**ACKNOWLEDGMENT**

We thank Lord Almighty for the blessings he had bestowed on us which resulted in finalization of our mini project.

We would like to express our sincere gratitude to this institution, Ilahia College of Engineering & Technology, Muvattupuzha & our Manager Mr.M.M.Mohammed Kunju for the worthy support extended to us. We also thank our honourable Principal Dr.Babu Kurian, for providing us with best facilities and atmosphere for our project development and its realization. We are grateful to the Head of the Department, Asst. Prof. Robin Abraham, for his constant encouragement and support.

We are highly obliged to our mini project coordinators Asst. Prof. Ansal K.A & Asst.Prof. Melbin George & also to our group guide Asst. Prof. Simy M Baby, for their valuable guidance and help during the course of this project work. We also like to express our gratitude to the lab staffs who helped us to make the project a success.

This acknowledgement will stand incomplete if our parents, friends, seniors & classmates are not thanked for enduring their prayers and support to us, which was inevitable for the realization of this work. Last but not the least we are thankful to each other in this project group for being co-operative, patient and hardworking for the successful completion of this project.
1. INTRODUCTION

The project titled “VOICE CONTROLLED WHEEL CHAIR” describes the design of a simple, low-cost microcontroller based robot for helping disabled persons. Robot is an intelligent agent that can perform tasks automatically or with guidance, typically by remote control. A robot is usually an electro-mechanical machine that is guided by computer and electronic programming.

An embedded system is a system which is going to do a predefined specified task is the embedded system and is even defined as combination of both software and hardware.

A general-purpose definition of embedded systems is that they are devices used to control, monitor or assist the operation of equipment, machinery or plant. "Embedded" reflects the fact that they are an integral part of the system. In many cases their embeddedness may be such that their presence is far from obvious to the casual observer and even the more technically skilled might need to examine the operation of a piece of equipment for some time before being able to conclude that an embedded control system was involved in its functioning. At the other extreme a general-purpose computer may be used to control the operation of a large complex processing plant, and its presence will be obvious.

All embedded systems are including computers or microprocessors. Some of these computers are however very simple systems as compared with a personal computer.

The very simplest embedded systems are capable of performing only a single function or set of functions to meet a single predetermined purpose. In more complex systems an application program that enables the embedded system to be used for a particular purpose in a specific application determines the functioning of the embedded system. The ability to have programs means that the same embedded system can be used for a variety of different purposes. In some cases a microprocessor may be designed in such a way that application software for a particular purpose can be added to the basic software in a second process, after which it is not possible to make further changes. The applications software on such processors is sometimes referred to as firmware.

The simplest devices consist of a single microprocessor (often called a "chip"), which may itself be packaged with other chips in a hybrid system or Application Specific Integrated Circuit (ASIC). Its input comes from a detector or sensor and its output goes to a switch or activator which (for example) may start or stop the operation of a machine or, by operating a valve, may control the flow of fuel to an engine.

As the embedded system is the combination of both software and hardware

Block diagram of Embedded System

Software deals with the languages like ALP, C, and VB etc., and Hardware deals with Processors, Peripherals, and Memory.

Memory: It is used to store data or address.
**Peripherals:** These are the external devices connected

**Processor:** It is an IC which is used to perform some task

Processors are classified into four types like:

1. Micro Processor (µp)
2. Micro controller (µc)
3. Digital Signal Processor (DSP)
4. Application Specific Integrated Circuits (ASIC)

**Micro Processor (µp):**

It is an electronic chip which performs arithmetic and logical operations with assistance of internal memory.

![Block Diagram of Micro Processor (µp)](image)

**Micro Controller (µc):**

It is a highly integrated micro processor designed for specific use in embedded systems.

1.1 **Introduction to applications of embedded system:**

Embedded controllers may be found in many different kinds of system and are used for many different applications. The list, which follows, is indicative rather than exhaustive. An item in the list may be relevant to a particular company because either (a) it is or involves a core process or product, (b) it is or involves an ancillary function or service performed by the company or (c) it refers to a product or service provided by a contractor under some form of agreement and the vulnerability of the supplier may need to be considered.
List of applications of embedded systems:
Manufacturing and process control
Construction industry
Transport
Buildings and premises
Domestic service
Communications
Office systems and mobile equipment
Banking, finance and commercial
Medical diagnostics, monitoring and life support
Testing, monitoring and diagnostic systems

Industrial functions of embedded systems:
A manufacturing company has provided the following list of embedded systems:
Multi-loop control and monitoring - DCS, SCADA, telemetry Panel mounted devices - Control, display, recording and operations.
Safety and security - Alarm and trip systems, fire and gas systems, buildings and facilities security. Field devices - measurement, actuation. Analytical systems - Laboratory systems; on-line/ plant systems. Electrical supply - supply, measurement, control, protection. Tools - for design, documentation, testing, maintenance.

Embedded systems compared with commercial systems:
The Year 2000 problem in embedded systems differs from the problem in commercial / database / transaction processing systems (often referred to as IT systems) in a number of ways. Firstly the user's problem may much lie much deeper than packages or applications software. It may lie in and be inseparable from systems and operating software and from hardware, i.e. in the platform on which the application software is based. When users of IT systems have hardware or operating software problems they can and should be made the concern of the computer supplier: typically, this is not the case with microprocessors and devices based on them.

Secondly in embedded systems the concern is often with intervals rather than with specific dates: the need may be for an event to occur at 100-day intervals rather than on the 5th day of each month. This has the implication that Year 2000 problems may reveal themselves both before and for some time after 1 January 2000 and not at all on the date itself.
The lifetime of embedded systems tends to be greater than that of commercial data processing systems: they remain in use for longer without alteration to their software. Because their software may therefore be older they are rendered more liable to Year 2000 problems.

1. HARDWARE DISCRIPTION

The basic elements used in ‘VOICE CONTROLLED WHEEL CHAIR’ are:

1.1. MICROCONTROLLER:

The microcontroller used is 89C51/52. The features of this µcontroller are as follows:

- 8051/52 pin and instruction compatible

- Four 8-bit I/O ports (or 6 in 64/68 pins packages)

- Three 16-bit timer/counters

- 256 bytes scratch pad RAM

- 7 Interrupt sources with 4 priority levels

- ISP (In System Programming) using standard VCC power supply.

- Boot FLASH contains low level FLASH programming routines and a default serial loader

- High-Speed Architecture

- 40 MHz in standard mode
- 20 MHz in X2 mode (6 clocks/machine cycles)

- 64K bytes on-chip Flash program / data Memory

- Byte and page (128 bytes) erase and write

- 10k write cycles

- On-chip 1024 bytes expanded RAM (XRAM)

- Software selectable size (0, 256, 512, 768, 1024 bytes)

- 768 bytes selected at reset for T87C51RD2 compatibility

- 2 k bytes EEPROM block for data storage

- 100K Write cycle

- Programmable Counter Array with:

  - High Speed Output,

  - Compare / Capture,

  - Power control modes:
• Idle Mode.

• Power-down mode.

A micro-controller can be compared to a small stand alone computer; it is a very powerful device, which is capable of executing a series of pre-programmed tasks and interacting with other hardware devices. Being packed in a tiny integrated circuit (IC) whose size and weight is usually negligible, it is becoming the perfect controller for robots or any machines requiring some kind of intelligent automation. A single microcontroller can be sufficient to control a small mobile robot, an automatic washer machine or a security system. Any microcontroller contains a memory to store the program to be executed, and a number of input/output lines that can be used to interact with other devices, like reading the state of a sensor or controlling a motor.
2.1.1 The 8051/52 microcontroller architecture:

The 8051/52 is the name of a big family of microcontrollers. The device which we are going to use along this tutorial is the 'AT89C51' which is a typical 8051 microcontroller manufactured by Atmel™. Note that this part doesn't aim to explain the functioning of the different components of an 89C51 microcontroller, but rather to give you a general idea of the organization of the chip and the available features, which shall be explained in detail along this tutorial.

The block diagram provided by P89V51RD2 in their datasheet showing the architecture the 89C51/52 device can seem very complicated, and since we are going to use the C high level language to program it, a simpler architecture can be represented as in the figure.

This figure shows the main features and components that the designer can interact with. You can notice that the 89C51/52 has 4 different ports, each one having 8 Input/output lines providing a total of 32 I/O lines. Those ports can be used to output DATA and orders do other devices, or to read the state of a sensor, or a switch. Most of the ports of the 89C51/52 have 'dual function' meaning that they can be used for two different functions: the first one is to perform input/output operations and the second one is used to implement special features of the microcontroller like counting external pulses, interrupting the execution of the program according to external events, performing serial data transfer or connecting the chip to a computer to update the software.

Each port has 8 pins, and will be treated from the software point of view as an 8-bit variable called 'register', each bit being connected to a different Input/output pin. You can also notice two different memory types: RAM and EEPROM. Shortly, RAM is used to store variable during program execution, while the EEPROM memory is used to store the program itself, that's why it is often referred to as the 'program memory'. The memory organization will be discussed in detail later.
The special features of the 89C51 microcontroller are grouped in the blue box at the bottom of figure. At this stage of the tutorial, it is just important to note that the 89C51/52 incorporates hardware circuits that can be used to prevent the processor from executing various repetitive tasks and save processing power for more complex calculations. Those simple tasks can be counting the number of external pulses on a pin, or generating precise timing sequences.

![Table of Pin Configuration](image)

**Fig. 2.2:- Pin configuration of 8051/52 microcontroller**

It is clear that the CPU (Central Processing Unit) is the heart of the microcontrollers. It is the CPU that will Read the program from the FLASH memory and execute it by interacting with the different peripherals discussed above.

The above figure shows the pin configuration of the 89C51/52, where the function of each pin is written next to it, and, if it exists, the dual function is written between brackets. The pins are written in the same order as in the block diagram of 89C51/52, except for the VCC and GND pins which I usually note at the top and the bottom of any device.
Note that the pin that has dual functions can still be used normally as an input/output pin. Unless your program uses their dual functions. All the 32 I/O pins of the microcontroller are configured as input/output.

Most of the function of the pins of the 89C51/52 microcontroller will be discussed in detail, except for the pins required to control an external memory, which are the pins number 29, 30 and 31. Since we are not going to use any external memory, pins 29 and 30 will be ignored throughout all the tutorial, and pin 31 (EA) always connected to VCC (5 Volts) to enable the micro-controller to use the internal on chip memory rather than an external one (connecting the pin 31 to ground would indicate to the microcontroller that an external memory is to be used instead of the internal one).

2.1.2. Memory organization

Typical register

<table>
<thead>
<tr>
<th>D_7</th>
<th>D_6</th>
<th>D_5</th>
<th>D_4</th>
<th>D_3</th>
<th>D_2</th>
<th>D_1</th>
<th>D_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

This base architecture is supported with on-chip peripheral functions like I/O ports, timers/counters, versatile serial communication port. So it is clear that this 8051/52 architecture was designed to cater many real time embedded needs. The following list gives the features of the 8051/52 architecture:

- Optimized 8 bit CPU for control applications.
- Extensive Boolean processing capabilities.
- 64K Program Memory address space.
- 64K Data Memory address space.
- 128 bytes of on-chip Data Memory.
- 32 Bi-directional and individually addressable I/O lines.
- Three 16 bit timer/counters.
- Full Duplex UART.
- 6-source / 5-vector interrupt structure with priority levels.
- On-chip clock oscillator.
Code Memory:

Code memory is the memory that holds the actual 8051/52 program that is to be run. This memory is limited to 64K and comes in many shapes and sizes: Code memory may be found on-chip, either burned into the microcontroller as ROM or EPROM. Code may also be stored completely off-chip in an external ROM or, more commonly, an external EPROM. Flash RAM is also another popular method of storing a program. Various combinations of these memory types may also be used—that is to say, it is possible to have 4K of code memory on-chip and 64k of code memory off-chip in an EPROM.

External RAM:

As an obvious opposite of Internal RAM, the 8051/52 also supports what is called External RAM. As the name suggests, External RAM is any random access memory which is found off-chip. Since the memory is off-chip it is not as flexible in terms of accessing, and is also slower. For example, to increment an Internal RAM location by 1 requires only 1 instruction and 1 instruction cycle. To increment a 1-byte value stored in External RAM requires 4 instructions and 7 instruction cycles. In this case, external memory is 7 times slower! What External RAM loses in speed and flexibility, it gains in quantity. While Internal RAM is limited to 128 bytes (256 bytes with an 8052), the 8051/52 supports External RAM up to 64K. Programming Tip: The 8051/52 may only address 64k of RAM. To expand RAM beyond this limit requires programming and hardware tricks. You may have to do this "by hand" since many compilers and assemblers, while providing support for programs in excess of 64k, do not support more than 64k of RAM. This is rather strange since it has been my experience that programs can usually fit in 64k but often RAM is what is lacking. Thus if you need more than 64k of RAM, check to see if your compiler supports it-- but if it doesn't, be prepared to do it by hand.

On-Chip Memory:

As mentioned at the beginning, the 8051/52 includes a certain amount of on-chip memory. On-chip memory is really one of two types: Internal RAM and Special Function Register (SFR) memory. As is illustrated in this map, the 8051/52 has a bank of 128 bytes of Internal RAM. This Internal RAM is found on-chip on the 8051/52 so it is the fastest RAM available, and it is also the most flexible in terms of reading, writing, and modifying its contents. Internal RAM is volatile, so when the 8051 is reset this memory is cleared. The 128 bytes of internal ram is subdivided as shown on the memory map. The first 8 bytes (00h - 07h) are "register bank 0". By manipulating certain SFRs, a program may choose to use register banks 1, 2, or 3. These alternative register banks are located in internal RAM in addresses 08h through 1Fh. We'll discuss "register banks" more in a later chapter. For now it is sufficient to know that they "live" and are part of internal RAM.

Register Banks:

The 8051/52 uses 8 "R" registers which are used in many of its instructions. These "R" registers are numbered from 0 through 7 (R0, R1, R2, R3, R4, R5, R6, and R7). These registers are generally used to assist in manipulating values and moving data from one memory location to
another. For example, to add the value of R4 to the Accumulator, we would execute the following instruction:

ADD A, R4

Thus if the Accumulator (A) contained the value 6 and R4 contained the value 3, the Accumulator would contain the value 9 after this instruction was executed.

**Internal RAM:**

Programming Tip: If you only use the first register bank (i.e. bank 0), you may use Internal RAM locations 08h through 1Fh for your own use. But if you plan to use register banks 1, 2, or 3, be very careful about using addresses below 20h as you may end up overwriting the value of your "R" registers!

**Bit Memory:**

The 8051/52, being a communications-oriented microcontroller, gives the user the ability to access a number of bit variables. These variables may be either 1 or 0. There are 128 bit variables available to the user, numbered 00h through 7Fh. The user may make use of these variables with commands such as SETB and CLR. For example, to set bit number 24 (hex) to 1 you would execute the instruction:

**Special Function Register (SFR) Memory:**

Special Function Registers (SFRs) are areas of memory that control specific functionality of the 8051/52 processor. For example, four SFRs permit access to the 8051/52’s 32 input/output lines. Another SFR allows a program to read or write to the 8051/52’s serial port. Other SFRs allow the user to set the serial baud rate, control and access timers, and configure the 8051/52’s interrupt system.

2.1.3 **8051/52 Microcontroller Register Banks Stacks**:

**The Accumulator**
The Accumulator, as its name suggests, is used as a general register to accumulate the results of a large number of instructions. It can hold an 8-bit (1-byte) value and is the most versatile register the 8051/52 has due to the sheer number of instructions that make use of the accumulator. More than half of the 8051/52’s 255 instructions manipulate or use the accumulator in some way.

For example, if you want to add the number 10 and 20, the resulting 30 will be stored in the Accumulator. Once you have a value in the Accumulator you may continue processing the value or you may store it in another register or in memory.

The "R" registers

The "R" registers are a set of eight registers that are named R0, R1, etc. up to and including R7. These registers are used as auxiliary registers in many operations. To continue with the above example, perhaps you are adding 10 and 20. The original number 10 may be stored in the Accumulator whereas the value 20 may be stored in, say, register R4. To process the addition you would execute the command:

ADD A, R4

After executing this instruction the Accumulator will contain the value 30. You may think of the "R" registers as very important auxiliary, or "helper", registers. The Accumulator alone would not be very useful if it were not for these "R" registers. The "R" registers are also used to temporarily store values. For example, let’s say you want to add the values in R1 and R2 together and then subtract the values of R3 and R4. One way to do this would be: MOV A,R3 ;Move the value of R3 into the accumulator ADD A,R4 ;Add the value of R4 MOV R5,A ;Store the resulting value temporarily in R5 MOV A,R1 ;Move the value of R1 into the accumulator ADD A,R2 ;Add the value of R2 SUBB A,R5 ;Subtract the value of R5 (which now contains R3 + R4)

As you can see, we used R5 to temporarily hold the sum of R3 and R4. Of course, this isn’t the most efficient way to calculate (R1+R2) - (R3 +R4) but it does illustrate the use of the "R" registers as a way to store values temporarily.

The "B" Register

The "B" register is very similar to the Accumulator in the sense that it may hold an 8-bit (1-byte) value. The "B" register is only used by two 8051 instructions: MUL AB and DIV AB. Thus, if you want to quickly and easily multiply or divide A by another number, you may store the other number in "B" and make use of these two instructions. Aside from the MUL and DIV instructions, the "B" register are often used as yet another temporary storage register much like a ninth "R" register.

The Data Pointer (DPTR)

The Data Pointer (DPTR) is the 8051’s only user-accessible 16-bit (2-byte) register. The Accumulator, "R" registers, and "B" register are all 1-byte values. DPTR, as the name suggests, is
used to point to data. It is used by a number of commands which allow the 8051 to access external memory. When the 8051 accesses external memory, it will access external memory at the address indicated by DPTR. While DPTR is most often used to point to data in external memory, many programmers often take advantage of the fact that it’s the only true 16-bit register available. It is often used to store 2-byte values which have nothing to do with memory locations.

**The Program Counter (PC)**

The Program Counter (PC) is a 2-byte address which tells the 8051 where the next instruction to execute is found in memory. When the 8051 is initialized PC always starts at 0000h and is incremented each time an instruction is executed. It is important to note that PC isn’t always incremented by one. Since some instructions require 2 or 3 bytes the PC will be incremented by 2 or 3 in these cases. The Program Counter is special in that there is no way to directly modify its value. That is to say, you can’t do something like PC=2430h. On the other hand, if you execute LJMP 2430h you’ve effectively accomplished the same thing. It is also interesting to note that while you may change the value of PC (by executing a jump instruction, etc.) there is no way to read the value of PC. That is to say, there is no way to ask the 8051 "What address are you about to execute?" As it turns out, this is not completely true: There is one trick that may be used to determine the current value of PC. This trick will be covered in a later chapter.

**The Stack Pointer (SP)**

The Stack Pointer, like all registers except DPTR and PC, may hold an 8-bit (1-byte) value. The Stack Pointer is used to indicate where the next value to be removed from the stack should be taken from. When you push a value onto the stack, the 8051 first increments the value of SP and then stores the value at the resulting memory location. When you pop a value off the stack, the 8051 returns the value from the memory location indicated by SP and then decrements the value of SP. This order of operation is important. When the 8051 is initialized SP will be initialized to 07h. If you immediately push a value onto the stack, the value will be stored in Internal RAM address 08h. This makes sense taking into account what was mentioned two paragraphs above: First the 8051 will increment the value of SP (from 07h to 08h) and then will store the pushed value at that memory address (08h). SP is modified directly by the 8051 by six instructions: PUSH, POP, ACALL, LCALL, RET, and RETI. It is also used intrinsically whenever an interrupt is triggered (more on interrupts later. Don’t worry about them for now!).

The 8051 is a flexible microcontroller with a relatively large number of modes of operations. Your program may inspect and/or change the operating mode of the 8051 by manipulating the values of the 8051's Special Function Registers (SFRs).
**SFRs** are accessed as if they were normal Internal RAM. The only difference is that Internal RAM is from address 00h through 7Fh whereas SFR registers exist in the address range of 80h through FFh.

Each SFR has an address (80h through FFh) and a name. As you can see, although the address range of 80h through FFh offers 128 possible addresses, there are only 21 SFRs in a standard 8051. All other addresses in the SFR range (80h through FFh) are considered invalid. Writing to or reading from these registers may produce undefined values or behavior.

**SFR Types**

As mentioned in the chart itself, the SFRs that have a blue background are SFRs related to the I/O ports. The 8051 has four I/O ports of 8 bits, for a total of 32 I/O lines. Whether a given I/O line is high or low and the value read from the line are controlled by the SFRs in green. The SFRs with yellow backgrounds are SFRs which in some way control the operation or the configuration of some aspect of the 8051. For example, TCON controls the timers, SCON controls the serial port. The remaining SFRs, with green backgrounds, are "other SFRs." These SFRs can be thought of as auxiliary SFRs in the sense that they don't directly configure the 8051 but obviously the 8051 cannot operate without them. For example, once the serial port has been configured using SCON, the program may read or write to the serial port using the SBUF register.

**TCON** (Timer Control, Addresses 88h, Bit-Addressable):-

The Timer Control SFR is used to configure and modify the way in which the 8051's two timers operate. This SFR controls whether each of the two timers is running or stopped and contains a flag to indicate that each timer has overflowed. Additionally, some non-timer related bits are located in the TCON SFR. These bits are used to configure the way in which the external interrupts are activated and also contain the external interrupt flags which are set when an external interrupt has occurred.

**TMOD** (Timer Mode, Addresses 89h):

The Timer Mode SFR is used to configure the mode of operation of each of the two timers. Using this SFR your program may configure each timer to be a 16-bit timer, an 8-bit auto-reload timer, a 13-bit timer, or two separate timers. Additionally, you may configure the timers to only count when an external pin is activated or to count "events" that are indicated on an external pin.
TL0/TH0 (Timer 0 Low/High, Addresses 8Ah/8Ch):

These two SFRs, taken together, represent timer 0. Their exact behavior depends on how the timer is configured in the TMOD SFR; however, these timers always count up. What is configurable is how and when they increment in value.

TL1/TH1 (Timer 1 Low/High, Addresses 8Bh/8Dh):

These two SFRs, taken together, represent timer 1. Their exact behavior depends on how the timer is configured in the TMOD SFR; however, these timers always count up. What is configurable is how and when they increment in value.

SCON (Serial Control, Addresses 98h, Bit-Addressable):

The Serial Control SFR is used to configure the behavior of the 8051's on-board serial port. This SFR controls the baud rate of the serial port, whether the serial port is activated to receive data, and also contains flags that are set when a byte is successfully sent or received.

SBUF (Serial Control, Addresses 99h):

The Serial Buffer SFR is used to send and receive data via the on-board serial port. Any value written to SBUF will be sent out the serial port's TXD pin. Likewise, any value which the 8051 receives via the serial port's RXD pin will be delivered to the user program via SBUF. In other words, SBUF serves as the output port when written to and as an input port when read from.

2.1.4. 8051 Ports Structures And Operations

There are four ports P0, P1, P2 and P3 each use 8 pins, making them 8-bit ports. All the ports upon RESET are configured as output, ready to be used as output ports. To use any of these ports as an input port, it must be programmed. A logic state (voltage) on any pin can be changed or
read at any moment. A logic zero (0) and logic one (1) are not equal. Logic one (0) represents almost short circuit to ground. Such a pin is configured as output. A logic one (1) is “loosely” connected to voltage power supply through resistors of high resistance. Since this voltage can be easily “pulled down” by an external signal, such a pin is configured as input.

Port 0:

It occupies a total of 8 pins (pins 32-39). It can be used for input or output. To use the pins of port 0 as both input and output ports, each pin must be connected externally to a 10K ohm pull-up resistor. This is due to the fact that P0 is an open drain, unlike P1, P2, and P3. Open drain is a term used for MOS chips in the same way that open collector is used for TTL chips. With external pull-up resistors connected upon reset, port 0 is configured as an output port. Another characteristic is expressed when it is configured as output. Namely, unlike other ports consisting of pins with embedded pull-up resistor (connected by its end to 5 V power supply), this resistor is left out here. This, apparently little change has its consequences: If any pin on this port is configured as input then it performs as if it “floats”. Such input has unlimited input resistance and has no voltage coming from “inside”.

Port 1:

Port 1 occupies a total of 8 pins (pins 1 through 8). It can be used as input or output. In contrast to port 0, this port does not need any pull-up resistors since it already has pull-up resistors internally. Upon reset, Port 1 is configured as an output port. This is a true I/O port, because there are no role assigning as it is the case with P0. Since it has embedded pull-up resistors it is completely compatible with TTL circuits.

Port 2:

Similar to P0, when using external memory, lines on this port occupy addresses intended for external memory chip. This time it is the higher address byte with addresses A8-A15. When there is no additional memory, this port can be used as universal input-output port similar by its features to the port 1. Port 2 occupies a total of 8 pins (pins 21-28). It can be used as input or output. Just like P1, P2 does not need any pull-up resistors since it already has pull-up resistors internally. Upon reset, Port 2 is configured as an output port.

Port 3:

Even though all pins on this port can be used as universal I/O port, they also have an alternative function. Since each of these functions use inputs, then the appropriate pins have to be configured like that. In other words, prior to using some of reserve port functions, a logical one (1) must be written to the appropriate bit in the P3 register. From hardware’s perspective, this port is also similar to P0, with the difference that its outputs have a pull-up resistor embedded. It occupies a total of 8 pins, pins 10 through 17. It can be used as input or output. P3 does not need any pull-up resistors, the same as P1 and P2 did not. Although port 3 is configured as an output port upon reset, port 3 has the additional function of providing some extremely important signals such as interrupts. This information applies both 8051 and 8031 chips system.
2.1.5 AT 89S51 Timers:

The 8051 comes equipped with two timers, both of which may be controlled, set, read, and configured individually. The 8051 timers have three general functions: 1) Keeping time and/or calculating the amount of time between events, 2) Counting the events themselves, or 3) Generating baud rates for the serial port.

The three timer uses are distinct so we will talk about each of them separately. The first two uses will be discussed in this chapter while the use of timers for baud rate generation will be discussed in the chapter relating to serial ports.

Timer SFRs

As mentioned before, the 8051 has two timers which each function essentially the same way. One timer is TIMER0 and the other is TIMER1. The two timers share two SFRs (TMOD and TCON) which control the timers, and each timer also has two SFRs dedicated solely to itself (TH0/TL0 and TH1/TL1). We’ve given SFRs names to make it easier to refer to them, but in reality an SFR has a numeric...
The SFRs relating to timers are:

<table>
<thead>
<tr>
<th>SFR Name</th>
<th>Description</th>
<th>SFR Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>TH0</td>
<td>Timer 0</td>
<td>8Ch</td>
</tr>
<tr>
<td>TL0</td>
<td>Timer 0</td>
<td>8Ah</td>
</tr>
<tr>
<td>TH1</td>
<td>Timer 1</td>
<td>8Dh</td>
</tr>
<tr>
<td>TL1</td>
<td>Timer 1</td>
<td>8Bh</td>
</tr>
<tr>
<td>TCON</td>
<td>Timer Control</td>
<td>88h</td>
</tr>
<tr>
<td>TMOD</td>
<td>Timer Mode</td>
<td>89h</td>
</tr>
</tbody>
</table>

**USING TIMERS TO MEASURE TIME**

Obviously, one of the primary uses of timers is to measure time. We will discuss this use of timers first and will subsequently discuss the use of timers to count events. When a timer is used to measure time it is also called an "interval timer" since it is measuring the time of the interval between two events.

**13-bit Time Mode (mode 0)**

Timer mode "0" is a 13-bit timer. This is a relic that was kept around in the 8051 to maintain compatibility with its predecessor, the 8048. Generally the 13-bit timer mode is not used in new development. When the timer is in 13-bit mode, TLx will count from 0 to 31. When TLx is incremented from 31, it will "reset" to 0 and increment THx. Thus, effectively, only 13 bits of the two timer bytes are being used: bits 0-4 of TLx and bits 0-7 of THx. This also means, in essence, the timer can only contain 8192 values. If you set a 13-bit timer to 0, it will overflow back to zero 8192 machine cycles later.

**16-bit Time Mode (mode 1)**

Timer mode "1" is a 16-bit timer. This is a very commonly used mode. It functions just like 13-bit mode except that all 16 bits are used. TLx is incremented from 0 to 255. When TLx is incremented from 255, it resets to 0 and causes THx to be incremented by 1. Since this is a full 16-bit timer, the timer may contain up to 65536 distinct values. If you set a 16-bit timer to 0, it will overflow back to 0 after 65,536 machine cycles later.

**8-bit Time Mode (mode 2)**

Timer mode "2" is an 8-bit auto-reload mode. When a timer is in mode 2, THx holds the "reload value" and TLx is the timer itself. Thus, TLx starts counting up. When TLx reaches 255
and is subsequently incremented, instead of resetting to 0 (as in the case of modes 0 and 1), it will be
reset to the value stored in THx.

2.4. MOTOR DRIVER IC:

The L293 and L293D are quadruple high-current half-H drivers. The L293 is designed to provide
bidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. The L293D is designed to
provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices
are designed to drive inductive loads such as relays, solenoids, dc and bipolar stepping motors, as
well as other high-current/high-voltage loads in positive-supply applications.

All inputs are TTL compatible. Each output is a complete totem-pole drive circuit, with a
Darlington transistor sink and a pseudo-Darlington source. Drivers are enabled in pairs, with drivers
1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN. When an enable input is high, the
associated drivers are enabled and their outputs are active and in phase with their inputs. When the
enable input is low, those drivers are disabled and their outputs are off and in the high-impedance
state. With the proper data inputs, each pair of drivers forms a full-H (or bridge) reversible drive
suitable for solenoid or motor applications.

On the L293, external high-speed output clamp diodes should be used for inductive transient
suppression. A VCC1 terminal, separate from VCC2, is provided for the logic inputs to minimize
device power dissipation. The L293 and L293D are characterized for operation from 0°C to 70°C.
Features:

Featuring Unit rode L293 and L293D
Wide Supply-Voltage Range: 4.5 V to 36 V
Separate Input-Logic Supply
Internal ESD Protection
Thermal Shutdown
High-Noise-Immunity Inputs
Functional Replacements for SGS L293 and SGS L293D
Output Current 1 A Per Channel (600 mA for L293D)
Peak Output Current 2 A Per Channel (1.2 A for L293D)
Output Clamp Diodes for Inductive
Transient Suppression (L293D)

Fig. 2.5:- Pin Configuration of L293D, Motor Driver IC

VOICE MODULE
EasyVR is the second generation version of the successful VRbot Module. It is a multi-purpose speech recognition module designed to easily add versatile, robust and cost effective speech recognition capabilities to virtually any application.

The EasyVR module can be used with any host with an UART interface powered at 3.3V – 5V, such as PIC and Arduino boards. Some application examples include home automation, such as voice controlled light switches, locks or beds, or adding “hearing” to the most popular robots on the market.

**features**

- A host of built-in Speaker Independent (SI) commands for ready to run basic controls, in the followings languages:
  - English (US)
  - Italian
  - German
  - French
  - Spanish
  - Japanese
- Supports up to 32 user-defined Speaker Dependent (SD) triggers or commands as well as Voice Passwords. SD custom commands can be spoken in ANY language.
- Easy-to-use and simple Graphical User Interface to program Voice Commands and audio.
- Module can be used with any host with an UART interface (powered at 3.3V - 5V)
- Simple and robust documented serial protocol to access and program through the host board
- 3 GPIO lines (IO1, IO2, IO3) that can be controlled by new protocol commands.
- PWM audio output that supports 8 ohm speakers.
- Sound playback feature.
Physical dimensions and pin assignment

<table>
<thead>
<tr>
<th>Connector</th>
<th>Number</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>GND</td>
<td>Ground</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VCC</td>
<td>Voltage DC input</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E5X</td>
<td>Serial Data Receive (TTL level)</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ETX</td>
<td>Serial Data Transmit (TTL level)</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>J2</td>
<td>PWM</td>
<td>Differential audio output (can directly drive 3.5 speaker)</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>J3</td>
<td>MIC_RET</td>
<td>Microphone reference ground</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MIC_IN</td>
<td>Microphone input signal</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>J4</td>
<td>IO1</td>
<td>General purpose IO</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IO2</td>
<td>General purpose IO</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IO3</td>
<td>General purpose IO</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

COMMAND DETAILS

Format of command strings accepted by the module. Please note that numeric arguments of command requests are mapped to upper-case letters (see above section).

CMD_BREAK
Abort recognition, training or playback in progress if any or do nothing

<table>
<thead>
<tr>
<th>'b'</th>
<th>Known issues:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(62h)</td>
<td>In firmware ID 0, any other character received during recognition will prevent this command from stopping recognition, that will continue until timeout or other recognition results.</td>
</tr>
</tbody>
</table>

**Expected replies:** STS_SUCCESS, STS_INTERR

### CMD_SLEEP

<table>
<thead>
<tr>
<th>'s'</th>
<th>Go to the specified power-down mode</th>
</tr>
</thead>
</table>

Sleep mode (0-8):

0 = wake on received character only
1 = wake on whistle or received character

**Expected replies:** STS_SUCCESS

### CMD_KNOB

<table>
<thead>
<tr>
<th>'k'</th>
<th>Set SI knob to specified level</th>
</tr>
</thead>
</table>

Confidence threshold level (0-4):

0 = loosest: more valid results
2 = typical value (default)

**Expected replies:** STS_SUCCESS

### CMD_LEVEL

<table>
<thead>
<tr>
<th>'v'</th>
<th>Set SD level</th>
</tr>
</thead>
</table>

Strictness control setting (1-5):

1 = easy
2 = default

**Expected replies:** STS_SUCCESS
### CMD_LANGUAGE

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'T' (Col)</td>
<td>Set SI language</td>
</tr>
</tbody>
</table>

**Language:**

- 0 = English
- 1 = Italian
- 2 = Japanese

**Expected replies:** STS_SUCCESS
### CMD_TIMEOUT

<table>
<thead>
<tr>
<th>'o'</th>
<th>Set recognition timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>Timeout (-1 = default, 0 = infinite, 1-31 = seconds)</td>
</tr>
</tbody>
</table>

**Expected replies:** STS_SUCCESS

### CMD_RECOG_SI

<table>
<thead>
<tr>
<th>'i'</th>
<th>Activate SI recognition from specified wordset</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>Wordset index (0-3)</td>
</tr>
</tbody>
</table>

**Expected replies:** STS_SIMILAR, STS_TIMEOUT, STS_ERROR

### CMD_TRAIN_SD

<table>
<thead>
<tr>
<th>'t'</th>
<th>Train specified SD/SV command</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>Group index (0 = trigger, 1-15 = generic, 16 = password)</td>
</tr>
<tr>
<td>[2]</td>
<td>Command position (0-31)</td>
</tr>
</tbody>
</table>

**Expected replies:** STS_SUCCESS, STS_RESULT, STS_SIMILAR, STS_TIMEOUT, STS_ERROR

### CMD_GROUP_SD

<table>
<thead>
<tr>
<th>'g'</th>
<th>Insert new SD/SV command</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>Group index (0 = trigger, 1-15 = generic, 16 = password)</td>
</tr>
<tr>
<td>[2]</td>
<td>Position (0-31)</td>
</tr>
</tbody>
</table>

**Expected replies:** STS_SUCCESS, STS_OUT_OF_MEM

### CMD_UNGROUP_SD

<table>
<thead>
<tr>
<th>'u'</th>
<th>Remove SD/SV command</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>Group index (0 = trigger, 1-15 = generic, 16 = password)</td>
</tr>
<tr>
<td>[2]</td>
<td>Position (0-31)</td>
</tr>
</tbody>
</table>

**Expected replies:** STS_SUCCESS

### CMD_RECOG_SD
**CMD_ACTIVATE_SD**

<table>
<thead>
<tr>
<th>'d'</th>
<th>Activate SD/SV recognition</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>Group index (0 = trigger, 1-15 = generic, 16 = password)</td>
</tr>
</tbody>
</table>

**Expected replies**: STS_RESULT, STS_SIMILAR, STS_TIMEOUT, STS_ERROR

**CMD_ERASE_SD**

<table>
<thead>
<tr>
<th>'e'</th>
<th>Erase training of SD/SV command</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>Group index (0 = trigger, 1-15 = generic, 16 = password)</td>
</tr>
<tr>
<td>[2]</td>
<td>Command position (0-31)</td>
</tr>
</tbody>
</table>

**Expected replies**: STS_SUCCESS
### CMD_NAME_SD

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'n'</td>
<td>Label SD/SV command</td>
</tr>
<tr>
<td>[1]</td>
<td>Group index (0 = trigger, 1-15 = generic, 16 = password)</td>
</tr>
<tr>
<td>[2]</td>
<td>Command position (0-31)</td>
</tr>
<tr>
<td>[3]</td>
<td>Length of label (0-31)</td>
</tr>
<tr>
<td>[4-n]</td>
<td>Text for label (ASCII characters from 'A' to &quot;)</td>
</tr>
</tbody>
</table>

**Expected replies:** STS_SUCCESS

### CMD_COUNT_SD

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'c'</td>
<td>Request count of SD/SV commands in the specified group</td>
</tr>
<tr>
<td>[1]</td>
<td>Group index (0 = trigger, 1-15 = generic, 16 = password)</td>
</tr>
</tbody>
</table>

**Expected replies:** STS_COUNT

### CMD_DUMP_SD

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'p'</td>
<td>Read SD/SV command data (label and training)</td>
</tr>
<tr>
<td>[1]</td>
<td>Group index (0 = trigger, 1-15 = generic, 16 = password)</td>
</tr>
<tr>
<td>[2]</td>
<td>Command position (0-31)</td>
</tr>
</tbody>
</table>

**Expected replies:** STS_DATA

### CMD_MASK_SD

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'m' (6Dh)</td>
<td>Request bit-mask of non-empty groups</td>
</tr>
</tbody>
</table>

**Expected replies:** STS_MASK

### CMD_RESETALL

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'r'</td>
<td>Reset all commands and groups</td>
</tr>
<tr>
<td>'R'</td>
<td>Confirmation character</td>
</tr>
</tbody>
</table>

**Expected replies:** STS_SUCCESS
### CMD_ID

<table>
<thead>
<tr>
<th>'x' (78h)</th>
<th>Request firmware identification</th>
</tr>
</thead>
</table>

**Expected replies:** STS_ID

### CMD_DELAY

<table>
<thead>
<tr>
<th>'y' (79h)</th>
<th>Set transmit delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>Time (0-10 = 0-10 ms, 11-19 = 20-100 ms, 20-28 = 200-1000 ms)</td>
</tr>
</tbody>
</table>

**Expected replies:** STS_SUCCESS
**CMD_BAUDRATE**

<table>
<thead>
<tr>
<th>'a'</th>
<th>Set communication baud-rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>(61h)</td>
<td>Speed mode:</td>
</tr>
<tr>
<td></td>
<td>1 = 115200</td>
</tr>
<tr>
<td></td>
<td>2 = 57600</td>
</tr>
</tbody>
</table>

**Expected replies:** STS_SUCCESS

**CMD_QUERY_IO**

<table>
<thead>
<tr>
<th>'q'</th>
<th>Configure, query or modify general purpose I/O pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>Pin number (1 = pin IO1, 2 = pin IO2, 3 = pin IO3)</td>
</tr>
<tr>
<td></td>
<td>Pin mode (0 = output low, 1 = output high, 2 = input*, 3 = input strong**, 4 = input weak***)</td>
</tr>
<tr>
<td></td>
<td>* High impedance input (no pull-up)</td>
</tr>
</tbody>
</table>

**Expected replies:** STS_SUCCESS (mode 0-1), STS_PIN (mode 2-4)

**CMD_PLAY_SX**

<table>
<thead>
<tr>
<th>'w'</th>
<th>Wave table entry playback</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1-2]</td>
<td>Two 5-bit values that form a 10-bit index to the sound table (index = [1] * 32 + [2])</td>
</tr>
<tr>
<td>[3]</td>
<td>Playback volume (0-31, 0 = min volume, 15 = full scale, 31 = double gain)</td>
</tr>
</tbody>
</table>

**Expected replies:** STS_SUCCESS, STS_ERROR

**CMD_DUMP_SX**

| 'h'   | Read wave table data                                 |

**Expected replies:** STS_TABLE_SX, STS_OUT_OF_MEM

2.7 **DC Gear Motor**:
DC motors in vehicle convert chemical energy into mechanical energy
Constant mechanical power output or constant torque
Rapid acceleration or deceleration
Extensively used as a positioning device because its speed as well as torque can be controlled precisely over a wide range
12v, 35rpm
Applications – in automobiles, robots, VCRs, movie camera, electric vehicles, in steel and aluminum rolling mills, electric trains, overhead cranes, control devices, etc.

Fig. dc gear motor

3. BLOCK DIAGRAM

![Block Diagram](extremeelectronics.co.in)
3.1. BLOCK DIAGRAM DISCRIPTION

Here we use a voice recognition module to control performance of the micro controller and there by control the movement or rotation of the motor. This is the basic working principle of the ‘voice controlled wheel chair’.

Voice is given to the voice recognition module through a microphone. Voice recognition module is trained by giving the 5 commands. The five commands are converted into hex file. These hex file points 5 address locations of AT89s52 microcontroller. When command is given the program in the corresponding address location is executed and chair moves accordingly.

An additional joystick is also used to control the rotation of motor, this is connected to the micro controller and the micro controller is also programmed to respond the input from the joystick.
4. CIRCUIT DIAGRAM

Fig. circuit diagram of voice controlled wheel chair

4.1 CIRCUIT DIAGRAM DESCRIPTION
5. PCB LAYOUT

Fig.5.1. PCB Layout

5.1. PCB

5.1.1. Circuit

Nowadays printed circuit boards are mentioned as PBCs. Making the electronic circuit manufacturing as easy one. In olden days vast area was required to implement a small circuit to connect the leads of the components and separate connectors were needed. But PCBs connect the two by copper-coated lines on the PCB boards. In the single-sided PCBs the copper layer is on both sides. Some cases, middle layer is also possible than the two sides.

The most popular board types are

SINGLE-SIDED BOARDS: They are mainly used in entertainment electronics where manufacturing costs have to be kept at the minimum.

DOUBLE-SIDED BOARDS: Double sided PCBs can be made with or without plated through holes. The production of boards with plated through holes is fairly expensive.

5.1.2. Manufacturing Process

First, the wanted circuit is drawn on paper and it is modified or designed PCB layout is to be drawn on the plain copper-coated board. These boards are available in two types: Phenolic, Glass epoxy

Most computers PCBs are glass epoxy. To draw circuit diagram we can use the black colour paints. Before that the required size of the plane PCB board is determined from the roughly drawn PCB layout. Using black paint the desired circuit is drawn on the board.

5.1.3. Layout Approaches

The first rule is to prepare each and every PCB layout as viewed from the component side. Another important rule is not to start the designing of a layout unless an absolutely clear circuit diagram is available, if necessary, with a component lists. Among the components the
larger ones are placed first and the space between is filled with smaller ones. Components requiring input / output connections come near the connectors. All components are placed in such a manner that disordering of the components is not necessary if they have to be replaced.

In the designing of a PCB layout it is very important to divide the circuit into functional subunits. Each of these subunits should be realized on a defined portion of the board. In the designing the inter connections which are usually done by pencil lines, actual space requirements in the artwork must be considered. In addition the layout can be rather roughly sketched and will still be clear enough for artwork designer.

5.1.4. Board Cleaning

The cleaning of the copper surface prior to resist applications in an essential step for any types of PCB process using etch or plating resist. Insufficient cleaning is one of the reasons most often encountered for difficulties in PCB fabrication although it might not always be immediately recognized as this. But it is quiet often the reasons of poor-resist adhesion, uneven photo-resist films, pinholes, poor plating adhesion, etc. Where cleaning has to be done with simplest means or only for a limited quantity of PCBs, manual-cleaning process is mainly used. In the process we require just a sink with running water, pumice powder, scrubbing brushes and suitable tanks.

5.1.5. Screen Printing

This process is particularly suitable for large production schemes. However the preparation of a screen can also be economically attractive for series of 1000 PCBs. Below, while photo printing is basically the non-accurate method to transfer a pattern on to a board surface. With the screen-printing process one can produce PCBs with a conduction of as low as 0.5 + or – and a registration error of 0.1mm on an industrial scale with a high reliability. In its basic form, a screen fabric with uniform meshes and opening is stretched and fixed on a solid frame of metal or wood. The circuit pattern is photographically transferred onto the screen, leaving the meshes in the pattern open, while the meshes in the rest of the area are closed. In the actual printing step, ink forced by the moving squeegee through the open meshes onto the surface of the material to be printed.

5.1.6. Plating

From a practical stand port, printed circuit boards may have to be stocked before being taken for assembly of components. It is expected that the circuit board retain its solder ability for long periods of several months so that reliable solder joints can be produced during assembly. Plating of a metal can be accomplished on a copper pattern by three methods. They are:
Immersion plating
Electro less plating
Electroplating

5.1.7. Etching

This can be done both by manual and mechanical ways by immersing the board onto a solution of formic chloride and hydrochloric acid and finally cleaning the board by soap. In all subtractive PCB processes, etching is one of the most important steps. The copper pattern is formed by selective removal of all unwanted copper, which is not protected by an etch resist. This looks very simple at first glance but in practice there are factors like under etching and overhang which complicate the matter especially in the production of fine and highly precise PCBs. Etching of PCBs as required in modern electronic equipment production, is usually done in spray type etching machines.

5.1.8. Components Placing

The actual location of components in the layout is responsible for the problems to be placed during routing of the interconnections. In a highly sensitive circuit the critical components are placed first and in such a manner as to require minimum length for the critical conductors. In less critical circuit the components are arranged exactly in the order of signal flow. This will result in a minimum overall conductor length. In a circuit where a few components have considerably more connecting points than the others. These key components are placed first and the remaining ones are grouped around them.

The general result to be aimed at is always to get shortest possible interconnections. The bending of the axial component leads is done in a manner to guarantee an optimum retention of the component of the PCB. The lead bending radius should be approximately two times the lead diameter. Horizontally mounted resistors must touch the board surface to avoid lifting of solder joints along with the copper pattern under pressure on the resistor body. Vertically mounted resistors should not be flush to the board surface to avoid strain on the solder joints as well as on the component lead junction due to different thermal expansion coefficients of lead and board materials, where necessary resilient spaces to be provided. Coated or sealed components should to be mounted in such to provide a certain length along the leads. Especially when plated through holes where the solder flows up in the hole, clean lead of at least 1 mm above the board are recommended.

5.1.9. Drilling

Drilling of component mounting holes into PCBs is by far the most important mechanical machining operation in PCB production processes. Holes are made by drilling wherever a superior hole finish for plated-through hole processes is required and where the tooling costs for a punching tool cannot be justified. Therefore drilling is applied for all the professional grade PCB manufacturers and generally in smaller PCB production laboratories. The importance of hole drilling into PCB has further gone up with electronic
component miniaturization and it needs smaller hole diameters and higher package density where hole punching is practically ruled out.

5.1.10. Soldering

Soldering is a process for the joining of metal parts with the aid of a molten metal (solder), where the melting temperature is suited below that of the material joined, and whereby the surface of the parts are wetted, without then becoming molten. Soldering generally implied that the joining process occurs at temperatures below 450-degree centigrade. Solder wets and alloys with the base metals and gets drawn, by capillary action, into the gap between them. This process forms a metallurgical bond between the parts of the joint. Therefore soldier act by Wetting of base metal surfaces forming joint flowing between these surfaces, which result in a completely filled space between them. Metallurgical bonding to these surfaces when soldered. Soldering consists of the relative positioning of the surfaces to be joined, wetting these surfaces with molten solder and allowing the solder to cool down until it has solidified. During these soldering operation, an auxiliary medium is mostly used to increase the flow properties of molten solder or to improve the degree of wetting. Such a medium is called flux.
Flowing characteristics are required in a flux:
It should provide a liquid cover over the materials and exclusive air up
To the soldering temperature.
It should dissolve any oxide on the metal surface or on the solder and carry such unwanted elements away.
It should be readily displaced from the metal by the molten soldering Operation.
Residues should be removable after completion of the solder. To achieve a soldered joint the solder and the base metal must be heated above the melting point of the solder used. The method by which the necessary heat is applied, among other things depends on:
Nature and type of the joint
Melting point of the solder

5.1.11. Flux

Generally applied soldering methods are iron soldering, torch soldering, mass soldering, electrical soldering furnace soldering and other methods. Components are basically mounted only one side of the board. In double-sided PCBs, the component side is usually opposite to the major conductor pattern side, unless otherwise dictated by special design requirements. The performance and reliability of solder joints give best result covered with solder and herewith contributing to the actual solder connections. However, lead cutting after soldering is still common in particular in smaller industries where hand soldering is used. With the soldered PCB many contaminants can be found which may produce. Difficulties with the functioning of the circuit. The problems usually arise at a much later than during the final functioning testing of the board in the factory. Among the contaminants, we can typically find flux, chips of plastics, metals and other constructional materials, plating sails, oils greases environmental soil and other processing materials.

The following performances are expected from the cleaning procedure with the appropriate cleaning medium:
Dissolution or dissolving of organic liquids and solids, e.g., oils, greases, resin flux.
Removal of plating salts and silicone oils.
Displacing of particulate and other insoluble matters, e.g., chips, dust, and lint.
No severe attacks on boards and components to be cleaned, no alteration of ink or paint notations and last but not the least, compatibility with healthy environmental working conditions.
5.1.12. **CAD in Lab**

First the PCB layout is designed by ORCAD. The printout is taken from the computer (of large size) for out clearance. This layer is given to the photography section to get the layout. This photographic image is exposed in the following three methods:

- Polybluem
- Chromium
- Five star

The exposed mesh is placed on plain copper coated board in correct alignment by using wooden clamps paint flow through the board and the layout lines are made on the boards.

6. **SOFTWARE DISCRIPITION**

6.1. **KIEL C**:

Keil compiler is a software used where the machine language code is written and compiled. After compilation, the machine source code is converted into hex code which is to be dumped into the microcontroller for further processing. Keil compiler also supports C language code.

**STEPS TO WRITE AN ASSEMBLY LANGUAGE PROGRAM IN KEIL AND HOW TO COMPILE IT:**

1. Install the Keil Software in the PC in any of the drives.

2. After installation, an icon will be created with the name “Keil uVision3”. Just drag this icon onto the desktop so that it becomes easy whenever you try to write programs in keil.

3. Double click on this icon to start the keil compiler.

4. A page opens with different options in it showing the project workspace at the leftmost corner side, output window in the bottom and an ash coloured space for the program to be written.
5. Now to start using the keil, click on the option “project”.

6. A small window opens showing the options like new project, import project, open project etc. Click on “New project”.

7. A small window with the title bar “Create new project” opens. The window asks the user to give the project name with which it should be created and the destination location. The project can be created in any of the drives available. You can create a new folder and then a new file or can create directly a new file.

8. After the file is saved in the given destination location, a window opens where a list of vendors will be displayed and you have to select the device for the target you have created.

9. The most widely used vendor is Atmel. So click on Atmel and now the family of microcontrollers manufactured by Atmel opens. You can select any one of the microcontrollers according to the requirement.

10. When you click on any one of the microcontrollers, the features of that particular microcontroller will be displayed on the right side of the page. The most appropriate microcontroller with which most of the projects can be implemented is the AT89C51. Click on this microcontroller and have a look at its features. Now click on “OK” to select this microcontroller.

11. A small window opens asking whether to copy the startup code into the file you have created just now. Just click on “No” to proceed further.

12. Now you can see the TARGET and SOURCE GROUP created in the project workspace.
13. Now click on “File” and in that “New”. A new page opens and you can start writing program in it.

14. After the program is completed, save it with any name but with the .asm extension. Save the program in the file you have created earlier.

15. You can notice that after you save the program, the predefined keywords will be highlighted in bold letters.

16. Now add this file to the target by giving a right click on the source group. A list of options open and in that select “Add files to the source group”. Check for this file where you have saved and add it.

17. Right click on the target and select the first option “Options for target”. A window opens with different options like device, target, output etc. First click on “target”.

18. Since the set frequency of the microcontroller is 11.0592 MHz to interface with the PC, just enter this frequency value in the Xtal (MHz) text area and put a tick on the Use on-chip ROM. This is because the program what we write here in the keil will later be dumped into the microcontroller and will be stored in the inbuilt ROM in the microcontroller.

19. Now click the option “Output” and give any name to the hex file to be created in the “Name of executable” text area and put a tick to the “Create HEX file” option present in the same window. The hex file can be created in any of the drives. You can change the folder by clicking on “Select folder for Objects”.

20. Now to check whether the program you have written is errorless or not, click on the icon exactly below the “Open file” icon which is nothing but Build Target icon. You can even use the shortcut key F7 to compile the program written.

21. To check for the output, there are several windows like serial window, memory window, project window etc. Depending on the program you have written, select the appropriate window to see the output by entering into debug mode.

22. The icon with the letter “d” indicates the debug mode.

23. Click on this icon and now click on the option “View” and select the appropriate window to check
for the output. 24. After this is done, click the icon “debug” again to come out of the debug mode. 25. The hex file created as shown earlier will be dumped into the microcontroller with the help of another software called Proload.

Fig. 6.1:- Screenshot of Keil C51 Compiler
6.2 EASY VR COMMANDER

The Easy VR Commander software can be used to easily configure your EasyVR module connected to your PC through an adapter board, or by using the microcontroller host board with the provided “bridge” program (available for ROBONOVA controller board, Arduino 2009/UNO, Parallax Basic Stamp). The user can define groups of commands or passwords and generate a basic code template to handle them. It is required to edit the generated code to implement the application logic, but the template contains all the functions or subroutines to handle the speech recognition tasks.

Fig.6.2 screen shot of EASY VR COMMANDER

Speech recognition
The recognition function of the EasyVR works on a single group at a time, so that users need to group together all the commands that they want to be able to use at the same time.

When EasyVR Commander connects to the module, it reads back all the user-defined commands and groups, which are stored into the EasyVR module non-volatile memory.

The user can add a new command by first selecting the group in which the command needs to be created and then using the toolbar icons or the “Edit” menu.

A command should be given a label and then it should be trained twice with the user’s voice: the user will be guided throughout this process (see Figure 2) when the "Train Command" action is invoked.

![Command Training](image)

**Figure 2 – Guided training dialog**

If any error happens, command training will be cancelled. Errors may happen when the user’s voice is not heard correctly, there is too much background noise or when the second word heard is too different from the first one.

The software will also alert if a command is too similar to another one by specifying the index of the conflicting command in the "Conflict" column. For example, in the following Figure 3 the command "TEST_CMD_ONE" sounds too similar to "TEST_CMD_ZERO" (i.e. they have been trained with a similar pronunciation).
Figure 3 – Conflicting commands

The current status is displayed in the EasyVR Commander list view where groups that already contain commands are highlighted in bold.

The selected group of commands can also be tested, by using the icon on the toolbar or the “Tools” menu, to make sure the trained commands can be recognized successfully.
Download a new Sound Table

The EasyVR can also play one of the sounds or sentences saved on the internal flash memory. A predefined “beep” sound is also always available, even when no sounds have been downloaded to the module.

The custom sounds are organized in a so-called “sound table”, that users can prepare and build with the special QuickSynthesis™ tool. Please refer to this application's own manual for details about creation of a sound table.

Once the sound table has been created, it can be processed by the EasyVR Commander and downloaded to the module. The user must first disconnect from the module and do the steps required to start it in “boot-mode” (see the section FLASH UPDATE).

Now the command “Update Sound Table” is enabled, either on the toolbar or the “Tools” menu, and it can be used to start the update process.
First the user will be prompted to open the QuickSynthesis project file just created and a new sound table will be generated.

The project must have been built already with the QuickSynthesis tool, before the sound table generation can be completed successfully. If a recent build is not available the user will receive a warning message, the project can be opened in QuickSynthesis again and a fresh build started (make sure the project file has been saved before the build).

Once back in the EasyVR Commander the project can be reloaded by pressing the “Refesh” button. If the
process completes successfully, the “Download” button will be enabled and the flash update process can start.

The download process will connect at a higher speed to the EasyVR module, so the “bridge” program running on your host device might not work (in particular Robonova and Basic Stamp cannot be used for this purpose) and you might need a true “serial adapter”.

The full speed used is 230400 bit/s, but the option “Slow transfer” can be used to reduce it to 115200. One adapter that can go to full speed is the VoiceGP DevBoard. Otherwise any USB/Serial adapter with TTL/CMOS interface can be used for updating the flash.

After the download completes, a new connection can be established with the EasyVR module (in “normal-mode”) and the new sounds will be displayed by the EasyVR Commander, in the special group “SoundTable” (the last one in the list with a yellow icon).

They can be played back and tested using the “Play Sound” command on the toolbar or in the “Tools” menu. See also how to do that in your application in the code example USE CUSTOM SOUND PLAYBACK.
7. ADVANTAGES

It reduces the amount of user effort required to pull and push the wheels

Safe and secure

Users who cannot use their arms can use the voice to control the wheel chair

8. DISADVANTAGES

Limitation in Languages

Speech problems

9. FUTURE ASPECTS

GPS Navigation can be embedded into this for using this outdoor.

Movement of wheelchair can be controlled by motion of head.

10. CONCLUSION

The voice controlled wheel chair is implemented, this gives a new life to the disabled persons.

11. BIBLIOGRAPHY

WEB SITES:-

1. www.atmel.com
2. www.datasheetcatalog.com
4. www.electronicsforyou.com
5. www.datasheetarchive.com

TEXT REFERED:-

1. Linear Integrated Circuits- Ramakant A. Gayakwad
2. Electronics Devices and Circuits-J B Gupta
5. The 8051 microcontroller and embedded systems – Muhammad Ali Mazidi
6. The 8051 microcontroller architecture, programming, & applications – Kenneth J Ayala