Module 13 – Input/output techniques

Overview

Microcomputer architecture (concept map)

Peripheral communications (concept map)

Objectives

At the completion of this module you will be able to:

- explain the concepts involved in interfacing a microprocessor to a peripheral device
- describe how the keypad and display are interfaced to a typical microprocessor
- demonstrate an understanding of programming techniques used to drive a typical peripheral device
- explain the principles used to achieve analog-to-digital and digital-to-analog conversion
- describe Direct Memory Access (DMA) and explain its use in CPU operation
- demonstrate a knowledge of interrupts and show how they may be detected, prioritised, initiated and handled by a CPU.

13.1 Introduction

All modern microcomputers are ‘bus structured’ systems. They basically have three buses and these are:

- a unidirectional Address Bus (16 or more lines)
- a bidirectional Data Bus where the number of lines equals the word size of the computer, e.g. 8 lines. This bus is used for all data transfer ‘in’ and ‘out’ of the CPU.
- a control bus, which consists of a number of control lines. These will include clock signals, control signals from the CPU to devices connected to the bus, e.g. Read/Write, request and acknowledge lines which are inputs to the CPU and are activated by devices connected to the buses.

To design a microprocessor system, the elements that comprise the system are all interfaced to the microprocessor using the address, data and control buses.

The structure of the buses supplied by the various makes of microprocessors is non-standard. In the past, a limiting feature with microprocessor design was the constraints provided by a standard 40 pin dual-in-line package. Today, ICs are being produced in 40 plus pin packages, but still there are limitations. Manufacturer must decide how best to utilise the limited number of pins. In many cases I/O pins serve a dual purpose, e.g. input/output (software configured) on the same pin.

A basic block diagram showing the bus connections of a typical microcomputer system is provided in figure 13–1.

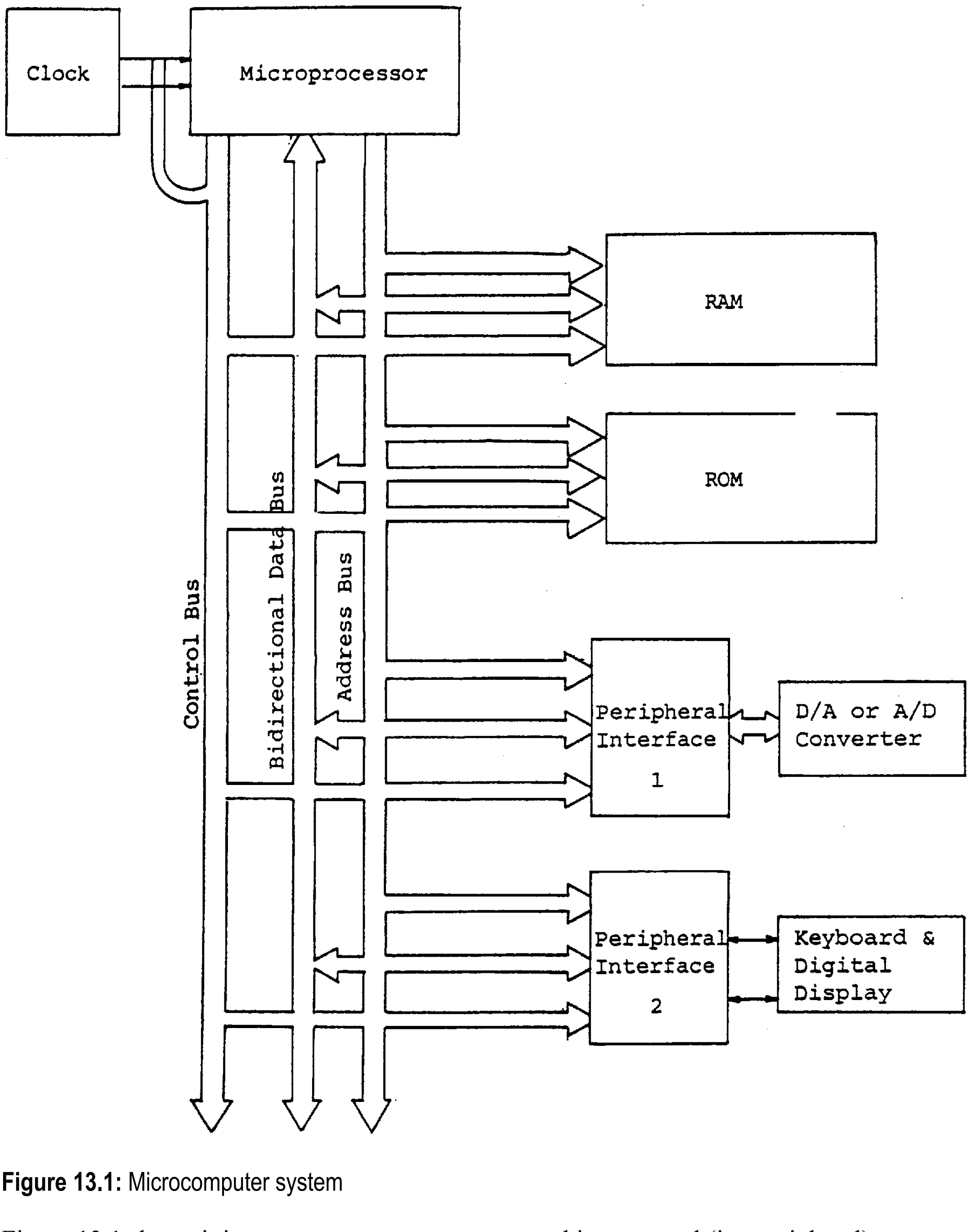
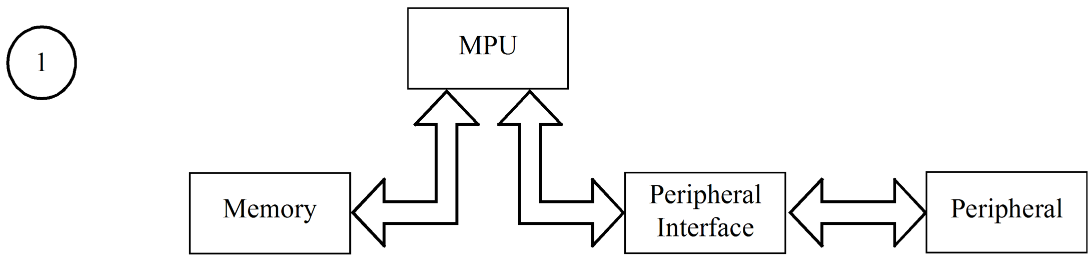


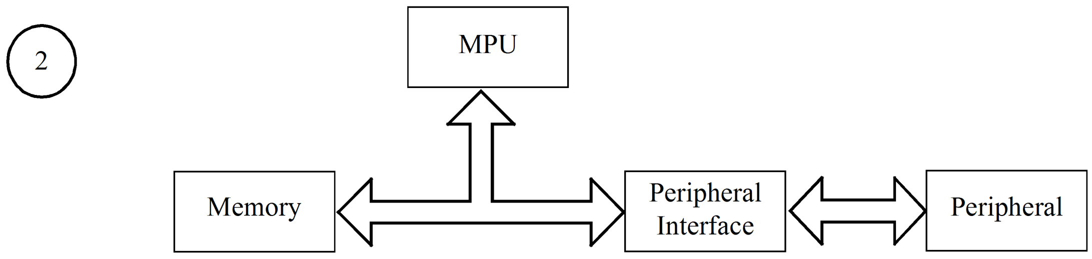
Figure 13.1: Microcomputer system

Figure 13.1 shows it is most common to connect anything external (i.e. peripheral) to a microprocessor (MPU) through an interface device. This is done so that the transfer of data to/from the peripheral can be program controlled.

This is achieved classically in two ways:



All information is controlled via the MPU, or



In some situations it is possible to have the peripheral directly access the memory to speed up certain operations. This is known as Direct Memory Access (DMA) and will be discussed later.

In both cases there is a need for an I/O interface between the peripheral device and the computer bus. The device takes a variety of forms, one example being the Motorola 'Peripheral Interface Adaptor (PIA)'. This device is separate to the MPU.

Motorola used the PIA approach for the early MC6800 series of microprocessors. More recently, to reduce circuit size and complexity, Motorola developed the MC68HC11 series of micro-controllers, which are totally integrated. The micro-controller includes, within the one integrated circuit package a microprocessor, RAM and ROM memory, peripheral I/O hardware and a timer system.

Before discussing the peripheral interface in detail it is important to understand how device registers outside the MPU (e.g. PIA registers) are accessed.

13.2 I/O addressing

So far, the term 'register' has been used to refer to a data storage unit within the MPU. These MPU registers are accessed (addressed) via opcodes, e.g. LDAA #$04. However, many more registers exist within a microcomputer system to provide; RAM and ROM; I/O and Timer functions. These registers cannot be accessed directly via opcodes. Instead, they must be accessed by a unique address placed on the address bus. Data can then be written to or read from that address when the MPU issues a read/write request. As these registers store data, they are called the systems 'memory'.

e.g. LDAA $0004. This instruction loads the contents of memory address $0004 into the MPU accumulator A register.

Blocks of memory locations are often reserved for specific purposes such as; ROM and RAM; I/O control and Data; etc. Once assigned to a specific purpose those memory locations cannot be used for any other purpose, e.g. locations $0000 to $003F are reserved for the ‘68HC11’s I/O registers.

Motorola microprocessors communicate with I/O hardware as if it were memory. That is, the I/O hardware occupies one or more locations within the systems memory, as shown above. This I/O interface technique is called ‘memory mapped I/O’. The same instructions used to transfer data to and from memory are used for I/O communications, e.g. LDAA $0000 and STAA $0001.

Memory map example

Table 13.1: EZ-Micro memory map

EZ-Micro MC68HC11D0 development board (expanded mode)

Memory type	Address	Label	Description	
I/O Registers	0000	PRA	Reserved for keypad	Total available memory space in this processor
	|			
	0003	PRC	Reserved for address/data	
	0004	PRB	Reserved for address bus	
	0006	DDRB	Reserved	
	0007	DDRC	Reserved	
	0008	PRD	Port D data	
	0009	DDRD	Port D direction	
	|		(PD0 & PD1 reserved)	
	003F			
RAM	0900		***Use for program data***	
	|			
	0FFF			
	1000		***Use for program code***	
	|			
	1FFF			
Display I/O	2000	LED1	***7 Segment displays***	
	3000	LED2		
	4000	LED3		
ROM	5000		Reserved	
	|			
	7FFF			
ROM	8000			
	|			
	9FFF			
RAM	F000			
	|			
	FFF2	IRQ	IRQ pin interrupt vector	
	FFFE	RESET	Reset button vector	
	FFFF			

13.2.1 Address bus decoding

Figure 13–1 showed that a single microprocessor is able to communicate with many devices connected to its bus system. To select a device to communicate with, the microprocessor outputs the address of that device on the address bus. Through address bus decoding the device is enabled, allowing data exchange with the microprocessor.

The address bus may be partially or fully decoded depending on the application. Partial decoding is the preferred method as it costs less to implement. On the down side, partial decoding is wasteful of memory address space as one device register may occupy several address locations. To illustrate this point refer to the following diagram.

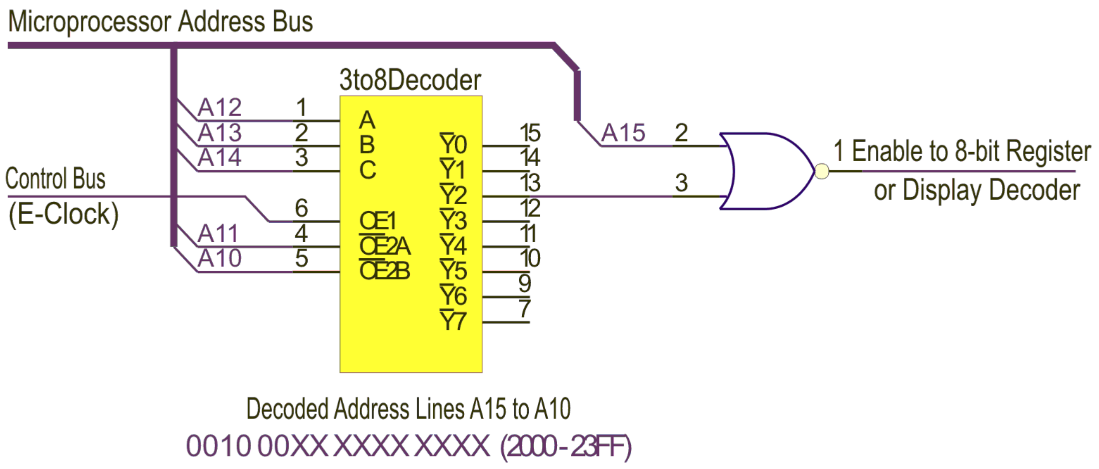

Microprocessor Address Bus
3to8Decoder
A12
A13
A14
1
2
3
A
B
C
Control Bus
(E-Clock)
A11
A10
6
4
5
OE1
OE2A
OE2B
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
15
14
13
12
11
10
9
7
A15
2
3
1 Enable to 8-bit Register or Display Decoder
Decoded Address Lines A15 to A10
0010 00XX XXXX XXXX (2000-23FF)

Figure 13.2: Partial address bus decoding

The 8-bit register, of figure 13-2, is enabled by addressing any location between 2000_{16} and $23FF_{16}$ ($0010\ 00XX\ XXXX\ XXXX_2$). That is, all 1023_{10} memory address locations are valid for this 8-bit register. Address lines A12, A13 and A14 are decoded to select one of the decoder's 8 output lines (in this case Y2). A10 and A11, together with the control bus (E) are used to enable the decoder chip. An active low (0) will appear on output line Y2 when:

A12 (A)	= 0
A13 (B)	= 1
A14 (C)	= 0
A11 (OE2A)	= 0
A10 (OE2B)	= 0
E (OE1)	= 1

Note, address lines marked X (don't care) are not decoded, therefore may be a 1 or 0.

The decoder's other seven outputs could be used to enable other 8-bit registers or peripheral devices requiring an enable from the microprocessor. For example, Y5 could be selected by placing 5000_{16} to $53FF_{16}$ ($0101\ 00XX\ XXXX\ XXXX_2$) on the address bus.

Activity 13.1

Refer to figure 13–2. Given the following logic levels:

A10 (OE2B)	= 0
A11 (OE2A)	= 0
A12 (A)	= 0
A13 (B)	= 1
A14 (C)	= 1
A15	= 0

Determine the output enabled and the range of addresses decoded.

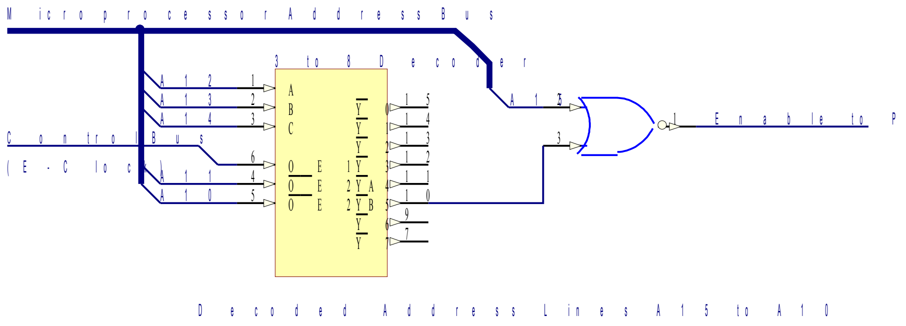
Note - Answer to activity 13.1

Y6, 6000_{16} to $63FF_{16}$

(0110 00XX XXXX XXXX)

Self assessment 13.1

1. Describe and illustrate the concept of 'memory mapped I/O'.
2. Explain what is meant by 'Address Bus Decoding'. Include in your answer the advantages and disadvantages of 'partial' decoding.
3. Explain the operation of the following address decoder circuit and determine the decoded address range to enable the Peripheral Device (logic 1).



13.3 The Motorola MC68HC11

As previously mentioned, The MC68HC11 family (16 members based on pin configuration, amount and type of onboard memory) of micro-controllers has an internal structure that includes I/O hardware. This means that all buses connecting the microprocessor to the PIAs are internal and, in single-chip mode, all I/O registers occupy fixed memory addresses.

Any future reference to the MC68HC11 or ‘HC11 will stand for the MC68HC11D3 version of this micro-controller family.

The ‘HC11 has two modes of operation, ‘single-chip’ and ‘expanded mode’. In single-chip mode the micro-controller uses only the resources contained within the chip to perform its function. In expanded mode the internal address, data and control buses are brought out to pins. In this way the micro-controller’s functionality is greatly enhanced, albeit at the expense of two I/O ports used for address and data bus purposes.

The following block diagram depicts the internal organization of the ‘HC11. This particular family member has 4 user configurable I/O ports; A, B, C and D.

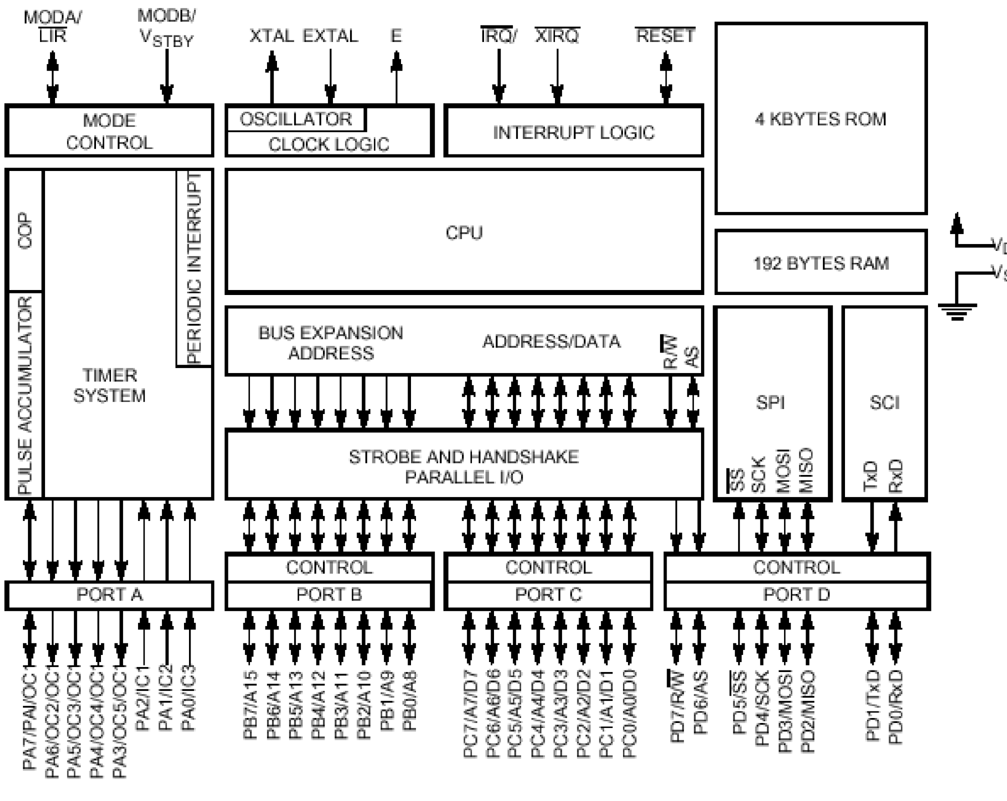


Figure 13.3: MC68HC11D3 block diagram

Port A (address $0000) is a multi-purpose port sharing functions that include general purpose I/O, the main timer system, and the pulse accumulator system. Three pins are fixed input (PA0-PA2), three are fixed output (PA4-PA6) and two are bi-directional (PA3 and PA7).

Port B (address $0004) is a general purpose I/O port, but can also be used as a high order address bus in expanded mode. Each I/O pin associated with port B can be configured as either input or output.

DDRB (address $0006) As with the MC6821, configuring the I/O pins is done by setting bits in the data direction register B (DDRB). To configure a pin as an input, the corresponding bit in the DDRB is set to 0. To configure a pin as an output, the corresponding bit in the DDRB is set to 1.

Port C (address $0003) is also a general purpose I/O port, but in the expanded mode can be used as a multiplexed low order address and data bus.

DDRC (address $0007) As with Port B, the I/O pins of Port C are configured by setting the bits in the DDRC.

Port D (address $0008) is also a general purpose I/O port, but in the expanded mode the pins take on a variety of control and data transfer roles, which will be explained in more detail later.

DDRD (address $0009) As with other ports, the I/O pins of port D are configured by setting the bits in the DDRD.

13.4 MC68HC11 parallel I/O

As previously mentioned, the 'HC11 can be operated in 'single-chip' or 'expanded' modes of operation. Mode selection is normally hard wired by applying appropriate logic levels to pins MODA and MODB during reset.

13.4.1 Single-chip mode

Single-chip mode is the simplest configuration for applications requiring minimal program memory and maximum general-purpose parallel I/O availability. The MPU relies only on the internal resources of the chip and requires only a small number of external components for proper operation.

In this mode the address, data and control buses are contained within the chip as is; RAM and ROM; and I/O and Timer controls. User configured address decoding is not possible, so all memory addressed devices have fixed addresses, e.g. Port B data register occupies address $0004.

Handshaking pins are provided to control data transfers via the I/O ports. Data is latched in on the rising or falling edge of the signal on pin STRA (PD6/AS). The active edge of this signal is software selectable. STRB (PD7/R/W) to indicate to the MPU that valid data is available on a port configured for output.

Note – Pin STRA (PD6/AS) is an input in single-chip mode and an output in expanded mode. STRB (PD7/R/W) is an output in both modes.

13.4.2 Expanded mode

In expanded mode the address, data and control buses are brought out via the I/O ports. Ports B and C are used for address and data lines. This is made possible by assigning the high byte of the address to port B and the low byte to port C. The low byte of port C is time multiplexed with the data bus. Pins PD6/AS and PD7/R/W control the flow of information (address or data) through port C.

13.4.3 Programming example 1 (based on the EZ-Micro development board)

In this example the ‘HC11 will output data to a seven segment LED display. It will be assumed that the ‘HC11 is operating in the **expanded** mode and the seven segment display will occupy memory address $2000 to $23FF. Therefore, storing the appropriate data to address $2000 will cause the corresponding segments of the display to light.

The manufacturers data on the seven segment display advise that a logic ‘0’ applied to any of the segments ‘a’ to ‘g’ will cause the segment to light, assuming the display’s common anode is a logic ‘1’. A BCD to 7-segment display decoder will be used to drive the display. The input to the display decoder will be connected to the ‘HC11 data bus and each decoder output will drive one segment of the display. So placing the required number on the data bus will cause the appropriate segments to light when the display decoder is enabled.

To enable the display decoder the address of the display must appear on the address bus. The address is then decoded, as shown in figure 13.2. The address decoder output is then used to enable the display decoder. The segments will light briefly, in fact, to briefly to be seen. For this reason the display must be repeatedly written to. In this way our brain ‘sees’ the display as being continuously lit.

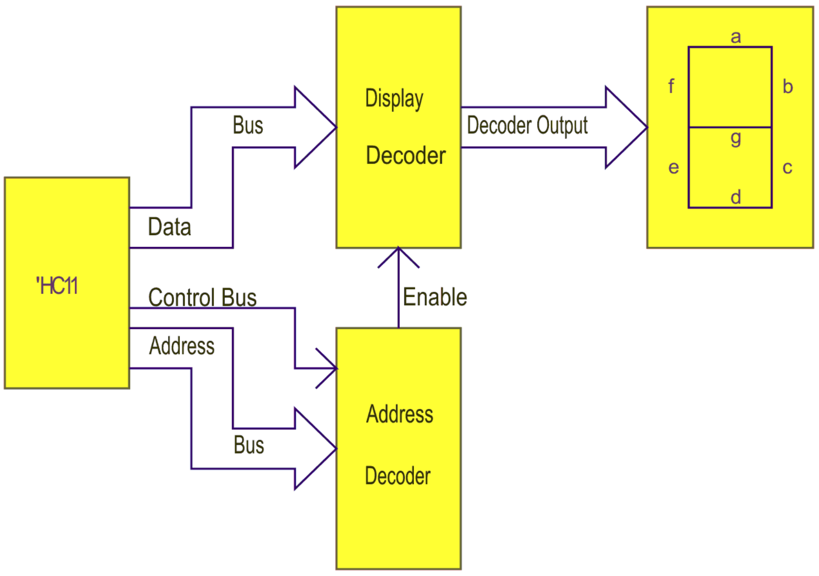
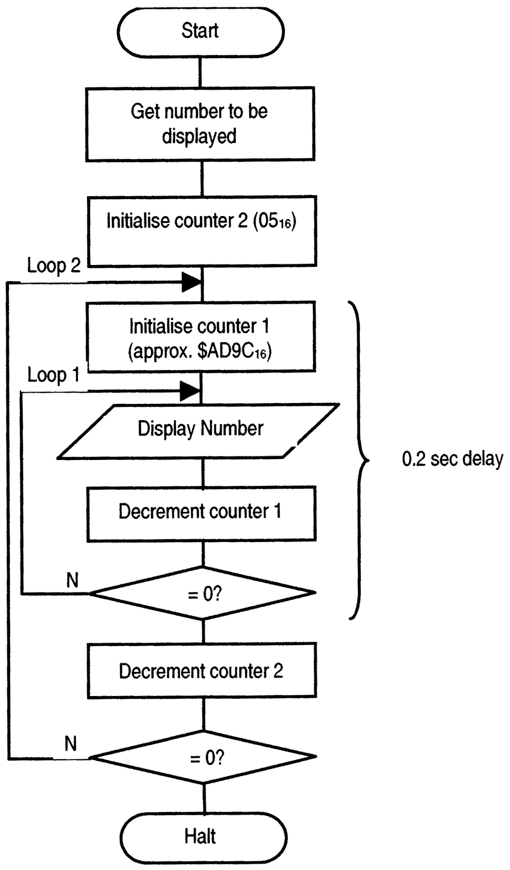


Figure 13.4: MC68HC11D3 block diagram

Suppose we now wish to program the system to display the decimal number 3 on the display for a period of 1 second and then turn it off.

The flow chart is:


Start
Get number to be displayed
Initialise counter 2 (05_{16})
Loop 2
Initialise counter 1 (approx. $AD9C$_{16}$)
Loop 1
Display Number
Decrement counter 1
N
= 0?
0.2 sec delay
Decrement counter 2
N
= 0?
Halt

The memory map is:

EZ-Micro MC68HC11D0 development board (expanded mode)

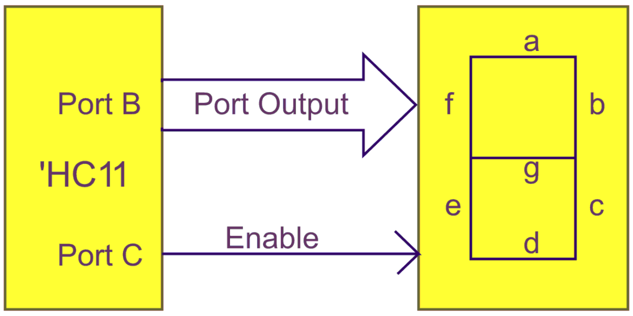
Memory type	Address	Label	Description
RAM	0900		***Use for program data***
	|		
	0FFF		
			Use for program code
	1000	START	Program start
	|		
	1010	HALT	Program end
Display I/O	2000	LED1	***7 segment displays***
	3000	LED2	
	4000	LED3	
RAM	FFFE	RESET	Reset button vector
	FFFF		

The program listing for the EZ-Micro development board is:

Address	Opcode	Label	Mnemonic	Operand	Comment
2000		LED1	EQU	$2000	Address of display
1000			ORG	$1000	Program start
1000	86 03	START	LDAA	#03	Number to be displayed
1002	C6 05		LDAB	#05	Counter for loop 5
1004	CE AD 9C	LOOP2	LDX	#$AD9C	Counter for 0.2sec delay
1007	B7 20 00	LOOP1	STAA	LED1	Display number
100A	09		DEX		Decrement counter 1
100B	26 FA		BNE	LOOP1	Refresh display
100D	5A		DECB		Decrement counter 2
100E	26 F4		BNE	LOOP2	Display (0.2 x 5 sec)
1010	7E 10 10	HALT	JMP	HALT	

13.4.4 Programming example 2 (based on the THRSimll software)

In this example the ‘HC11 will again output data to a seven segment LED display. However, in this example the ‘HC11 will be operating in **single-chip** mode. In this mode the ‘HC11 is connected directly to the LED display, i.e. the external display decoder and the address decoder are not used. This mode of operation reduces circuit complexity, however, it requires a little more program code to setup the output ports to achieve the same result.



Port B (address $0004) will be used to drive the display segments, so the I/O pins associated with port B will need to be set for output. This is done by setting the all bits of DDRB ($0006) to a logic 1 ($FF). Each output pin of port B will then be used to drive one segment of the LED display.

Port C (address $0003) will be used to enable (illuminate) the display, so one I/O pin associated with port C will need to be set for output. This is done by setting the bit 0 of DDRC ($0007) to a logic 1 ($01).

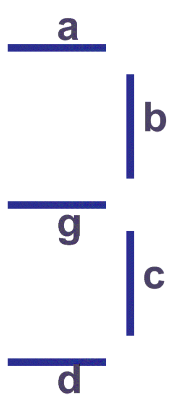
To run this example on the THRSimll software, port addresses will need to be moved to page $1000. To do this, change the four port addresses in the program listing, e.g. $07 becomes $1007 and so on. When you assemble the program the address modes will change from direct to extended, e.g. STAA DDRB (97 06) becomes (B7 10 06)

Note: each register has a unique address, therefore it is not necessary to write to bit 2 of either control register.

A logic 1 on a Port B output line will cause that segment to illuminate when the LED display is enabled by a logic 0 on Port C pin, PC0. For this reason it will be necessary to determine the bit pattern, to be output on port B, that will cause the required segments to illuminate.

Suppose we wish to now program the system to display the decimal number three on the display for a period of 1 second (approx) and then turn it off again. Unlike the previous example the display will remain illuminated until it is turned off.

The bit pattern for decimal three will be segments a, b, c, d and g illuminated as shown:



These segments correspond to:

PB0, PB1, PB2, PB3 and PB6 set to logic 1, and the other lines PB4, PB5 set to logic 0.

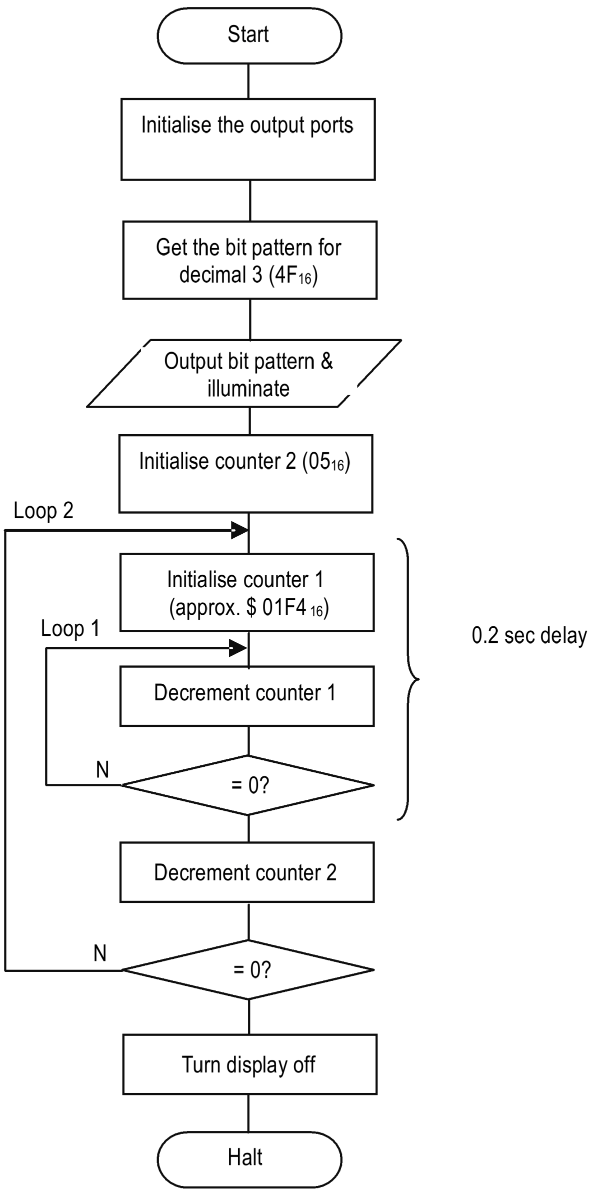
The peripheral register B word would be:

	7	6	5	4	3	2	1	0	
PRB	0	1	0	0	1	1	1	1	$4F
		g	f	e	d	c	b	a	

To illuminate the display we require a logic 0 on the enable (E) line of the 7 segment display. This means a logic 0 on PC0. The peripheral register C word will therefore be:

	7	6	5	4	3	2	1	0	
PRC	0	0	0	0	0	0	0	0	$00

The flowchart is:


Start
Initialise the output ports
Get the bit pattern for decimal 3 ($4F_{16}$)
Output bit pattern & illuminate
Initialise counter 2 (05_{16})
Loop 2
Initialise counter 1 (approx. \$ $01F4_{16}$)
Loop 1
Decrement counter 1
N
= 0?
0.2 sec delay
Decrement counter 2
N
= 0?
Turn display off
Halt

The memory map is:

THRSim11 – 68HC11 simulator

Memory type	Address	Labels	Description
I/O Registers	1003	PRC	Port C data
	1004	PRB	Port B data
	1006	DDRB	Port B direction
	1007	DDRC	Port C direction
ROM (Program)			***Use for program code***
	E000	START	Program start
	|		
	E020	HALT	Program end
	FFFE	RESET	Reset button vector
	FFFF		

The program listing below uses the peripheral register addresses for the actual 68HC11 device, not the simulator. To simulate the program, add $1000 to the peripheral register addresses, eg. PRB = $1004):

Address	**Opcode**	**Label**	**Mnemonic**	**Operand**	**Comment**
		PRB	EQU	$04	Peripheral register B
		DDRB	EQU	$06	Data direction register B
		PRC	EQU	$03	Peripheral register C
		DDRC	EQU	$07	Data direction register C
			ORG	$E000	Program start
E000	86 FF	START	LDAA	#$FF	All port B line output
E002	97 06		STAA	DDRB	Set port B lines
E004	86 01		LDAA	#$01	PCO line output
E006	97 07		STAA	DDRC	Set port C line
E008	86 4F		LDAA	#$4F	Get display word
E00A	97 04		STAA	PRB	Output display word
E00C	86 00		LDAA	#$00	Get enable word
E00E	97 03		STAA	PRC	Enable the display
E010	C6 05		LDAB	#$05	Counter for loop 2
E012	CE 01 F4	LOOP2	LDX	#$1F4	Counter for 0.2sec delay
E015	09	LOOP1	DEX		Decrement counter 1
E016	01		NOP		No operation, to
E017	01		NOP		increase loop time
E018	26 FB		BNE	LOOP1	Refresh display
E01A	5A		DECB		Decrement counter 2
E01B	26 F5		BNE	LOOP2	Display (0.2 X 5 sec)
E01D	86 01		LDAA	#$01	Get disable word
E01F	97 03		STAA	PRC	Disable the display
E021	7E E0 21	HALT	JMP	HALT	

13.4.5 Programming example 3

In this example the ‘HC11 receives input from a keypad. Again, it will be assumed that the ‘HC11 is operating in the single-chip mode.

The keypad is comprised of 12 keys arranged in 3 columns by 4 rows. Each key forms a single-pole single-throw switch, as shown in figure 3-5. The 4 rows are connected to the ‘HC11 output lines and are normally set to logic ‘1’; the three columns are connected to input lines and are also normally tied to logic ‘1’ via a pull up resister.

Since the keypad is being used as an input device, a program must be written to scan the keypad to determine, which key has been pressed. Scanning the keypad may take the following form.

- Output logic '0' to Row 1
- Read Column 1; if '0' then key 1 pressed
- Read Column 2; if '0' then key 2 pressed
- Read Column 3; if '0' then key 3 pressed
- Output logic '0' to Row 2
- Read Column 1; if '0' then key 4 pressed
- Read Column 2; if '0' then key 5 pressed
- etc.

This process continues until all rows and columns have been read then the process repeats, continuously scanning for a pressed key.

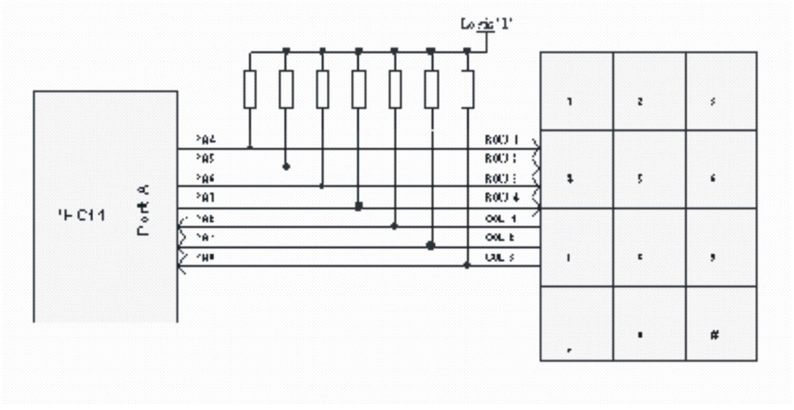


Figure 13.5: Keypad interface

The software required to scan the keypad and output the value of the key pressed to the display is beyond the scope of this course.

Activity 13.2

a. Refer to the ELE1301 course page on 'Study Desk' and complete the following experiments:

   i. Home experiment 13-1 - Parallel I/O program 1 – Time delays

   ii. Home experiment 13-2 - Parallel I/O program 2 – LED displays

   iii. Home experiment 13-2 - Parallel I/O program 3 – Subroutines.

Self assessment 13.2

1. The Motorola 68HC11 has two modes of operation, i.e. single-chip and expanded. Explain why user configured address decoding is not possible in single-chip mode, whereas it is possible in expanded mode.
2. Referring to figure 13-3, a microprocessor application requires 32Kbytes of ROM, which mode of operation is best suited to this application and why?
3. Sketch the basic block diagram of a 7 segment display interfaced to the 68HC11 in the single-chip mode.
4. Assuming the display configuration drawn in part (3) above and a cycle time of 40 microseconds, write a program, for the 68HC11, to activate the decimal number 5 for a period of 10sec. Start the program at $E000. Include in your answer a flow chart, memory map, assembly code, and the machine code for the program.
5. A basic microprocessor system uses a clock frequency of 1MHz (1 usec cycle time). Describe, with the aid of a flow chart, how a delay of approximately one second could be achieved by decrementing a predetermined decimal number to zero. Show all your calculations.

13.5 Digital and analog signals

Most microprocessor based systems are used to control external devices or processes. They transmit data to or receive data from these devices. Many of these devices involve either mechanical or electrical transducers but they all may either require an analog voltage to operate, such as a motor, or generate a voltage depending upon some form of motion such as a position sensor.

In these cases the signals to/from the peripheral devices must be converted to either a digital or analog form.

So far we have concentrated on digital data where the data is defined by specific values, e.g. a logic 0 may be represented by 0 volts and a logic 1 by 5 volts. Analog data, on the other hand, is continuous in nature and may be any value at any time, e.g. temperature varies continuously with time, so all values of temperature within a particular time period are valid analog values.

A selection of integrated circuit devices is available for designers to incorporate in their systems. These devices are known as Analog-to-Digital (A/D) converters or Digital-to-Analog (D/A) converters. They vary according to their design, which typically includes parameters such as operating speed, number of bits used and resolution.

13.5.1 Digital to analog conversion

The basic digital to analog converter uses principles associated with an electronic operational amplifier as shown below.

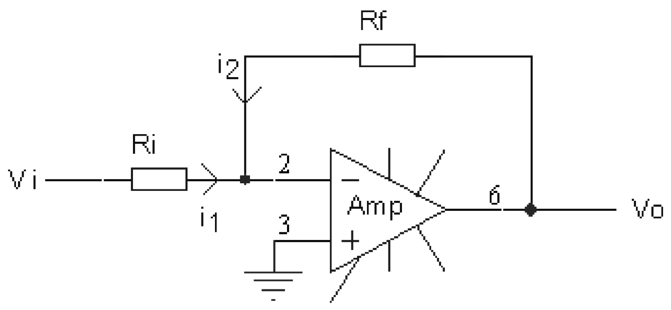


Figure 13.6: Inverting amplifier

Using Kirchoffs laws and summing the currents at the amplifier (A) input, we can derive an expression for the output voltage Vo in terms of the input voltage Vi; the feedback resistance Rf and the input resistance Ri.

Now $i_1 = i_2$

$$\therefore \quad \frac{Vi}{Ri} = \frac{Vo}{Rf}$$

$$\text{i.e.} \quad Vo = \frac{Rf}{Ri}\,Vi$$

$$\text{i.e.} \quad Vo = G \quad Vi \text{ where } G = \text{gain, given by } \frac{Rf}{Ri}$$

Consider an amplifier with Rf = 100kΩ, Ri = 10kΩ and an input voltage of 1 Volt. We have:

$$Vo = \frac{100}{10} ⫲ 10\,\text{Volts}$$

This principle can be extended to a summing amplifier circuit connected to electronic switches shown as flip flops, FF1 to FF4 in the following diagram.

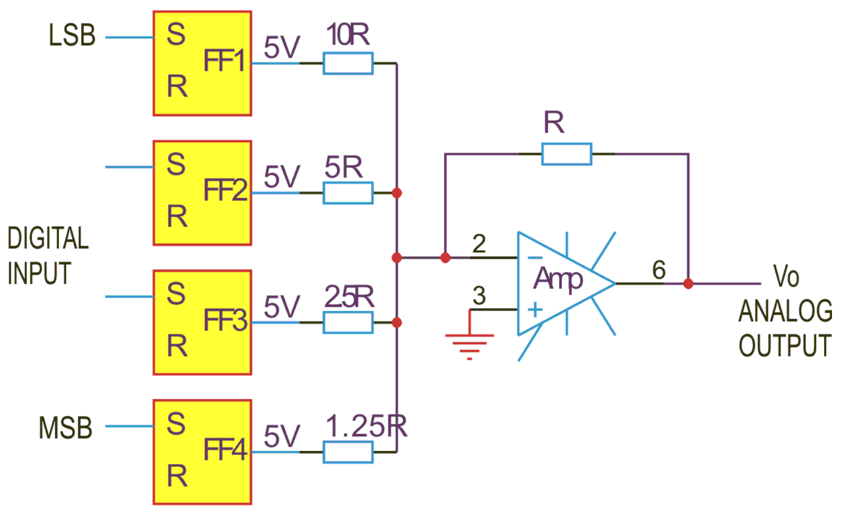


Figure 13.7: Basic digital to analog converter (summing amplifier)

In the circuit shown initially all flip flops are reset and Vo = 0Volts.

If now FF1 is set, its 5Volt output is applied to the amplifier and $Vo = \frac{R}{10R}\ 5 = 0.5$ Volts

If now FF1 is reset and FF4 is activated then $Vo = \frac{R}{1.25R}\ 5 = 4$ Volts

It can be seen that FF1 input is the least significant bit and FF4 is the most significant bit. The binary word 1101 applied to FF's 1 to 4 gives

Vo = 4 + 2 + 0 + 0.5 = 6.5 volts.

Thus a digital word (1101) produces an analog voltage (5.5). This type of D/A converter is known as the weighted resistor network system. Other types of D/A converters are available such as resistor ladder network systems and multiplying D/A converters.

The multiplying D/A converter has a variable reference voltage applied to it and it produces an output that is the product of the digital input value times the reference voltage. The details of these converters are beyond the scope of this course.

Microprocessors are often interfaced to D/A converters and special integrated circuits with addressable input buffers have been developed.

Consider the system shown:

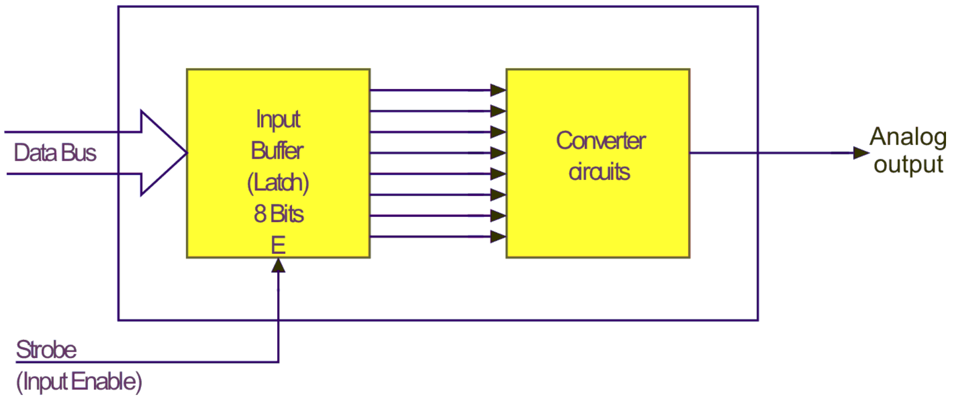


Figure 13.8: D/A converter

A D/A converter requires a binary word at its input to remain present and stable as long as the analog output voltage is required. This process would require the data bus to be held constant for long periods which would prevent the CPU from executing other tasks. For this reason the D/A unit normally has a temporary store register (latch) to accept the data bus word and maintain it until cleared by some other means. This immediately releases the data bus for other tasks.

From the foregoing it is clear the resolution of output voltages from a D/A converter depends upon the number of binary bits used to drive it. Many systems only require 8 bits however 10, 12, 16 or 32 bits could be used. In these cases as the D/A requires more bits than the data bus can provide in one byte, special techniques can be used to achieve the correct input word for conversion.

One technique uses two latches, which are sequentially filled before conversion is allowed to commence.

Consider for instance a 10 bit D/A converter to be interfaced to an 8 bit CPU data bus.

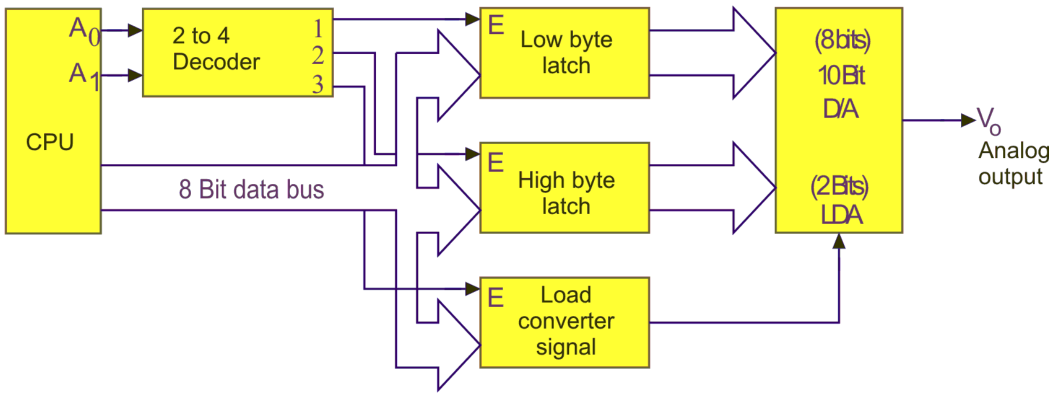


Figure 13.9: 10 Bit D/A converter

In this system, the CPU sends the first 8 bits of the 10 bit word from a memory location on to the data bus. It goes to all three latches simultaneously. An address word is then used which when decoded by the 2 to 4 decoder, enables via the E line, the low byte latch.

The CPU then sends the next 2 bits of the 10 bit word to the high byte latch enabling it as before with another address. This now means that a 10 bit binary word is sitting at the D/A input ready to be converted. The CPU now addresses the third latch which provides a load D/A converter (LDA) signal and all 10 bits are converted to an analog voltage output simultaneously.

13.5.2 Analog to digital conversion

An analog to digital converter (A/D) accepts an analog signal such as a voltage and when commanded, converts it into a single binary word.

Typically there are three types of analog to digital converters. These are:

- successive approximation
- integration
- direct comparison.

13.5.3 Successive approximation conversion

This type of conversion uses a D/A converter to compare its output voltage with the unknown voltage to be converted to a binary word. The hardware for such a system is shown in the diagram.

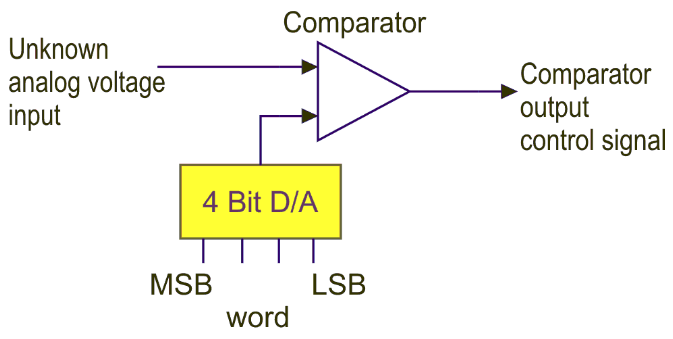

Comparator
Unknown analog voltage input
Comparator output control signal
4 Bit D/A
MSB
LSB
word

Figure 13.10: Successive approximation converter

In other words we ‘guess’ a binary word and compare the D/A output with the unknown voltage input. We then proceed to refine our guess as many times as is necessary until the two analog voltages are as equal as possible depending upon the number of binary bits available.

Consider the operation of the system. Suppose the comparator output is a logic 0 if the unknown voltage is greater than the D/A voltage and logic 1 if it is less than the D/A voltage.

The ‘guessing’ algorithm is to start with the most significant bit of the D/A on (i.e. word 1000) and examine the comparator output. If it is a logic 1, we are greater than the unknown so we turn the MSB off and turn the next bit on and try again. If on the other hand we are less than the unknown we leave that bit on and than activate the next bit on and compare etc. This process is continued until all bits have been tested. The binary word at the D/A input is the required word for that analog signal.

Consider an example. Suppose our D/A has outputs associated with the binary word as shown:

	MSB			LSB
Binary Word	1	1	1	1
Analog Voltages	.5V	.25V	.125V	.0625V

Suppose we have an unknown voltage as shown on the graph:

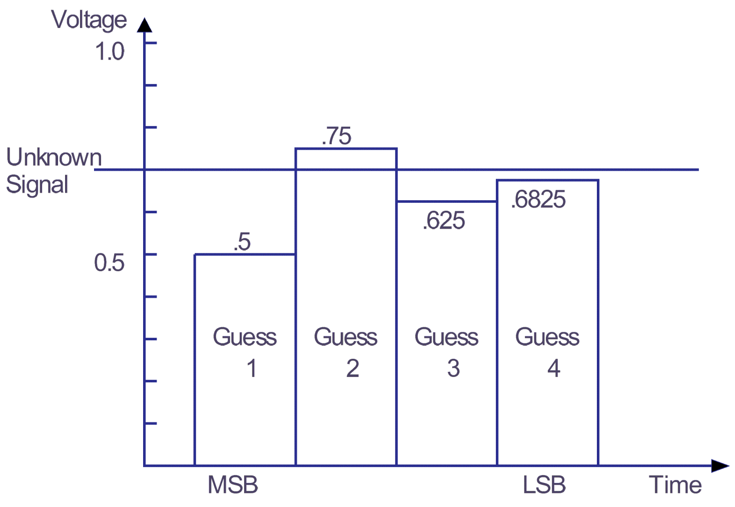


Figure 13.11: Voltage guesses

We start by turning on the MSB, it provides an output of 0.5 V which is below our unknown so the comparator output is a logic 1. We leave that bit on and activate the next most significant bit. This bit contributes to 0.25 V therefore our D/A output is now 0.5 + 0.25 = 0.75 V. This causes the comparator output to be a logic 0 indicating our guess is above the unknown. We turn the second bit off and turn on the third bit. This gives 0.5 + 0.125 = 0.625 V. This is below the unknown so we leave it on.

The last bit (LSB) is then turned on giving 0.5 + 0.125 + 0.0625 = 0.6875 V. This is as close as we can come to the unknown as we have used all of the 4 bits. Thus the required binary word corresponding to the unknown analog voltage is 1011. Obviously greater accuracy in conversion would arise from using 8 or 10 binary digits in the system.

Most members of the MC68HC11 family of micro-controllers contain this type of A/D converter. However, that member depicted in figure 13-5 has no A/D converter.

13.5.4 Integration conversion

In this category of A/D converters, two types are found in practice. These are the integrating A/D converter and the single/dual slope A/D converters.

13.5.5 Integrating A/D converters

This type of converter is slow in comparison to some other types. The principle is shown in the diagram.

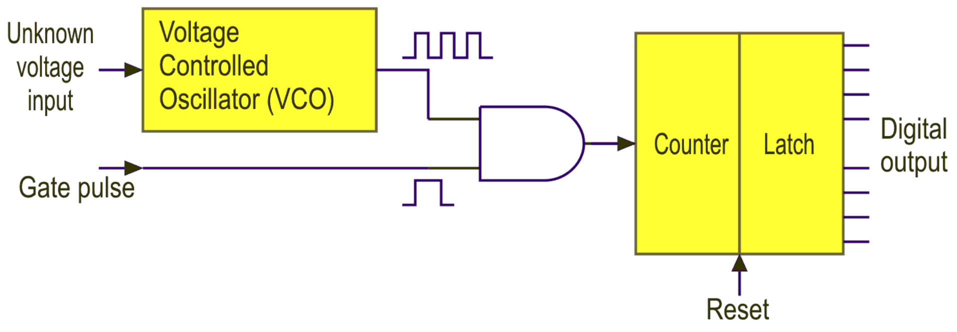


Figure 13.12: Integrated A/D converter

The voltage controlled oscillator (VCO) generates an output pulse train whose frequency is proportional to the unknown analog voltage input. Small input voltages produce a low frequency output and large input voltages produce higher frequency outputs from the VCO. These signals are gated to a counter enabled by a fixed duration timing or gate pulse.

The counter having been previously reset to zero will begin counting until the gate pulse inhibits it. The count is then transferred to the latch register and the digital output is maintained allowing the counter to be reset for the next conversion.

Waveforms associated with this system are shown below.

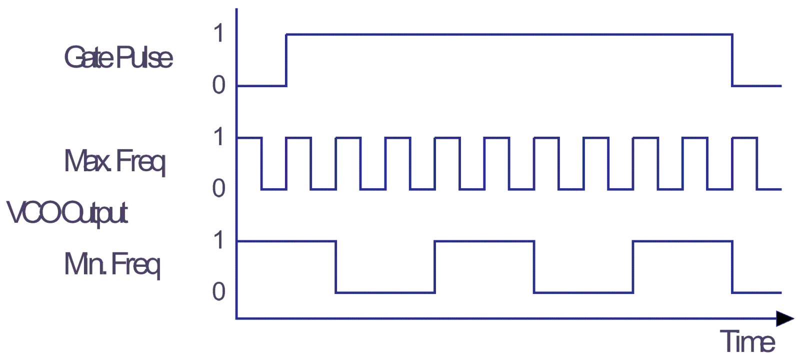


Figure 13.13: Timing waveforms

13.5.6 Single Slope A/D converter

This A/D converter utilises a saw-tooth waveform generator (integrator) and a digital counter with precise timing. In operation, the analog input voltage is compared to the ramping voltage of the integrator. When the analog input voltage is greater than the integrator output, the output of the comparator is a logic 0, allowing the integrator output to continue rising at a linear rate. During this time the counter is counting up at a frequency fixed by a precision clock. The higher the analog input voltage, the longer it takes the integrator to reach an equivalent voltage, hence a higher count. When the integrator reaches the same level as the analog input voltage the comparator output toggles to a logic 1. This causes the contents of the counter to be loaded into the shift register. It also causes the integrator to reset to 0 volts and subsequently the comparator output toggles to a logic 0 - clearing the counter and allowing the integrator to ramp up again as illustrated in the following diagram.

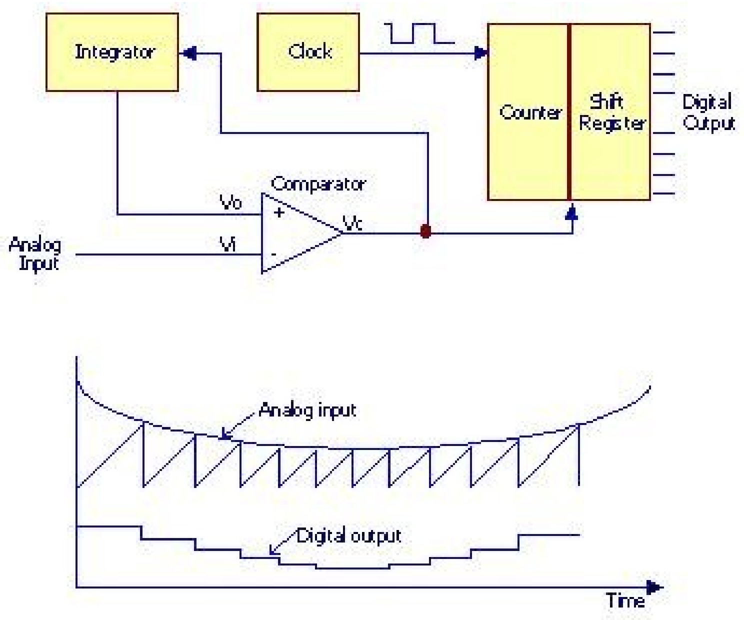


Figure 13.14: Single slope A/D converter

The accuracy of the single slope A/D converter depends on to independent circuits, ie. The integrator and the counter. Any variation between the two, which is inevitable over time, will cause inaccurate conversions. This is referred to as calibration drift. To overcome this deficiency the dual slope A/D converter can be utilised.

13.5.7 Dual slope A/D converter

In this system, a time comparison between a known reference voltage ($-V_{REF}$) and the unknown input voltage (Vu) is made. The block diagram of this dual-slope converter is shown.

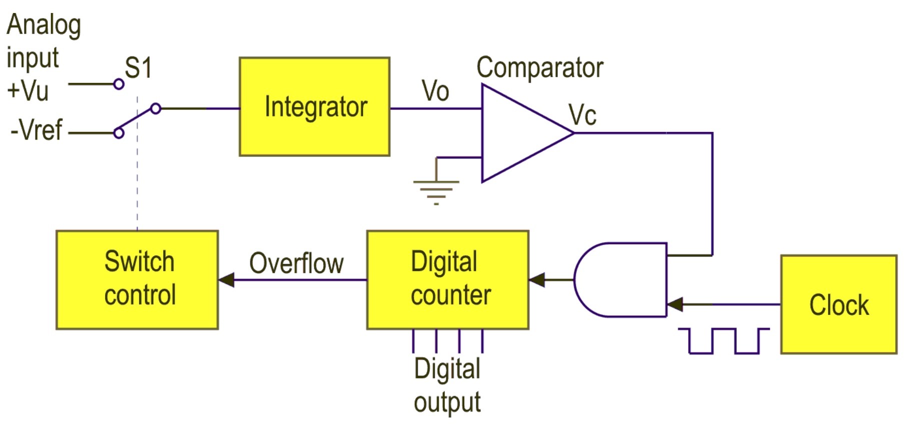


Figure 13.15: Dual slope A/D converter

In operation, the counter is initially reset to zero and S1 is connected to the unknown input voltage (Vu). The comparator output is Vc, a logic zero, thus inhibiting the AND gate.

Immediately Vu is applied to the integrator its output Vo begins to rise linearly and this causes the comparator output Vc to change to a logic one. The clock pulses (at a known frequency) are then transferred to the counter, which immediately begins counting. When the counter reaches its maximum state (i.e. 1111) it will reset itself to zero and provide an overflow signal, which changes S1 to the reference voltage (-Vref). This causes the integrator to ramp in the opposite direction until it reaches zero. When the comparator detects this its output returns to a logic 0 and the counter stops.

The waveforms associated with the system are shown below:

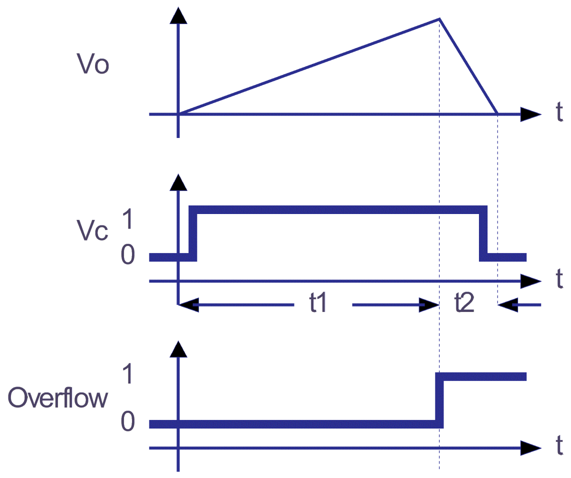


Figure 13.16: Converter waveforms

The counter output provides the required output digital word and is proportional to the unknown input voltage given by the ratio:

$$Vu = \frac{t2}{t1} Vref$$

13.5.8 Direct comparison conversion

This system is used where extreme speed is required to perform a conversion. The circuit uses several comparators each with their own reference voltage (VREF) within a given range of operation. Each comparator provides a logic output depending upon the input level compared with the reference.

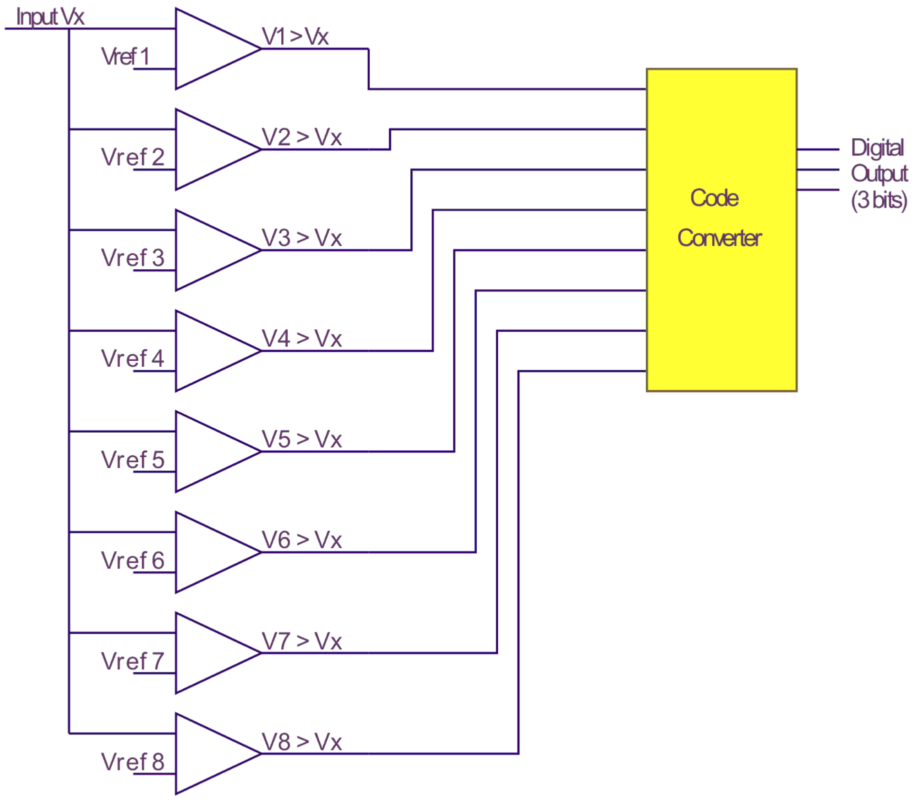


Figure 13.17: Direct comparison converter

An 8 to 3 encoder is used in this case to convert the comparators logic outputs to a 3 bit binary number. A 4 bit binary output would require 16 comparators etc. The need for many comparators, reference voltages and a complex encoder scheme make this method expensive to produce. It can however convert analog signals to digital word in nanoseconds. This system is sometimes known as a parallel or flash converter.

Self assessment 13.3

1. Explain with the aid of diagrams, the operation of a successive-approximation type of analogue-to-digital converter.
2. Explain, with the aid of a block diagram, how a 10 bit D/A converter could be interfaced to an 8 bit data bus of a microprocessor.
3. Explain with the aid of diagrams, the operation of a dual-slope type of A/D converter.

13.6 Direct memory access

Some microprocessors have specific input/output instructions, e.g. IN transfers data from the addressed device to the accumulator and vice-versa for OUT. The Motorola microprocessors have no input/output instructions at all. They treat peripheral devices as memory locations so any instruction transferring data to and from memory can achieve input/output. We have seen the use of these instructions (e.g. LOAD and STORE) in addressing ports. This type of transfer is called 'memory-mapped input/output'. It is ideal for slow speed devices and for those that only require single data transfers.

Many peripheral devices require transfer of 'blocks of data' to or from memory at high speed. Such devices include magnetic disks or tapes. If the peripheral could access the memory directly and not via the CPU then obviously the data transfer speed could be increased to the hardware speed alone. This procedure is known as Direct Memory Access (DMA). DMA is an operation in which the CPU hands over control of the buses to a special control unit known as the DMA controller.

The controller must specify the following information for a block transfer:

- The device address
- The starting address of the block in memory
- The number of data words to be transferred
- The direction of transfer (e.g. IN or OUT).

DMA controllers are complex and essentially another processor. In some cases the original CPU can even act as this processor. Most systems however use separate hardware and all must:

- Advise the CPU of the DMA request
- Control the buses so as not to interfere with the CPU. This is known as bus contention
- Complete the transfer
- Advise the CPU of the end of the transfer.

When a DMA request is received by the processor it usually halts its current operation and indicates to the DMA controller it is not using the buses and waits until control is returned to it.

One simple DMA technique is to use CPU cycles when the CPU is not accessing the memory. The DMA controller can then use the buses without informing the CPU. This is called 'cycle stealing'. A major problem of bus contention can occur under this technique. Motorola MC6800 series of microprocessors get over this problem by generating a special signal whenever the CPU is using the memory. This signal is known as Valid Memory Address (VMA).

Self assessment 13.4

1. State the purpose of DMA. Include in your answer the information that the DMA controller must supply to the CPU.
2. Explain what is meant by bus contention and how it is avoided.
3. State the purpose of Motorola’s VMA signal.
4. Explain the DMA term ‘cycle stealing’. Include in your answer a major problem associated with cycle stealing and how Motorola overcome this problem.

13.7 Interrupts

A major difference exists between most peripherals when they are connected to a CPU and that is ‘speed of operation’. In general most peripherals are very slow compared with a CPU. Signals between the CPU and the peripheral indicating when each other is ready to transfer data, are known as ‘Interrupts’.

An interrupt can also be a signal to a CPU to halt its normal operations and do something more important. These signals are used for alarm inputs, external control of processes or debugging aids in software development. Generally interrupts fall into three categories:

- internal
- external
- software

Internal interrupts, sometimes called error ‘traps’ are initiated by the illegal use of an instruction or code, register overflows or incorrect memory operations.

External interrupts are generated by signals from peripheral devices. They may be requesting data transfers or indicating a failure has occurred. A good example of this is a power failure.

In this case the interrupt may be used to initiate a special routine to quickly store some vital data in a non-volatile memory in the few milliseconds before the power supply drops to zero.

A software interrupt is initiated by a special mnemonic included in the instruction set, e.g. SWI for the Motorola processor. This is also used in some situations to halt the processor at the end of a program.

When an interrupt is received by the CPU it:

- Completes the current instruction in its program
- Stores all of its registers contents in memory
- Executes a special program called the Interrupt Service Routine (ISR)
- Returns to the point where it left the original program.

Various methods are used to transfer control to the ISR, these include:

i. Jump to a specified memory location and get the address of the start of the ISR previously put there by the programmer (this may or may not be a permanently reserved location)
ii. Obtain a new value for the PC (which is the start of the ISR) from a specific register within the CPU
iii. Execute a JSR instruction to an address supplied by some external hardware on the data bus. This externally supplied address is called the **vector address or interrupt vector**
iv. On receipt of **interrupt request** the CPU supplies an **interrupt acknowledge** signal which gates back from the external hardware a **specific instruction**.

Methods i. or ii. are relatively simple to operate and require no external hardware. Methods iii. or iv. require external hardware which needs to be synchronised. However, where multiple interrupts are concerned these last methods can supply vector addresses peculiar to their own ISR's. Multiple interrupts will be explained later.

13.7.1 Returning to the main program

This relies on how the original PC value was stored on the jump to ISR sequence.

One method is the **jump and mark** technique. The **start address** of the ISR is held in a particular memory location while the **return address** to the main program is stored in the first address of the ISR. Using **indirect addressing** a jump will cause a return to the main program at the end of the ISR.

This method does not allow 'nested interrupts' nor can it be used on the 'HC11 because **indirect addresses are not available**.

An alternative method as used by Motorola is to use the stack.

On interrupt, the PC all internal registers are put on the stack and operation is identical to JSR and RTS operations previously described.

A processor, with only one interrupt request input pin, may have more than one source of interrupt and may have to respond differently to each. For this reason recognition by the processor of the interrupt source needs to be considered.

Three common methods are used to examine interrupts. These are:

a. **polling**
b. **daisy chaining**
c. **vectoring**

a. **Polling**

Polling means the CPU checks each interrupt line until it finds an active one. Usually the CPU has only one **interrupt request** line with several devices connected, via OR-gates, to it as shown in the diagram.

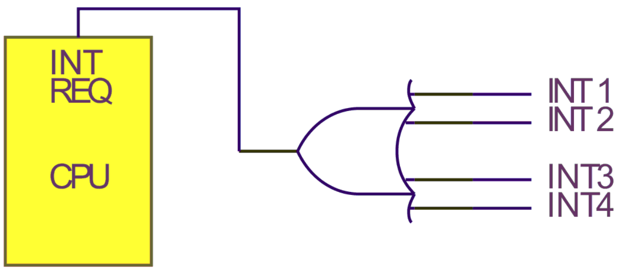


Figure 13.18: Multiple interrupt system

Also each peripheral has a 'control and status register' (CSR) which has an INT bit set when an interrupt request is made.

A software routine is then used to check each peripheral CSR INT BIT in turn (i.e. polling) to identify the interrupting peripheral. This is shown in the following software flowchart.

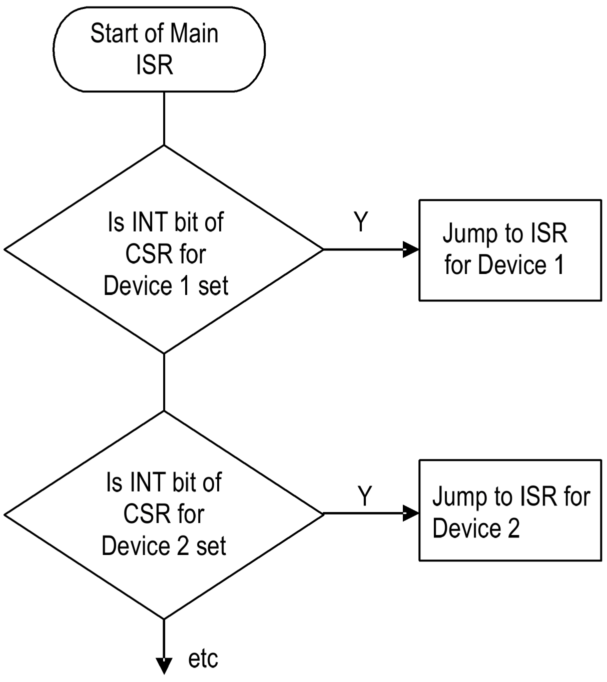


Figure 13.19: Polling software routine

An alternative to a separate control and status register in each peripheral is to have a special register called an 'input port'.

Each peripheral can then set a bit in this register.

A similar polling technique can check each bit in this register and go to the appropriate ISR.

Polling requires very little hardware to achieve but can only be used for a limited number of peripherals because of the serial nature of checking each one.

b. **Daisy chaining**

An improvement on the polling routine for interrupts is called the 'daisy chain' method. It is significantly faster than conventional polling techniques:

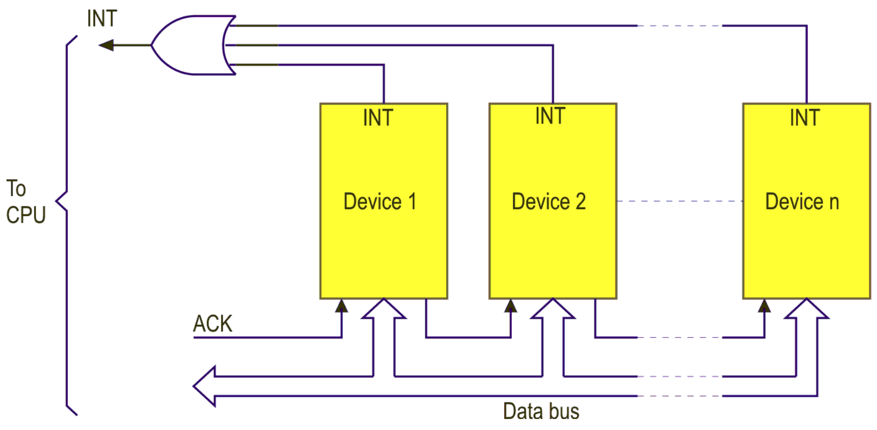


Figure 13.20: Daisy chain interrupt system

Let us consider the operation of this system. On receipt of an interrupt signal (INT) after storing its registers, the microprocessor generates an interrupt acknowledge signal (ACK). This is gated to device 1. If device 1 caused the interrupt it will place its identification code on the data bus where it will be read by the microprocessor.

If it did not generate the interrupt, it propagates the ACK to the next device, which follows the same procedure.

c. **Vectoring**

This is the fastest and most sophisticated method of handling interrupts. It means each interrupt source must provide data (i.e. vector), which the CPU can use for identification of the source or better yet, the branching address for the interrupt handling routine.

If the input/output device controller just provides the identity code of a device, it is a fairly simple task in software to look-up a table containing a branching address for each device.

This is simple from a hardware point of view but doesn't achieve the highest possible performance efficiency.

The highest possible performance occurs when not only does the CPU receive an interrupt, it also receives the **direct 16 bit branching address** to the appropriate ISR.

It can then directly branch to the required location in memory and start servicing the device.

13.7.2 Priorities

When handling multiple interrupts a problem can arise in that several interrupts may be triggered simultaneously. Under these conditions, the CPU must then decide in which order they should be serviced.

If a priority is normally attached to each device, the CPU can then service each device in order of that priority. It is normal to assign level 0 as the highest priority, level 1 the next, and so on.

Typically,

level 0 could be used for a power-fail restart
levels 1 and 2 for display devices
levels 3 and 4 for disks
level 5 for printers
level 6 for teletypes
level 7 for external switches, etc.

Priorities may be enforced in hardware or in software.

Software priorities are enforced by using a look-up table of device code versus priority level and servicing the highest priority first.

Hardware priority may be achieved using a system as shown in the diagram. The system uses a special external register called an enabling register. This register is fed from the data bus and its outputs are used to enable or inhibit a set of AND gates.

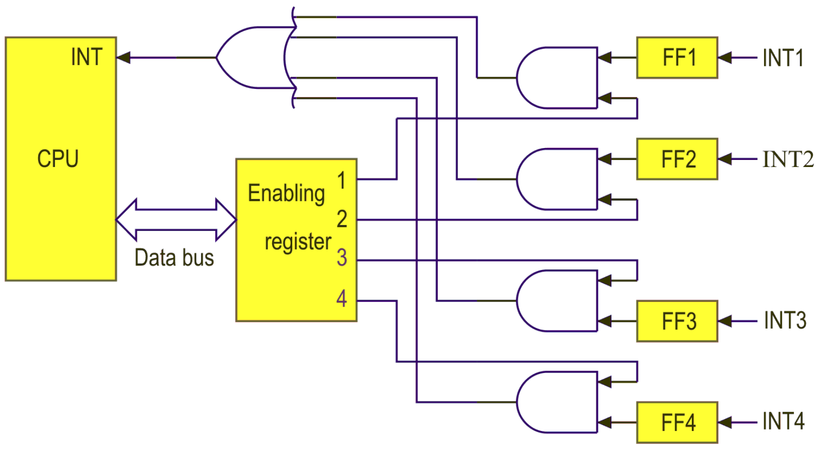


Figure 13.21: Priority interrupt system

The CPU will only respond to those interrupts whose enabling signals are set by a '1' in the register.

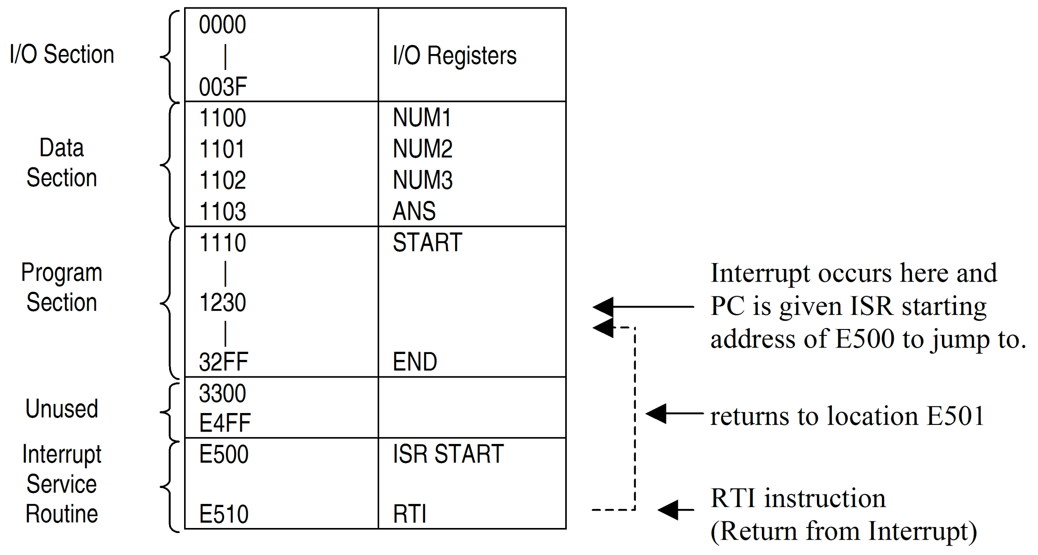
13.8 Device initiated input/output by interrupts

This type of input/output makes efficient use of computer time because the CPU doesn't have to repeatedly check if the peripheral is ready for a transfer.

Instead, CPU can do other tasks and when the device is ready, it sends a signal to the CPU interrupt input.

This causes CPU to suspend its current execution and perform a special Interrupt Service Routine (typically this would be a program to transfer data) and when finished, it returns to the main program. Consider a memory map:

e.g.



13.8.1 Return address

After executing the ISR, how does MPU know where to return to?

As outlined previously, this is taken care of automatically by the CPU as it temporarily stores the current address of the PC on the stack **before** it branches to the ISR.

Actually it stores PC, Index Reg, AccA, AccB and CCR in that order, on the stack.

When the RTI is executed it restores these values to their respective sources in the reverse order as the stack is a Last In First Off (LIFO) device.

13.8.2 Disabling the interrupt

What happens when an input/output device interrupts the MPU while it is executing a program, which requires continuous processing?

This might happen if the CPU was in the middle of a timing loop for a delay for instance, or if the CPU was communicating with another device.

In these situations, an interrupt could have undesirable results. All CPU’s have therefore provision for disabling the interrupt input. This is done by setting the interrupt mask bits (X and I) in the CCR and by local enable mask bits in the on-chip peripheral control registers.

Instructions are provided so that the programmer can set or reset these bits. These are Set Interrupt Mask (SEI) and Clear Interrupt Mask (CLI). These are put around a program to protect it. e.g.

```
(SEI
(Delay
(Routine
(CLI
```

13.8.3 Types of interrupt inputs

The 68HC11 has 2 types of interrupt inputs – *maskable* and *non-maskable*.

A maskable interrupt has just been described and is ignored if the interrupt mask flag (I), has been set. A **non-maskable** interrupt will interrupt the CPU regardless of the status of the I flag. It cannot be disabled by the programmer as it often involves a safety feature.

The parameters of these interrupt inputs are:

- The CPU can ignore IRQ if the I flag has been previously set. It cannot ever ignore a NMI (XIRQ input).
- NMI has priority over IRQ if signals were received simultaneously at these inputs.

A major use of NMI could be in a power failure shut-down routine. Here an external circuit detects a power drop (or loss) and signals the 68HC11, via the XIRQ line, to branch to an ISR to store register contents in a battery backed RAM.

IRQ is often used in handshaking between the CPU and a peripheral.

13.8.4 Address of the ISR

When an interrupt occurs how does the MPU know what address to branch to for the ISR?

The answer varies from CPU to CPU and is given by each manufacturer.

On interrupt, Motorola systems obtain a 16 bit interrupt vector from 2 fixed memory locations. This interrupt vector is then loaded into the PC as the address of where the first instruction of the ISR resides.

Activity 13.3

a. Refer to the ELE1301 course page on ‘Study Desk’ and complete the following experiment:

   Home experiment 13-4 – Parallel I/O program 4 – Interrupts

Self assessment 13.5

1. Explain the reason for requiring ‘interrupts’ in a microprocessor system, giving a clear example of the actions taken by a CPU when it detects an interrupt.

a. Explain one (1) method a microprocessor may use following an interrupt to determine the appropriate interrupt service routine.

13.9 FPGAs (not examinable)

What is an FPGA? An FPGA (field programmable gate array) is a programmable logic device. In other words, it is a digital logic device that can be programmed to implement an entire digital system, including the processor, peripheral components and the interface logic. The use of FPGAs has greatly increased in recent years, particularly in automotive, telecommunications, military and commercial products where reductions in size and cost are of paramount importance. FPGAs are introduced here to simply to raise your awareness of state-of-the-art techniques used in computer engineering and embedded systems design.

FPGAs can be programmed to perform relatively simple logic functions or very complex high speed logic functions. An FPGA is an integrated circuit containing many (1500 to 75,000) identical logic cells. Each logic cell may combine a few binary inputs (typically 3 to 10) to one or two outputs according to the boolean function specified in the user's program. The individual cells are interconnected by a matrix of wires and programmable switches. A user's design is implemented by specifying the simple logic function for each cell and selectively closing the switches in the wire matrix.

The beauty of FPGAs is that you do not require thousands of discrete integrated circuits, microprocessors and I/O devices to perform complex logic functions. An FPGA can be programmed to perform the required function by describing the function on a PC, using a schematic and/or specialised software such as VHDL, which stands for VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. Once you have described your logic function on a PC you can compile and download it to the FPGA. The FPGA will then perform the function as you described it.

You can search the web for more information on this topic. A good place to start is fpga4fun.com