the design and installation of equipment is the installation of equipment so that it can be repaired later. Often the designer or installer is not the person that has to repair the equipment. It is important to install the equipment so that the repairperson can easily gain access and replace the device if necessary.

A few common problems that can happen in the installation of equipment are:

- Too many wires in a cable run
- Wires installed over the top of connectors
- Spliced cables in the cable run
- Equipment installed on top or too close to other equipment
- Equipment installed too close to the floor or ceiling
- Cable markers not used or loose
- Lack of incorrect documentation

Too many cables in the cable run often happens as wires are added a little bit over time. As more and more cables are installed the cable run gets fuller and fuller until it is impossible to get to the bottom cables. Lots of small cable runs are usually better than one large cable run.

As equipment is added to an installation it is often easy to install wires over existing equipment or connectors. Sometimes the wires can be moved to get at the equipment underneath, but often the repairperson must remove the equipment on top first. This can cause even more problems for the repairperson.

In my opinion installers should ***NEVER*** splice wires. If the wire is not long enough during the installation, the wire should be pulled out and replaced with one that is long enough. Since about 60% of problems with equipment are due to bad connections, it is important to limit the number of connections. A splice is just another possible bad connection. Often it is hard to find these splices when they go bad as they can be hiding in the cable run.

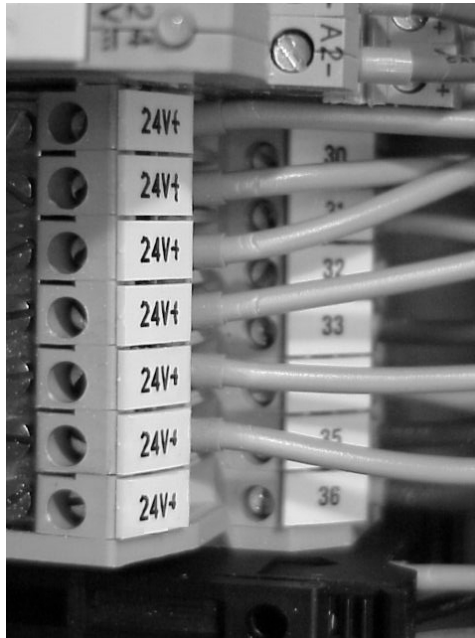
Equipment should be spaced out when mounted in a box. If the devices are located too close it is sometimes impossible to connect or disconnect wires or equipment. If equipment is mounted in 19-inch racks, the racks should be located where it is easy to get to the front and back of the equipment. Not only is equipment that is badly located often hard to repair but it can be dangerous to the technician or engineer.

If equipment is installed too close to the floor or ceiling it can be very difficult for the repairperson to gain access to the equipment. This could be an annoyance for the repairperson but also could be dangerous. Typically equipment is located between eye and knee level. The equipment should also be accessible from the back if possible.

The proper cable markers for the job should be installed correctly on both ends of the cables. These numbers should be noted in the documentation. The cable markers should be located where the repairperson can see them, but not where they can be knocked off or affected by heat or other environmental situations.

It is becoming less frequent that an installation is undocumented. The installer is usually required by the contract to document the complete installation. The problem with most documentation is that it is written from the installer's point of view instead of from the technician's point of view. The installer should document the installation as though he or she was going to repair the system. Colors of wires used, where they go and how they get there is very important, but any changes or unusual installation practices should be

documented or noted by the installer. This is especially true if the installer knows that the repairperson may have a problem later because of the installation.



**Figure 9.16**  
*Cable markers*

## 9.7 Conduit installation

As mentioned before plastic cable runs are becoming more popular than steel cable trays and the same thing is happening in conduit. Because the installation is easier and it costs less, the electrical industry is moving to plastic continuous tubing or conduit. When the distance is long, it can be difficult to install the wires through the conduit. It is often easier to insert the cables into the conduit before the conduit has been installed, but it is possible to insert the wires afterwards. Getting the wires through the conduit can be accomplished by first connecting a small plastic grocery type bag to the wires. A vacuum cleaner is then connected to the other end of the conduit and turned on. The vacuum created in the conduit draws the plastic bag with the wires attached through the conduit. It may be necessary to insert liquid soap into the conduit to facilitate the movement of the plastic bag and wires. It is best to use smooth walled plastic conduit.



**Figure 9.17**  
*Conduit*

Once the wires have been pulled through the conduit, it can be mounted or buried. If the conduit is being mounted, it should be placed out of direct sunlight if possible. If the conduit is placed out of doors, special UV resistant conduit should be used. If the conduit is mounted on equipment or walls it should be placed where it will not interfere with persons or other equipment. Plastic conduit should never be installed where it can be trod on by anyone. Conduit should be placed under a path rather than over, if it is to cross a walkway. If the path is a concrete pad on the ground, a water hose can be used to blow a hole under the concrete and then the conduit pushed under the concrete.

## 9.8 Troubleshooting techniques

In some ways it is difficult to define how to troubleshoot a piece of equipment. This is because every device is different. But having said that, there is a common method that most experienced troubleshooters use. The first thing the troubleshooter does is find out if the equipment has ever worked. There is a big difference between a device that suddenly fails and one that has never worked. If a device just stops working then there is usually only one problem. If it has never worked there may be tens or even hundreds of problems.

If the device has simply stopped the troubleshooter might check (in order) the following possible problems:

- Is there power to the equipment?
- If there is power, is there any smoke? (don't let the smoke out)
- Are the proper lights on, if any? If not, why not?
- Turn the power off and check the connections. Are any loose? Fix them.
- Turn the power back on.
- Now that the power is back on is it working correctly? No, then go on.
- Is anything unusually hot?

- If not, this is the time to get out the schematics and manuals.
- Separate and test different sections and sub systems.
- Starting in the middle of the device, divide the device in half functionally while testing each sub-system until the problem is found.
- Fix the problem. Does it work? If not start back with number 1.

I have seen an experienced troubleshooter spend hours trying to find a problem and then realize that the equipment is unplugged. Don't skip any steps.

## 9.9 Safety considerations

Safety should always come first on all installations. It is better that the job is not done than someone gets hurt or injured. If you think it is unsafe to do a job then refuse to do it. Most accidents happen because people do things they know are wrong and they get complacent. Doing something the unsafe way a hundred times may not cause an accident, but sometimes doing something wrong once causes an accident. It is up to the safety officer to remind people of potential accident situations, but it is up to everyone to prevent accidents.

Every job should have a safety officer. It is his or her job to help the employees on the job to reduce accidents.

This can be accomplished by doing the following:

- Follow company safety requirements
- Identify any unsafe situations or practices
- Have a first aid kit on hand
- Educate and remind employees of unsafe situations or practices
- Verify that you have the proper safety equipment
- Think safety!



**Figure 9.18**  
*Safety first*

Obviously the most common safety issue in the electronics field is electricity. A very small current (50 mA) can kill. It is very important to make sure that the power is off before working on equipment. Although this sounds simple, it is surprising how many people die every year from being shocked. The main problem is that electricity is invisible most of the time. It is a good idea to carry a voltage stick that indicates the presence of high voltage. Place the voltage stick next to the wires or equipment before touching it. The voltage stick is a non-contact high voltage-sensing device available in most electronic stores. When high voltage is sensed the voltage stick lights up. It is a very cheap and easy device to use and it could save your life.

## 9.10 Conclusion

Installation of electronic equipment is changing every day due to the huge amount of innovations happening in the electronics industry. The good thing is that some things change very little if at all. In the past steel conduits and cable runs were used because of safety reasons, but now they are being used because of noise problems. Screw, crimp and solder connections have been around for a long time. They are still being used in the same way and little has changed. Screw connectors are increasing in popularity and therefore it is necessary to understand the correct way to install them. We are soldering connections less and less, but there are still times when it is required.

Bad connections cause a high percentage of problems with electronic equipment. This is usually because of poor installation practices or external forces. Some of the things that can cause failures are, vibration, extreme hot and cold, insufficient tightening of screw connectors and incorrect installation techniques. By following proper and good maintenance procedures, failures can be greatly reduced. Scheduled visual and mechanical checks should be done often. Besides mechanical failures, another installation problem can be noise coupled to the wires through poor installation of cable runs or trays.

Enclosed steel cable runs are the best from a noise point of view. Plastic or open cable trays give no protection against externally radiated noise. The chance of noise coupling is higher if wires are laid in parallel for long distances. Tie wrapping cables together adds to the possibility of noise coupling. Also, if the tie wraps are too tight they can damage the wire. Torque tie wrap guns are used to fasten the tie wrap and therefore reduce potential damage.

Good installation practices will always be subjective and often if the installation looks good then it is assumed that it is a good installation. This is not necessarily true. Parallel wires look good, but can cause so much noise coupling that the system may not work. A good looking system that doesn't work isn't good for anything. Functionality is always second after safety in any installation. But having said that, it is still possible to have a neat, safe and tidy system that works correctly. Clean installations are usually safer and more reliable than messy ones.

## End notes

### 10.1 Conclusion

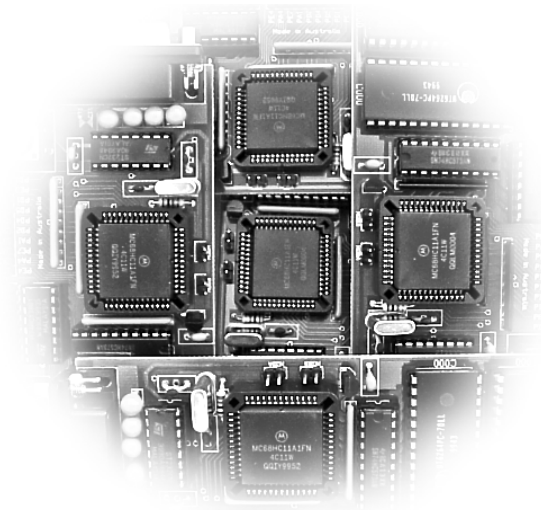
The world of microcontrollers has long been shrouded in mystery because of the public's perception of complicated computers. It was not long ago that most people thought you had to be a genius to even operate a computer. The personal computer and the Internet have put an end to that. Now we see grandparents E-mailing their grandchildren and children using computers every day in school. The mystery has passed and we are better for it.

A similar thing has been happening within the electronics industry. The use of microcontrollers has been very specialized, but now we see a huge expansion of the use of microcontrollers in the form of PLCs, DCSs and Data Loggers. These microcontrollers are extremely easy to use and are becoming more dependable every day. By understanding the inter-workings of microcontrollers we have a better feel for how and what these controllers do. When things go wrong we then have a better awareness of the methods for troubleshooting and repairing controllers. As mentioned in the preceding chapters microcontrollers can stop running for seemingly no apparent reason. It was shown that high voltage or static could cause a poorly designed microcontroller to freeze up. We also saw how this problem can be minimized by watchdog timers and filling the unused programming area of a microcontroller with NOPs and JSRs instructions. It is hoped that this book and the course that goes with it has de-mystified the world of microcontrollers for the reader.

### 10.2 CPU design and functions

This book was never written as a microcontroller design manual. The design of microcontroller systems though can be understood and even done on a simple level by almost anyone. We saw how the central core of the microcontroller was the CPU. The CPU implements the instructions contained within the program. The program lives in a variety of places including the RAM, EPROM and EEPROM. These memory devices not only hold the program but also the temporary data, input results and output information.

The program carries out arithmetic and logic functions using the input and output data that is held in its database. This data may have to be transferred to another microcontroller. This transfer is done on some type of communication system. It was seen that some microcontrollers like the 68HC11 have synchronous and asynchronous communication ports on board. For parallel communications the microcontroller uses one of the parallel ports supplied on the microcontroller. We found that these ports are either unidirectional or bi-directional. And that they can be controlled by registers in the microcontroller. The program places the values that are needed to control the port into the relevant control register. The ports can be configured in this way to be either an input port or output port.



**Figure 10.1**  
*CPUs*

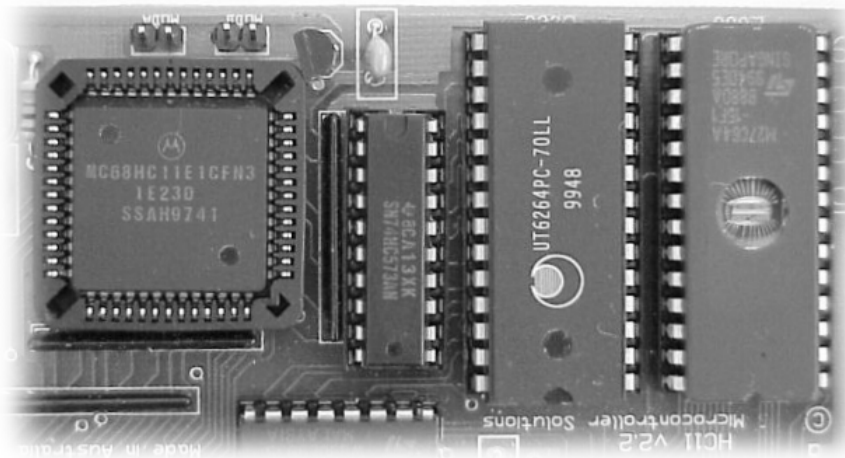
## 10.3 Assembly language programming

It is not necessary to understand assembly language programming (thank God) to use a microcontroller. But having a little bit of information on the way that microcontrollers are programmed can help us to understand the inter-workings of devices like PLCs and DSCs. The programmer uses instructions to tell the microcontroller what to do. A group of these instructions becomes a subroutine. A program is a group of linked subroutines written to accomplish the tasks defined in the specifications. It is best if the programmer creates a flow chart before starting the coding. The coding is the act of writing the code or instructions in the subroutines and the best subroutines are those that are completely stand-alone. That is, they are not dependent on the values taken from another subroutine. In a well-written program the subroutines can be moved around within the program with no effect to the function of the program.

Most programmers use BASIC or C++ to program microcontrollers. These high level languages are used because they are convertible and easier to program than assembly language. C++ also has the benefit of making low-level changes while at the same time being a high level language. As memory becomes cheaper and smaller, the use of C++ and other high level languages will increase. The problem with using high level languages is that they use a lot of memory.

## 10.4 Memory

RAM, EPROM and EEPROMs are used within the microcontroller system to hold data. The amount of memory included in microcontrollers is minimal and often less than is needed by most projects. Often designers have to add external chips to increase the memory so that it is large enough for the project. Battery backed RAM or EEPROMs are the most popular. The biggest difference between these two types of memory devices is the way they are programmed. Although the end result of using either RAM or EEPROMs in the microcontroller system is the same, the battery backed RAM is more susceptible to being spiked than the EEPROM. Therefore, the EEPROM is safer to use in potentially static or high voltage areas. The battery in the RAM will eventually drain and the RAM chip will need to be replaced. The battery is large and it takes up more room on the PCB than a normal RAM or EEPROM. The EEPROM is harder to program than the RAM and only has about 10 000 write cycles before it becomes unstable. Often designers put both of these chips on the PCB.



**Figure 10.2**  
*CPU, BUFFER, RAM and EPROM*

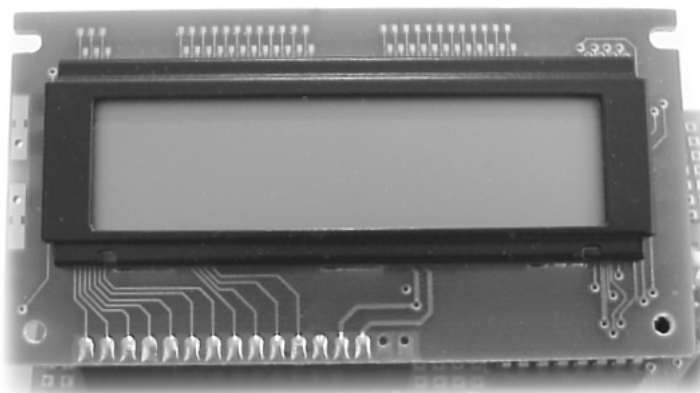
## 10.5 Inputs and outputs

A microcontroller that could not talk to the outside world would not be of much use. When defining or troubleshooting microcontroller I/O systems it is very important to understand the benefits and drawbacks of single ended and differential systems. Both analog and digital circuits are affected differently when they are connected single ended or differentially. The designer should understand that building an analog single ended circuit might be cheaper in the short run, but more expensive when the system doesn't work. Whereas differential analog circuits may be more expensive in the beginning, but will have a better chance of working than a single ended system. Digital inputs are different from analog circuits. Most digital inputs are connected in a single ended method. But switched digital inputs have their own requirements such as switch de-bounce.

There are three ways to control electronic devices, digital output, analog output and high speed switching. Either relays or transistor are used to turn devices on and off. The microcontroller puts a one or zero in a register that tells the relay to activate the device.



This type of control is very popular because it is simple and easy. But often the engineer needs to control an analog device. The most common way to control an analog device today is with digital switching. Digital switching control is used because it is more precise and gives the programmer extreme control. Two devices that are used to read and write to a microcontroller are the LCD display and a keypad. The keypad is scanned by first placing zeros one at a time on the rows of the keypad. The columns are then read and the result translated into ASCII characters. When the programmer wants to output data that can be read by the user, ASCII characters are placed in the LCD register. The contents of the register are then displayed on the LCD. Most LCD displays have microcontrollers on board and can be considered smart devices themselves.

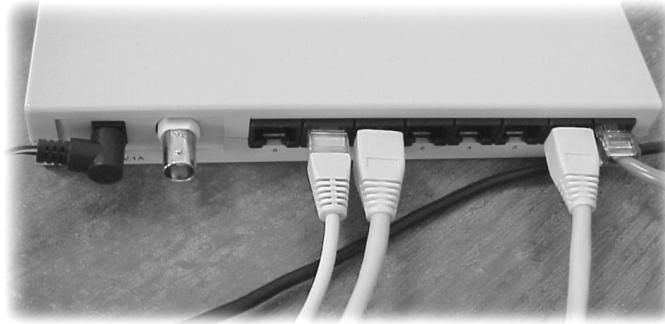


**Figure 10.3**  
*LCD display*

## 10.6 Data communication

Stand-alone devices have limited usefulness. The real power in a system is in connecting the devices together in a network of some type. When devices are connected together they can share resources and be accessed from one location. Often in troubleshooting a system the most cost and time is involved in traveling to the location. With networked systems the technician or engineer can access remote devices with radio, wires or even with a mobile phone. The Internet will allow anyone, with permission, to control and monitor a device from anywhere in the world. Since data communications has become such an important part of microcontroller design and troubleshooting, the need for systems like Ethernet and RS-485 have increased.

As long as we are using serial data transmission as the main method of long distance communication languages like ASCII and hex will be used to allow easy interpretation of the data. Voltage standards like RS-232, -422 and -485 will continue until replaced with more complicated systems such as Ethernet, Firewire and USB. The protocols that are in use today are often fieldbus protocols. These protocols define the rules on how the devices talk to each other. Overall protocol methods like master/slave, CSMA/CD and Tokenbus are the most popular methods of controlling the conversation. These methods have been incorporated into fieldbus systems like Modbus, Profibus and Foundation Fieldbus. With the increasingly popular communication medium of fiber optics, there will be huge changes in the way we do data communications in the not too distant future.

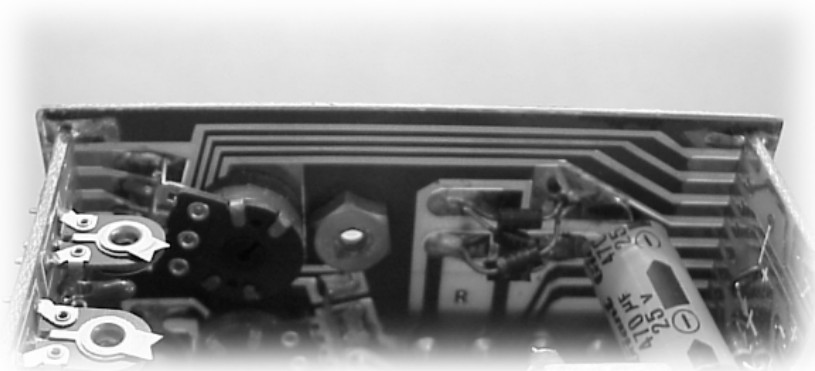


**Figure 10.4**  
*Ethernet*

## 10.7 Noise reduction

Noise reduction is very important from both a design and a troubleshooting point of view. It is the designer's job to design systems that not only are resistant to noise, but also radiate as little noise as possible. The troubleshooter needs to understand both the cause of noise problems and the best way of reducing noise in their systems. Noise is usually coupled into a system through the wiring. This makes it important for both the designer and troubleshooter to give a lot of attention to things like wiring methods and location. The location of the wires can have a great impact on how much noise is coupled from one system to another. When wires from one system lay in parallel with wires from another system, they can very easily couple noise into each other.

Noise reduction is a very important factor in PCB design. The main weapon that we have in the reduction of noise, both created and induced, into a PCB is the ground plane. The ways power and common grounds are laid out on a PCB have changed in the last few decades. Countries around the world have enacted legislation to require manufacturers to follow guidelines to reduce radiated noise from PCBs. Often these same rules not only reduce radiated noise, but also reduce the amount of noise being induced into the digital or analog circuit. Segregation of sub-systems on the PCB and Faraday boxes also goes a long way to reducing emitted and induced noise. Analog systems should be separated from digital circuits and a Faraday box should surround high noise producing systems.



**Figure 10.5**  
*Noise reduction on a PCB*

## 10.8 Grounding solutions

Earth grounding is used to reduce noise on PCBs and systems by supplying a better return path for potential noise. It is better from a noise position that high currents and voltage spike return to ground than be induced into our circuit. One problem with ground is that if high noise sources are physically close to our circuit, the noise can be coupled through the ground connection into our circuit. Therefore, every ground connection should be looked upon as a potential noise injection point and not a black hole for noise. The most common earth grounding system is the parallel single point ground, but there are modified versions. The digital and analog commons on the PCB are usually connected together at a single point and then connected to earth ground. A professional should always do this as often these commons have different voltages as referenced to earth ground.

Lightning protection costs the industry millions of dollars a year in damaged equipment. Therefore, reducing lightning damage has become high priority in the quest for better working systems. Lightning damage reduction usually takes the form of lightning rods and some type of earth grounding system. The most common of these is the trenched cable method. This is where an unshielded cable is buried in trenches around the building or antenna tower. The lightning rods are connected to the earth grounding system by way of aluminum tubing running up the side of the building or tower. Often a combination of spiked and trench cable earthing is done to increase the quality of the earth ground.



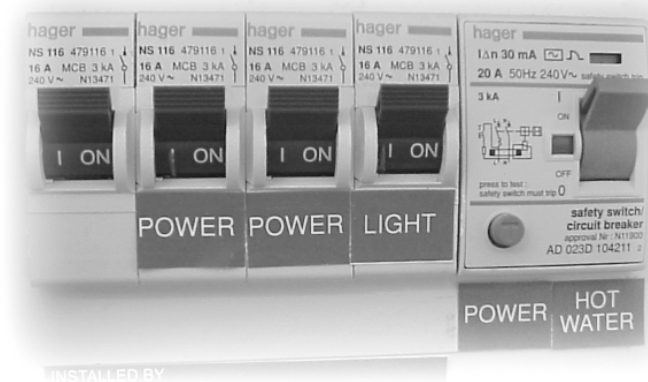
**Figure 10.6**  
*Ground*

## 10.9 Installation techniques

Installation of equipment can be looked upon from many different directions. The user, designer, installer and troubleshooter all look at the installation differently. The good designer views the installation from all angles and from everyone's point of view. The user wants controls that are easy to operate and reliable, the installer wants a good clean professional installation and the troubleshooter wants to be able to get to the equipment

and repair it if necessary. The most common problem in electronic equipment is the bad connection. Whether the connection is a screw, crimp or soldered connection, the connection can fail. A proper installation of these connections will reduce the possibility of failure later on in the life of the equipment.

Noise is another common problem that is encountered in electronic systems. Proper installation of cables in steel trays and ladders can reduce the coupling of noise from one wire to another. Often the problem with the system is not the installation but the type of materials that are specified for the job. Plastic cable trays and open ladder runs do nothing to limit noise being transmitted from one wire to another. Whereas enclosed steel cable trays will reduce substantially the noise coupled from one system to another through the wiring. The noise reduction of steel cable trays or conduits can be enhanced if the different types of cables are segregated according to their respective types of signals. Digital inputs and especially outputs should be kept away from analog signals. And mains power cables should be kept away from everything.



**Figure 10.7**  
*Mains power*

## 10.10 Final words

Understanding embedded controllers can help anyone that comes into contact with the myriad of microcomputer-controlled equipment available today, do their job better. One problem that happens in any industry is that people often become experts in their field, but have limited information about other fields. Since all systems are dependent on multiple disciplines, it is important for everyone involved in the system to have a bit of understanding of the other fields involved. If nothing else is gleaned from this book, it is hoped that the reader has received a different point of view of embedded microcontroller systems. It is also hoped that this book has taken some of the mystery out of embedded controllers.

---

# PRACTICAL 1

---

## Setting up the 68HC11 emulator board

### Introduction

The evaluation board that is available with IDC Technologies and on which this practical is based is specially designed for use as a development unit for university students. It is a very powerful tool for learning how embedded controllers work and are programmed. It is a very functional device and has most of the bells and whistles that are found on more expensive systems. The unit has the ability to receive a program from a terminal package and then run that program from RAM on the EVM board. An LCD display is attached to let the user see data sent from the program. It also has a keypad that lets the user input data into the program. This EVM unit is not a toy, but a completely useful development board.

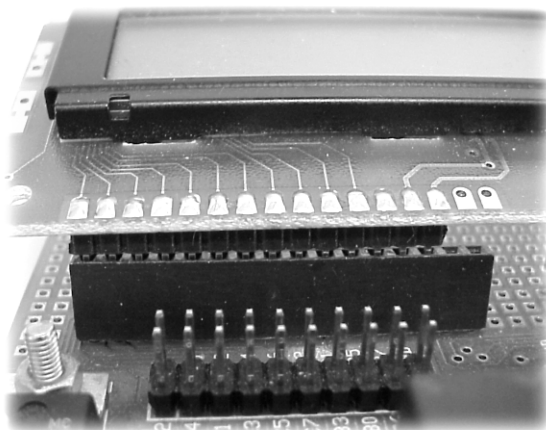
### Safety first

When working with this EVM board it is important that the user follow some simple safety rules:

- **DO NOT PLUG IN THE POWER** to the unit until you have checked that the installation is correct.
- **USE THE ANTI-STATIC WRIST STRAP** provided.

### Setup

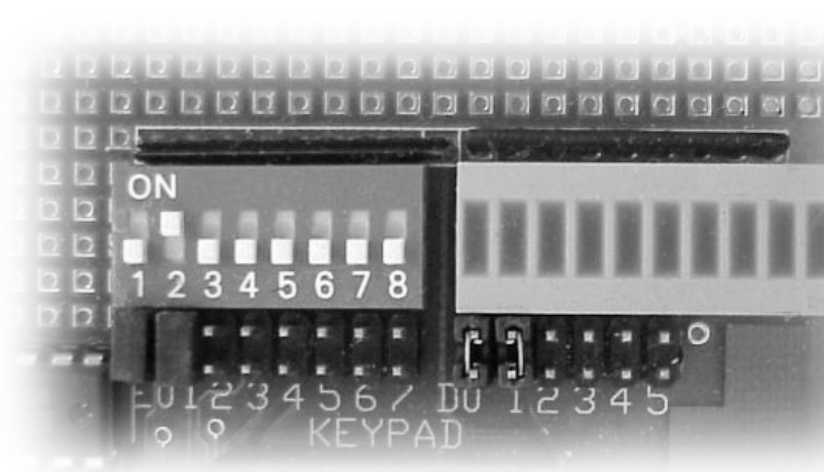
- Attach the anti-static strap to your wrist
- Remove the EVM from its case or box
- Install the LCD display in the connector as shown in Figure A1
- Do not install the keypad yet



**Figure A1**  
*LCD installation (Note; Pins are on the left)*

### **Switch and jumper setup**

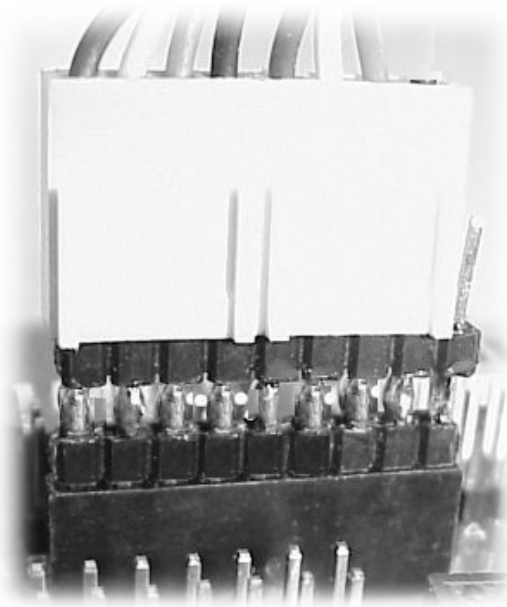
- Verify that the switches are setup as shown in Figure A2
- Verify that the jumpers are set as shown in Figure A2
- Switch 2 should be on, all others should be off
- Jumpers should be on switches 1, 2, 5 and LEDs 1, 2, 5
- Ignore the numbers on the PCB under the jumpers
- Count the switches and jumpers from the left



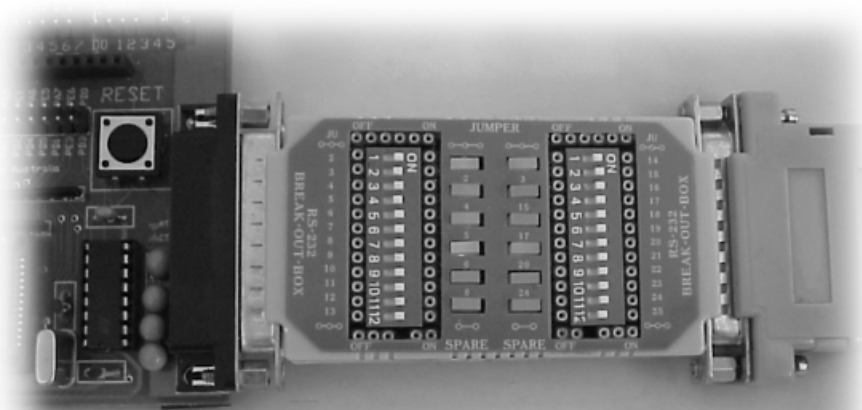
**Figure A2**  
*Switch and jumper settings*

### **Installing the RS-232 cable**

- Install the RS-232 into the computer
- Install the RS-232 into the breakout box
- Install the breakout box into the EVM board as shown in Figure A4



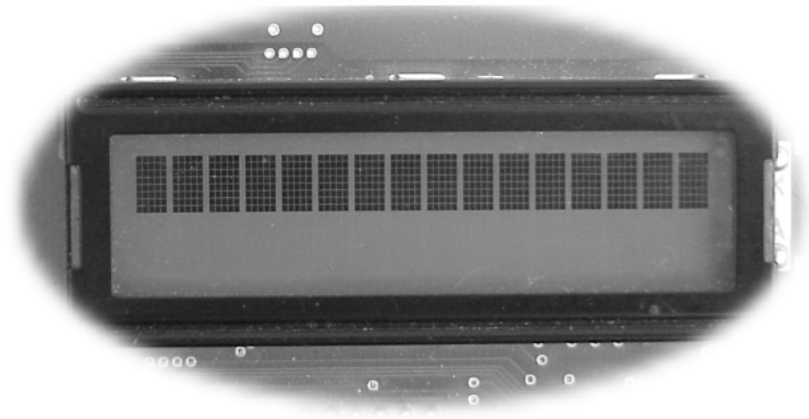
**Figure A3**  
*Keypad installation*



**Figure A4**  
*Breakout box (yours may look different)*

## Power up

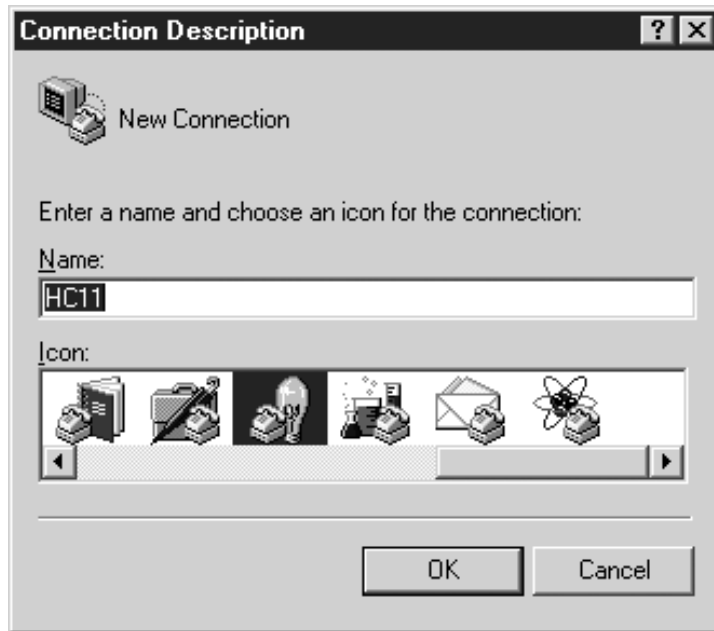
- Plug in the power supply into the EVM board but not the mains supply
- Plug in the power supply into the mains
- Does one red LED light and the LCD display show a row of black squares as shown in Figure A5?



**Figure A5**  
*LCD display after power on*

### Loading the software

Start the computer and after it has booted up into Windows and run the program called Hyper-terminal from the icon on the screen or under accessories if there is no icon. When Hyper-terminal is booted up the screen should look like...



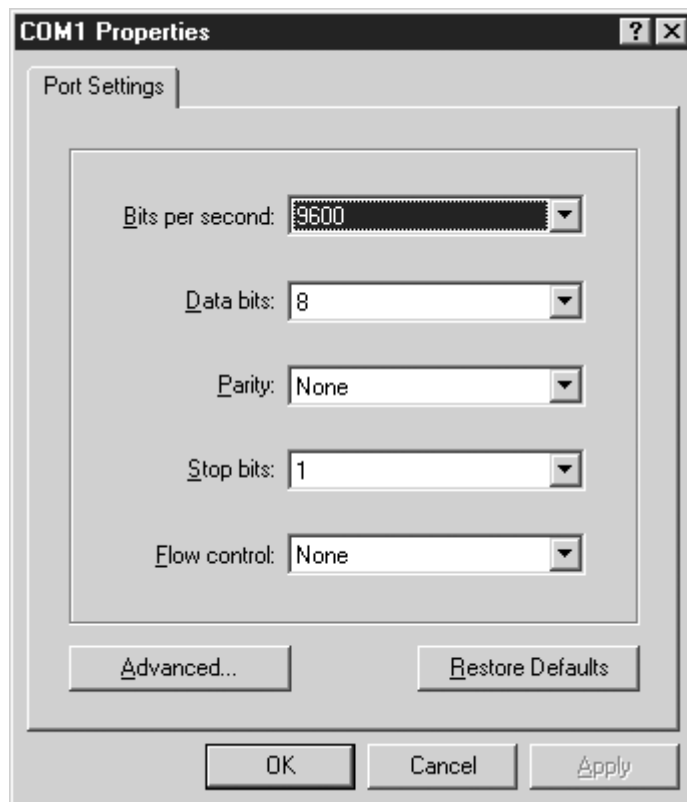
**Figure A6**  
*Hyper-terminal setup screen*

Add the HC11 and choose an icon and then click OK. Follow the setup screens as shown below as setup as shown.

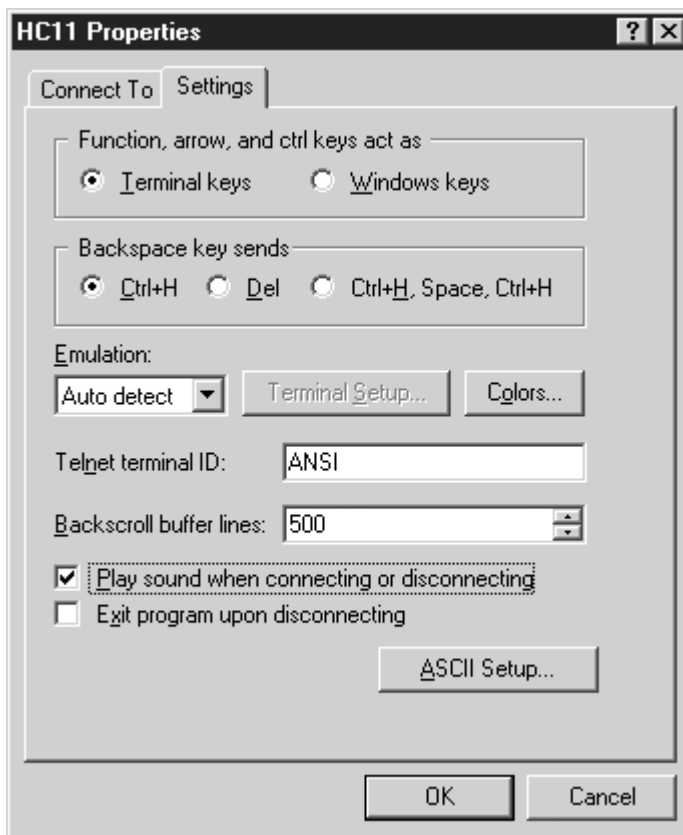




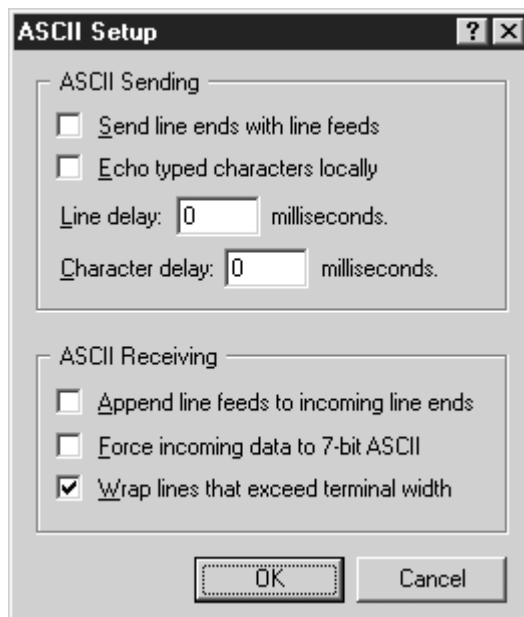
**Figure A7**  
*Set communication port to comm 1*



**Figure A8**  
*Comm port setup*



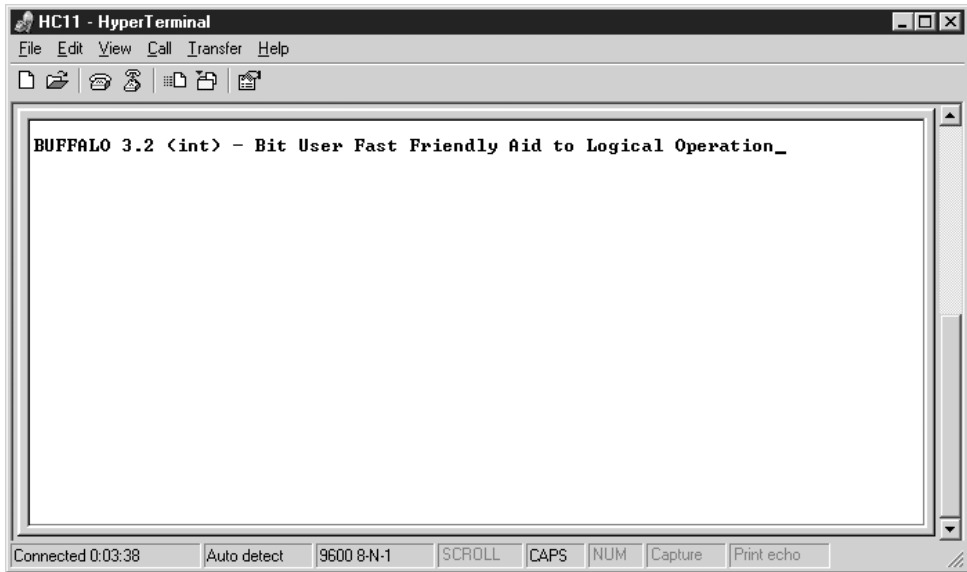
**Figure A9**  
*Terminal setup*



**Figure A10**  
*ASCII transfer setup*

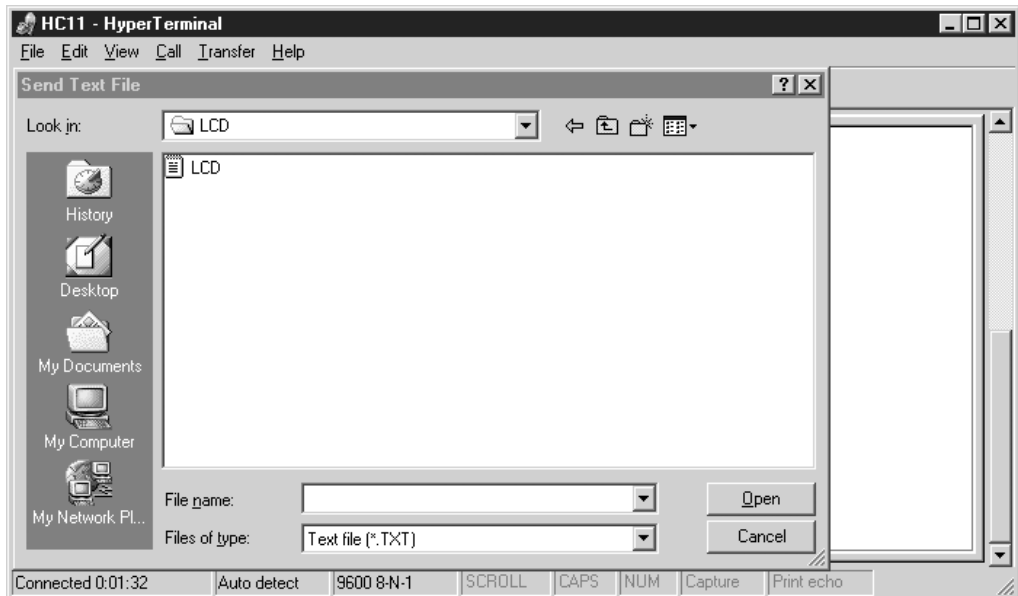
Sending a program to the EVM

Press the RESET button on the EVM and the following line should be displayed in Hyper-terminal.



**Figure A11**  
*Hyper-terminal front screen*

Type LOAD T and then press ENTER.  
Click on the Transfer menu and then choose Send Text File.



**Figure A12**  
*Select LCD file*

Double click the program LCD.TXT. When the program is done loading type G C000 and then press the ENTER key. The display should change to the IDC test program as seen in Figure A13.

Does the display show the IDC test sequence characters?



**Figure A13**

*Successful completion of the practical*

Congratulations! You have loaded and run your first embedded controller program.

---

# PRACTICAL 2

---

## Activating LEDs on the EVM

### Objective

This practical will introduce you to programming the 68HC11 microcontroller by showing how to turn on some LEDs in software. The practical is done in three steps. First you will enter the code as written below into the assembler program. Next or at the same time the comments will be added to the program. Finally you will load and run the program using the EVM board.

### Writing the program

Click on the assembler program icon. Create a new file by clicking on the new file icon on the top menu. Type in the program exactly as written below. Add the appropriate comments to the right of each line. Comments can be added as it is being written or after it is completed. The comments should be short but include sufficient information on what that line is trying to accomplish. Do not just repeat the instruction. Two examples of comments of an instruction could be...

```
LDAA    #$39           Load A with $39
```

This is not a very good comment. It just repeats the instruction. A better one would be.

```
LDAA    #$39           Load the ASCII character 9  
Or even better  
LDAA    #$39           Load character 9 ASCII in Accumulator A
```

Remember that the comments are there so that anyone can understand what the instruction is trying to do.

Capitalization does not matter, but the spacing of instructions is very important. The names for lines, such as START below, should always be placed on the far left-hand side of the page. Place one tab space (at least 8 spaces) between the line name and the

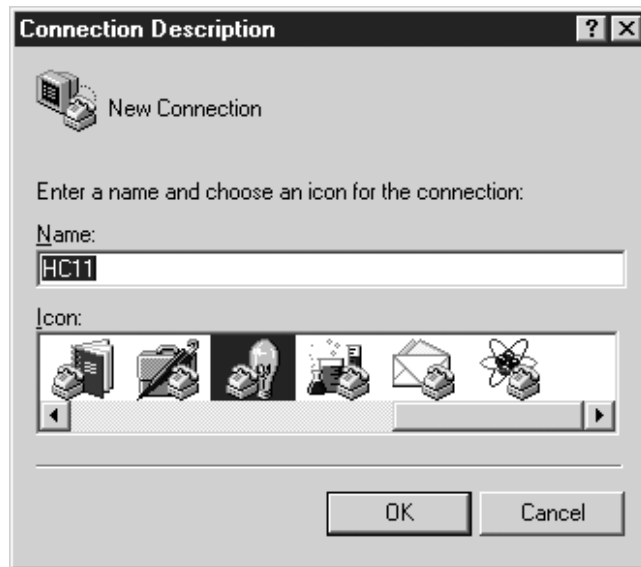
instruction. If there is no line name then put 2 tab spaces. There should be at least 1 tab space between the instruction and the operand of the instruction if any. There should be at least 2 tab spaces between the operand and the comment. Stars (shift 8) are used to indicate a comment line and to separate different parts of the program. The comment lines are not loaded into the microcontroller.

```
*****
*      WRITTEN BY - JOHN M. PARK
*      DESCRIPTION - IDC LED TEST
* DATE - 6/01
*****
* PROGRAM TO TURN ON SOME LEDS
* $1009 PORT D DATA DIRECTION REGISTER
* $1008 PORT D = LEDS
*****
          ORG  $C000
          LDS  #47
*****
          LDA  #$34
          STA  $1009
START    LDA  #$00
          STA  $1008
          JSR  $FFB8          BUFFALO Instruction! What does it do?
          JSR  DELAY
          LDA  #$34
          STA  $1008
          JSR  DELAY
          JSR  START
*****
* DELAY FOR LCD
*****
* DELAY LOOP – INDEX REGISTER X - 2 = 1 SECOND (approximate)
*****
DELAY    LDY  #$FFFF
          LDX  #$0002
LOOP     DEY
          CPY  #$0000
          BNE  LOOP
          DEX
          CPX  #$0000
          BNE  LOOP
          RTS
*****
```

## Loading the software

**NOTE:** if you have setup Hyper-terminal in the previous practical, then skip down to SENDING A PROGRAM TO THE EVM.

Start the computer and after it has booted up into windows and run the program called Hyper-terminal from the icon on the screen or under accessories if there is no icon. When Hyper-terminal is booted up the screen should look like...

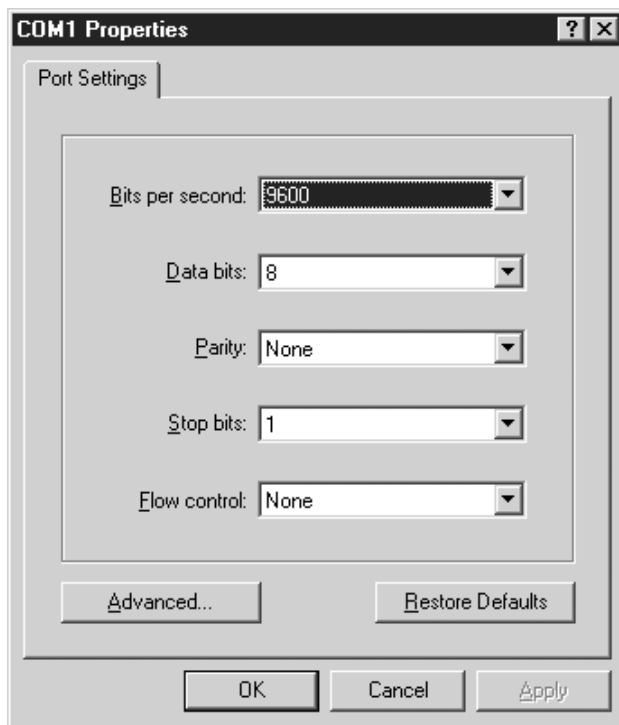


**Figure B1**  
*Hyper-terminal setup screen*

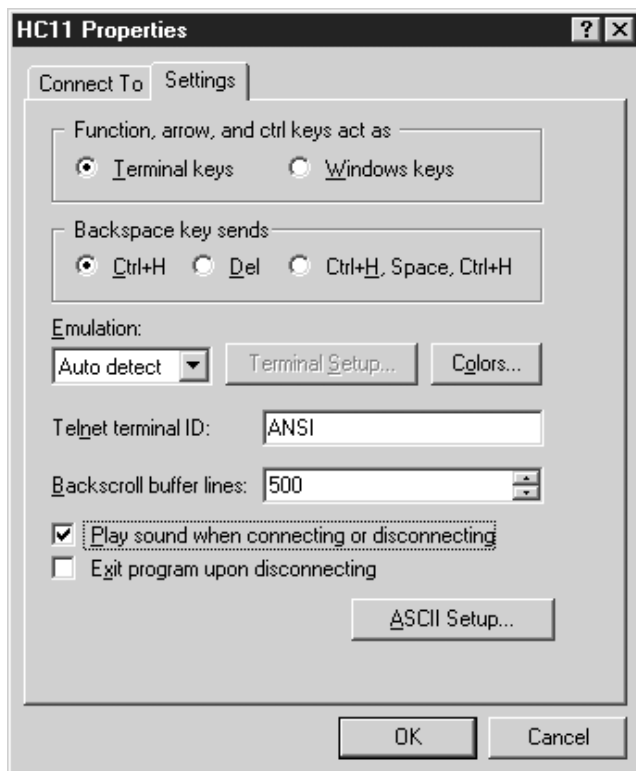
Add the HC11 and choose an icon and then click OK. Follow the setup screens as shown below as set up as shown.



**Figure B2**  
*Set communication port to comm 1*

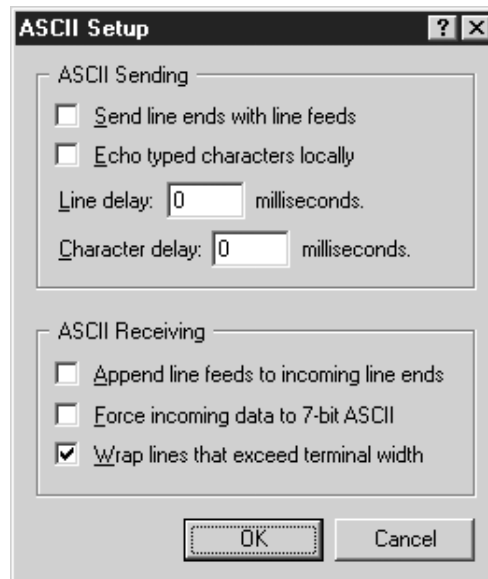


**Figure B3**  
*Comm port setup*



**Figure B4**  
*Terminal setup*

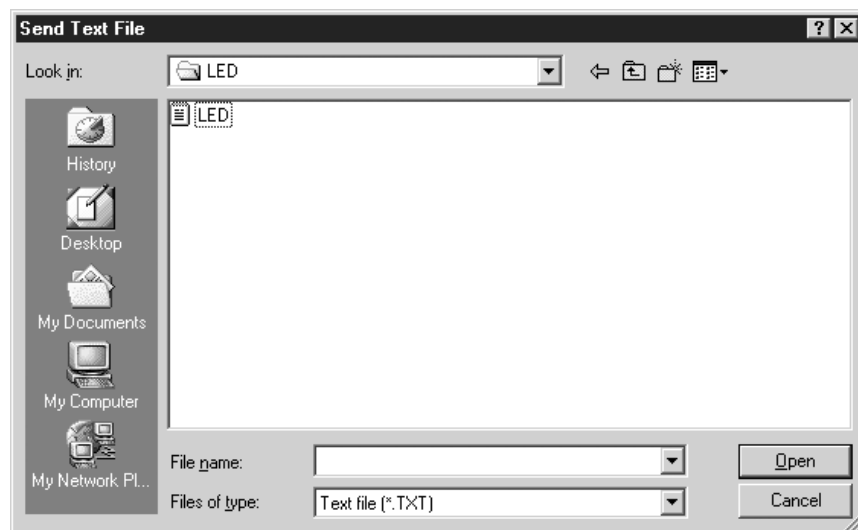




**Figure B5**  
*ASCII transfer setup*

## Sending a program to the EVM

The first step in sending a program that you have written to the EVM is to save the program by going to 'save as' under the file menu in the PF32 program. Save the program as LED.ASM. If there is a file by that name already in the directory, write over it.



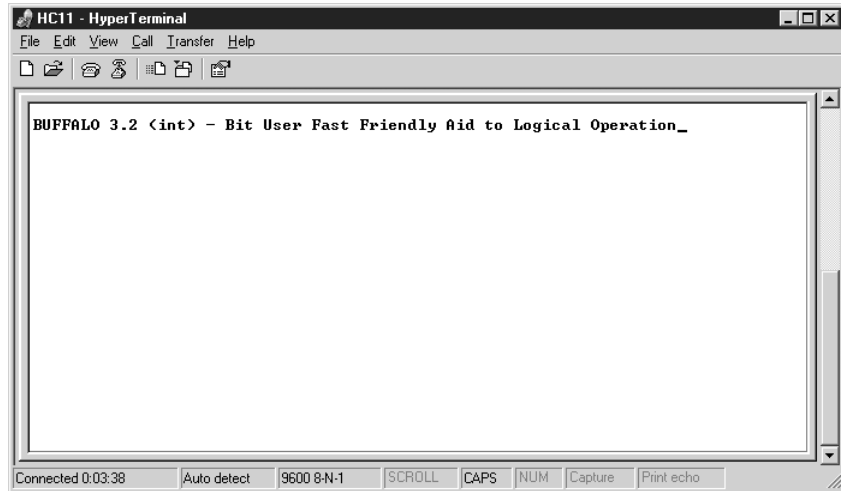
**Figure B6**  
*PF32 screen*

After the file has been saved click on the DOS button and type ASM11 LED.ASM. This will compile the program into different formats. One of the formats will be a .S19 file. This file has the code that you will send to the EVM. Unfortunately the .S19 format is not compatible with Hyper-terminal. While still in DOS change the extension of the .S19 file to .TXT. This is done with the following command.

REN LED.S19 LED.TXT

Using the ALT TAB change to the Hyper-terminal program.

Press the RESET button on the EVM and the following line should be displayed in Hyper-terminal.

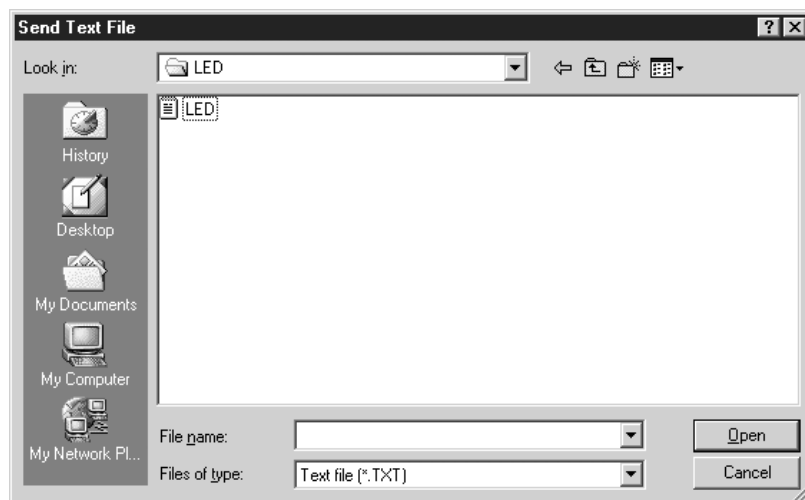


**Figure B7**  
*Hyper-terminal front screen*

Type LOAD T and then press ENTER.

Click on the Transfer menu and then choose Send Text File.

The screen should change to



**Figure B8**  
*LED.TXT*

Once the program is loaded, run the program by typing the following.

G C000

The LEDs on the LCD screen should now start flashing.

---

# PRACTICAL 3

---

## Reading switches on the EVM

### Objective

This practical will introduce you to programming the 68HC11 microcontroller by reading a switch and toggling a LED. The practical is done in three steps. First the delegate will enter the code as written below into the assembler program. Next or at the same time the comments will be added to the program. Finally you will load and run the program using the EVM board provided.

### Writing the program

Click on the assembler program icon. Create a new file by clicking on the new file icon on the top menu. Type in the program exactly as written below. Add the appropriate comments to the right of each line. Comments can be added as it is being written or after it is complete. The comments should be short but include the information on what that line is trying to accomplish. Do not just repeat the instruction in the comments. Two examples of comments of an instruction could be...

```
LDAA  #$39          Load A with $39
```

This is not a very good comment. It just repeats the instruction. A better one would be.

```
LDAA  #$39          Load the ASCII character 9  
Or  
LDAA  #$39          Load 9 ASCII in Accumulator A
```

Remember that the comments are there so that anyone can understand what the instruction is trying to do.

Capitalization does not matter, but the spacing of instructions is very important. The names for lines, such as START below, should always be placed on the far left-hand side of the page. Place one tab space (at least 8 spaces) between the line name and the

instruction. If there is no line name then put 2 tab spaces. There should be at least 1 tab space between the instruction and the operand of the instruction if any. There should be at least 2 tab spaces between the operand and the comment. Stars (shift 8) are used to indicate a comment line and to separate different parts of the program. The comment lines are not loaded into the microcontroller.

```

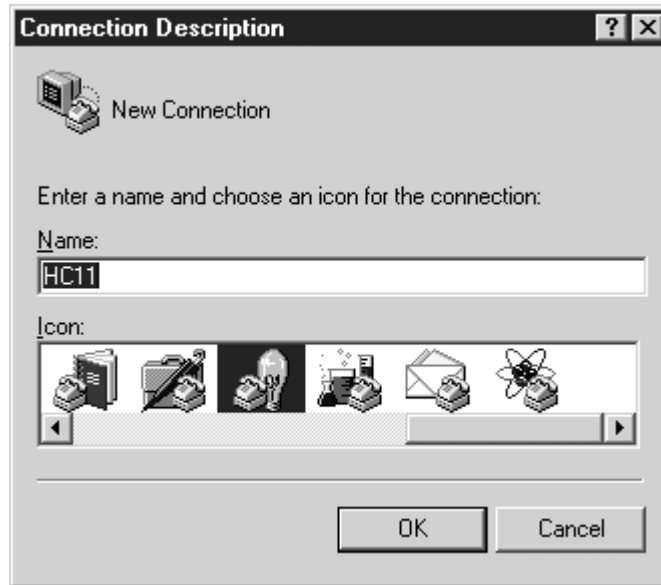
*****
*           WRITTEN BY - JOHN M. PARK
*
*           DESCRIPTION - IDC SWITCH TEST
*
*           DATE - 6/01
*****
*           PROGRAM TO TURN ON A LED USING A SWITCH ON THE EVM
*           NO DEBOUNCE
*****
           ORG           $C000
           LDS           $47
*****
           LDAA          #$10
           STAA          $1009
LOOP      LDAA          $100A
           STAA          $1008
           LDX           #$FFFF
LOOP1     DEX
           CMPX          #$0000
           BNE           LOOP1
           JSR           LOOP
           RTS
*****

```

### Loading the software

**NOTE:** If you have setup Hyper-terminal in the previous practicals, then skip down to SENDING A PROGRAM TO THE EVM.

Start the computer and after it has booted up into Windows and run the program called Hyper-terminal from the icon on the screen or under accessories if there is no icon. When Hyper-terminal is booted up the screen should look like...

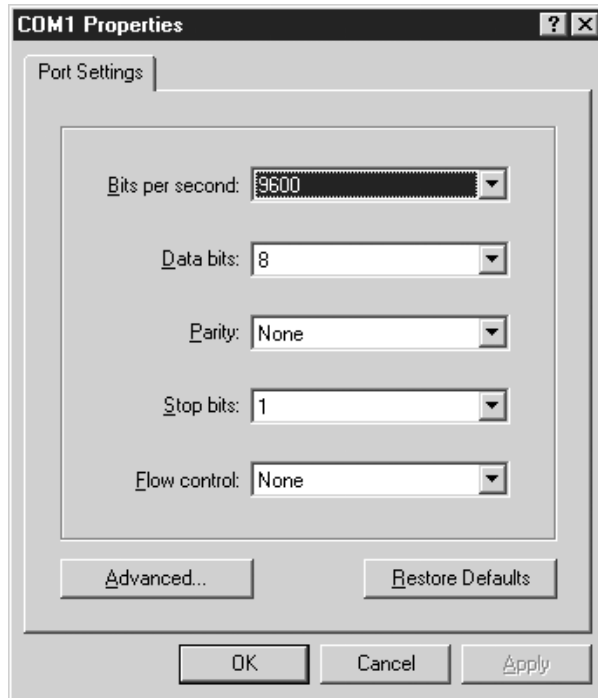


**Figure C1**  
*Hyper-terminal setup screen*

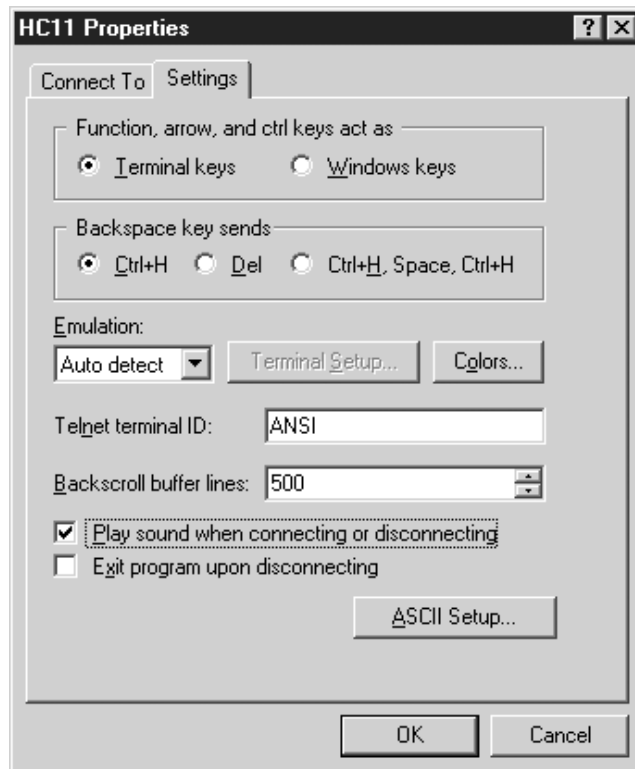
Add the HC11 and choose an icon and then click OK. Follow the setup screens as shown below as set up as shown.



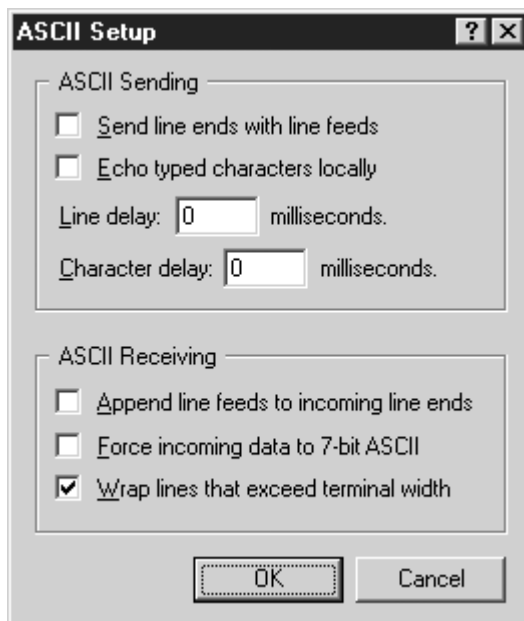
**Figure C2**  
*Set communication port to comm 1*



**Figure C3**  
*Comm port setup*



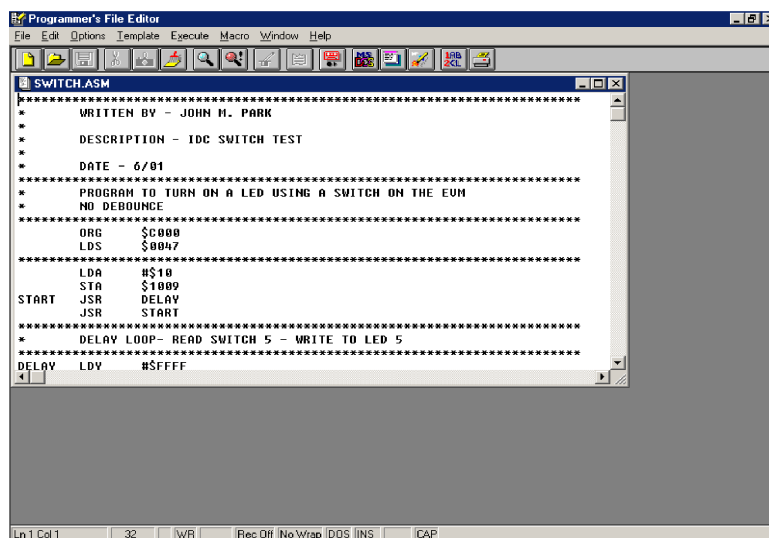
**Figure C4**  
*Terminal setup*



**Figure C5**  
ASCII transfer setup

## Sending a program to the EVM

The first step in sending a program that you have written to the EVM is to save the program by going to 'save as' under the file menu in the PF32 program. Save the program as SWITCH.ASM. If there is a file by that name already in the directory, write over it.



**Figure C6**  
PF32 screen

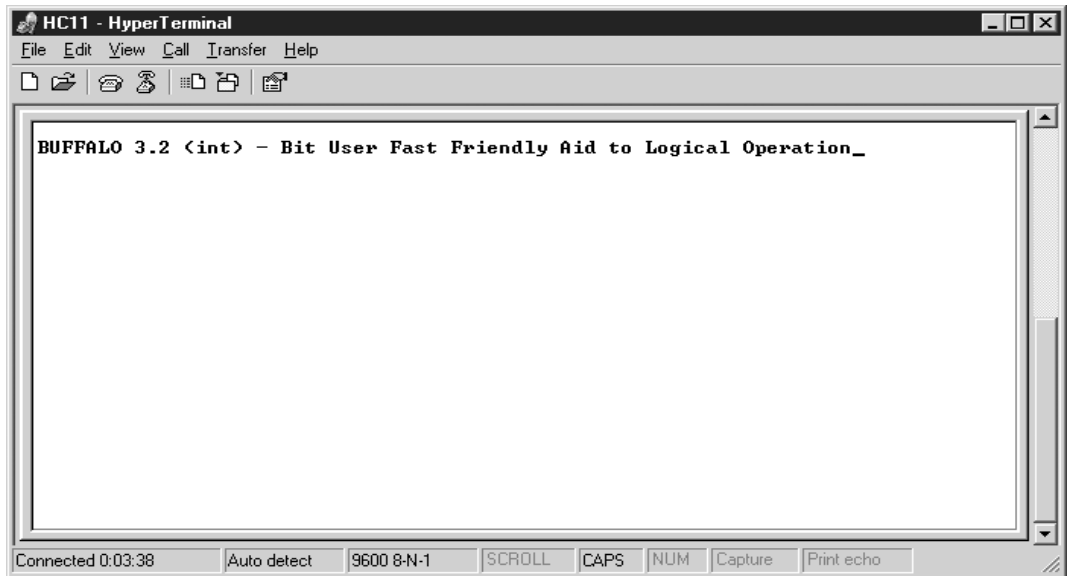
After the file has been saved click on the DOS button and type ASM11 SWITCH.ASM. This will compile the program into different formats. One of the formats will be a .S19 file. This file has the code that you will send to the EVM. Unfortunately the .S19 format

is not compatible with Hyper-terminal. While still in DOS change the extension of the .S19 file to .TXT. This is done with the following command.

```
REN SWITCH.S19 SWITCH.TXT
```

Using the ALT TAB change to the Hyper-terminal program.

Press the RESET button on the EVM and the following line should be displayed in hyper-terminal.



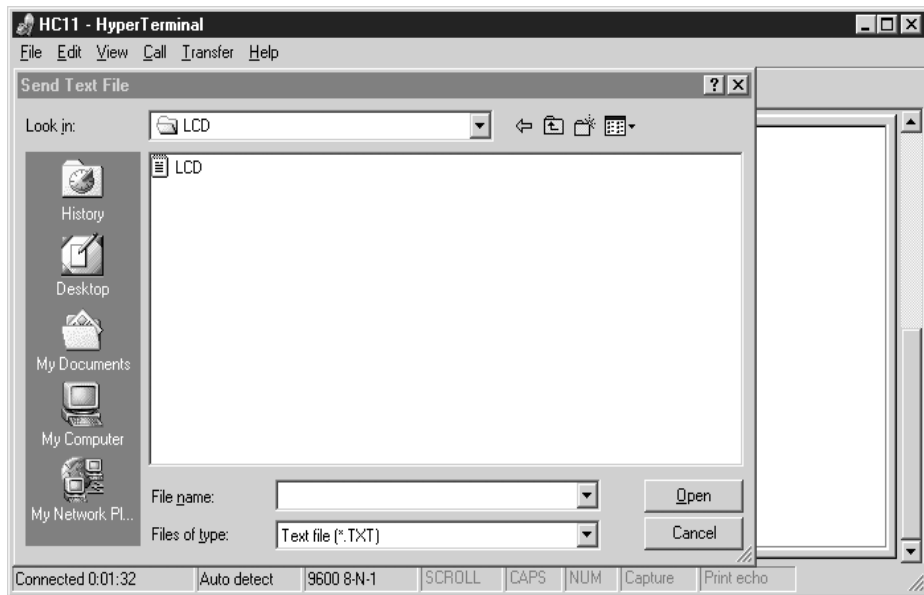
**Figure C7**  
*Hyper-terminal front screen*

Type LOAD T and then press ENTER.

Click on the Transfer menu and then choose Send Text File.

The screen should change to





**Figure C8**  
*SWITCH.TXT*

Once the program is loaded, run the program by typing the following.  
G C000

---

# PRACTICAL 4

---

## Sending characters to an LCD display

### Objective

This practical will introduce you to programming the 68HC11 microcontroller by sending information to the LCD display on the EVM. This practical is done in three steps. First you will enter the code, as written below, into the assembler program. Next or at the same time the comments will be added to the program. Finally you will load and run the program using the EVM board.

### Writing the program

Click on the assembler program icon. Create a new file by clicking on the new file icon on the top menu. Type in the program exactly as written below. Add the appropriate comments to the right of each line. Comments can be added as it is being written or after it is complete. The comments should be short but include the information on what that line is trying to accomplish. Do not just repeat the instruction in the comments. Two examples of comments of an instruction could be...

```
LDAA  #$39          Load A with $39
```

This is not a very good comment. It just repeats the instruction. A better one would be.

```
LDAA  #$39          Load the ASCII character 9
```

Or

```
LDAA  #$39          Load 9 ASCII in Accumulator A
```

Remember that the comments are there so that anyone can understand what the instruction is trying to accomplish.

Capitalization does not matter, but the spacing of instructions is very important. The names for lines, such as START below, should always be placed on the far left-hand side of the page. Place one tab space (at least 8 spaces) between the line name and the instruction. If there is no line name then put 2 tab spaces. There should be at least 1 tab space between the instruction and the operand of the instruction if any. There should be at least 2 tab spaces between the operand and the comment. Stars (shift 8) are used to indicate a comment line and to separate different parts of the program. The comment lines are not loaded into the microcontroller.

```

*****
*           WRITTEN BY - JOHN M. PARK
*
*           DESCRIPTION - IDC DISPLAY TEST
*
*           DATE - 6/01
*****
*           ORG   $C000
*           LDS   #50
*****
*           PROGRAM
*****
*           JSR   COMSET
RESTART    JSR   MENU
*           JSR   RESTART
*****
*           SET UP THE LCD AND CLEAR SCREEN
*           $8000 = LCD CONTROL REGISTER
*****
COMSET     JSR   DELAY           DELAY FOR LCD
*           LDB   #$38           LOAD FIRST CONTROL BYTE
*           STB   $8000          CONTROL REG
*           JSR   DELAY           DELAY FOR LCD
*           LDB   #$0C           LOAD SECOND CONTROL BYTE
*           STB   $8000          STORE IN LCD CONTROL REG
*           JSR   DELAY           DELAY FOR LCD
*           LDB   #$02           RESET SCREEN
*           STB   $8000          DO IT
*           RTS
*****
*           START OF MENU SYSTEM - PRINT START MESSAGE
*****
MENU       LDX   #MESSAGE1      'IDC TEST PROGRAM'
*           JSR   DISPLAY
*           LDX   #MESSAGE2
*           JSR   DISPLAY
*           LDX   #MESSAGE1
*           JSR   DISPLAY
*           LDX   #MESSAGE3
*           JSR   DISPLAY
*           LDX   #MESSAGE1
*           JSR   DISPLAY

```

```

LDX #MESSAGE4
JSR DISPLAY
LDX #MESSAGE1
JSR DISPLAY
LDX #MESSAGE5
JSR DISPLAY
LDX #MESSAGE1
JSR DISPLAY
LDX #MESSAGE6
JSR DISPLAY
LDX #MESSAGE7
JSR DISPLAY
LDX #MESSAGE7
JSR DISPLAY
RTS

```

```

*****
* DISPLAY ONE LINE OF MESSAGE DEFINED BY THE X INDEX REGISTER
* ROW IS DEFINED AS 40 (28 HEX) CHARACTERS
* $8001 = LCD SCREEN RAM
*****

```

```

DISPLAY      LDB  #$28
DISLOOP     JSR  LCDBUSY
            JSR  DELAY
            LDA  $00,X
            CMPA #$23
            BEQ  ENDD
            STA  $8001
            INX
            DECB
            BNE  DISLOOP
ENDD        RTS

```

```

*****
*      DELAY FOR LCD
*****

```

```

DELAY LDY  #$1000
LOOP   DEY
      BNE  LOOP
      RTS

```

```

*****
* CHECK BIT 7 TO SEE IF THE LCD IS BUSY
*****

```

```

LCDBUSY  LDAA $8000
         ANDA #$80
         CMPA #$80
         BEQ  LCDBUSY
         RTS

```

```

*****
*      MESSAGES MUST BE 16 CHARACTERS LONG
*****

```

```

MESSAGE1  FCC  'IDC TEST PROGRAM'

```

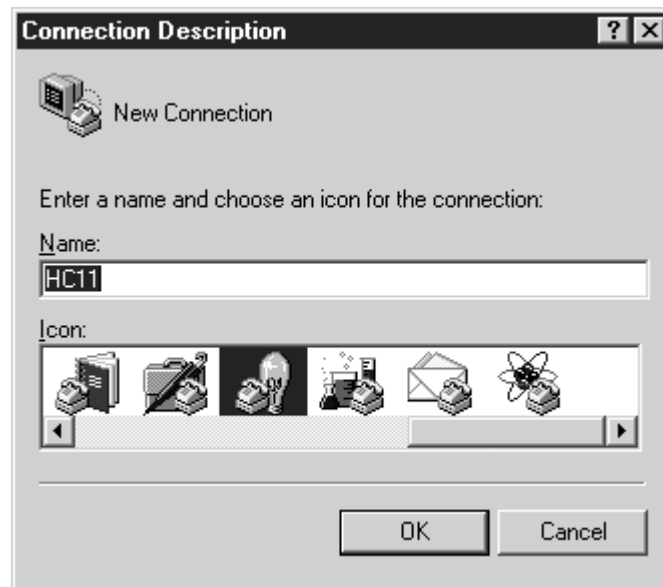
```

MESSAGE2   FCC   '!! GOOD WORK !!   '
MESSAGE3   FCC   '00000001 <-S/N   '
MESSAGE4   FCC   'IDC-0001 <-MODEL '
MESSAGE5   FCC   ' 1.0 < VERSION   '
MESSAGE6   FCC   '$$$ IT WORKS $$$ '
MESSAGE7   FCC   '
    
```

## Loading the software

**NOTE:** If you have setup hyper-terminal in the previous practicals, then skip down to SENDING A PROGRAM TO THE EVM.

Start the computer and after it has booted up into Windows and run the program called Hyper-terminal from the icon on the screen or under accessories if there is no icon. When Hyper-terminal is booted up the screen should look like...

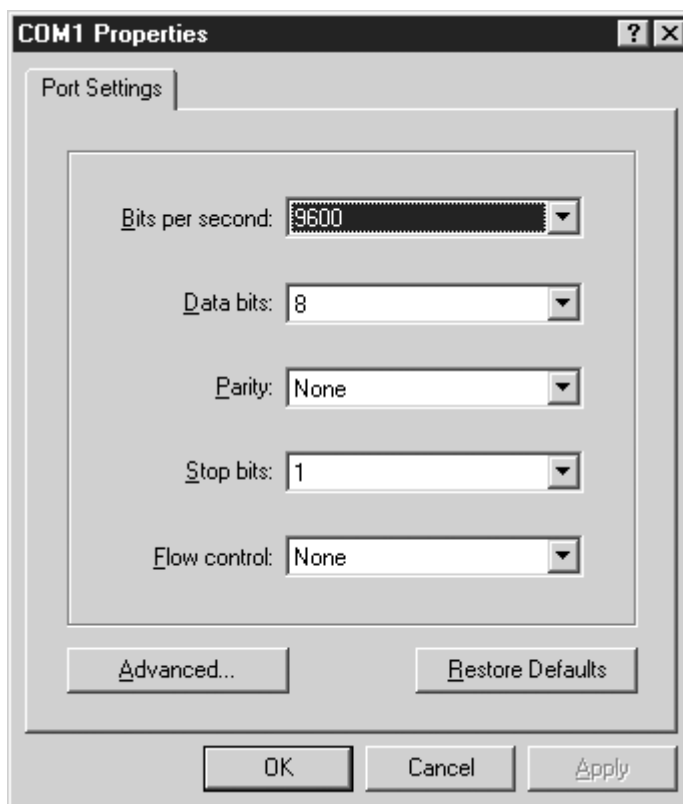


**Figure D1**  
*Hyper-terminal setup screen*

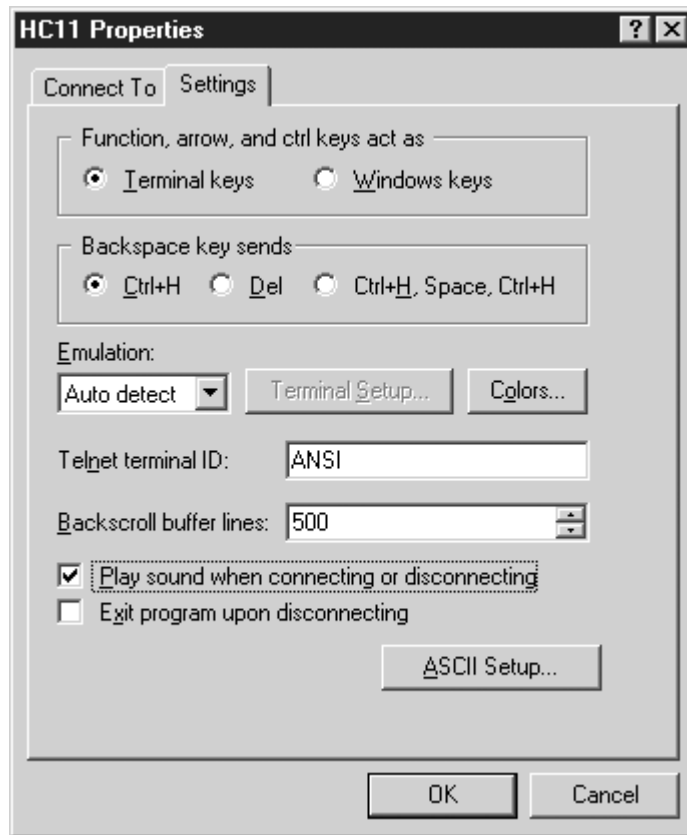
Add the HC11 and choose an icon and then click OK. Follow the setup screens as shown below as set up as shown.



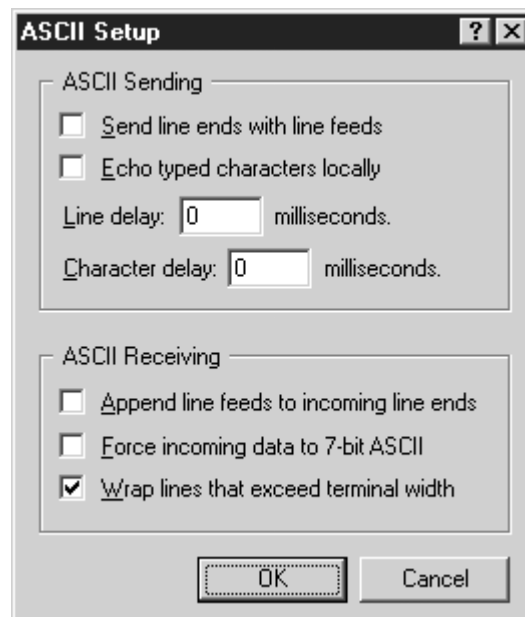
**Figure D2**  
*Set communication port to comm 1*



**Figure D3**  
*Comm port setup*



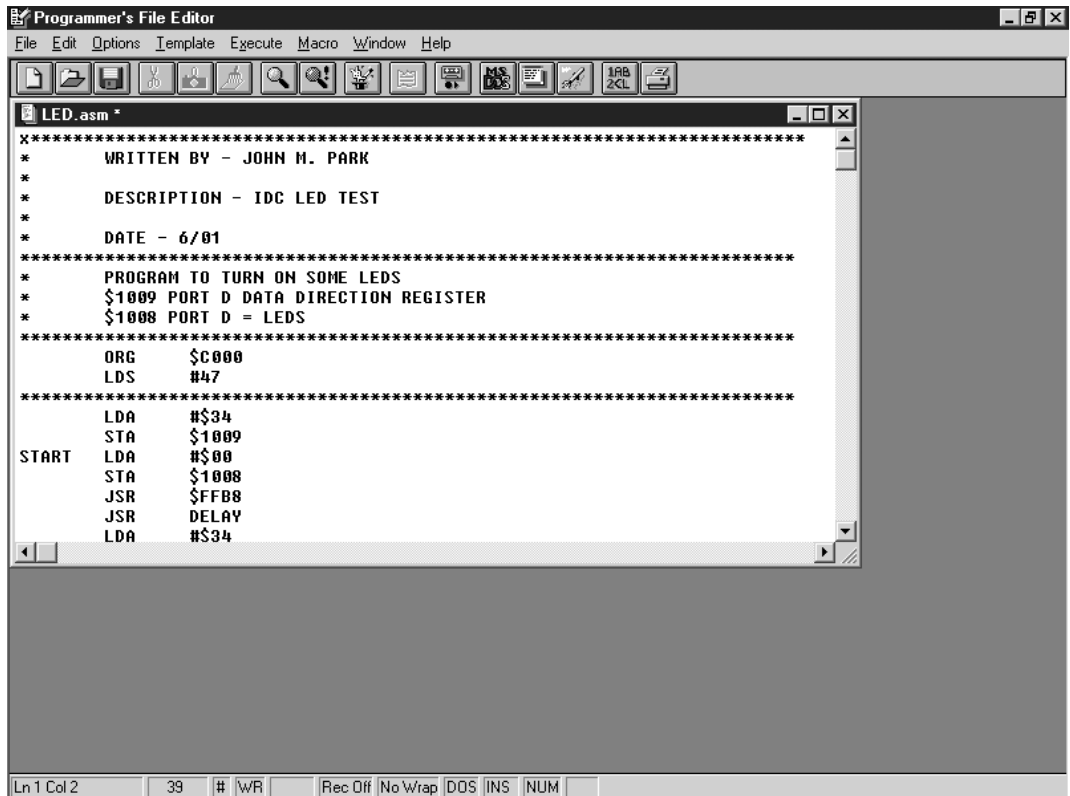
**Figure D4**  
Terminal setup



**Figure D5**  
ASCII transfer setup

## Sending a program to the EVM

The first step in sending a program that you have written to the EVM is to save the program by going to 'save as' under the file menu in the PF32 program. Save the program as LCD.ASM. If there is a file by that name already in the directory, write over it.



```

Programmer's File Editor
File Edit Options Template Execute Macro Window Help
LED.asm *
*****
*      WRITTEN BY - JOHN M. PARK
*
*      DESCRIPTION - IDC LED TEST
*
*      DATE - 6/01
*****
*      PROGRAM TO TURN ON SOME LEDS
*      $1009 PORT D DATA DIRECTION REGISTER
*      $1008 PORT D = LEDS
*****
          ORG      $C000
          LDS      #47
*****
          LDA      #34
          STA      $1009
START    LDA      #00
          STA      $1008
          JSR      $FFB8
          JSR      DELAY
          LDA      #34
  
```

Ln 1 Col 2    39    #    WR    Rec Off    No Wrap    DOS    INS    NUM

**Figure D6**  
*PF32 screen*

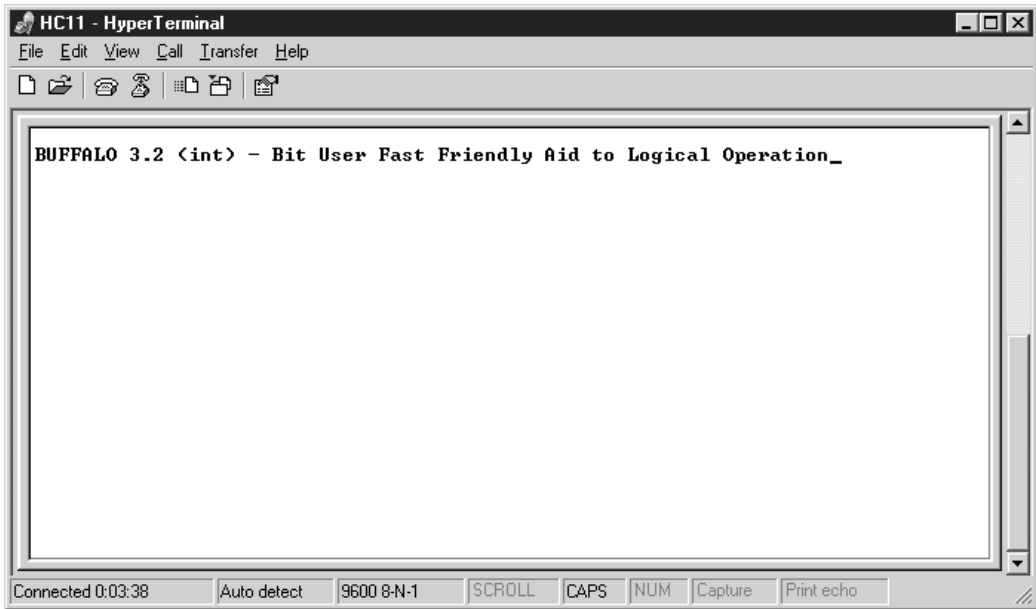
After the file has been saved click on the DOS button and type ASM11 LCD.ASM. This will compile the program into different formats. One of the formats will be a .S19 file. This file has the code that you will send to the EVM. Unfortunately the S19 format is not compatible with Hyper-terminal. While still in DOS change the extension of the .S19 file to .TXT. This is done with the following command.

```
REN LCD.S19 LCD.TXT
```

Using the ALT TAB change to the Hyper-terminal program.

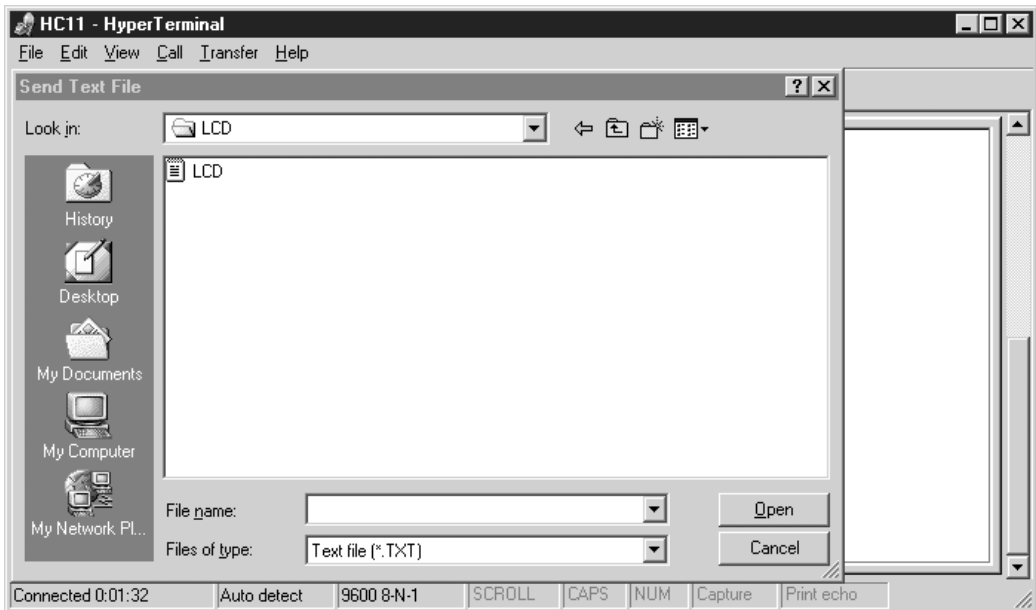
Press the RESET button on the EVM and the following line should be displayed in hyper-terminal.





**Figure D7**  
*Hyper-terminal front screen*

Type LOAD T and then press ENTER.  
Click on the Transfer menu and then choose Send Text File.  
The screen should change to



**Figure D8**  
*LCD.TXT*

Once the program is loaded, run the program by typing the following.  
G C000  
The LEDs

---

# PRACTICAL 5

---

## Reading keypad input

### Objective

This practical will introduce you to programming the 68HC11 microcontroller by reading information from the keypad on the EVM. The practical is done in three steps. First you will enter the code, as written below, into the assembler program. Next or at the same time the comments will be added to the program. Finally you will load and run the program using the EVM board provided.

### Writing the program

Click on the assembler program icon. Create a new file by clicking on the new file icon on the top menu. Type in the program exactly as written below. Add the appropriate comments to the right of each line. Comments can be added as it is being written or after it is complete. The comments should be short but include the information on what that line is trying to accomplish. Do not just repeat the instruction in the comments. Two examples of comments of an instruction could be...

```
LDAA    #$39           Load A with $39
```

This is not a very good comment. It just repeats the instruction. A better one would be.

```
LDAA    #$39           Load the ASCII character 9  
Or  
LDAA    #$39           Load 9 ASCII in Accumulator A
```

Remember that the comments are there so that anyone can understand what the instruction is trying to accomplish.

Capitalization does not matter, but the spacing of instructions is very important. The names for lines, such as START below, should always be placed on the far left-hand side of the page. Place one tab space (at least 8 spaces) between the line name and the

instruction. If there is no line name then put 2 tab spaces. There should be at least 1 tab space between the instruction and the operand of the instruction if any. There should be at least 2 tab spaces between the operand and the comment. Stars (shift 8) are used to indicate a comment line and to separate different parts of the program. Comment lines are not loaded into the microcontroller.

```

*****
*           WRITTEN BY - JOHN M. PARK
*
*           DESCRIPTION - KEYPAD TEST
*
*           DATE - 6/01
*****
*           ORG   $C000
*           LDS   #47
*****
KEY_B      EQU   $1000           KEY PRESS (PORT A)
TCTL1     EQU   $20             TIMER CONTROL REGISTER
PACTL     EQU   $26             PORT A REGISTER
*****
*           PROGRAM
*****
*           JSR   COMSET
RESTART    JSR   KEYP
*           JSR   RESTART
*****
*           SET UP THE LCD AND CLEAR SCREEN
*           $8000 = LCD CONTROL REGISTER
*****
COMSET     JSR   DELAY           DELAY FOR LCD
*           LDAB #38             LOAD FIRST CONTROL BYTE
*           STAB $8000           CONTROL REG
*           JSR   DELAY           DELAY FOR LCD
*           LDAB #$0C           LOAD SECOND CONTROL BYTE
*           STAB $8000           STORE IN LCD CONTROL REG
*           JSR   DELAY           DELAY FOR LCD
*           LDAB #$02           RESET SCREEN
*           STAB $8000           DO IT
*****
* KEYPAD SETUP
* ADJUST DIRECTION BITS IN DDRA3 IN 'PACTL'   $1026
*****
*           LDX   #KEY_B
*           LDAA  #$00
*           STAA  TCTL1,X
*           BSET  PACTL,X,$08  ADJUST PA3 TO BE OUTPUT
*           RTS
*****
* DISPLAY ONE key press to the screen
* $8001 = LCD SCREEN RAM
*****
DISPLAY    PSHA
DISLOOP    JSR   LCDBUSY

```

```

        PULA
        JSR   DELAY
        STA   $8001
        RTS

*****
*       DELAY FOR LCD
*****
DELAY      LDY   #$10f0
LOOP       DEY
           CMPY  #$0000
           BNE  LOOP
           RTS

*****
* CHECK BIT 7 TO SEE IF THE LCD IS BUSY
*****
LCDBUSY    LDAA  $8000
           ANDA  #$80
           CMPA  #$80
           BEQ  LCDBUSY
           RTS

*****
* KPSCAN - SCAN KEYPAD ONCE - MAIN SCAN SUBROUTINE
* RETURN: RAW DATA IN ACCA DATA FORM XRRRRCCC
* R=ROW C = COL ($07 = NO KEY PRESSED)
* NOTE NEED TO INITIALIZE PORT A BEFORE USING THIS ROUTINE
*****
KEYP       LDX   #KEY_B           POINT TO PORT A
           LDAA  #$07             LOAD A WITH %00000111
KPSK2      STAA  $00,X           STORE IN PORT A
           JSR   DELAY           DELAY
           LDAA  $00,X           LOAD A WITH WHAT IS IN PORT A
           SUBA  #$80
           CMPA  #$07           CHECK TO SEE IF ANY KEY PRESS
           BEQ  KEYP

*****
*       CHECK KEYPRESS AND ROTATE ROW CHECK
*****
CHECK1     LDAA  #$F7             LOAD 11110111
           LDAB  #$04             SET THE COUNT FOR 4
GETCHR     PSHA
           PSHB
           STAA  $00,X           STORE ZERO CHARACTER BYTE IN PORT A
           JSR   DELAY           DELAY
           LDAA  $00,X           READ KEYPAD
           PSHA
STACK      PUSH POSSIBLE GOOD CHARACTER ON THE
           ANDA  #$07             DELETE FIRST FIVE BITS / IS IT CORRECT ROW?
           CMPA  #$07
           BNE  CHECK2          IF CORRECT ROW THEN EXIT TO CHECK
CHARACTER  PULA
           PULB
           PULL BAD CHARACTER FROM STACK
           PULL B COUNT FROM THE STACK

```

```

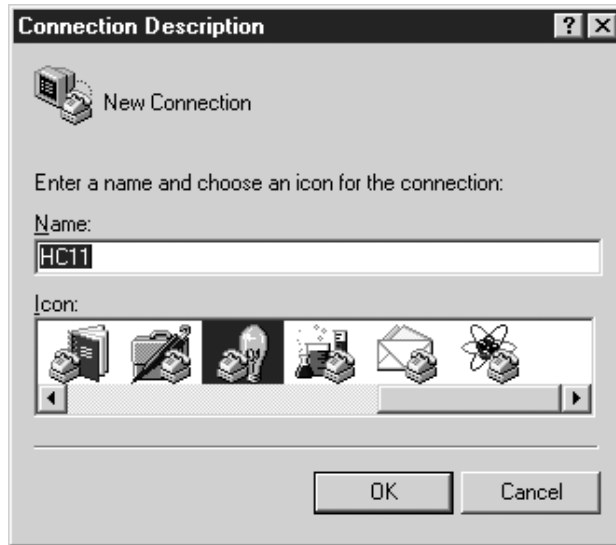
DECB          DECREMENT THE COUNT
CMPB #00     CHECK THE COUNT
BEQ  END2    IF DONE RESET
PULA         PULL 0 POSITION FROM STACK
ROLA        IF NOT CORRECT ROW OR DONE THEN
            ROTATE 0 LEFT
            TRY NEXT ROW
    BRA  GETCHR
END2        JSR  KEYP
*****
*****
*0=$BB, 1=$EB, 2=$EB, 3=$EE, 4=$F5, 5=$F3, 6=$F6, 7=$DD, 8=$DB, 9=$DE
* = $BD, # = $BE, NO KEY = $7F
*****
* KPGETC - SCAN KEYPAD WAIT FOR KEY TO BE PRESSED
* RETURN : ASCII DATA IN ACCA
*****
CHECK2      PULA          GET THE CHARACTER
            PULB          RESET THE STACK
            PULB          RESET THE STACK
            LDY #LOOK1    POINT TO LOOK UP
            LDX #LOOK2    POINT TO SECOND TABLE
KPGC2
*          SUBA #00
            CMPA $00,Y    SCAN VALUE SAME AS LOOK UP VALUE?
            BEQ  KPGC1
            LDAB $00,Y    IF VALUE POINTED$7F TOO FAR
            CMPB #07F     VALUE SCANNED IS NOT IN THE TABLE1
            BEQ  KEYP     GET VALUE FROM KEYBOARD AGAIN
            INX
            INY          ADJUST BOTH POINTERS
            BRA  KPGC2
KPGC1      LDAA $00,X     FIND EQUIVALENT
            JSR  $FFB8
            JSR  DISPLAY
            RTS
*****
* LOOK UP TABLE FOR RAW DATA
* 0=$BB,1=$EB,2=$EB,3=$EE,4=$F5,5=$F3,6=$F6,7=$DD,8=$DB,9=$DE
* *=$BD,#=$BE,NO KEY=$7F
*****
LOOK1      DB    $BD,$EB,$ED,$EE,$F3,$F5,$F6,$DB,$DD,$DE,$BB,$BE,$7F
LOOK2      DB    $30,$31,$32,$33,$34,$35,$36,$37,$38,$39,$2A,$23
*****

```

## Loading the software

**NOTE:** If you have setup Hyper-terminal in the previous practicals, then skip down to SENDING A PROGRAM TO THE EVM.

Start the computer and after it has booted up into Windows and run the program called Hyper-terminal from the icon on the screen or under accessories if there is no icon. When Hyper-terminal is booted up the screen should look like...

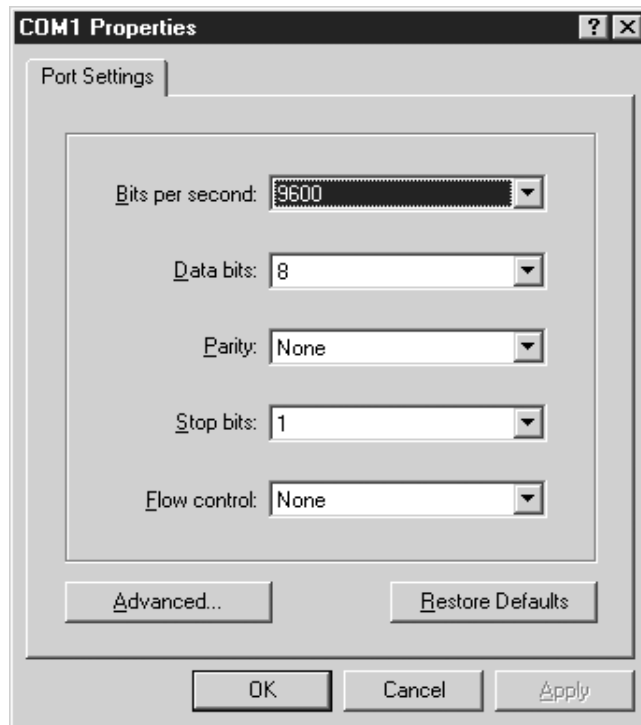


**Figure E1**  
*Hyper-terminal setup screen*

Add the HC11 and choose an icon and then click OK. Follow the setup screens as shown below as set up as shown.



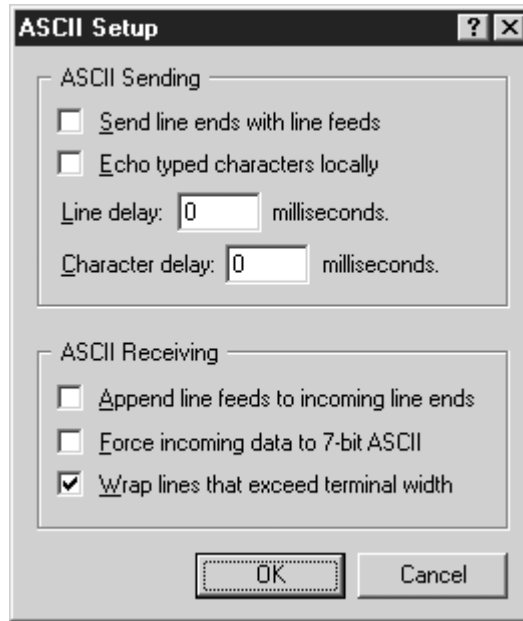
**Figure E2**  
*Set communication port to comm 1*



**Figure E3**  
Comm port setup



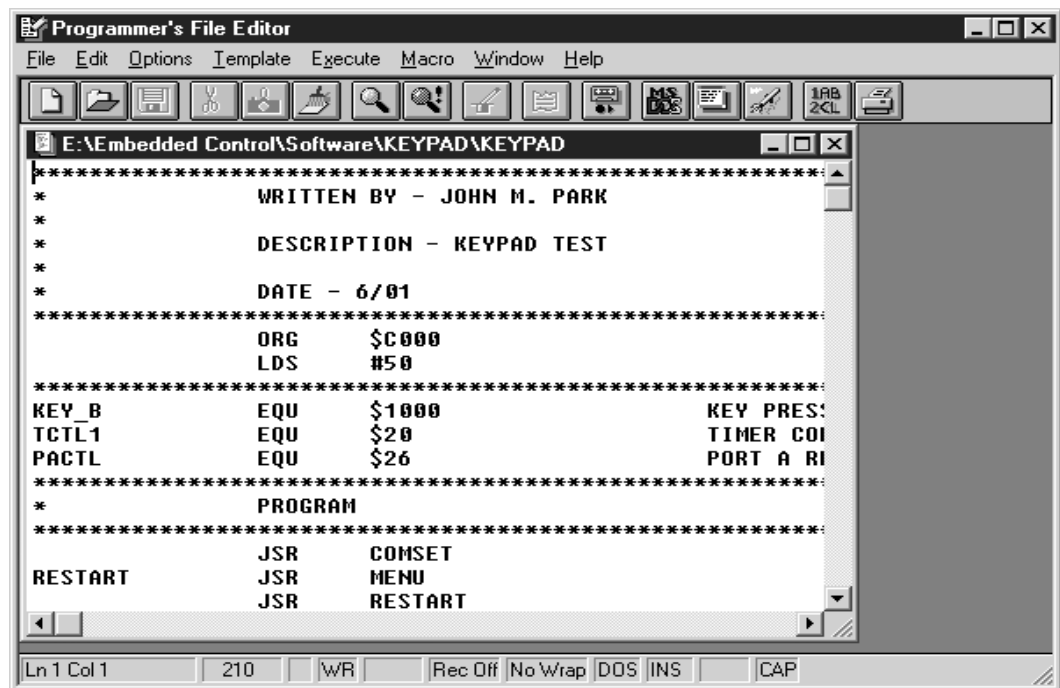
**Figure E4**  
Terminal setup



**Figure E5**  
*ASCII transfer setup*

## Sending a program to the EVM

The first step in sending a program that you have written to the EVM is to save the program by going to 'save as' under the file menu in the PF32 program. Save the program as KEYPAD.ASM. If there is a file by that name already in the directory, write over it.



**Figure E6**  
*PF32 screen*

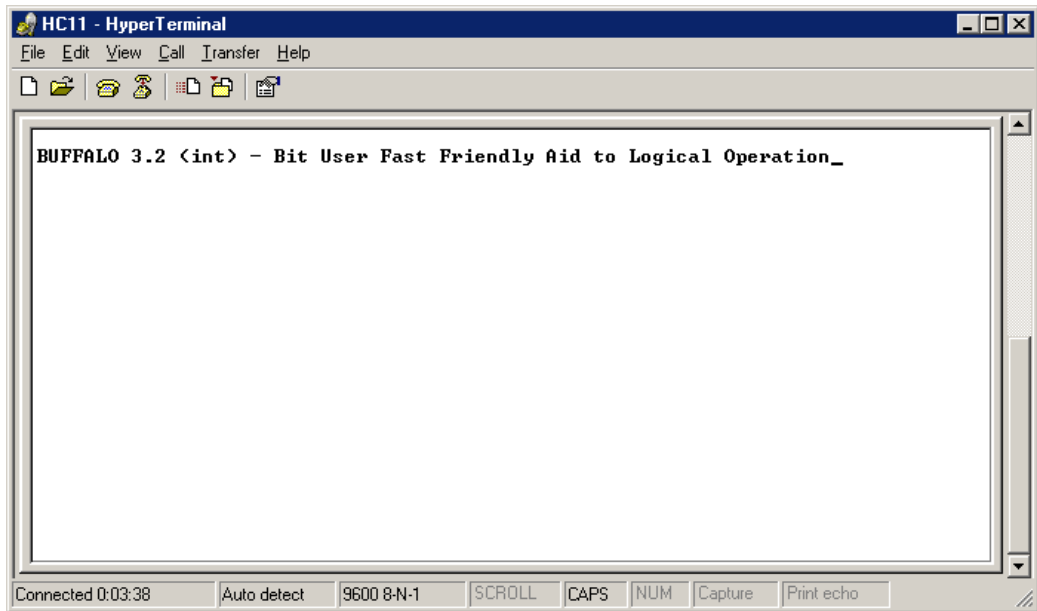


After the file has been saved click on the DOS button and type ASM11 KEYPAD.ASM. This will compile the program into different formats. One of the formats will be a .S19 file. This file has the code that you will send to the EVM. Unfortunately the S19 format is not compatible with Hyper-terminal. While still in DOS change the extension of the .S19 file to .TXT. This is done with the following command.

```
REN KEYPAD.S19 KEYPAD.TXT
```

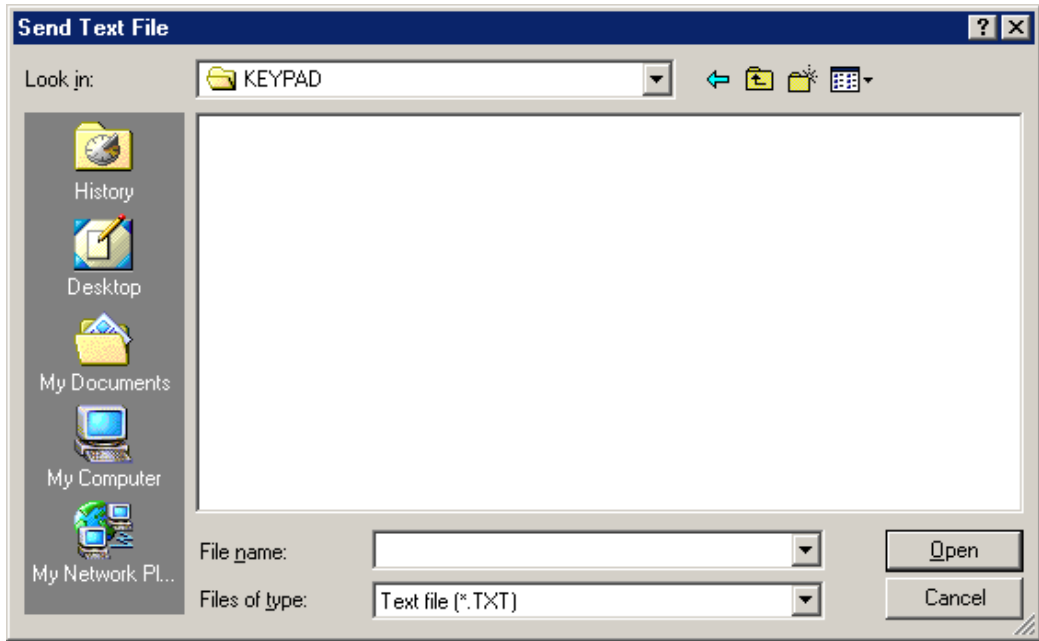
Using the ALT TAB change to the Hyper-terminal program.

Press the RESET button on the EVM and the following line should be displayed in Hyper-terminal.



**Figure E7**  
*Hyper-terminal front screen*

Type LOAD T and then press ENTER.  
Click on the Transfer menu and then choose Send Text File.  
The screen should change to



**Figure E8**  
*LED.TXT*

Once the program is loaded, run the program by typing the following.  
G C000

---

# PRACTICAL 6

---

## Using the PAT software

### Objective

The objective of this practical is to give you a basic understanding of the PAT software. The complete PAT manual can be acquired from IDC Technologies.

### Overview

You will learn how the PAT software can be utilized to monitor a data transmission. This will be accomplished by building a working simulation of a computer-to-equipment communication system. Two computers will be connected together to simulate the computer-to-equipment system. You will walk through the different sections of the PAT software including, port settings, communication port testing and protocol transfer. Then you will change the baud rate to 2 and view the bits on a breakout box.

### Installation

Locate the following equipment:

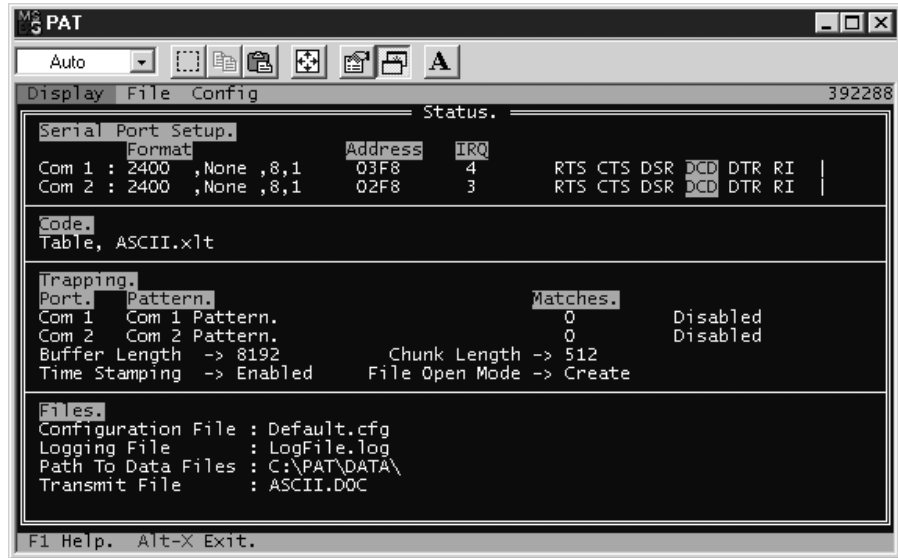
- Two computers
- Pat software
- One laplink cable
- Two 25-pin female to 9-pin male cables
- Two breakout boxes

### Hardware set up

- Connect the 9 to 25 cable cables to the 9-pin Com 1 port of the two computers
- Connect each end of the laplink cable to the 9 to 25 cable cables

## Software setup

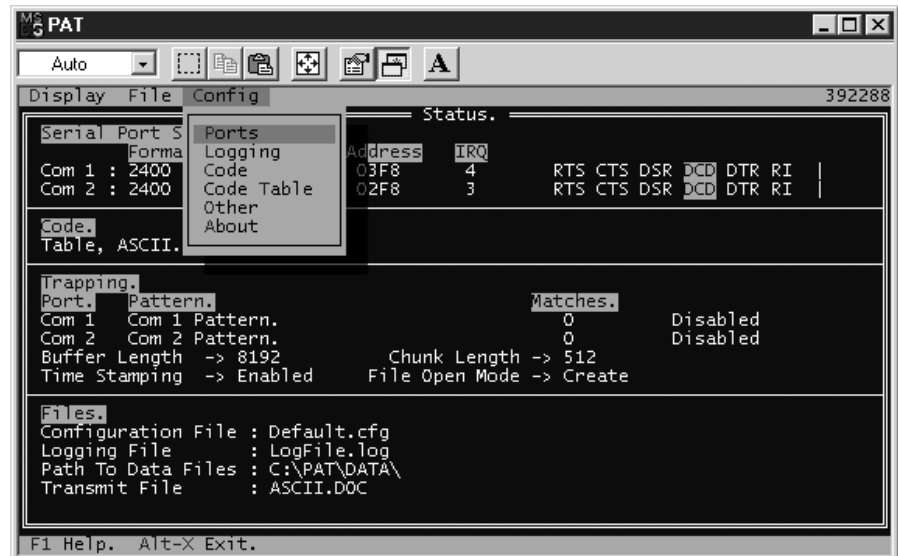
Run the PAT software by clicking the PAT icon on the desktop. If there is no PAT icon on the screen then run DOS from Programs in Windows. Once the PAT software has booted up and the two computers are on the screen, then press the space bar. The screen should change to...



**Figure F1**  
PAT front screen

## Communication port setup

Using the right arrow, move the cursor to the Config menu as seen in Figure F2



**Figure F2**  
Config menu

TAB over the PORT menu. Pressing the ENTER key on the keyboard, and open the PORT menu.



**Figure F3**  
*PAT port menu*

Change the Baud rate on the top line to 2. First delete the default Baud rate using the back space and arrow keys on the keyboard. Then type in the number 2.

When done press Enter.

Move back to the Interactive screen. Disconnect the cable that connects the computers.

Use the down arrow to move down the Display menu to Interactive.



**Figure F4**  
*Interactive menu*

Press Enter on Interactive and the screen should change to



**Figure F5**  
*Setup for interactive on com 1*

Press the Enter key and the screen should change to...



**Figure F6**  
*Interactive screen*

### ASCII character test

Choose three characters at random and using the ASCII Table F1 write the binary equivalent for each character.

Character \_\_\_\_\_ Binary \_\_\_\_\_

Character \_\_\_\_\_ Binary \_\_\_\_\_

Character \_\_\_\_\_ Binary \_\_\_\_\_

Draw the wave form for one of the characters that has been chosen.

Character \_\_\_\_\_ Waveform \_\_\_\_\_

	HEX	0	1	2	3	4	5	6	7
HEX	BIN	000	001	010	011	100	101	110	111
0	0000	(NUL)	(DLE)	Space	0	@	P	`	p
1	0001	(SOH)	(DC1)	!	1	A	Q	a	q
2	0010	(STX)	(DC2)	"	2	B	R	b	r
3	0011	(ETX)	(DC3)	#	3	C	S	c	s
4	0100	(EOT)	(DC4)	\$	4	D	T	d	t
5	0101	(ENQ)	(NAK)	%	5	E	U	e	u
6	0110	(ACK)	(SYN)	&	6	F	V	f	v
7	0111	(BEL)	(ETB)	'	7	G	Q	g	w
8	1000	(BS)	(CAN)	(	8	H	X	h	x
9	1001	(HT)	(EM)	)	9	I	Y	i	y
A	1010	(LF)	(SUB)	*	:	J	Z	j	z
B	1011	(VT)	(ESC)	+	;	K	[	k	{
C	1100	(FF)	(FS)	,	<	L	\	l	
D	1101	(CR)	(GS)	.-	=	M	]	m	}
E	1110	(SO)	(RS)	.	>	N	^	n	~
F	1111	(SI)	(US)	/	?	O	_	o	DEL

**Table F1**  
*ASCII table*

Press the corresponding key on the computer and watch the breakout box. Notice that one of the LED flashes a red light as the positive or zero bits are being sent. Does the LED flashing match the waveform?

---

# PRACTICAL 7

---

## Viewing character data transmission

### Objective

The objective of this practical is to give you an understanding how to transfer a character between two devices using RS-232 on the PAT software.

### Overview

You will learn how to monitor a character utilizing the PAT software. This will be accomplished by building a working simulation of a computer to equipment communication system. Two computers will be connected together to simulate the computer to equipment system. You will change the baud rate, test the communication port and then send characters from one computer to another.

### Installation

Locate the following equipment:

- Two computers
- Pat software
- One laplink cable
- Two 25-pin female to 9-pin male cables
- Two breakout boxes

### Hardware set up

- Connect the 9 to 25 cable cables to the 9-pin Com 1 port of the two computers
- Connect the breakout boxes to the 25-pin connector on the 9 to 25-pin cable
- Connect each end of the laplink cable to the breakout boxes



## Software setup

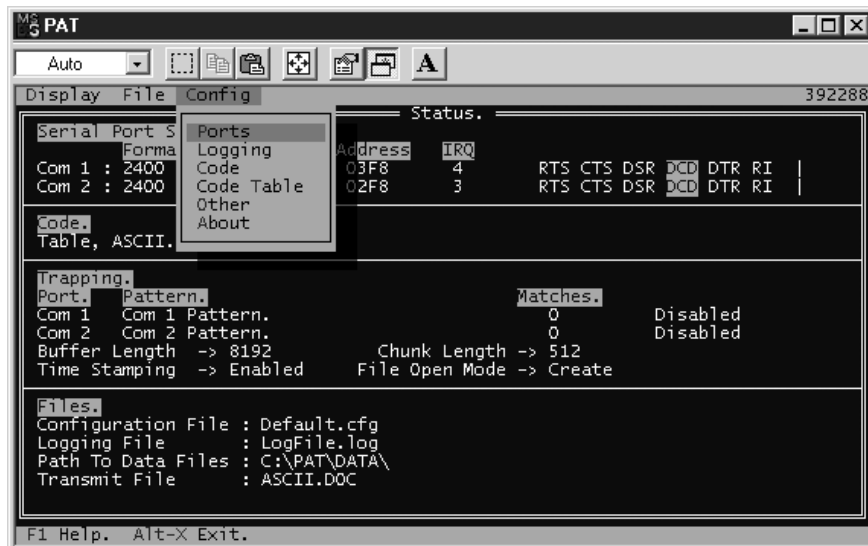
Run the PAT software by clicking the PAT icon on the desktop. If there is no PAT icon on the screen then run DOS from Programs in windows. Once the PAT software has booted up and the two computers are on the screen, then press the space bar. The screen should change to...



**Figure G1**  
PAT front screen

## Changing the baud rate

Use the right arrow to move over to the Port menu.



**Figure G2**  
Port menu

Press Enter and the screen should change to

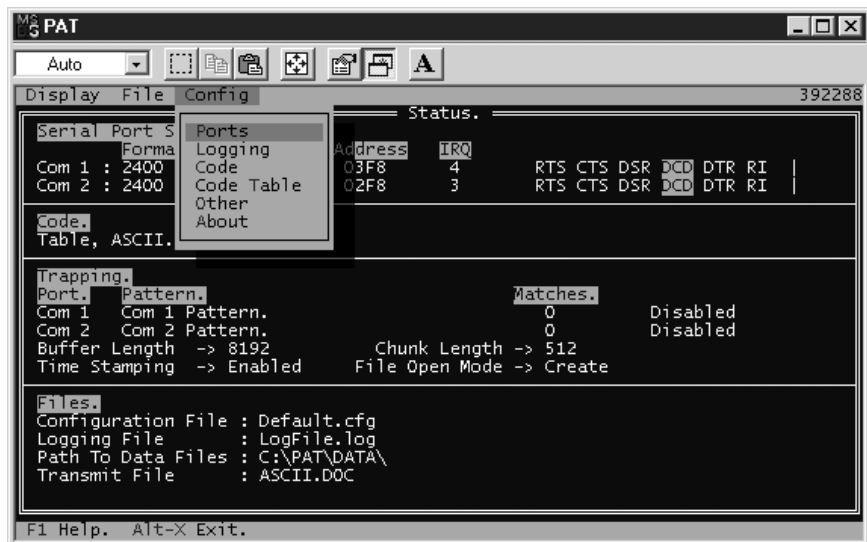


**Figure G3**  
*Baud rate*

Change the Baud rate to 9600 as shown.

Note that there are two ways to change the Baud rate. The Baud rate on top is adjustable to any number, while the one on the bottom is changeable only in standard Baud rates. Use the down arrow to choose the Baud rate. The top Baud rate will change when you change the bottom one. If the Baud rate is 9600 already then change it to 2400. Notice that the PAT software defaults to 8N1 and can be changed by using the Tab key to move down to Parity, Data Bits or Stop Bits.

Once the Baud rate has been changed press the Enter key. The screen should change back to...



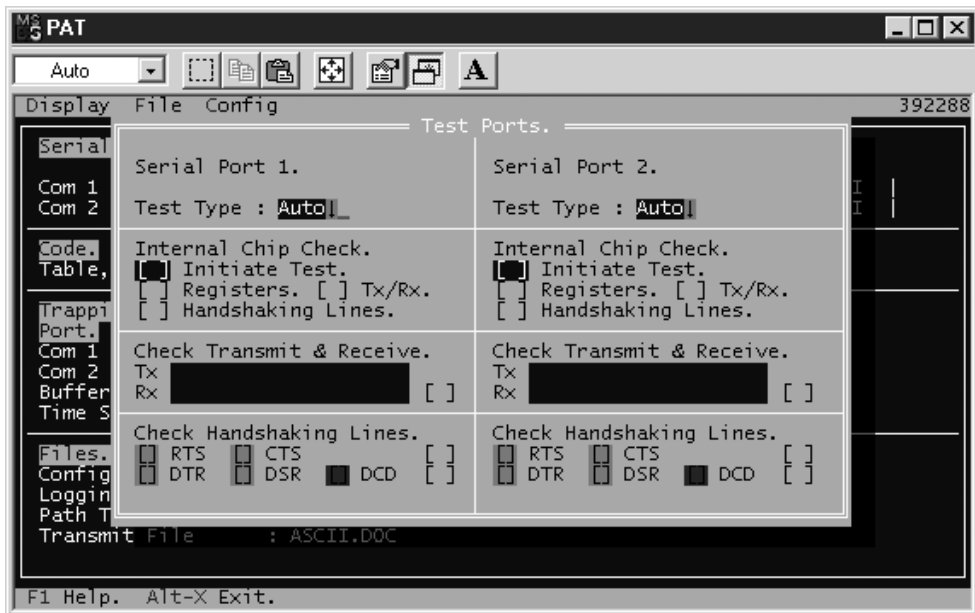
**Figure G4**  
*Port menu screen*

## Testing the communication port

Using the arrow key on the keyboard move to the Test Ports menu.

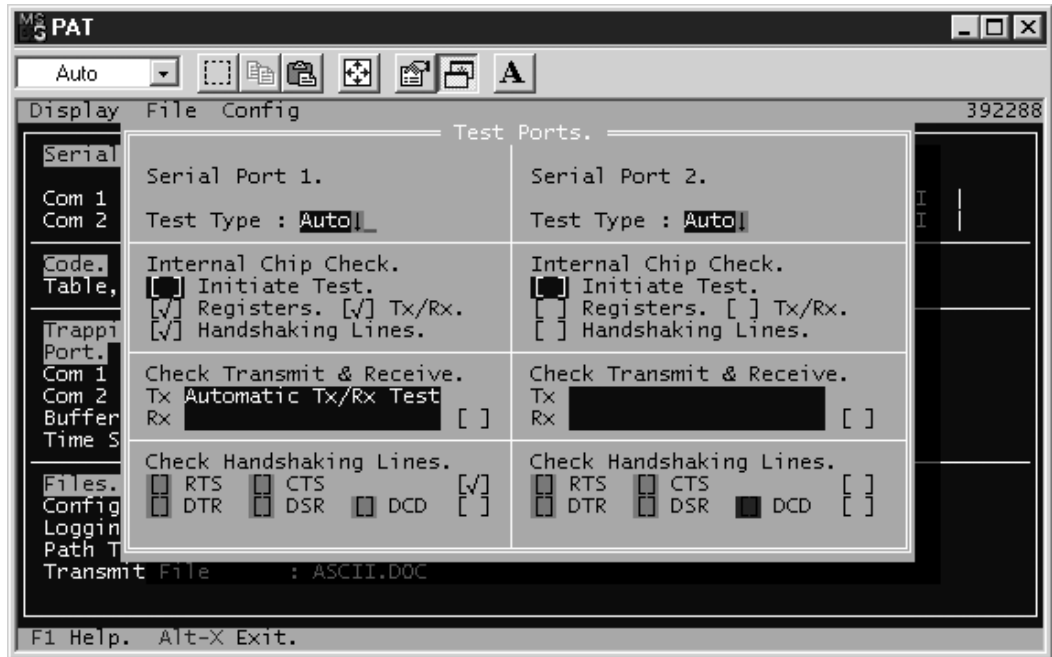


**Figure G5**  
Test ports menu



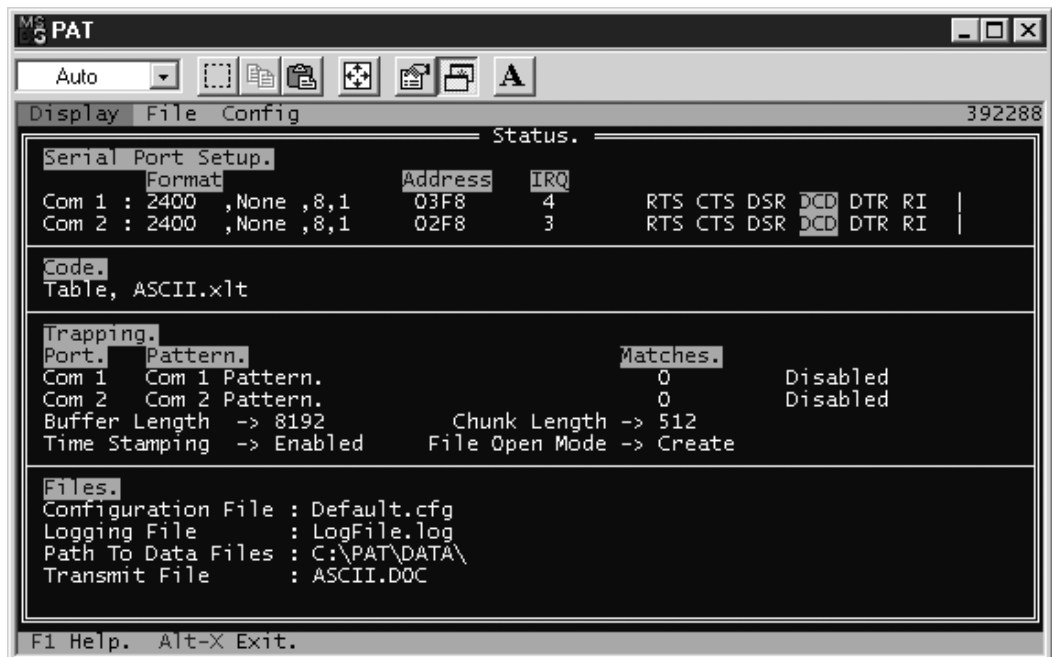
**Figure G6**  
Test ports screen

Press the Space key on the keyboard and notice that the PAT software will automatically test the communication port. The string Automatic Tx/Rx Test should show up on the screen as shown in the following figure.



**Figure G7**  
Test string

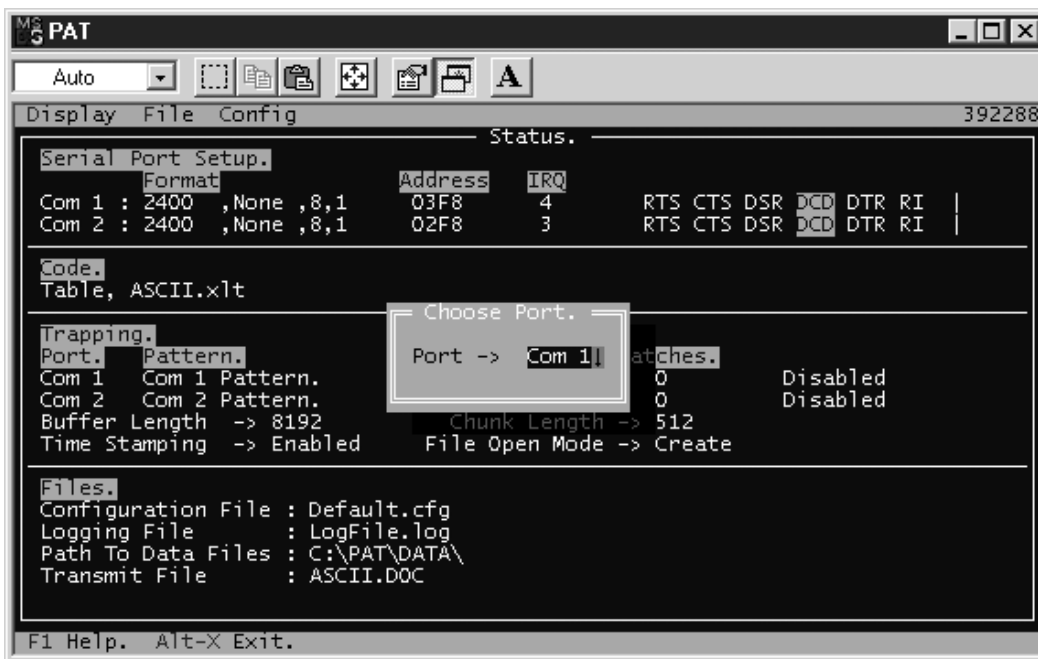
Notice that the RTS and DTR indicators change from green to red as they are toggled. Sending data from one computer to the other. Go to the Interactive screen by following the next figures.



**Figure G8**  
Display menu



**Figure G9**  
Interactive menu



**Figure G10**  
Select comm one



**Figure G11**  
*Interactive screen*

Type something on each computer and verify that the data is seen correctly on both computers. Is it correct? If not, why? Are the Baud rates the same?

Once the data transmission is verified press the End key on the keyboard.

---

# PRACTICAL 8

---

## Troubleshooting a data communication system

### Objective

The objective of this practical is to show you how to troubleshoot and solve possible problems with a data communication system. This will be done using two computers, the breakout boxes and the PAT software.

### Overview

You will set up the computers, breakout boxes and the PAT software as described in Practical 7. The two computers will be connected together to simulate the computer to equipment system. You will troubleshoot problems that are defined by the practical. This practical is done using two groups, one on each computer. These groups are defined in the practical as group 1 and group 2.

### Installation

Locate the following equipment:

- Two computers
- Pat software
- One laplink cable
- Two 25-pin female to 9-pin male cables
- Two breakout boxes

### Hardware set up

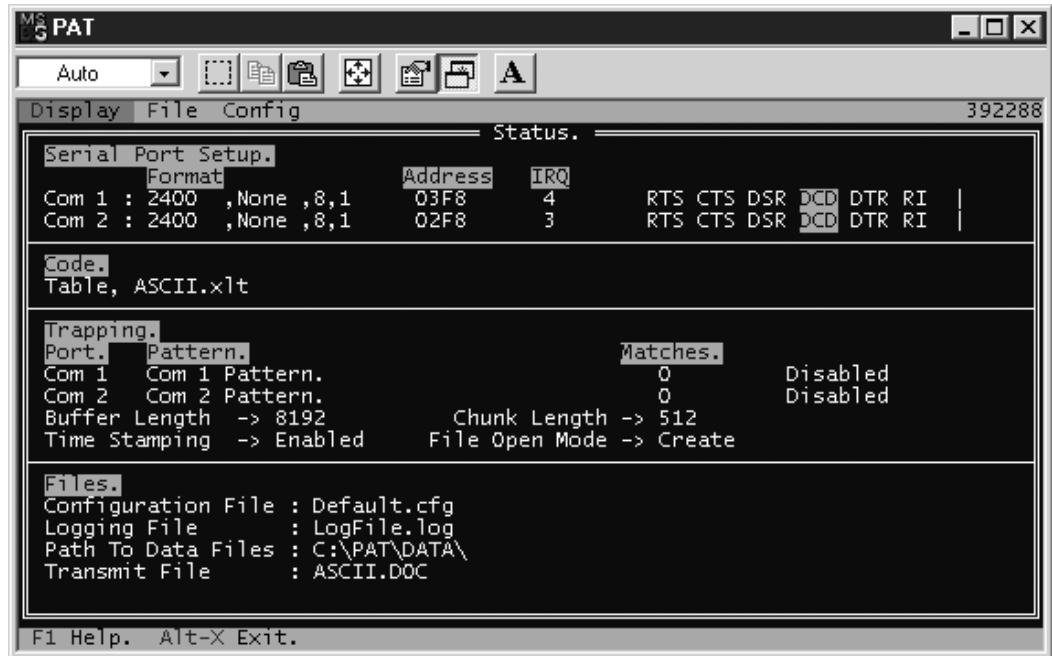
- Connect the 9 to 25 cable cables to the 9-pin Com 1 port of the two computers
- Connect the breakout boxes to the 25-pin connector on the 9 to 25-pin cable

- Connect each end of the laplink cable to the breakout boxes

## Software setup

It is assumed that the delegate knows how to use the PAT software and has transferred data using the interactive mode in the software.

Run the PAT software by clicking the PAT icon on the desktop. If there is no PAT icon on the screen then run DOS from Programs in Windows. Once the PAT software has booted up and the two computers are on the screen, then press the space bar. The screen should change to...



**Figure H1**

*PAT front screen*

### Wrong baud rate

Group 1 set up their Comm port for 2400 and group 2 set their Comm port in the PAT software for another baud rate. Group 2 will send a set of 8 characters to group 1. Group 1 will then try to determine what is the baud rate of group 2 using the interactive mode on the PAT software.

**Hint:** How many characters are received, compared to the number sent?

Once the baud rate is known the groups will switch and group 2 will set their baud rate for 2400 baud and group 1 will set their baud rate for an unknown baud rate.

Other group's baud rate \_\_\_\_\_

### Wrong number of bits or parity

Group 1 will set up their Comm port for 8N1. Group 2 will set their Comm port something else. Group 2 will then send characters to group 1 and they will try to determine what is group 2's setup.

**Hint:** What types of errors are shown on the PAT software. Parity errors?

When the setup is known, the two groups will switch as in previous example.

Other group's setup \_\_\_\_\_



Wrong baud rate and setup

Group 1 will setup their Comm port for 2400 baud and 8N1. Group 2 will set their Comm port for some other setting. Group 2 will send 8 characters to group 1 as in the last test. Group 1 will try to determine the baud rate and setup using the PAT software. Once the baud rate and setup is known, the groups will switch.

Other group's baud rate \_\_\_\_\_ and setup \_\_\_\_\_

Lights on the breakout box

Disconnect the cable that connects both computers. What is the name, line number, color and Binary value that relates to the lights that are illuminated on the breakout box?

Name \_\_\_\_\_ Number \_\_\_\_\_ Color \_\_\_\_\_ Binary \_\_\_\_\_

Name \_\_\_\_\_ Number \_\_\_\_\_ Color \_\_\_\_\_ Binary \_\_\_\_\_

Name \_\_\_\_\_ Number \_\_\_\_\_ Color \_\_\_\_\_ Binary \_\_\_\_\_

Name \_\_\_\_\_ Number \_\_\_\_\_ Color \_\_\_\_\_ Binary \_\_\_\_\_

Plug in the cable and connect the computers to each other. What has changed?

Name \_\_\_\_\_ Number \_\_\_\_\_ Color \_\_\_\_\_ Binary \_\_\_\_\_

Name \_\_\_\_\_ Number \_\_\_\_\_ Color \_\_\_\_\_ Binary \_\_\_\_\_

Name \_\_\_\_\_ Number \_\_\_\_\_ Color \_\_\_\_\_ Binary \_\_\_\_\_

Name \_\_\_\_\_ Number \_\_\_\_\_ Color \_\_\_\_\_ Binary \_\_\_\_\_

What's the function of the RTS and CTS lines?

\_\_\_\_\_

What is the voltage of the TX line? \_\_\_\_\_

In the Port setup screen in the PAT software assert the RTS line. Disconnect the computers from each other and read the voltage of the RTS line. What is the color, voltage and binary value of the RTS line now?

RTS color \_\_\_\_\_ Voltage \_\_\_\_\_ Binary \_\_\_\_\_

---

# PRACTICAL 9

---

## Troubleshooting a protocol problem

### Objective

The objective of this practical is to show you how to troubleshoot and solve possible protocol problems that can happen when using a data communication system. This will be done using two computers, the breakout boxes and the PAT software.

### Overview

You will set up the computers, breakout boxes and the PAT software as described in Practical 7. The two computers will be connected together to simulate the computer to equipment system. You will troubleshoot problems that are defined by the practical. Some of the problems are written while others are actual physical problems. Two groups do this practical, one on each computer. These groups are defined in the practical as group 1 and group 2.

### Installation

Locate the following equipment:

- Two computers
- Pat software
- One laplink cable
- Two 25-pin female to 9-pin male cables
- Two breakout boxes

### Hardware set up

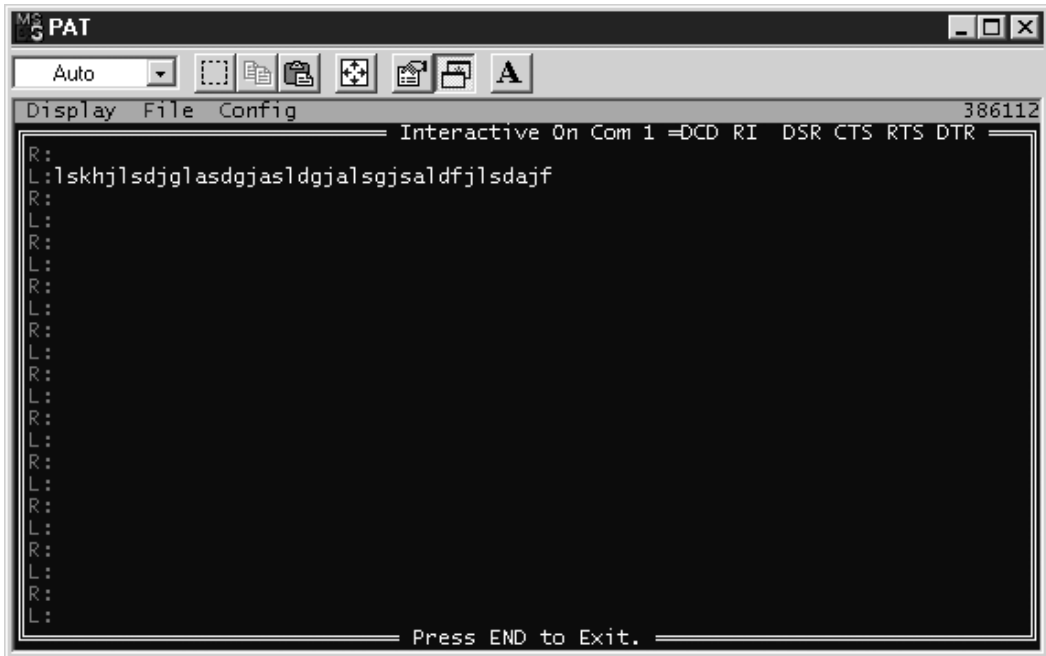
- Connect the 9 to 25 cable cables to the 9-pin Com 1 port of the two computers
- Connect the breakout boxes to the 25-pin connector on the 9 to 25-pin cable
- Connect each end of the laplink cable to the breakout boxes

## Software setup

Initially both groups will boot up and run the PAT software. They will test the hardware connection. Then one group as defined by the instructor will run the PAT software and the other group will run the MODBUS software.

### Testing the connection

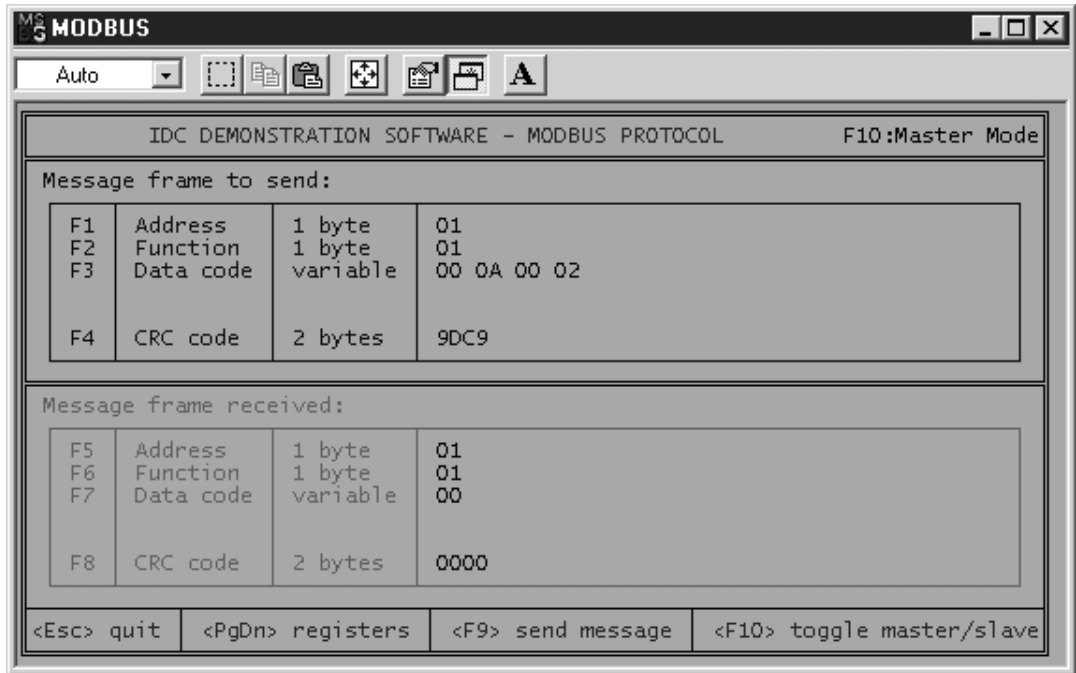
Both groups will boot up the PAT software and get into the Interactive mode. The PAT software should be setup for a baud rate of 9600 using 8N1. Both groups will attempt to send some data. If the data is correct, then the connection is correct between the computers.



**Figure 11**  
*Sending data*

### Sending a packet

Group 1 will exit the PAT software and get into the MODBUS software.



**Figure I2**

Press the F3 key when in the MODBUS software and setup as shown in Figure I2.  
000A 0002

Press F9 to send the packet to the other group. They should be in the interactive mode and receiving the packet. What do the characters look like?

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

Is there something wrong? Is the PAT software setup correctly?

\_\_\_\_\_

How many characters are you receiving? \_\_\_\_\_

**Hint:** What language are you using to look at the data?

### Logging a packet

Once correct characters are being received, setup the PAT software to log the data. Move to the Logging screen.

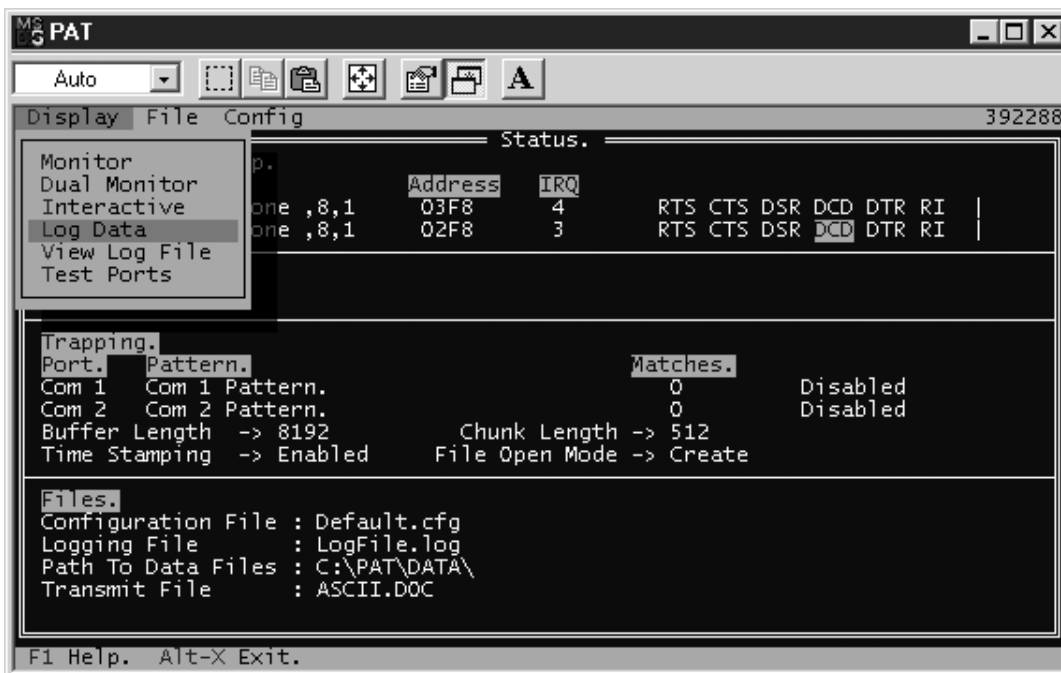


Figure 13  
Log data

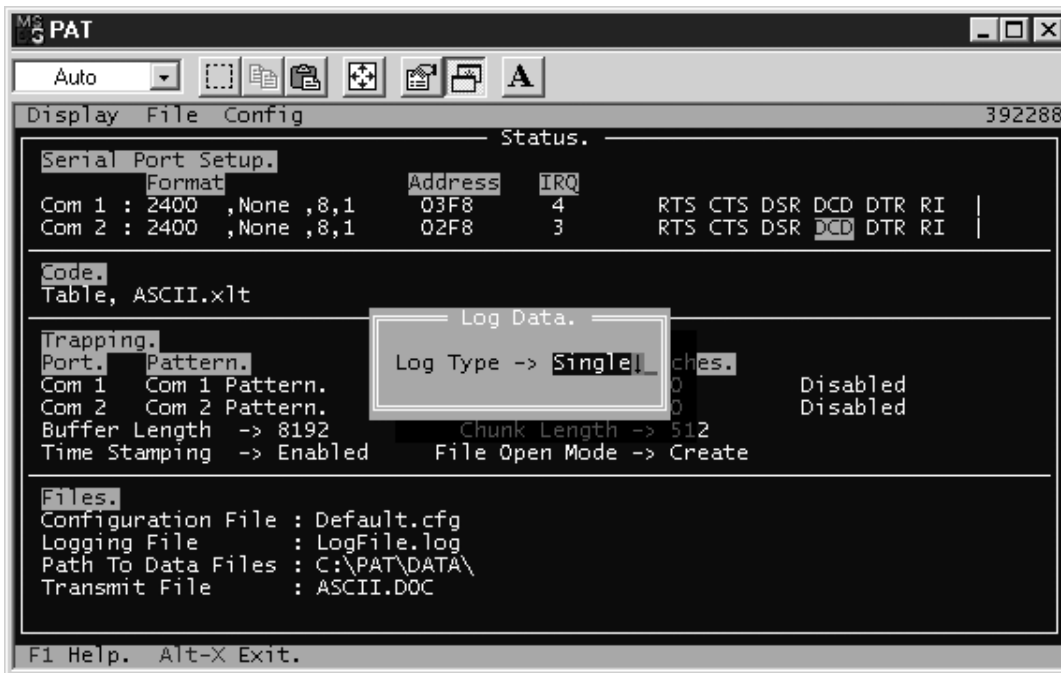
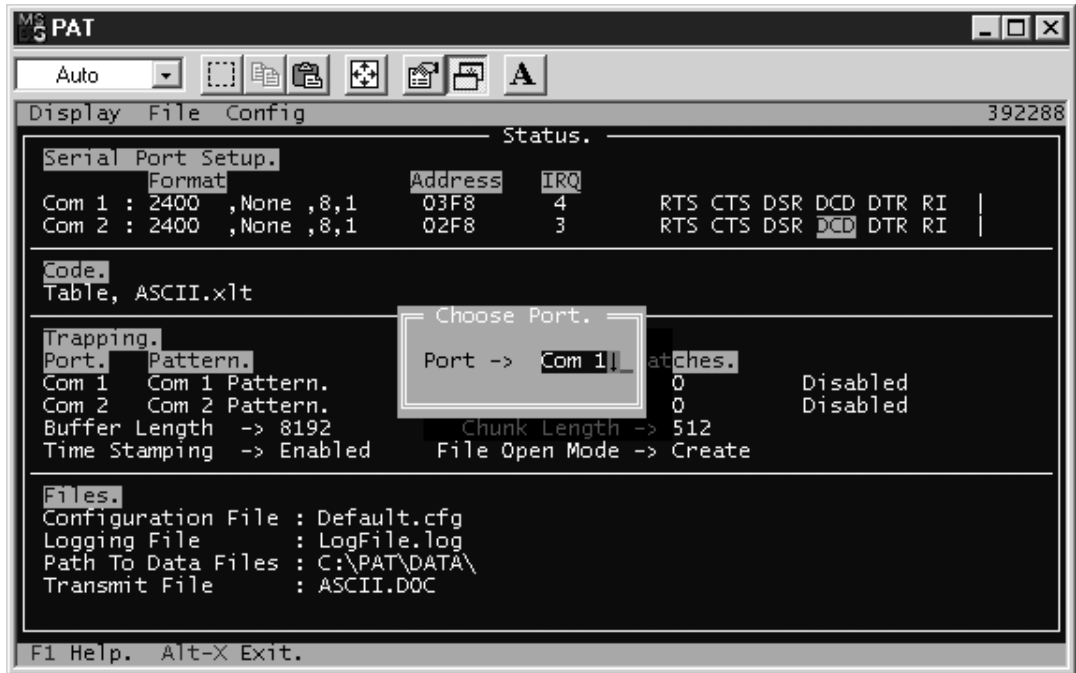
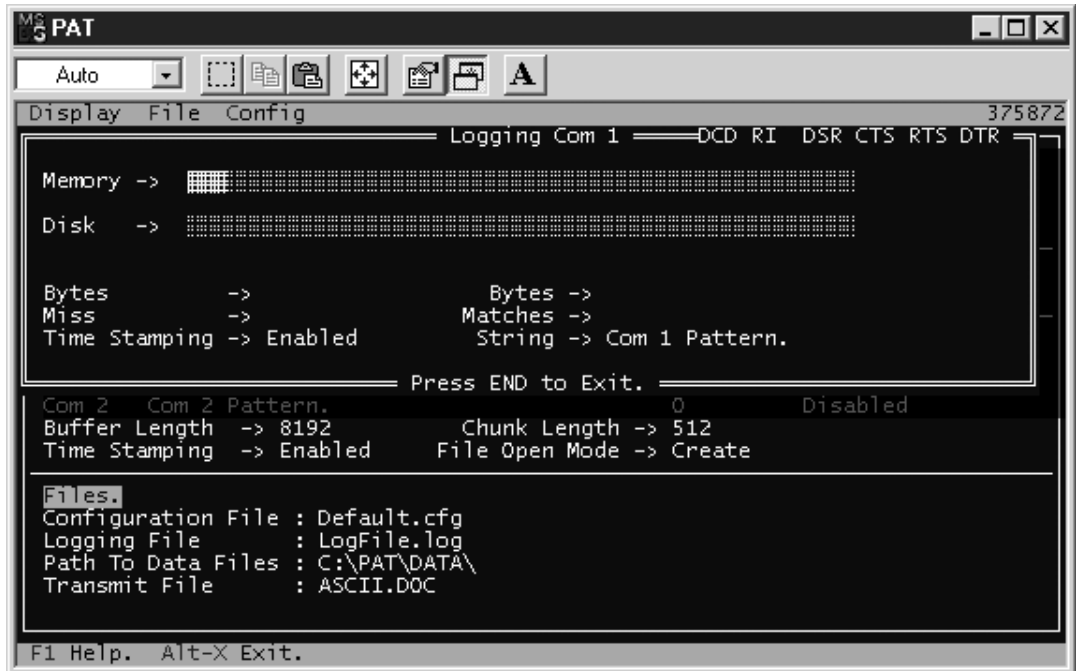


Figure 14  
Single log



**Figure I5**  
*Com port 1*



**Figure I6**  
*Logging data*

Once you are in this screen, indicate to the other group to send you a packet.  
 How many bytes did the other group send? \_\_\_\_\_  
 How many bytes of data do you receive? \_\_\_\_\_

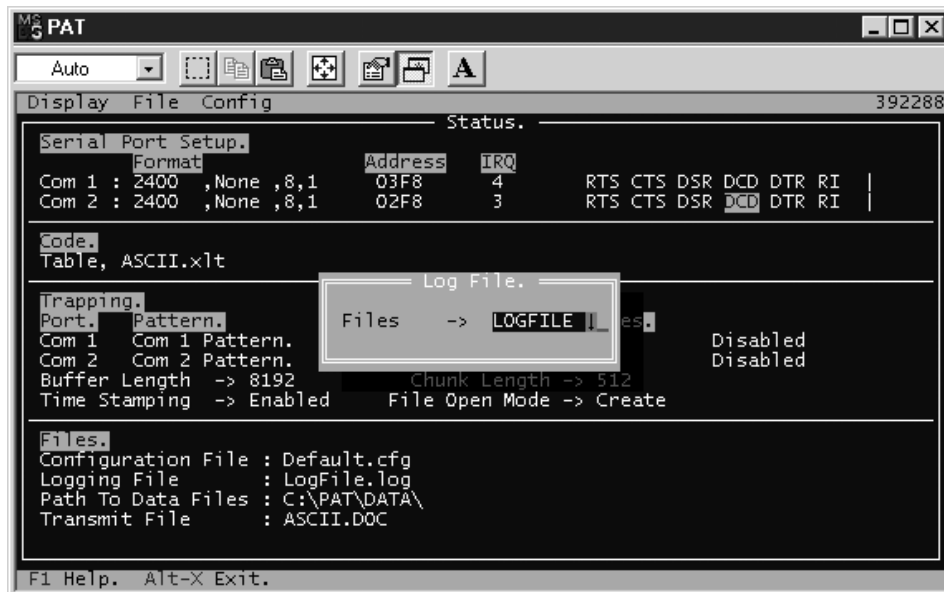
If the number of bytes is correct, then press Alt-X to get out of the logging mode. The PAT software automatically saves the data. To view the data go to the View Data screen.

**Hint:** What language are you using to view the data?

### Viewing a logged packet



**Figure 17**  
Log data



**Figure 18**  
Log File

The MODBUS packet should be on the screen. If it is not there or if it is not correct. Why?

# Bibliography

**Motorola reference manual**

M68HC11RM/AD

Revision 3

[www.motorola.com/sps](http://www.motorola.com/sps)

**EMC for product designers**

Tim Williams

Newnes Publishing

[www.newnespress.com](http://www.newnespress.com)

**Microcontroller technology: the 68HC11**

Peter Spasov

Prentice Hall Publishing

**The microcontroller 68HC11:**

**Applications in control, instrumentation and communication**

Michael Kheir

Prentice Hall Publishing

**Microprocessors and microcomputers: hardware and software**

Ronald J.Tocci

Prentice Hall Publishing

**MC 68HC11: an introduction**

Han-Way Huang

West Publishing



# Index

- 68HC11 17, 27, 29–30, 65, 77, 91, 180
- AC 87, 132, 135, 143
- Accumulator 17, 22–6, 41, 54, 56, 58–9, 71, 93
  - add or subtract bits 24
  - addressing modes 24
    - direct 24
    - indexed 25
    - inherent 25
    - relative 25
  - arithmetic shift 26
  - bit masking 26
  - bits *see* Bits
  - branch if equal (BEQ) 26
  - branch if not equal (BNE) 26
  - branch if higher (BHI) 27
  - branch if higher or same (BHS) 27
  - branch if lower (BLO) 27
  - branch if lower or same (HLS) 27
  - load accumulator( LDAA) 24–5, 61
  - logic shift 26
  - loop *see* Loop
  - register *see* Register
  - shift 26
    - most significant bit 26
    - rotate right 26
    - shift right 26
    - shifting of bits 26
  - store accumulator (STAA) 24
  - test bits 26
  - transfer accumulator 25
- Address:
  - 16 bit address 52–3
  - address bus *see* Bus
  - data bus *see* Bus
  - mode 52
  - pointer *see* Pointer
- AMD chip 61
- Amplifier 88, 126–9
- Analog input 4, 6, 18, 29, 45–6, 80–1, 87, 89–90, 98, 125
  - amplification 88
  - analog filters 88
  - filtering 88
  - measuring filters 89
  - Nyquist rate 89
  - resolution 90, 98
  - sample rate 89
- AND gates 22–3, 26, 41, 56
  - ANDA 26
  - in a microcontroller 22
  - logical 26
  - masking 22, 26
  - physical 26
- Antenna 11
- Arithmetic operations 51
- ASCII 16, 20–1, 25, 27–8, 41, 55, 93, 101, 182
- Assembly language 1, 6, 20, 42, 61, 63, 180
  - BASIC *see* BASIC
  - C++ *see* C++
  - database management 63
  - error checking 63
- Asynchronous 10, 18, 30–2, 41, 105, 117, 180
  - asynchronous communication 30,31
  - interface chips 10
  - large overheads 10
  - noise *see* Noise
  - slow speeds 10
  - start and stop bit 31
- BASIC 3, 6, 42, 61–2, 180
  - GW BASIC 61
  - interpreter 61
  - to convert 61
  - Visual BASIC 61
- Battery power 18
- Baud rate 36–8, 74, 114
  - in PCB 37
- Baudot code 101
- Binary 19, 20, 21, 61
  - binary coded decimal (BCD) 21
  - binary number 9, 16
    - AND *see* AND
    - gates *see* Gates
    - NOT *see* NOT
    - OR *see* OR
    - sub-set of hex 21
    - to do arithmetic functions 21
    - XOR *see* XOR
  - converting to hex 21
- Bits 19–20, 22–9, 31, 56, 60, 76, 90
  - 8-bit 26–7, 29, 41, 53, 60
  - 16-bit 26–8, 41, 57
  - shifting and rotating 56

- in stepper motor or lights 56
  - sequence 56–7
  - shift and rotate functions 57
- BNE STAR 125
- BUFFALO 52–3, 63, 67–9
  - BUFFALO memory map 68
- common utility 67
- Bus:
  - address bus 20, 53
  - control bus 20
  - data bus 6, 20, 29, 53, 86
- C++ 3, 6, 42, 61–2, 180
  - cross compiler 61
  - in embedded programming 61
  - to convert 61
- Cables 169–72
  - cable runs 11, 13, 169, 178
    - affect noise quality 13
  - cable ties 172
  - cable trays 169
  - coaxial cable 162
  - cooling 173
  - excess heat 173
  - horizontal cable run 171
    - optimal protection 171
  - ladder cable run 169–70
  - mounting cables 172
  - physical damage 173
  - random pattern 171
  - types 169
    - plastic cable runs 169–70
    - plastic cable tray 185
    - steel cable tray 169, 175, 185
  - vertical cable runs 170
  - wire management 174
    - common problems 174
- Call instructions 60
- Capacitance 132–3, 150
- Capacitor 87, 132, 137, 139, 150
- Central processing unit (CPU) 2–3, 6, 52–3, 73, 179
  - data bus *see* Bus
  - design 179
  - internal parallel address 6
- Clock bytes 31
- Code standards 101
  - ITU universal code 101
  - voltage standards 101, 111, 114
- Commands 25
  - TAB or TBA 25
- Common ground 145
- Common mode resistance ratio (CMRR) 82
- Condition code register 60
  - difference between 60
- Controller 1, 11–4
- Conduits 13
- Counter inputs 4
- Crystals 36–7
  - baud rates *see* Baud rates
  - determine the speed 36
  - faraday shield *see* Faraday shield
  - frequency 36–8
    - to stabilize clocking 36
  - low frequency crystal 38
  - resistor *see* Resistors
  - to reduce noise 37
    - see also* Noise
- CSMA/CD 106, 108–9, 119, 182
  - low traffic level 108
  - collision detection 108
  - in industrial controller system 108
  - node 108
  - problem 108
- Data:
  - data acquisition systems 87, 162
  - data communication 1, 8, 182
  - data loggers 179
    - program instructions 6
- db9 connector 116
- DC current measurement 87, 89
- DCS 11, 179–80
- De-bounce 83, 93
- Decibel 122
- Decimal 19–20
- Differential 80, 82, 126–7, 129, 181
  - common mode rejection ratio 129
  - differential analog circuits 82
    - differential nature 82
    - more resistance to noise 82
  - differential digital circuits 82
    - advantage 82
    - disadvantage 82
    - opto-isolator 82
    - to reduce noise 82
  - test equipment 127
- Digital:
  - digital circuit decoupling 137
  - capacitor decoupling 137

- digital controller 90
  - analog devices 90
  - DC stepper motor *see* Stepper motor
  - fuzzy logic 91
  - rotor 90
  - serial bus system 91
- digital input 4, 18, 45–6, 55, 81, 83, 98, 181, 185
  - bouncing 83
  - de-bounce *see* De-bounce
  - filters 88
  - NC *see* Switches
  - NO *see* Switches
  - problems 83
    - to delay the reading 83
- digital output 18, 47, 85, 181
  - controls 85–6
- Diode 87
- Distributed control system (DCS) 2
- EEPROM 2–4, 6, 8, 36, 41, 49, 63, 65–6, 74–8, 179, 181
- Electric field 150, 152–3
- Electro magnetic force (EMF) 86
  - back EMF 86
  - bad connections 3
  - clearing 75
  - destroy 86
  - in a microcontroller system 74
  - incorrect installation 3
  - memory location 75
  - very stable 74
  - writing 76
- Electronic circuits 11
  - noise reduction *see* Noise reduction
- Electronic equipment 3
  - cause catastrophic problems 3
- Embedded controllers 1–3, 9, 13–5, 185
  - cause catastrophic problems 3
  - intelligent electronic devices 2
    - control and monitor devices 2
- EMC 135–6, 138, 140, 146
  - cable trench grounds 158
    - in dry sandy areas 159
    - uninsulated cable 158–9
    - vertical ground stakes 158
  - grounding *see* Grounding
- EMI 12, 136, 148
- EPROM 3–4, 6, 8, 41, 52, 63, 65, 68, 179, 181
- Ethernet 9–10, 3–2, 100–1, 105–7, 182
  - cyclic redundancy 31, 35
- Evaluation module (EVM) 51, 67, 91
  - buffalo program 52
  - setting up 52
- Expanded mode 65
- External:
  - external address bus 3
  - external control bus 3
  - external data bus 3
  - external electromagnetic noise 169
  - external forces 3
  - external memory chips 6
- Faraday shield or box 37, 136–8, 151–3, 183
  - earth ground 151, 156
  - ferrite material 152
  - floating 151
    - non-grounded 151
  - incoming electromagnetic field 151
  - negatively charged 153
    - Edisson effect 153
    - corrosion due to electrolysis 153
      - to reduce corrosion 153
- Fiber optic 100, 117, 120, 172, 182
  - fiber optic cable 18, 30, 32, 100, 117–8, 172
    - immune to external noise 30
    - laser diode 117
    - light sensitive receiver 117
      - to melt connectors 117
      - to connect industrial devices 118
    - zip cable 118
- Fieldbus 106, 108, 118, 182
- Firewire 9, 32, 182
- Flow charts 43–5, 47
  - preparation phase 47
  - sub-subroutines 47
- Full-duplex 106–7
  - multi-dropped full-duplex 107
- Gas discharge tube (GDT) 155
- Gates 22–4
  - address location 22
- Grounded circuits 124
- Grounding 12
  - noise conduits 12
  - EMC grounding 146
    - Faraday box *see* Faraday box
    - on a PCB 146

- track placement 150
- ground specification 141
  - cabinet installations 143
  - common mistakes 141
  - common presumption 143
  - grid grounds 143
  - lightning strike 144
  - parallel grounds 143
  - real world example 142
  - single point 143, 145
- grounding solutions 10, 140
  - types of earth grounds 144
- lighting and static voltage protection 12
- new EMC requirements 12
- PCB *see* PCB
- safety ground 157
  - earth ground spike 157
  - lightning rods 159
  - tower lightning protection 159
- solution 140, 184
  - earth grounding 184
  - high priority 184
  - lightning protection 184
  - lightning rod 184
  - unshielded cable 184
- GW BASIC *see* BASIC
- Half-duplex 106, 119
  - most common mode 106
- Hardware vs software 33
  - cyclic redundancy check (CRC) 55
  - hardware reset 33
    - advantage 34
    - arithmetic checksum 35
    - design 34
    - disadvantage 34
    - error free 35
    - resets all the chips 33
    - self explanatory 34
  - software resets 34
    - stops and moves 34
  - types of resets 33
- HC11 17–8, 27, 37, 41, 46, 52, 63, 67
  - transmit and service lines 18
  - two communication modes 18
    - SCI *see* SCI
    - SPI 18
- Hex 19–21, 41, 101
  - conversion to binary 20
  - hexadecimal 16, 20, 93, 101
- I/O 2, 6, 9, 14, 29, 73, 78, 181
- Impedance 126, 130–1
- Inductance 150
- Industrial electronic equipment 1
- Inputs 1, 6, 8, 22–4, 28, 45, 47, 49–50, 73, 79, 181
  - analog inputs *see* Analog input
  - differential *see* Differential
  - digital inputs *see* Digital input
  - in data acquisition systems 80
  - planning phase 45
  - single end *see* Single end
- Installation 11, 48, 116–7, 162–3, 167, 169, 172, 174–5, 177
  - cable *see* Cable
  - conduit installation 175–6
    - plastic conduit 176
    - special UV resistance 176
  - connector problems and solutions 167–8
  - crimp connectors 13, 163, 166, 178
    - advantage 166
    - problems 166
  - failure in working equipment 13
  - planned maintenance 169
  - re-install 163
  - screw connectors 13, 163, 168–9, 178
    - flap type screw connectors 165
    - types 165
  - soldered connectors 163, 167, 178
    - first and most important 167
  - very common problems 163
    - excessive vibrations 168
    - vibration free location 168
  - visual and mechanical check 169
- Instruction:
  - BRA 60–1, 69
  - CMPA 26, 60
  - IMP 60–1, 69
  - RTS 60–2
- Intelligent electronic devices 2
  - control and monitor devices 2
- Interface manager 45
- Interrupt 60, 64, 68–71
  - hardware interrupts 70
    - data acquisition module (XIRQ) 70
  - maskable vs non-maskable 70
    - SCI interrupts *see* SCI
  - maskable interrupts 71, 78
  - non-maskable interrupts 71–2, 78
  - software vs hardware 70

- software interrupts 70
- types 64
- IRQ 69
- Jumps 60
  - conditional jumps 60
  - unconditional jumps 60
- Keypads 2, 29, 45–6, 91–4, 98, 182
  - accumulator *see* Accumulator
  - converts the key press 93
  - de-bounce *see* De-bounce
  - reading software 91
  - subroutine to check 93
  - to the evaluation modules 91
  - types 91
- LCD 6, 41, 51, 64, 79, 94–8, 182
  - data communication 99–103, 106
  - Ethernet *see* Ethernet
  - fiber optic *see* Fiber optic
  - on-board RAM 95
  - master slave bus 107
    - alarm situation 108
    - pooling method 107
    - round robin method 107
    - talk directly 107
    - master/slave 107–9, 182
  - protocol *see* Protocol
  - serial communication 100, 120
  - simplex 106, 119
  - speed problem 100
  - timed setup 95
  - three basic parts 101
  - timed system 109
    - faster data transfer 109
    - link active scheduler (LAS) 109
    - problem 109
    - to write to 97
- LED 57, 90, 117
- Lenz's law 86
- Load, store and transfer 53
- Logical operations 55
  - to check 55
- Loop 26, 58
  - CBA 26
  - CMPA *see* Instructions
  - CMPB 26
  - CPD 26
- Memory 1, 4, 6, 8–9, 24–7, 33–5, 41, 49, 59, 61, 63–9, 71–5, 77, 180–81
  - BUFFALO *see* BUFFALO
  - in a microcontroller 63
  - internal and external 63
  - internal RAM 65
    - see also* RAM
  - memory-checking 35
  - memory corrupt 34
  - memory map 1, 9, 64, 67–8, 78
  - vectoring 64
    - interrupt *see* Interrupt
    - is a way of 64
    - vectors *see* Vector
- Metal oxide varistor (MOV) 155
- Microcontroller:
  - converts 21
  - design 3
  - displays 2
  - Ethernet *see* Ethernet
  - equipment ground 156
  - in embedded controllers 16
  - keypads *see* Keypads
  - main components 3
  - memory *see* Memory
  - programming 5
    - also see* Programming
  - power systems *see* Power systems
  - registers *see* Registers
  - relays *see* Relays
  - safety considerations 177
    - installation *see* Installation
    - to reduce accidents 177
    - voltage-sensing device 178
  - sample rate 9
  - temporary storage locations 17
  - to control mains/solar 18
  - troubleshooting techniques 176, 182
    - common method 176
    - possible problems 176
- Microwave 2, 142
- Modbus 101, 106, 108, 182
- Morse code 20, 100
- Motorola chip 61
- Multimeter 141
- N type connectors 162
- Nanometers 133
- Nibble 19,20

- Noise 3, 11, 31, 37, 81–3, 100, 110, 112, 115, 121–3, 126, 129–35, 138–40, 143–4, 147–50, 162, 169–71, 178, 183–5
  - capacitive coupled noise 132–3
    - adjacent equipment 132
    - communication lines 133
    - problems 132–3
    - separated by 132
  - causing phantom characters 31
  - conductive coupled noise 131
    - external equipment 131
    - problems 137
    - transmission lines 131
  - electronic noise 121, 132, 139
  - external noise 130–1
    - types 130
  - internal and external 122–3, 139
  - low impedance 129
  - magnetically coupled noise 133–5
    - from adjacent equipment 135
    - induced noise 134–5
    - line of force 133
    - magnetic field 133, 135, 150, 152–3
  - more susceptible to noise 31
  - noise problem 128
  - noise reduction 11, 14, 135–6, 169, 183, 185
    - batter effect 13
    - broadcasting wire 11
    - equipment acts like an antenna 11
    - exposed conductor 11
    - four areas of noise reduction 12
    - in PCB design 135
    - less susceptible to noise 11, 31
    - methods 135
    - noisy circuits, filters 11
    - preventing noise 11
    - victim wire 11
  - noise transmitted 123
  - on PCB 124
  - power of noise 122
  - radio antenna 123
  - signal to noise ratio 88, 122
  - source 122
  - to other chips 37
- NOT gates 22, 24, 41
  - to inverse 24
  - used in conjunction with 24
  - used to reverse 24
- Number systems 17, 19, 41
  - binary *see* Binary
  - decimal *see* Decimal
  - hex *see* Hex
- Ohm's law 87
- OR gates 22–3, 26, 41
  - ORA 26
- Oscillator 18, 36
- OSI model 103, 105–6, 119
  - application layer 103–4, 106
  - data link layer 103, 105–6, 119
    - logical link control layer 105
    - low level device driver 105
    - media access control layer 105
    - UART 105, 110
    - USART 105
  - physical layer 103, 105–6, 119
  - presentation layer 105
  - session layer 105
  - transport layer 105
- Outputs 1, 6, 8, 22–4, 28, 45, 47, 49–50, 56, 73, 79, 181
  - digital outputs *see* Digital output
  - electromagnetic 47
- Overhead ratio 109
- Packet 31, 41, 109
- Pascal 61
- PCB 11–2, 18–9, 22, 37–8, 64, 135–38, 140, 143, 145–54, 160–1, 181, 183–4
  - design 146
    - clean ground 148
    - multi-layer-boards 149
    - recommendation 146
    - rule 146
  - ground plane 183
  - radiate EMI frequencies 11
- Programmable logic controller (PLC) 2, 11, 179–80
- Pointer 57
- Ports 8, 17–18, 22–5, 27–30, 56–7, 59, 67, 73–4, 86, 115, 117, 180
  - bi-directional 8
  - definable port 8
  - expanded mode 29
  - external chips 29
  - I/O ports 6
  - input port 180
  - keypad *see* Keypad
  - output port 180
  - parallel ports 4, 6, 180

- register *see* Registers
- setup 8
- Power systems 32
  - COP watchdog (woof) 35, 41
    - circuits 36
    - computer operating property (COP) 35
    - no operation (NOP) 35, 179
  - battery-powered 36
  - black outs 18
  - brown outs 18
  - lockup 33
  - microcontroller-reset circuitry 36
  - power failure 36
  - power fluctuates 36
  - spikes 18
- Profibus 106, 109, 118, 182
- Programming 42
  - manipulation of data 46
  - overall structure 48
    - constants 48–9
    - initialization 48
    - strings *see* Strings
  - planning 43–4
    - problems 44
  - three basic parts 45
- Protocol 101–3, 105–7, 109–11, 117–9, 182
  - alternative 102
  - ground planes 149
    - basic types 149
    - compound grounding 149
    - multi-layer PCB 149
    - purpose 149
  - identify 146
  - open standard protocol 103
  - placement of protection 154
  - proprietary protocol 103, 118
  - protecting from lightning 153
  - translated 102
- PVC 13
- Random access memory (RAM) 2–4, 6, 8, 25, 27, 36, 41, 47, 49, 51, 59, 63–6, 74–5, 77–8, 95, 179, 181
  - battery 64
  - corrupt 64–5
  - external RAM 65
    - 64 K addressing 65
    - hold large amount of data 65
  - in a microcontroller 64
  - INIT register 65
    - measured in 65
    - non-volatile 64
    - rules of guaranteed quality 66
    - volatile 64
- Ratio 31
- Read external devices 6
- Receiver 31
- Register 4, 6, 17, 22, 24, 26–9, 41, 49, 54, 57, 58–60, 65, 68, 71, 73–6, 78, 86, 95, 115, 124, 180, 182
  - see also* Accumulator
  - 8-bit *see* 8-bit
  - 16-bit *see* 16-bit
  - bias-resisters 115
  - binary information 28
  - bits *see* Bits
  - control registers 73
  - data direction 28–9, 73
  - direction control register 73
  - index registers 16, 25–7, 42, 48, 57, 59
  - pointer *see* Pointer
  - ports *see* Ports
  - port D bit 0 (PD0) 28
  - reduce reflection 115
  - stack *see* Stack
  - termination resistor 115, 117
  - voltage resistor 115
- Relays 2, 6, 47, 85–7, 181
  - AC relays 87
  - DC relays 87
  - snubber networks 87
  - solid-state relays 47, 86
    - to eliminate back EMF 87
    - to increase 86
    - to turn on 47
- Repeater 114
- Resistance 87, 141–2
- Resistor 38, 82, 87
- ROM 4, 6, 8, 63, 65, 74
- RS-232 6, 9–10, 18, 30, 32, 41, 67, 99, 102, 110–2, 115, 117, 119–20, 124, 127, 182
  - 25 pin connectors 112
  - analyzer 114
  - breakout box 113
  - common confusion factor 11
  - connectors 111
  - data communication equipment (DCE) 111–3
  - data terminal equipment (DTE) 111–3
  - DTE to DCE 111

- DTE to DTE 113
- EIA 110
- link combinations 111
- problem shoot 114
- three-wire transmission method 110
- to communicate 110
- voltage standard *see* Voltage standard
- RS-422 9–10, 32, 102, 115, 182
- RS-485 6, 9–10, 18, 30, 32, 41, 99, 101–2, 114–6, 119–20, 127, 182
  - for multi-drop communication 114
  - preferred connector 116
  - problem 116
  - to reduce noise 115
- RS-485 Vs RS-422 115
- RTS-STAR 25
  
- SCI 18, 67, 71, 74
- Sensor 11, 13, 29, 80, 84, 87–8, 90, 125
- Serial communications 6, 30–1
  - fiber optic cable *see* Fiber optic
  - parallel communication 30
  - two basic transmission modes 30
- Serial data communication 4, 29
- Shifts and rotates 16, 26, 53, 56
- Single end 80–2, 102, 124–6, 139, 181
  - analog single end circuit 181
  - common mode 128
  - differential amplifier 127, 129
  - floating source 127
  - grounded 124–5
  - measurement 124, 126–7, 129
  - measuring a signal 125
  - single ended analog circuits 80
    - choke base isolator 81
    - pseudo-differential 80
    - to reduce noise on 81
  - single ended digital circuits 81
    - advantage 81
    - disadvantage 81
    - via opto-couplers 81
  - types of single ended test 125
- Smart sensor 2
- Software design 43
  - flow chart *see* Flow chart
  - most common problems 42
- Solar power 18
- Stack 28, 41, 54, 57, 59, 68
  - correct way to use 28
  - first in/last out 59
  - functions 59
  - in a microcontroller 28
  - LDS 28,
  - memory block 59, 61, 63–4
  - often compared 59
  - push instruction 28
  - to free up 59
  - to hold data and address 59
    - see also* Address
    - user-defined 59
- Stepper motor 56, 90–1
- Strings 48, 51
  - string storage area 51
- Subroutine 45, 47–51, 59, 61, 63, 67, 69–70, 83, 93–4, 105, 180
- Switches 6, 84, 85
  - electronic switches 83–4
    - highly susceptible 84
    - problem 84
  - magnetic switch 84
  - mechanical switches 83–4
  - normal close (NC) 84
  - normal open (NO) 84
  - switch sensing 83
  - switch sensor system 84
- Synchronous 10, 18, 30–2, 41, 105, 117
  - as a packet 31
  - edge triggering 31
  - lack of 10
  - for higher data 10
  - overhead ratio 37
  - overheads 31
  - synchronous communications 31
  - to maintain high-speed 10
  - USB *see* USB
  
- Token bus 106, 109, 119
  - connects multiple nodes 109
  - pseudo-master 109
- Torque 90, 168–9, 172, 178
- Transistors 83–4, 86, 181
- Transistor transistor logic (TTL) 47
- Transorbs 155
- Troubleshooting 108, 113, 159, 173
  
- USB 9, 32, 182
  
- Vectors 64, 67–70, 78
  - pseudo-vector 68–70, 78
  - interrupt vectors 68, 78



Visual BASIC *see* BASIC

Voltage 10, 29

World wide web (WWW) 101

XOR gates 22–3, 41

Zener diode 155

## THIS BOOK WAS DEVELOPED BY IDC TECHNOLOGIES

### WHO ARE WE?

IDC Technologies is internationally acknowledged as the premier provider of practical, technical training for engineers and technicians.

We specialise in the fields of electrical systems, industrial data communications, telecommunications, automation & control, mechanical engineering, chemical and civil engineering, and are continually adding to our portfolio of over 60 different workshops. Our instructors are highly respected in their fields of expertise and in the last ten years have trained over 50,000 engineers, scientists and technicians.

With offices conveniently located worldwide, IDC Technologies has an enthusiastic team of professional engineers, technicians and support staff who are committed to providing the highest quality of training and consultancy.

### TECHNICAL WORKSHOPS

#### TRAINING THAT WORKS

We deliver engineering and technology training that will maximise your business goals. In today's competitive environment, you require training that will help you and your organisation to achieve its goals and produce a large return on investment. With our "Training that Works" objective you and your organisation will:

- Get job-related skills that you need to achieve your business goals
- Improve the operation and design of your equipment and plant
- Improve your troubleshooting abilities
- Sharpen your competitive edge
- Boost morale and retain valuable staff
- Save time and money

#### EXPERT INSTRUCTORS

We search the world for good quality instructors who have three outstanding attributes:

1. Expert knowledge and experience – of the course topic
2. Superb training abilities – to ensure the know-how is transferred effectively and quickly to you in a practical hands-on way
3. Listening skills – they listen carefully to the needs of the participants and want to ensure that you benefit from the experience

Each and every instructor is evaluated by the delegates and we assess the presentation after each class to ensure that the instructor stays on track in presenting outstanding courses.

#### HANDS-ON APPROACH TO TRAINING

All IDC Technologies workshops include practical, hands-on sessions where the delegates are given the opportunity to apply in practice the theory they have learnt.

#### REFERENCE MATERIALS

A fully illustrated workshop book with hundreds of pages of tables, charts, figures and handy hints, plus considerable reference material is provided FREE of charge to each delegate.

#### ACCREDITATION AND CONTINUING EDUCATION

Satisfactory completion of all IDC workshops satisfies the requirements of the International Association for Continuing Education and Training for the award of 1.4 Continuing Education Units.

IDC workshops also satisfy criteria for Continuing Professional Development according to the requirements of the Institution of Electrical Engineers and Institution of Measurement and Control in the UK, Institution of Engineers in Australia, Institution of Engineers New Zealand, and others.

#### CERTIFICATE OF ATTENDANCE

Each delegate receives a Certificate of Attendance documenting their experience.

#### 100% MONEY BACK GUARANTEE

IDC Technologies' engineers have put considerable time and experience into ensuring that you gain maximum value from each workshop. If by lunch time of the first day you decide that the workshop is not appropriate for your requirements, please let us know so that we can arrange a 100% refund of your fee.

#### ONSITE WORKSHOPS

All IDC Technologies Training Workshops are available on an on-site basis, presented at the venue of your choice, saving delegates travel time and expenses, thus providing your company with even greater savings.

#### OFFICE LOCATIONS

AUSTRALIA • CANADA • IRELAND • NEW ZEALAND • SINGAPORE • SOUTH AFRICA • UNITED KINGDOM • UNITED STATES

**idc@idc-online.com • www.idc-online.com**

### Visit our Website for FREE Pocket Guides

IDC Technologies produce a set of 4 Pocket Guides used by thousands of engineers and technicians worldwide.



- Vol. 1 - **ELECTRONICS**
- Vol. 2 - **ELECTRICAL**
- Vol. 3 - **COMMUNICATIONS**
- Vol. 4 - **INSTRUMENTATION**



To download a **FREE copy** of these internationally best selling pocket guides go to:  
**[www.idc-online.com/freedownload/](http://www.idc-online.com/freedownload/)**