2 Bus Systems and Devices

2.1 Introduction

Many of today's advanced embedded processors have the hundreds of kilobyte sized on-chip memories of desktop computers just a few decades ago. They also run at gigahertz speeds and have a high level of integration with other functional blocks on the same chip or in what is a System on Chip or SoC. These memories are used for programs and data executing within the processor and are optimized for speed so that the busses in the processor are normally not easily accessed.

However these processors need to interface to external devices which have grown in complexity. A common situation is to acquire and store data or interface with large displays. An external bus interface is required for such situations. So this external bus interface is used for i) specialized memory needs, for example, static RAM, NAND and NOR type flash memory ii) controlling a large graphics LCD.

Thus many vendors provide an external bus interface and most of the time, a high speed bus is not needed. As we will see, just using a few low cost TTL chips are enough to do the interfacing. However, the protocols used in these simple bus interfaces still lives on in modern systems, which build upon these protocols with advanced techniques in computer architecture. We will look at a bus and see how it interfaces with the most common kinds of devices, namely memory chips, buffers and latches.

2.2 ARM based Hardware architecture

We have seen that the ARM series of processors has become dominant in mobile computing and most of these processors are used in a specific application. However there have been versions of the ARM that have a more open interface. One of these is the Raspberry Pi (RPi), a low-cost single-board computer with a SoC from Broadcom - this SoC is widely used in set top boxes used in televisions.

The RPi is the size of a credit card and can run the Linux operating system. Its hardware can interface with low-level electronics and so that high-level Linux software can analyse as well as provide control to external devices. Thus it has been successfully used in Internet of Things (IoT) applications, as well as robotics, cyber-physical systems, 3D printing and much more. Besides the low cost, the success of the RPi as an affordable computing environment is due to the effort made by the producers of the RPi to make its operating software trouble-free and easy to use especially for new users.

For industrial use, the designers came up with the Compute Module 3 (CM3) which has much less built in peripherals than the RPi. Much more processor pins are available for use, lowers the cost of the board making it more flexible for custom designs.

2.3 Hardware of the SoC

The Broadcom SoC used in the CM3 has a CPU which uses an ARM Cortex A core which follows the ARMv8-A architecture. The following is a list of hardware blocks which typically make up the SoC.

- GPU (Graphics Processing Unit - Broadcom VideoCore IV)
- Memory
- Timers
- DMA (Direct Memory Access)
- Interrupt Controller
- GPIO (General Purpose Input Output)
- USB (Universal Serial Bus)
- PCM (Pulse Code Modulation)
- I2S (Inter-IC Sound)
- PWM (Pulse Width Modulation)
- Serial Communication
    - I2C (Inter-Integrated Circuit)
    - SPI (Serial Peripheral Interface)
    - UART (Universal Asynchronous Receiver/Transmitter)

On the Broadcom SoC, an on-chip bus transfers data between these various blocks using hardware based on a bus architecture designed by ARM called AMBA (Advanced Microcontroller Bus Architecture). This in turn specifies other interfaces to external devices like the Advanced eXtensible Interface (AXI) which allows interfacing to external memory devices. There are other on-chip interfaces in use by other companies.

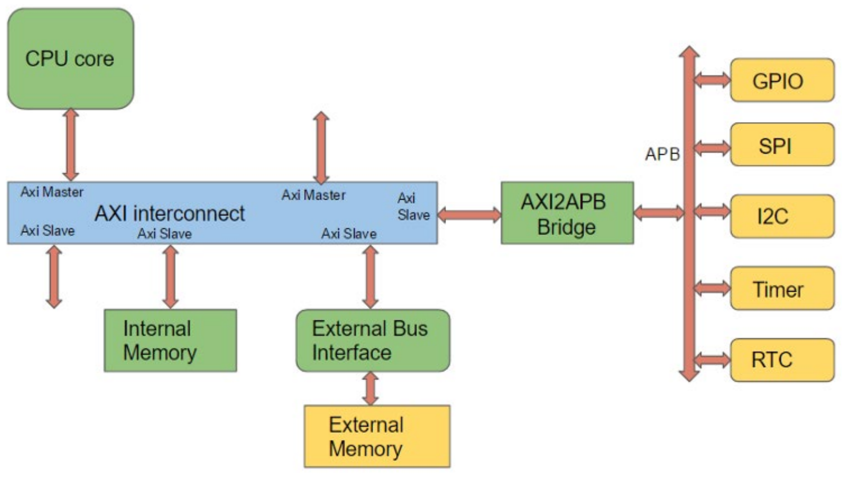


Fig 2.1 ARM AMBA block diagram

The AXI specification allows connection to external memory using the external bus interface and we will look at how it can be used to access flash memory using the NAND type. For the Broadcom devices, this is known as the Secondary Memory

Interface or SMI. The NAND Flash memory interface requires a smaller number of address pins and six are specified in the SMI with the requisite control pins. The "8080" or "6800" bus interface is used because of their simplicity and modest hardware requirements.

The 8080 from Intel and 6800 from Motorola were among the very first microprocessors on the market in the 1970's. These devices are not in use today, but their interfacing protocol is still very much in use and this motivates for learning about how they work.

The 8080 was electrically cumbersome to use due to its multiple voltages and support chips and an improved version, the 8085 was introduced soon. In fact it is the bus timing signals of the 8085 which is commonly referred to as the “8080 bus” in current use. Currently, there are several variations on this specification. In this case, we will describe how the *8080 mode* parallel bus works.

2.5 The 8080 mode bus

In the earliest commercial microcomputer systems, vendors have always provided an interface bus. They realized that it was not possible to anticipate all the needs of users who needed to interface to all kinds of devices. The 8080 interface specifies a 16 bit address bus, an 8 bit data bus and the use of separate read and write signals for control.

Current devices that interface using the 8080 mode are graphical liquid crystal displays (LCD) and various types of flash memories. In these devices the address, data bus sizes can vary but the timing of control signals have a common sequence.

2.5.1 Description of bus

The 8080 and 8085 originally specified the following bus widths and control signals.

Table 2.1 Description of selected signals of the 8080 mode Bus

Signal name	Description
A0-A15	This is the address bus and gives a total of 64 Kb of memory. When outputting to an I/O device, only 8 bits are used.
D0-7	These are the data bus bits for devices, signifying that a byte is the normal size of data being accessed.
$\overline{\text{RD}}$	*Read* instructs a selected device to drive data onto the data bus.
$\overline{\text{WR}}$	*Write* instructs a selected device to store the data on the data bus.

$IO/\overline{M}$	*IO/M* indicates whether the current instruction is mean for memory or I/O devices.

As mentioned earlier, for devices in current use, the full width of the address and data busses can vary. For example, the data bus may be 8, 9 or 16 and 18 bits.

The timing of a typical instruction of the 8080 and 8085 is that it executes over 3 clock cycles. Also the data bus is multiplexed allowing more pins to be available for control purposes. Thus the full width of the address bus appears only at cycle T2.

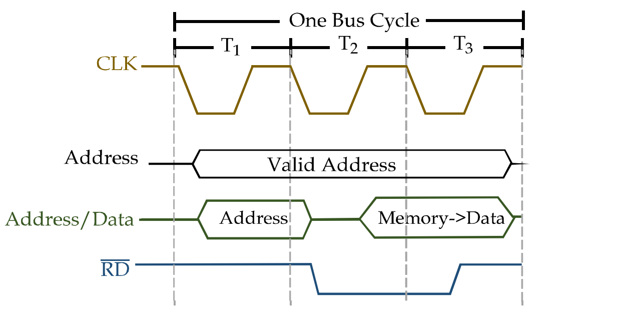
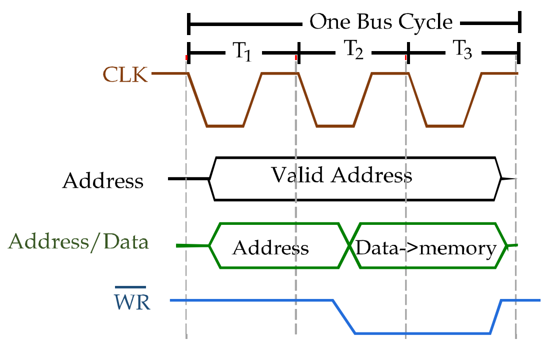
 One Bus Cycle T_1 T_2 T_3 CLK Address Valid Address Address/Data Address Memory->Data $\overline{RD}$	 One Bus Cycle T_1 T_2 T_3 CLK Address Valid Address Address/Data Address Data->memory $\overline{WR}$
T1 – Lower 8 bits of address appears on data bus. T2 – Read signal goes low, data from memory appears on data bus. T3 – Read signal goes high, data goes into CPU.	T1 – Lower 8 bits of address appears on data bus. T2 – Write signal goes low, data from CPU appears on data bus. T3 – Write signal goes high, data goes into memory.

Fig 2.2 Timing diagram of a typical 8080/8085 instruction

2.5.2 Memory and I/O mapped addressing

The 8080 bus makes a distinction between addressing memory and I/O devices by having an IO/M signal. This is because the 8080 and 8085 have different instructions for accessing them as they allow for greater flexibility in accessing I/O. Firstly, memory mapped I/O allows the use of the interfacing device as if it were a memory device using standard load/store/move instructions in assembler, or high level language assignment statements. Displays are a typical example of this sort of device.

Second, I/O mapped instructions take the form of "input" or "output" instructions. A comparison between the two modes of addressing is shown in Table 2.2.

Table 2.2 Comparison between Memory mapped and I/O mapped devices

Memory mapped	I/O mapped
Uses microprocessor main memory address space	Uses separate memory space apart from processor's main memory.
May interfere with allocating memory to programs as we have to note that there are "special" locations that are not standard memory.	No concerns about special address ranges
Use flexible addressing modes to access data.	Normally all data must pass through accumulator - can be slow.

2.6 Memory Devices on the 8080 Mode Bus

2.6.1 Introduction to Types of Memory Devices

There are many different types of semiconductor memories, but we will focus on two: namely, Random Access Memory, or RAM, and Read Only Memory, or ROM. Each has its own set of technologies and uses.

<u>RAM - Random Access Memory</u>
The term RAM has come to mean a memory device where data can be stored as well as retrieved, and where the process of storing data at a particular location takes about the same time as the process of retrieving it from that location.

Most RAMs are volatile devices i.e. the stored data is lost when the power source is removed. However, there are non-volatile RAMs on the market which contain either a capacitor, or a small re-chargeable battery cell, so that the stored data can be retained for some time after the power is removed.

2.6.1.1 ROM - Read Only Memory

The logic function is stored in the circuit permanently without the need of electrical power to sustain the memory it is non-volatile memory). Also, the input combination has no effect on how long it takes to read the output combination random access memory). So, a ROM can be defined as a fixed memory whose contents cannot be altered during normal operation.

Below we summarize some of the main features of ROMs, comparing them with RAMs.

	RAM	ROM
Volatile	yes	no
Random Access	yes	yes
Read	yes	yes
Write by processor	yes	No

There are a number of types of ROM, based on the methods of storing the data in the ROM, and whether or not that data can be erased.

In ROM devices, the data is stored as part of the manufacturing process, and can never be changed after that. Thus it requires costly manufacturing processes and is not economical unless many thousands are required.

Applications :

ROMs are used to store unchanging data like user prompts, error messages, character generators for video displays, or as the bootup program for computers. But ROMs are widely used in simpler forms, for example, even as a simple 4 line to 16 line decoder or a BCD to 7 segment decoder.

EPROM - Erasable PROM/One Time Programmable ROMs

An EPROM or Erasable PROM is a device in which the data can be erased after it has been stored and so it can be reused many times. The erasing is done by shining ultraviolet light on to the chip through a quartz window for 20 to 30 minutes. As a result, the entire data contents are erased each time. Because of the quartz window, the chip has to be made of ceramic. In order to make using of existing processes and lowering the cost of the package, manufacturers offer EPROMs in plastic packaging that cannot be erased. These are known as One Time Programmable ROMs (OTPROM).

Applications :

For design testing and development, it may be necessary to revise and test a program many times before it is considered error free. Because a PROM can only be programmed once, it is much too expensive for this purpose. An EPROM or Erasable PROM, which can be re-used many hundreds of times is used in this case. Although the initial cost of the IC package is greater than for a PROM, it becomes effectively much cheaper the more it is re-used. Because they are now relatively cheap. EPROMs have replaced PROMs in many applications.

PROM - Programmable ROM

To overcome the problem of expense when only small quantities are required, programmable ROMs are produced. Here, the data is not stored during manufacture, but later, in the field, by a user with the necessary programming equipment. Once stored, the data can never be changed. The process involves the "blowing" of small nichrome fuses or links in the device, and is thus non-reversible. Hence these devices are often called fusible link PROMs. They are normally bipolar devices, requiring a different manufacturing process.

Applications :

The main use of PROMs is for small-scale production where the economies of scale are not available. Because a chip manufacturer can produce the basic (unprogrammed) device in large quantities, they are comparatively inexpensive, but the user of the PROM must then pay the additional costs of programming the device. A custom-made address decoder is a very useful application of a small Field-programmable ROM.

EEPROM - Electrically Erasable PROM

An EEPROM or E^{2}PROM (Electrically Erasable PROM) is a special type of erasable semiconductor memory where the data can be erased and re-written electrically in circuit. This class of devices has recently become very important as storage devices. But it is not like a RAM. The process of storing data takes very much longer than reading it, usually about 30 milliseconds writing time. A later type of EEPROM is Flash memory. The main difference is that FLASH updates its contents in terms of group of bytes or sectors whereas EEPROM does so a byte at a time. FLASH can therefore store large amounts of data relatively quickly. Also, EEPROM is available with a serial interface, which makes it smaller, both physically and in storage capacity. Parallel devices are larger on both areas, thus addressing different applications.

Flash memory can be purchased as bare memory chips, which generally use NOR gate technology used to store programs as the memory can be accessed randomly. Interfacing to the chip involves hardware design and it may not be possible to remove the bare chip, but it can take very little space. In terms of software, little effort is needed.

However using NAND gate technology, flash memory is commonly available as "memory cards". Examples are Compact Flash (CF), Secure Digital (SD), MultiMedia Card (MMC) formats and others. This is because NAND flash memory favours sequential access of data within blocks of memory such as in these devices. Hardware on the cards make interfacing easier. For example, CF cards appears like hard disks while SD/MMC cards can use the Serial Peripheral Interface (SPI), a commonly used protocol to interface to it. This can make transfer of data easier as card readers are common, but interfacing is more complicated.

Flash memory also appears in standalone portable storage devices. For example, portable music players can appear as a hard disk when attached through the USB interface.

Applications :

The form of flash memory to use is dependent on the application. As part of a system, it is natural to use flash memory for updates to programs as data stored within can be done easily and automatically. An example of this is the ability of modern equipment to update their firmware by "flashing" an update from the manufacturer.

However, if we wish to use flash memory for storing of large amounts of data, it makes more sense to use a CF card and access it like a hard disk.

Serial EEPROM is mainly used to store settings and configuration information where its low cost, low power consumption and small size are advantages here.

This illustrates the fundamental difference between RAM and EEPROM. In a RAM the processes of storing and retrieving data take about the same time - in the order of tens to hundreds of nano-seconds. While an EEPROM can be read from in about the same time, the process of storing data takes much longer.

$$\begin{tikzcd}
 & \text{EEPROM} \arrow[dl] \arrow[dr] & & \\
\text{Serial} & & \text{Parallel} \arrow[dl] \arrow[dr] & \\
 & \text{Flash} \arrow[dl] \arrow[dr] & & \text{EEPROM} \\
\begin{array}{l}\text{Flash chip}\\\text{NOR gate}\end{array} & & \begin{array}{l}\text{Flash devices}\\\text{NAND gate}\end{array} &
\end{tikzcd}$$

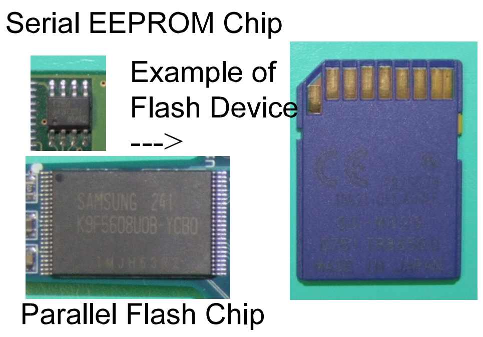


Fig 2.3 EEPROM family

In summary, the diagram describes the EEPROM family. The type used will depend on the applications.

2.6.1.2 RAM Devices

RAM devices can be classified into two types: static and dynamic. RAM's are manufactured using both MOS and bipolar technologies. MOS uses NMOS and CMOS. These are usually simpler devices, and yield higher densities than bipolar process. Bipolar devices include Schottky TTL, ECL, and IIL types. Usually faster than MOS devices, and accordingly have higher power dissipation.

Static RAM’s

Static RAM's use a flip-flop as the storage element, one F/F for each bit of data storage. So an 8K RAM has 8192 F/Fs plus the necessary address decoding and I/O buffering circuits. Information is retained in the F/F for as long as power is applied.

Dynamic RAM’s

Dynamic RAMs use the ability of MOS capacitors to store charge as the basic storage element. 1's or 0's are indicated by the presence or absence of charge on the capacitor. If the capacitor is charged above a certain level (that is, its voltage is above a certain threshold), it represents a logic 1, otherwise it is a logic 0. An ideal capacitor, once charged, maintains a constant voltage, but charge will leak away in a practical capacitor, and so the voltage will change.

The capacitor's voltage (the data) must be read regularly, and if it is a logic 1, it must be restored to full voltage, before too much charge leaks away. This is called refreshing. If a dynamic RAM is not refreshed often enough, the voltage of the logic 1's will fall below the threshold and all stored bits of data will revert to 0's. For example, an 8K dynamic RAM contains 8192 MOS capacitors plus associated decoders, buffers, etc. All these capacitors must be refreshed regularly, typically every 2 milliseconds.

2.7 I/O Devices on ISA bus

A more common use of plug-in cards is to provide many more ports to be used for interfacing with external devices. These ports normally interface with devices that are much slower than the main processor so that the relatively slower speed of data transfer is not an issue. As we saw earlier, the Intel architecture uses 16 bit addressing for I/O. On the PC, only 10 address pins are actually used, to reduce the hardware decoding requirements.

2.7.1 BASIC INPUT/OUTPUT HARDWARE

Various hardware elements can perform some of the functions of the I/O section. Simple devices include flip-flops (latches) and buffers. In this chapter we will look at some of the basic elements used in interfacing circuitry.

2.7.1.1 Buffers

Most microprocessors are MOS products, hence the driving capabilities of such devices

are usually restricted to at most 1 TTL load or 5 MOS loads. This severely restricts the operation of the microprocessor in driving other devices, for example, RAMs, ROMs and other support devices. In order to increase the driving power, or driving current, buffers are used.

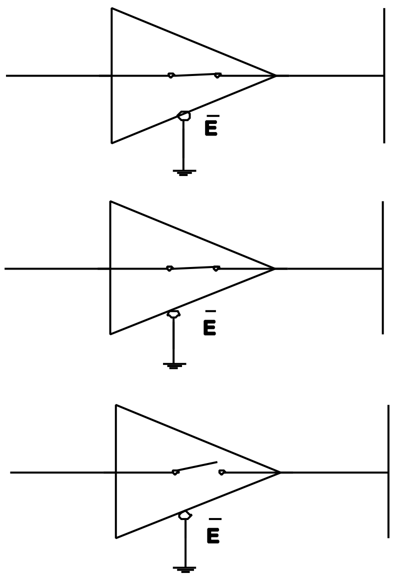
Buffers are TTL ICs which allow a signal to pass through them without altering the logic value of the signal. However, the buffer increases the DC drive characteristics of the signal. Hence, the microprocessor, with the aid of a buffer is capable of driving many more circuits.

Another advantage of using buffers is that the microprocessor is isolated from any outside signals. If there is a change in voltage along the signal lines, the buffers are first affected, thereby preventing any damage to the microprocessor. Being mainly TTL ICs, they introduce a small delay in the transmission of data.

Two variations of the buffer exist. The first is simply a one-in one-out buffer which takes an input and increases the driving current of the signal at the output for all logic levels.

The second type of buffer is called a Tri-state buffer. This device has an additional control signal. When the control is active, the device performs exactly as a buffer. When the control is de-activated, the device prevents any output from appearing. The output of the device turns into a high-impedance state giving neither a logic 1 or 0. The table below shows the logic states of such a buffer :

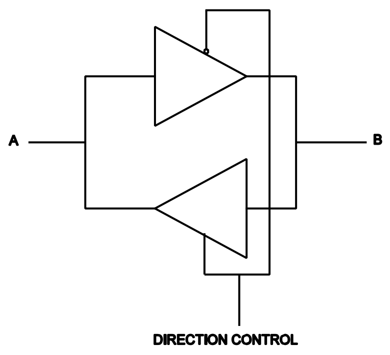
Input	Control	Output	State
0	Active	0	buffers input
1	Active	1	buffers input
0	Non-active	High impedance	Tri-state
1	Non-active	High impedance	Tri-state



Tri-state buffers are useful in circuits which only allow the passage of signals when a particular state is active. This state can be connected to the Control input of the buffer.

A variation to the tri-state buffer is the bi-directional buffer. This buffer allows the passage of data from one point to the other depending on a Direction Control Signal. This allows a system to control the direction of flow of data from one point to the other. The figure below shows the operation of such a buffer.

When the Direction Signal is ‘low’, data flows from Point A to Point B. When the signal is ‘high’, the data flows from Point B to Point A.



A typical application of the bi-directional buffer is in the buffering of the microprocessor's data bus. Using the _RD line as the Control Signal, the microprocessor is able to control the flow of data into the microprocessor only when data is required. At all other times, the direction of flow of data is from the microprocessor to the other devices.

2.7.1.2 Latches

Latches are mainly made up of D-type flip-flops. The latch takes in a signal via a strobe (clock pulse). The latch then holds the signal until another strobed signal is applied or until the power is removed.

The latch has an advantage over the buffer in the sense that the signal is held at the particular logic level until a change is requested. The buffer holds the logic level as long as the signal is also at that level. Latches are thus used in preference to buffers in situations where the receiving end requires a little more time to process the signal. The figure below shows the operation of different latches in response to changing signals.

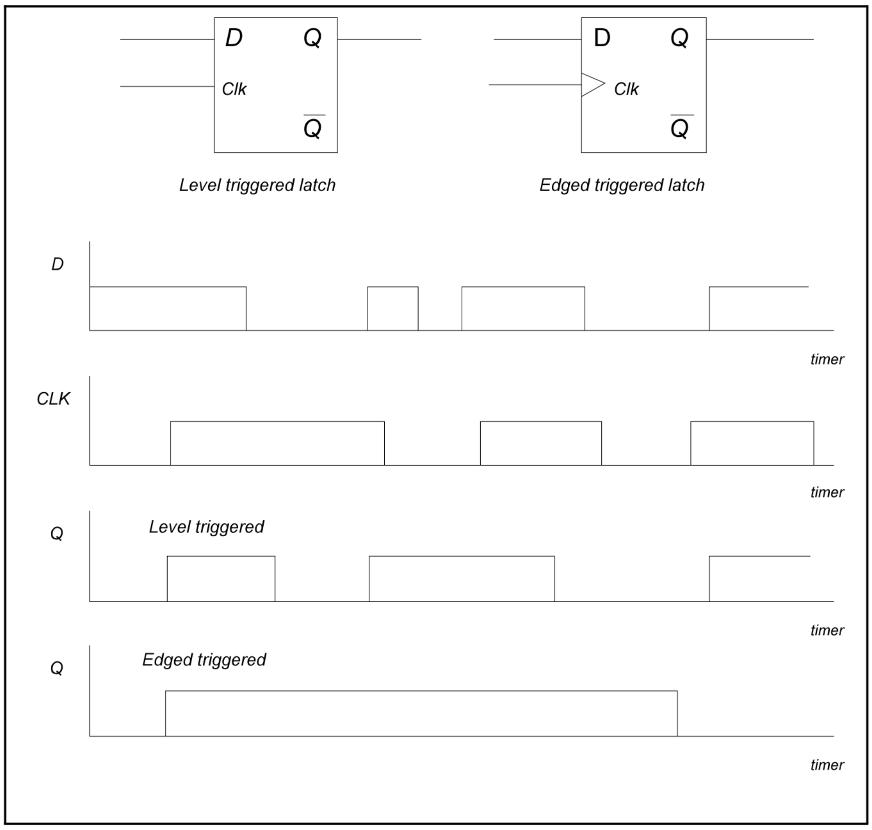


Fig 2.4 Signal Response of Latches

Latches can be level-triggered (changes output as long as the strobe is active) or edge-triggered (changes output only on either a rising or falling edge of the strobe signal). In cases where ambiguity is to be avoided, edge-triggered latches are used.

2.7.1.3 Input/Output Ports

Basic Input Ports

The basic input port is a tri-state buffer connected to a data bus line. The control signal of the buffer is usually the combination of _IORD and the selected port address, which is obtained using address decoding.

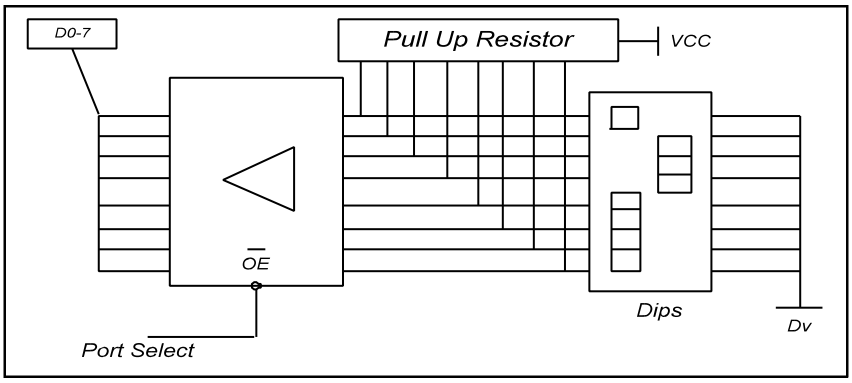


Fig. 2.5 An Example of an Input Port using a Buffer

Basic Output Ports

The basic output port is a latch. The CPU places the necessary data to be output on the data bus lines and then clocks the data into the latch with the control signal which is usually the combination of the _IOW and the selected port address, which is obtained using address decoding.

The latch is preferred over the buffer as external devices are usually slower than the CPU. Hence, the port has to hold the data until the device is ready to read it.

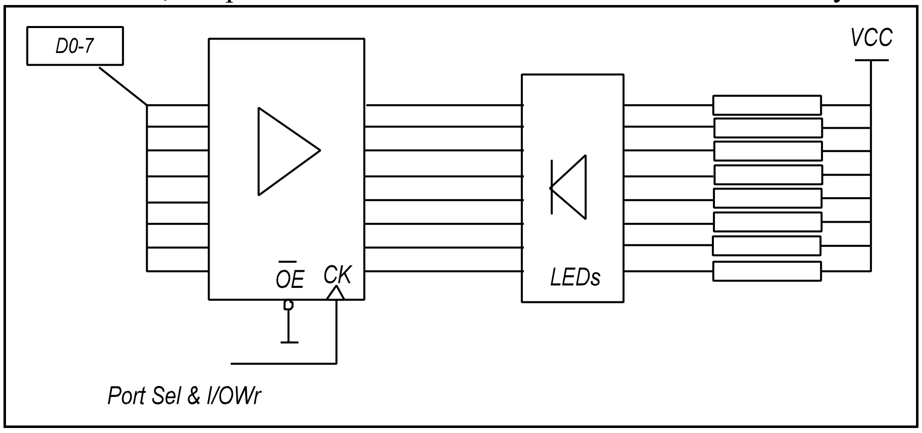


Fig. 2.6 An Example of an Output Port using a Latch

2.8 The BIOS and boot up process

As a final note and application, we see how ROM is used in a typical microprocessor where memory is mainly DRAM. We will also see how a basic "monitor" program considerably enhances the use of a bare computer system.

The 8080 follows a given sequence when it is RESET. Recall that ROM is non-volatile, so that if a processor starts, it is guaranteed to have valid code when it jumps to that location.

In most other microprocessors, what happens after that is purely a matter of what the programmer puts into the ROM code. Starting from the early microprocessor systems available, the vendor would often put in basic routines in a so called “Monitor Program” and include it in the ROM.

These allowed users to do fundamental tasks like simple output to the screen, keyboard handling, output to printer, serial communications and so on. This allowed users to quickly bring up their system to productive use and allowed a measure of standardization to programs as they used common routines to do input and output.

In some systems – typically PCs, this monitor program is known as the Basic Input Output System or BIOS. More routines could be added, for secondary storage devices like hard disk.

A BIOS may occupy as much as 64K of ROM which is not practical for small processors, but monitor programs can be as small as 2K. However if the system is simple enough, a programmer can make do without any built in monitor program. Consider the boot up process for the 8080 processor:

<u>Microprocessor function</u>
i) Performs a jump to location 0H.
ii) Executes the code found there, which is normally contained in ROM.

<u>Monitor program (BIOS) function:</u>
i) It searches for a secondary storage device like a floppy or hard disk.
ii) It reads in the boot sector (normally 512 bytes in length) for the device.
iii) Puts the contents into RAM.
iv) Jumps to that code and executes it.
v) This code will have instructions to load other larger files which will continue to initialise the system.

This process is called “bootstrapping”. Other processors will jump to other locations upon start up but the startup process follows the same sequence.