POSITIONED TV ANTENNA BASED ON MICROCONTROLLER

DEGREE PROJECT REPORT

NAME: AHMED TARIK

ENROLLMENT: 01-113082-005

NAME: ISRAA KHALID SIDDIQUI

ENROLLMENT: 01-113082-014

SUPERVISOR:

MR. KHAWAJA QASIM MAQBOOL

DEPARTMENT OF GRADUATE STUDIES AND APPLIED SCIENCES

BAHRIA UNIVERSITY ISLAMABAD


بسم الله الرحمن الرحيم

BAHRIA UNIVERSITY ISLAMABAD

Dated: _____

CERTIFICATE

We accept the work contained in the degree project report title '**Channel Positioned TV Antenna Based on Microcontroller'** as a confirmation to the required standard for the partial fulfillment of the degree of BSETM.

_____

Project Coordinator

_____ _____ _____

Supervisor **Internal Examiner** **External Examiner**

_____

Head of Department

ACKNOWLEDGMENT

We are thankful to Almighty Allah for His blessings and for our parents and siblings who put-up with us till the submission of our final year project. We gratefully acknowledge the contribution, support and help of our supervisor Mr.Khawaja Qasim Maqbool who helped us towards the completion of our project. He contributed a lot in the completion of our project. Thank you Sir.

And without our team work, this project would not have been a success. Our hard work and continuous struggle made this project to reach its destination. Bahria University provided us a platform to complete our final year degree and final year project successfully. Thank you very much.☺

DEDICATED

We dedicate this project to our parents who are always there for us and who stood with us through thick and thin, who taught us:

"Walk on a rainbow trail,

Walk on a trail of songs,

And all about u,

Will be beauty.

There is a way out

Of every mist,

Over a rainbow trail."

Thank you for always being so loving and caring.

"**Constant hard work leads to success**"

THESIS IN BRIEF

Project Title: *Channel Positioned TV Antenna based on Microcontroller*

Undertaken by: *Ahmed Tarik*

Israa Khalid Siddiqui

Supervised by: *Mr Khawaja Qasim Maqbool*

ABSTRACT

Nowadays engineers are trying to solve daily routine problems with the concept of engineering and making the lives of people easier. Conventional 'rabbit ear' antenna has its limitation which results in loss of quality of picture and sound; therefore a device called 'Channel Positioned TV Antenna Based On Microcontroller' was designed and developed for people still using the antenna, but the conventional antenna is replaced with 'Yagi-uda' antenna that has been mounted on a servo motor and is being controlled by a 4x4 keypad that can be replaced with the remote of the television. The antenna will rotate with the help of keypad due to which the viewer won't need to rotate the antenna manually. The channels will be saved and viewed with the help of keypad.

TABLE OF CONTENTS

Table of Figures

Chapter 1 Introduction

1.1 PROJECT MOTIVATION

A visit to Rawalakot was the motivation towards making this project. Rawalakot is a hilly area in Kashmir. There are a lot of hills and mountains due to which there is a bad reception of TV channels and radio channels. The signals of mobile phones are also weak only because of the obstacles.

The communication between the towers installed needs to be in line of site with each other for the boosting of the signals and routing of the signals but due to hills and mountains, that is a big obstruction in the way of communication. And if the communication between towers is weak, the reception and signal strength of channels will also be poor.

There are other technologies too like fiber optic and coaxial cables etc but all these mediums require subscription and are costly as well; therefore, using a wireless medium that has no connection fee or monthly subscription fee is a better option.

Secondly a compromise of reception and voice was a big issue due to which the people who cannot afford fiber optic or coaxial cable subscription were unable to watch TV with a clear reception.

1.2 PROJECT OBJECTIVES

Conventionally 'rabbit ear' antennas were used by the people who watch TV using an indoor antenna. This antenna is not a very good antenna as its directivity and the signal to noise ratio as compared to other antennas is very less therefore the first objective was to replace the conventional antenna with a better antenna. An antenna whose immunity towards noise and is a better option than rabbit ear antenna. This objective was achieved by replacing the rabbit ear antenna with Yagi – Uda antenna.

The reason for selecting a yagi – uda antenna is that its directivity can be increased by increasing the number of directors. And for the practical implementation of this antenna, a Matlab code was used for the plotting of the radiation pattern of yagi – uda antenna and it was observed that by adding directors, how its directivity increases as the reception of the receiver antenna's signals is directly related to the directivity of the antenna.

Then the objective was to make a device that can rotate the antenna and the user does not require rotating the antenna manually. The reason behind this objective was that the viewer using antennas required to rotate and adjust the antenna again and again for better reception due to which the viewer was used to get fed–up of rotating the antenna and still unable to watch the channel with a better reception or sound.

1.3 PROJECT CHALLENGES

There were a lot of challenges that were faced related to the project. Firstly the PIC that was decided for the development of the project was difficult for interfacing as there was no information regarding its instruction set and special function registers in detail and there is not enough information in the datasheet of the PIC that was initially used. And there was no book available regarding the PIC16F877A.

Reasons of changing IC:

Further reasons regarding the change in PIC IC are as follows:

- PIC16F877A belongs to 16 family of PIC. It does not have many special function registers as compared to PIC 18 family.
- Another reason is that PIC 18 have a lot of available space for RAM and availability of codes for special function registers where as there are less in PIC 16F877A.
- PIC 16 is less flexible than PIC 18. PIC 16 does not support all the commands where as PIC 18 supports many of the commands.
- PIC 18 has three external interrupt pins whereas PIC 16 has only one pin. Due to more interrupt pins than PIC 16, the interface that we require regarding keypad can be easily fulfilled whereas while using PIC 16, the bits will be required to rotate serially for comparison and checking of an interrupt generation (pressing of a button). This would result in more delay regarding execution of further commands.

Due to all the reasons mentioned above, PIC18F452 is better than PIC 16F877A.

Then the second problem that became a challenge was the installation of the software for programming in PIC. The software was easily available on www.microchip.com but it was a bit tricky to install the software.

After the installation of the software, the embedding of software in the microcontroller was a challenge. The challenge was that there was no one to guide how to embed software in the microcontroller and there was no trainer for implementing hardware. And all because of fewer resources and difficulties in implementation of the project i.e. the hardware part, PIC microcontroller was changed with 89C51 because 89C51 was the only microcontroller that was taught and the related resources related to it were available easily.

Furthermore the challenge faced is regarding the stepper motor. Although the advantages of a stepper are far more than a servo motor but the biggest drawback of a stepper is that it forgets where it was previously and if one switches off and turns on the motor again, it doesn't

remembers where was the initial point from where it started due to which a feedback loop is made with it. This makes the circuitry tedious and difficult to handle. The programming also gets complex. Therefore stepper has been replaced with a servo motor.

Chapter 2 Literature Review

2.1 BACKGROUND

In rural and urban areas the method of watching television involves usage of conventional methods. Digital televisions require broadband connections and satellite receivers for subscription. Therefore, this turns out to be a major drawback of digital televisions. Whereas, conventional method using an antenna does not require such subscriptions so people feel comfortable with conventional methods.

Using conventional antenna has some limitations where the signal strength is not strong enough for all the channels. This results in poor reception and weak quality of sound. Therefore, for avoiding this problem, the antenna would be required to rotate in different directions for different channels.

About a decade ago, the rotation of antenna was done manually where the viewer had to compromise with the picture quality and sound. But nowadays, rotor antennas are available. They have the ability of rotating the antenna and how these products distinguish from other products shall be discussed in detail in Chapter 3.

The main concept of channel positioned antenna is the facility to rotate the antenna in the desired direction. After the rotation of antenna, the foremost important objective is to save and recall multiple positions at a given instance in order to obtain better reception. This task would be performed by using a remote controlled antenna where all the data is processed by the microcontroller 89C51. For the prototype, the remote controller has been replaced by a keypad.

Chapter 2 Literature Review

2.2 READING LIST

As this project is on Microcontroller, therefore, books regarding Microcontrollers and the content and examples regarding the project and microcontroller were searched and studied.

Initially we have studied the data sheet of the microcontroller that we are going to use as without studying it, we cannot interface or use it in our project. Some of the features of PIC16F877a observed and read in the datasheet are mentioned below. Appendix A

And on the same side, for programming, the book named **"PIC Microcontroller and Embedded Systems"** by Muhammad Ali Mazidi was under study.

The observation was that there were few requirements that were not being fulfilled by PIC16F877A and complete information was not available for the programming of the microcontroller so the PIC was changed to PIC18F452 because the book for the 18 family of PIC was only available and that was under study too.

After a detail study, when the programming and implementation stage came, interfacing issues started to create problems that led to the changing of the microcontroller once again. The microcontroller was changed to 89C51.

Then for 89C51 programming, book called **"Microcontroller and Embedded Systems"** by Muhammad Ali Mazidi was taken as a reference and for queries was also consulted as it is very easy but yet tedious.

Chapter 3 Project Methodology

3.1 THE CONCEPT

The product of Channel Positioned TV Antenna Based on Microcontroller consists of the following components:

Yagi – Uda antenna

The function of this part was only to observe the radiation pattern of Yagi Uda antenna and see to why there was a need for changing the conventional rabbit ear antenna with Yagi Uda antenna. Different results were analyzed with the help of internet as how the antenna's directivity enhances. And the observations observed are:

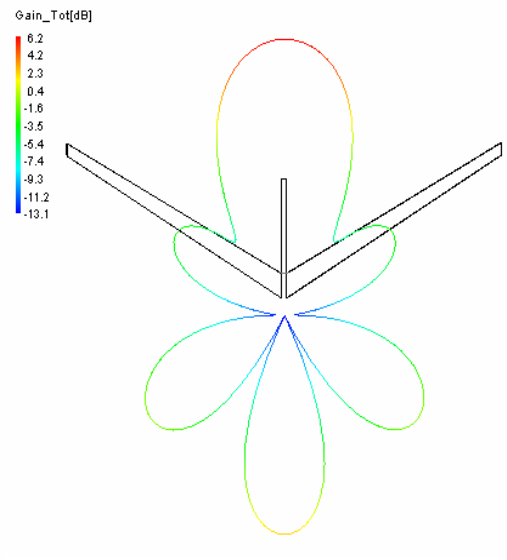


Fig 1. Radiation Pattern Rabbit Ear

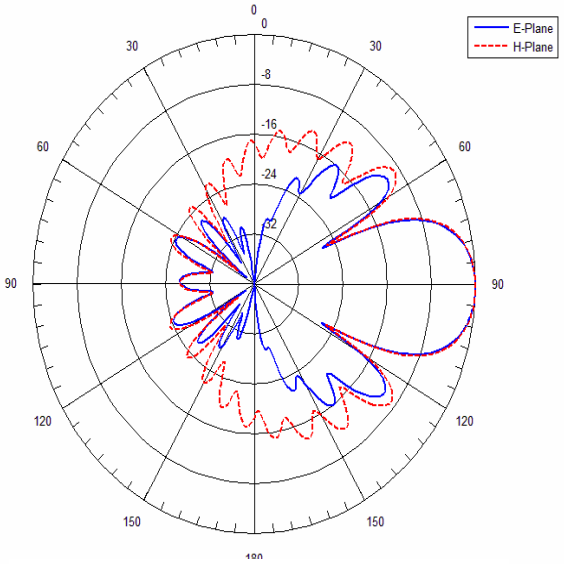


Fig 2. Radiation Pattern Yagi Uda

This can be observed that the directivity of a rabbit ear antenna (Fig 1) is more than yagi uda antenna (Fig 2) but the front to back ratio of rabbit ear antenna is less as compared to that of yagi uda antenna.

Front-to-back ratio is a factor that is very important while observing any type of antenna. Front-to-back ratio is a ratio between the main lobe and the reflected wave's lobe of an antenna. In a rabbit ear antenna, the reflected waves are a lot as there is no reflector to reflect the reflected waves towards the transmitter where as a yagi uda antenna consists of a reflector for reflecting the waves back so that maximum signal can be transmitted or received.

Chapter 3 Project Methodology

Drive Unit

The function of this unit is to drive the antenna and to give feedback to the control box in order to identify the position of the antenna. These two components are connected which results in both rotating together.

The Drive unit would consist of the following:

- Servo Motor

Control Box

This unit is to be known as the 'Heart of the product'. It is responsible for many of the functions performed by this unit. The major function of this unit is to detect the signals from remote controller to providing the necessary signal to drive the motor.

The control box would consist the following:

- Microcontroller (89C51)
- 4x4 Matrix keypad
- 8 bit LCD
- EEPROM

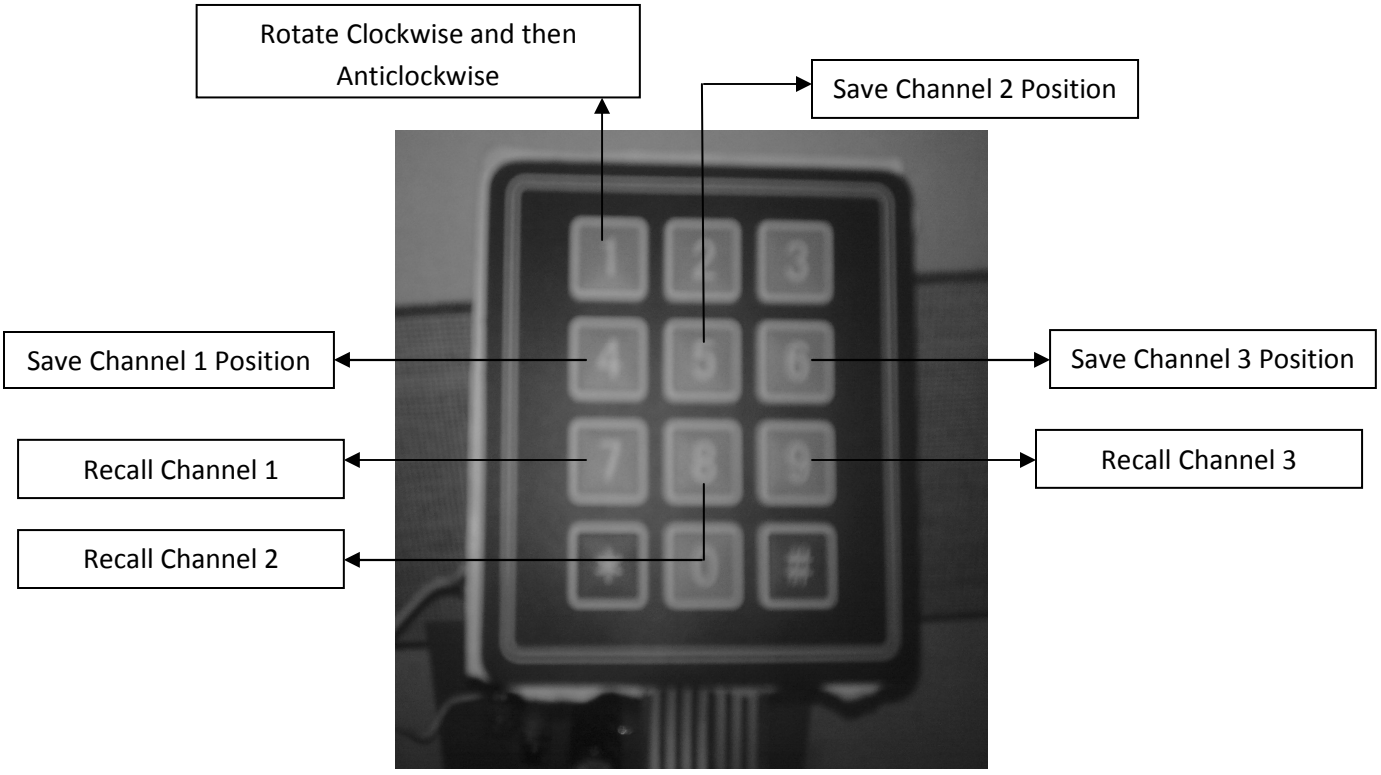


Fig 3. Prototype

Chapter 3 Project Methodology

The second and the main part of the concept is the hardware. The project or the hardware is basically a device for controlling the antenna rotation. What happens is that when one is using a conventional antenna rather than any type of cable, one has to rotate it manually. And while rotating the antenna, one has to touch it due to which the reception is good but as soon as one leaves the antenna, the reception becomes worse. This is because the extra power or the reflected back current causing interference and distortion gets grounded with the help of one's body and as soon as the person adjusting the antenna leaves it, the channel gets disturbed. This is what happens while using any type of antenna but usually while using rabbit ear antenna. And therefore, on the whole, this product has been designed and implemented so that the issue regarding the rotation of the antenna and adjusting of the antenna according to the most suitable reception can be automated and one does not need to rotate again n again. The rotation is only a click far.

In the prototype, the whole device is being controlled with the keypad. The description is as follows:

'1' for once rotating the motor clockwise and after one rotation, start rotating anti-clockwise for scanning the channels
'4' for saving channel 1 in the EEPROM for recalling afterwards
'5' for saving channel 2 in the EEPROM for recalling after wards
'6' for saving channel 3 in the EEPROM for recalling after wards
'7' for recalling channel 1 saved by the viewer in the EEPROM
'8' for recalling channel 2 saved by the viewer in the EEPROM
'9' for recalling channel 3 saved by the viewer in the EEPROM

What happens is that when the user turns on the device, there a button is interfaced with the external interrupt zero i.e. INT0. This is for starting the device. A message will appear on the LCD connected in the circuit. Message that will appear is 'Enter Password'. This means that enter the desired channel number. The user can press the desired button according to the description of the buttons mentioned above.

When a user presses '1' after the message appeared 'Enter Password', the motor starts rotating. First it rotates clockwise and then anticlockwise. This is for the scanning of the channels. As soon as the viewer gets the best reception of a channel, press '4', '5' or '6'for saving the position of the antenna in the EEPROM. Whatever channel position was saved on channel '4', '5' and '6' can be recalled by channels '7','8' and '9' respectively from the EEPROM.

Chapter 3 Project Methodology

3.2 HARDWARE DETAIL

In this section there is a detailed explanation regarding the hardware used in this project. The hardware used in this project is as follow:

- 89C51
- Servo Motor
- 4x4 Matrix keypad
- Yagi Uda Antenna
- LCD
- EEPROM

89C51 MICROCONTROLLER

The AT89C51 is a low-power, high-performance CMOS 8-bit microcomputer. It belongs to the 8051 family of Microcontrollers. Important features are:

- Provides a Flash Programmable and Erasable Read Only Memory (PEROM) of 4 Kbytes.
- Consists of 128 bytes of RAM
- There are 32 I/O lines
- Two 16-bit timer/counters
- Five vector two-level interrupt architecture
- A full duplex serial port
- On-chip oscillator and clock circuitry.

Description:

The AT89C51 is designed with such logic that its operation is down to zero frequency and supports two software power savings modes. The Idle Mode stops the CPU and allows the RAM timer serial port and interrupt system to continue functioning. The Power-down Mode saves the RAM contents but stops the oscillator disabling all other chip functions until the next hardware reset. [Appendix A]

Chapter 3 Project Methodology

SERVO MOTOR

In our project we have used servo motor to rotate the antenna in the desired position. Servo motors tend to be 10% to 30% more expensive than similar stepper system but there are numerous advantages of servo motor. No feedback loop circuit is required which doesn't make the circuitry tedious and easy to handle. Servo motors are accurate and have high resolution. Servo motor resolution depends upon the encoder used. Typical encoders produce 2,000 to 4,000 pulses per revolution and encoders with up to 10,000 pulses per revolution are available. Since servos, which are closed loop, can and do achieve the available resolution, they are able to maintain positional accuracy. These motors can produce speeds and powers two to four times than the stepper motor. Servo motors have very good torque characteristics at higher speeds where as stepper motors are very poor in this. Here is an image of a servo motor.

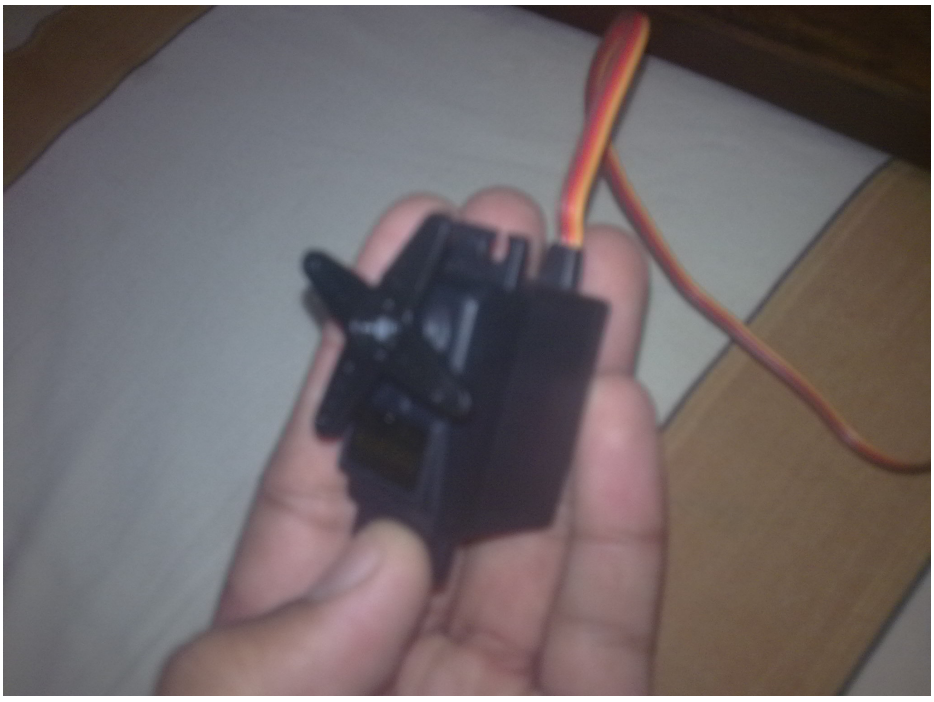


Fig 4. Servo Motor

By nature, servo motors have constant positional feedback. The positional feedback is used to correct any discrepancy between a desired and an actual position. This constant corrective action results in faster speeds and increased power up to three times at high speeds. So conclusion is that servo motors are accurate and having high resolution. Servo motors are expensive but are very helpful and makes the work easier. We have used servo motor namely (Tower_Pro) in our project and we did not face any sort of problem with servo motor. It is easily available in the market and it costs Rs.950. It is reliable and has better results.

Chapter 3 Project Methodology

4X4 MATRIX KEYPAD

Tact switches are commonly used as digital input devices. Normally one tact switch requires one digital I/P pin of a microcontroller but if you want to interface a matrix of such switches (say a 16 digit keypad) assigning a digital I/O pin for each key won't be a good idea. You need to think about the way to minimize the required number of digital I/O pins of microcontroller. A very popular method is a keypad matrix where the keys are arranged into rows and columns so that a 4×4 (16) tact switches can be interfaced to a microcontroller using only 4+4 = 8 I/O pins.

How to identify the pins of 4x4 Matrix Keypad?

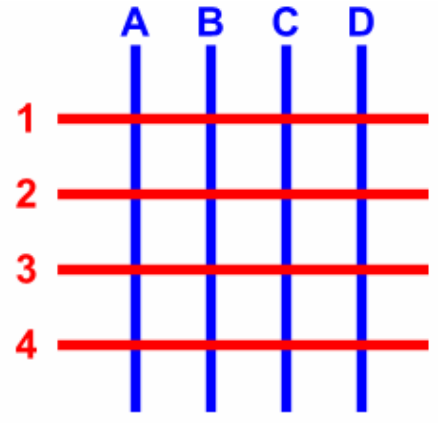
While interfacing a 4x4 keypad, the first thing that became an issue was how to interface it as at a glance, it is difficult to identify that which pins represent rows and which represent columns. After a lot of research it was found that first four pins (from left to right) represent rows and the rest of the four represent columns.

Key Matrices

In simple words the matrices are interface technique. Key matrices are also used to interface inputs like PC keyboard keys and it can also be used to control outputs like Light emitting diode. The I/O is divided into two sections:

- Columns
- Rows

Here is a 4 x 4 matrix

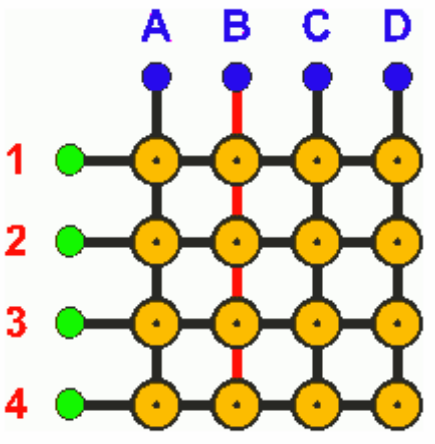


In the above figure the blue lines are representing columns whereas the red lines are representing rows. There are sixteen knots that the rows and columns intersect to each other. The columns and the rows are not in link with each other. Suppose that we want to make a key matrix. We will

connect a button to each knot. The buttons will be pressed to take a function on. When someone pushes the button, it will connect the column and the row that it belongs to.

Working

To understand the working of a keypad matrix, there are connection pins to each row and column wire. Then power will be given to only one column. Let it the column b. The red color wire indicates that it has power, and purple button indicates that the button is pressed. Then we will press to button number b3.



The column wire b has power. And the rest of them have no power, until the button b3 is pressed. This button makes contact between column b and row three. Because column B has power, and row 3 is also in power now as long as the button b3 is pressed. It means that if we know which column has power and watching the rows so we can judge which button was pressed if we detect power on row. So this is the working of a matrix keypad and it is use d in real life for example a remote control of a television or remote control of an air condition.

Chapter 3 Project Methodology

YAGI – UDA ANTENNA

Yagi Uda antenna is used in our final year project. High Frequencies (3–30 MHz), Very High Frequencies (30–300 MHz), and Ultra High Frequencies (300–3,000 MHz) ranges is the Yagi-Uda antenna respectively. Yagi Uda antenna consists of a number of linear dipole elements, [fig. 3.9]. In a Yagi-Uda antenna there is a folded dipole as a feeder that radiates. This radiator is exclusively designed to operate as an end-fire array, and it is achieved by having the scrounging elements in the forward beam act as directors while those in less number act as reflectors.

The main concept of using an antenna is to catch different number of channels at different frequencies. With the help of stepper motor this rotates the antenna in different directions according to the requirement. The radiation pattern of yagi uda antenna was observed by making a simulation in MATLAB. Factors like directivity, gain and beamwidth of the antenna were observed because these are the factors on which the antenna's selection is totally dependent.

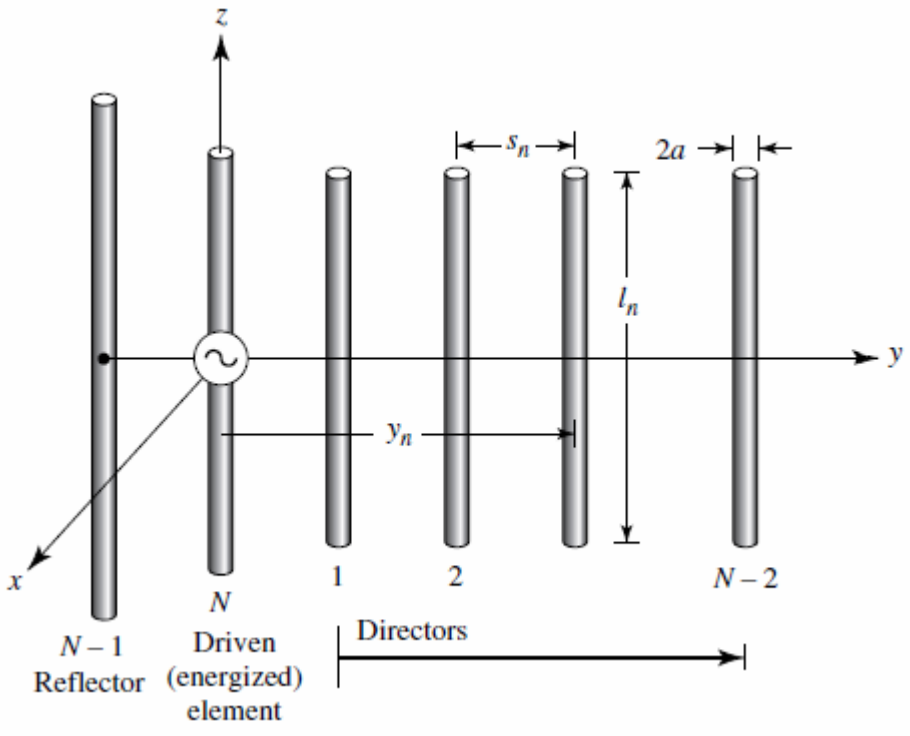


Fig 5. Yagi Uda Antenna

Chapter 3 Project Methodology

LIQUID CRYSTAL DISPLAY

Frequently, an 8051 program must interact with the outside world using input and output devices that communicate directly with a human being. One of the most common devices attached to an 8051 is an LCD display. Some of the most common LCDs connected to the 8051 are 16x2 and 20x2 displays. This means 16 characters per line by 2 lines and 20 characters per line by 2 lines, respectively.

Fortunately, a very popular standard exists which allows us to communicate with the vast majority of LCDs regardless of their manufacturer. The standard is referred to as HD44780U, which refers to the controller chip which receives data from an external source (in this case, the 8051) and communicates directly with the LCD.

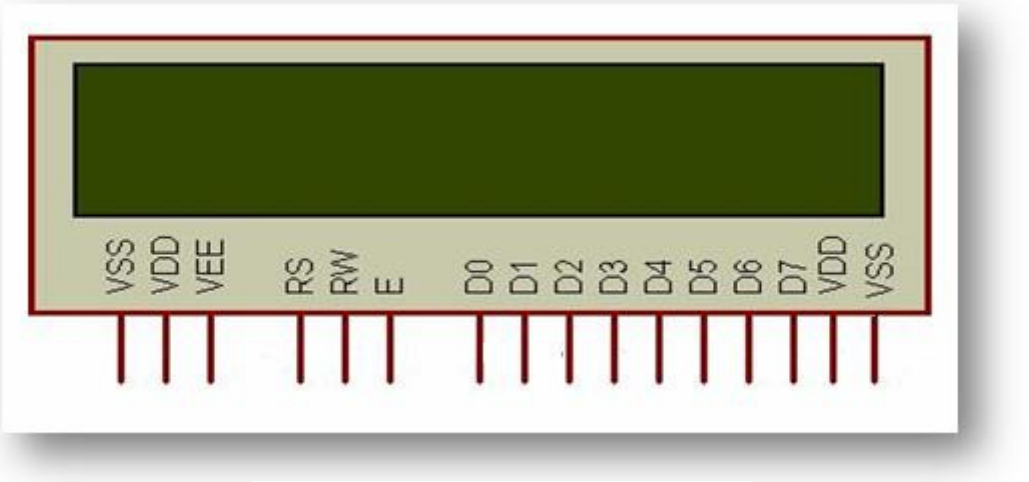


Fig 6. 8-bit LCD Pin Configuration

The 44780 standard requires 3 control lines as well as either 4 or 8 I/O lines for the data bus. The user may select whether the LCD is to operate with a 4-bit data bus or an 8-bit data bus. If a 4-bit data bus is used the LCD will require a total of 7 data lines (3 control lines plus the 4 lines for the data bus). If an 8-bit data bus is used the LCD will require a total of 11 data lines (3 control lines plus the 8 lines for the data bus).

The three control lines are referred to as **EN**, **RS**, and **RW**.

The **EN** line is called "Enable." This control line is used to tell the LCD that you are sending it data. To send data to the LCD, your program should make sure this line is low (0) and then set the other two control lines and/or put data on the data bus. When the other lines are completely ready, bring **EN** high (1) and wait for the minimum amount of time required by the LCD datasheet (this varies from LCD to LCD), and end by bringing it low (0) again.

The **RS** line is the "Register Select" line. When RS is low (0), the data is to be treated as a command or special instruction (such as clear screen, position cursor, etc.). When RS is high (1), the data being sent is text data which should be displayed on the screen. For example, to display the letter "T" on the screen you would set RS high.

The **RW** line is the "Read/Write" control line. When RW is low (0), the information on the data bus is being written to the LCD. When RW is high (1), the program is effectively querying (or reading) the LCD. Only one instruction ("Get LCD status") is a read command. All others are write commands--so RW will almost always be low.

Finally, the data bus consists of 4 or 8 lines (depending on the mode of operation selected by the user). In the case of an 8-bit data bus, the lines are referred to as DB0, DB1, DB2, DB3, DB4, DB5, DB6, and DB7.

Chapter 3 Project Methodology

EEPROM

EEPROM is also written as E^2PROM and is pronounced as "e-e-prom" "double-e prom," "e-squared" or simply "e-prom" stands for **E**lectrically **E**rasable **P**rogrammable **R**ead-**O**nly **M**emory and is a type of non volatile memory in computers and other electronic devices to store small amounts of data that must be saved when power is removed. When large amount of data is stored (such as in USB flash drives) a specific type of EEPROM such as flash memory is more economical than traditional EEPROM devices. EEPROM is basically read only memory (ROM) that can be erased and reprogrammed repeatedly through the application of higher than normal electrical voltage generated externally or internally in the case of modern EEPROMs. EPROM usually must be removed from the device for erasing and programming whereas EEPROMs can be programmed and erased in circuit. The reason that EEPROMs were slower was that it was limited to single byte operations but modern EEPROMs allow multiple byte operations. It also has a limited life that is the number of times it could be reprogrammed was limited to tens or hundreds of thousands of times. Here is an image of an EEPROM

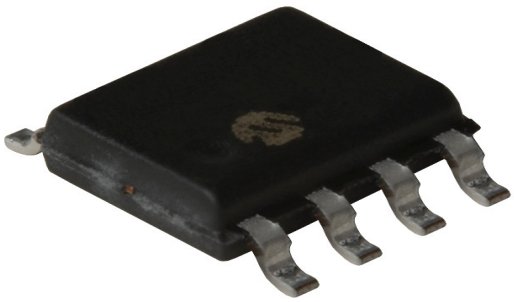


Fig 7. EEPROM

There are two interface types of EEPROM are which are as follows

1) Serial bus
2) Parallel bus

Chapter 3 Project Methodology

Serial bus devices

Most common serial interface types are Micro wire UNI/O and 1-Wire. These interfaces require one to four control signals for operations resulting in a memory device in an 8 pin or less package. The serial EEPROM typically operates in three phases OP Code Phase, Address Phase and Data Phase. The OP Code is the first eight bit input to the serial input pin of the EEPROM device then data to be read or written.

Parallel bus devices

Parallel EEPROM devices have an 8 bit data bus and an address bus wide enough to cover the complete memory. Most devices have chipped select and write protect pins. Some microcontroller has parallel EEPROM. Operation of a parallel EEPROM is simple and fast when compared to serial EEPROM, but these devices are larger due to the higher pin count twenty eight pins or more and have been decreasing in popularity in favor of serial EEPROM or Flash.

In our project we have used EEPROM to save the location of the keypad numbers and then to recall it in its memory. The function of EEPROM is to memorize the commands in its operational memory. It has a large versatile memory and it can save and recall many commands when desired. We have chosen by pressing 4 to save the first location. Similarly press 5 to save the second location and press 6 to save the third location. All the three locations are saved in the memory of EEPROM and if we press 7 it recalls location 1 saved by the viewer in the EEPROM similarly by pressing 8 it recalls location 2 saved by the viewer in the EEPROM and by pressing 9 it recalls location 3 saved by the viewer in the EEPROM.

3.3 SOFTWARE DETAIL

Initially the programming was to be done on PIC so the tool selected for programming was MPLab IDE. This is the recommended software by Muhammad Ali Mazidi and on the same side it is the tool that is made by Microchip Company themselves and the PIC ICs are only the development of this company. And the main benefit of this software is that this is the only software in which programming in Assembly plus in C, both ways can be done whereas, all the other softwares available are only C language.

How to install the software?

There is a trick behind installing this software. From the website www.microchip.com one can download the software and follow these steps:

- After downloading, unzip the whole file in the drive where you wish to install the software.
- After unzipping the file, run the setup and wait for the installation.
- If you won't unzip the file in the drive where the software is to be installed, the installation will generate a message of undefined path error.

But the software that was installed initially was MPLab IDE and is totally used for the PIC and after the hardware implementation problems; the microcontroller was changed to 89C51 and the software for programming for 89C51 is Keil µvision. This software is taught in detail in the 5th or 6th semester of the Engineering degree and is user friendly software. This software is easily available in the university.

Chapter 4 Project Specifications

4.1 BLOCK DIAGRAM

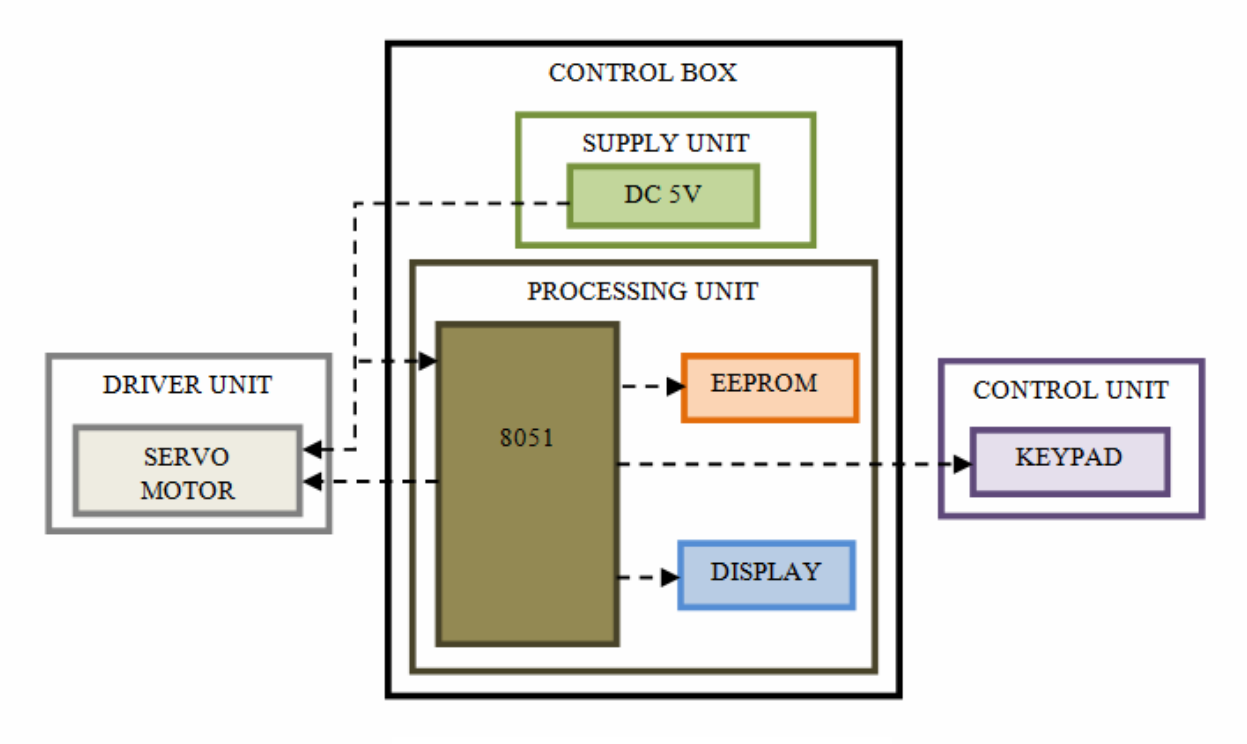


Fig 8. Block Diagram

Chapter 4 Project Specifications

Main File *main.c*

```
#include <At89x52.h>
#include <stdio.h>

sbit c1 = P1^0;
sbit c2 = P1^1;
sbit c3 = P1^2;
sbit r1 = P1^3;
sbit r2 = P1^4;
sbit r3 = P1^5;
sbit r4 = P1^6;
sbit w_p=P2^7;
extern void wait_lcd(void);
extern void clr_lcd(void);
extern void init_lcd(void);
extern void cur_pos(unsigned char);
extern void wrt_char(unsigned char);
extern void Send2LCD(unsigned char *, unsigned char);
extern    void                    WriteChar          (unsigned char, unsigned int);
extern    unsigned char      ReadChar           (unsigned int);
extern    void                    InitEEPRom         (void);
unsigned long int timer_counter=0;
unsigned long int amount=0;
//unsigned char no_of_candidate=3;//defining number of candidates
bit start_time=0;
unsigned char DataCount;

unsigned int Address;
unsigned char array[4];
unsigned long int wait_counter=0;
bit go_scan=0;
unsigned char array[4];
unsigned char val=0;
unsigned char val1=0;
char time_byte;
//////////////
void kb_delay(void)
{
        unsigned int i;
        for(i = 0; i < 1000UL;)
        {
                i++;
        }

}
//////////////
//////////////
void wait()
{
for(wait_counter=0;wait_counter<=10000;wait_counter++);
for(wait_counter=0;wait_counter<=10000;wait_counter++);
for(wait_counter=0;wait_counter<=10000;wait_counter++);
for(wait_counter=0;wait_counter<=10000;wait_counter++);
```

```
}
void InitSerial()
{
        SCON = 0x50;                    //mode 1, 10 bit
        TMOD = 0x20;                    //Timer 1 auto reload mode
        TH1  = 0xfd;                    //9600 baud rate over 11.0592MHz crystal    TH1 = 256 -
((Crystal/384)/Baud Rate)
        TR1  = 1;                               //Turn timer1 On
        EA   = 1;                               //Enable All Interrupts
//      ES   = 1;                       //Enable Serial Port Interrupt
}//end void InitSerial()

serial() interrupt 4
{
        register unsigned char RecCh;
        if(RI)
        {
                RecCh = SBUF;
                RI = 0;

        }//end if RI
}//end serial() interrupt 4

void SendChar(unsigned char ch)
{
        SBUF = ch;
        while(!TI)
        {
        }
        TI = 0;
}//end void SendChar(unsigned char ch)

void InitAll()
{

         TMOD = (TMOD & 0xF0) | 0x01;  /* Set T/C0 Mode */
         ET0 = 1;                       /* Enable Timer 0 Interrupts */
         TR0 = 1;
         IT0 = 1;   // Configure interrupt 0 for falling edge on /INT0 (P3.2)
    EX0 = 1;
         IT1 = 1;   // Configure interrupt 0 for falling edge on /INT0 (P3.2)
    EX1 = 1;               /* Start Timer 0 Running */
         EA = 1;                        /* Global Interrupt Enable */
        InitSerial();
        InitEEPRom();


}//end initall

void Delay(unsigned int dl)
{
        unsigned int i;
        for(i = 0; i < dl; i++)
                //do nothing
                ;
```

```
}
////////////////////
unsigned char keypad_scan(void)
{
unsigned int val;

 while(1)
 {
   P1=0xff;
   kb_delay();
   c1=0;c2=1;c3=1;
   kb_delay();
   val=P1;//&0xFE;

            switch(val)
            {
             case 246:
             {
                        while(!r1);
                        return ('*');
             }break;
             case 238:
             {
                        while(!r2);
                        return ('9');
             }break;

             case 222:
             {
                        while(!r3);
                        return ('6');
             }break;

             case 190:
             {
                                    while(!r4);
                                    return ('3');
             }break;

            }
             c1=1;c2=0;c3=1;
             kb_delay();
            val=P1;
            switch(val)
            {
             case 245:
             {
                                    while(!r1);
                                    return ('0');
             }break;
             case 237:
             {
                                    while(!r2);
                                    return ('8');
             }break;
```

```
            case 221:
            {
                                while(!r3);
                                return ('5');
            }break;

            case 189:
            {
                                while(!r4);
                                return ('2');
            }break;

           }
            c1=1;c2=1;c3=0;
            kb_delay();
           val=P1;
           switch(val)
           {
            case 243:
            {
                    while(!r1);
                    return ('#');
            }break;
            case 235:
            {
                                while(!r2);
                                return ('7');
            }break;

            case 219:
            {
                                while(!r3);
                                return ('4');
            }break;

            case 187:
            {
                                while(!r4);
                                return ('1');
            }break;

}           }
}
/////////////////
void timer0_ISR (void) interrupt 1
{
TH0=time_byte;
TL0=255;
if(start_time)
{
 if(++timer_counter==1)
 {
clk=1;
}
```

```
else
{clk=0;}
if(timer_counter>15)
{
            timer_counter=0;
}
}
}
////////////////////////////
void int0_isr (void) interrupt 0
{
clr_lcd();
 Send2LCD("Enter Password ",0);
 cur_pos(64);
 array[0]=keypad_scan();
 wrt_char( array[0]);
 wait();
 if(array[0]=='1')
 {
   go_scan=1;
 }
 else if(array[0]=='4')
 {
          go_scan=0;
          WriteChar(time_byte,1);
 }
  else if(array[0]=='5')
 {
          go_scan=0;
          WriteChar(time_byte,2);
 }
  else if(array[0]=='6')
 {
          go_scan=0;
          WriteChar(time_byte,3);
 }
  else if(array[0]=='7')
 {
          time_byte=ReadChar(1);
          start_time=1;
 }
  else if(array[0]=='8')
 {
          time_byte=ReadChar(2);
          start_time=1;
 }
  else if(array[0]=='9')
 {
          time_byte=ReadChar(3);
          start_time=1;
 }
 }/////////////////
```

```
void int1_isr (void) interrupt 2
{
 //reset=reset^1;
}
////////////////////////////
  void InitSeriall(void)
{
            SCON = 0x52;    // setup serial port control
            TMOD = 0x20;    // hardware (9600 BAUD @11.05592MHZ)
            TH1  = 0xFD;    // TH1
            TR1        = 1;                    // Timer 1 on
}
////////////////////////////
void main()
{
            unsigned char i;
            InitAll();
            wait();
            init_lcd();
            Send2LCD("    Channel        Positioning");
            wait();
            time_byte=250;
            start_time=0;

            start_time=1;
            time_byte=240;
            while(1)
            {
//  1  clock wise  still match eeep 4,5,6 to save
              if(go_scan)
              {
              do{
              start_time=1;
              for(time_byte=180;time_byte<=240;time_byte++)
     {
              wait();

     }
                for(time_byte=240;time_byte>=180;time_byte--)
     {
              wait();

     }
              }while(go_scan);
                        }
            }
}
```

Saving of data *eeprom.c*

```
#include<At89x52.h>
unsigned char  zdata, rdata; //data registers
unsigned char addr_lo; //2-byte address register
unsigned char addr_hi;
extern unsigned char Length;
//function prototypes
extern void read_random();
extern void write_byte();
extern void on_reset();

void WriteChar(unsigned char c, unsigned int A)
{
        addr_lo = A;
        addr_hi = A >> 8;
        zdata   = c;
        do
        {
                write_byte();
        }while(CY);
}//end WriteChar

unsigned char ReadChar(unsigned int A)
{
        unsigned char c;
        addr_lo = A;
        addr_hi = A >> 8;
        do
        {
                read_random();
        }while(CY);
        c = rdata;//RDDTA;
        return(c);
}//unsigned char ReadChar(unsigned char Address)

void InitEEPRom()
{
        unsigned char temp;
        temp = ReadChar(501);
        if(temp == 'A')
        {
                Length = ReadChar(500);
        }
        else
        {
                Reset();
        }
}
```

Initialization of LCD "*lcd.c*"

```
#include<At89x52.h>
sbit EN = P3^7;              //Pin 6
sbit RW = P3^6;              //pin 5
sbit RS = P3^5;              //pin 4
void wait_lcd()
{
        int set, cnt;
        set = 1;
        cnt = 0;
        EN  = 1;
        RS  = 0;
        RW  = 1;
        P0  = 255;
        while(set)
        {
                cnt++;
                if(P0_7 == 0)
                {
                        set = 0;
                        cnt = 0;
                }//end if
                else if(cnt == 255)
                        P0_7 = 0;
        }//end while
        EN = 0;
        RW = 0;
}//end wait lcd

void clr_lcd()
{
        EN = 1;
        RS = 0;
        P0 = 1;
        EN = 0;
        wait_lcd();
}//end clear lcd

void init_lcd()
{
        //entry mode set
  EN = 1;
  RS = 0;

  P0 = 0x06;

  EN = 0;
  wait_lcd();
        //Display On/Off
  EN = 1;
  RS = 0;
```

```
  P0 = 0x0f;
  EN = 0;
  wait_lcd();
        //Function Set
  EN = 1;
  RS = 0;
  P0 = 0x38;
  EN = 0;
  wait_lcd();
}

void cur_pos(unsigned char cp)
{
        EN = 1;
        RS = 0;
        P0 = 128 + cp;
        EN = 0;
        wait_lcd();
}//end cursor position

void wrt_char(unsigned char ch)
{
  EN = 1;
  RW = 0;
  RS = 1;
  P0 = ch;
  EN = 0;
  wait_lcd();
}

void Send2LCD(unsigned char *Msg, unsigned char cp)
{
        unsigned char index;
        cur_pos(cp);
        for(index = 0; *(Msg + index) != '\0'; index++)
        {
                wrt_char(*(Msg + index));
                cp++;
                if(cp == 16)
                {
                        cp = 64;
                        cur_pos(cp);
                }
        }
}//end send 2 lcd
```

Chapter 5 Conclusion

5.1 CONCLUSION

This project proved to be of great success and learning to us. The main aim of the project was to rotate an antenna in different directions to catch channels at different frequencies. With the help of a servo motor and matrix keypad the antenna rotates in different directions. Overall we were able to create successful prototype of the concept and believe that our product has a great scope in the local market. The project has been successfully completed and is in proper working condition. It is an advantageous concept of the future. Here is the image of our Final Year Project

5.2 PROJECT FOR LEARNING

This project has really been a learning curve. Many components with which we were not familiar were used such as servo motor, working of a matrix keypad. We put our efforts in the hardware and faced many problems during the project such as in the interfacing of LCD and servo motor but we did it successfully. We did our programming on Keil µvision2. Handling components and doing project in time is a big achievement. Project provides opportunity to learn and implement our skills.

Appendix A

Features

- Compatible with MCS-51™ Products
- 4K Bytes of In-System Reprogrammable Flash Memory
  - Endurance: 1,000 Write/Erase Cycles
- Fully Static Operation: 0 Hz to 24 MHz
- Three-level Program Memory Lock
- 128 x 8-bit Internal RAM
- 32 Programmable I/O Lines
- Two 16-bit Timer/Counters
- Six Interrupt Sources
- Programmable Serial Channel
- Low-power Idle and Power-down Modes

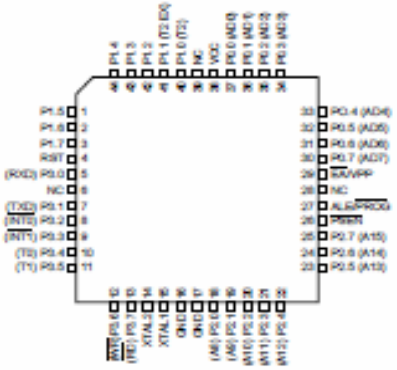
Description

The AT89C51 is a low-power, high-performance CMOS 8-bit microcomputer with 4K bytes of Flash programmable and erasable read only memory (PEROM). The device is manufactured using Atmel's high-density nonvolatile memory technology and is compatible with the industry-standard MCS-51 instruction set and pinout. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89C51 is a powerful microcomputer which provides a highly-flexible and cost-effective solution to many embedded control applications.

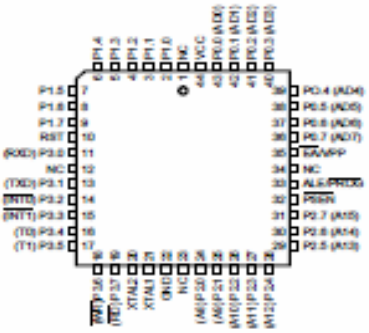
Pin Configurations

PDIP

PQFP/TQFP



PLCC



8-bit Microcontroller with 4K Bytes Flash

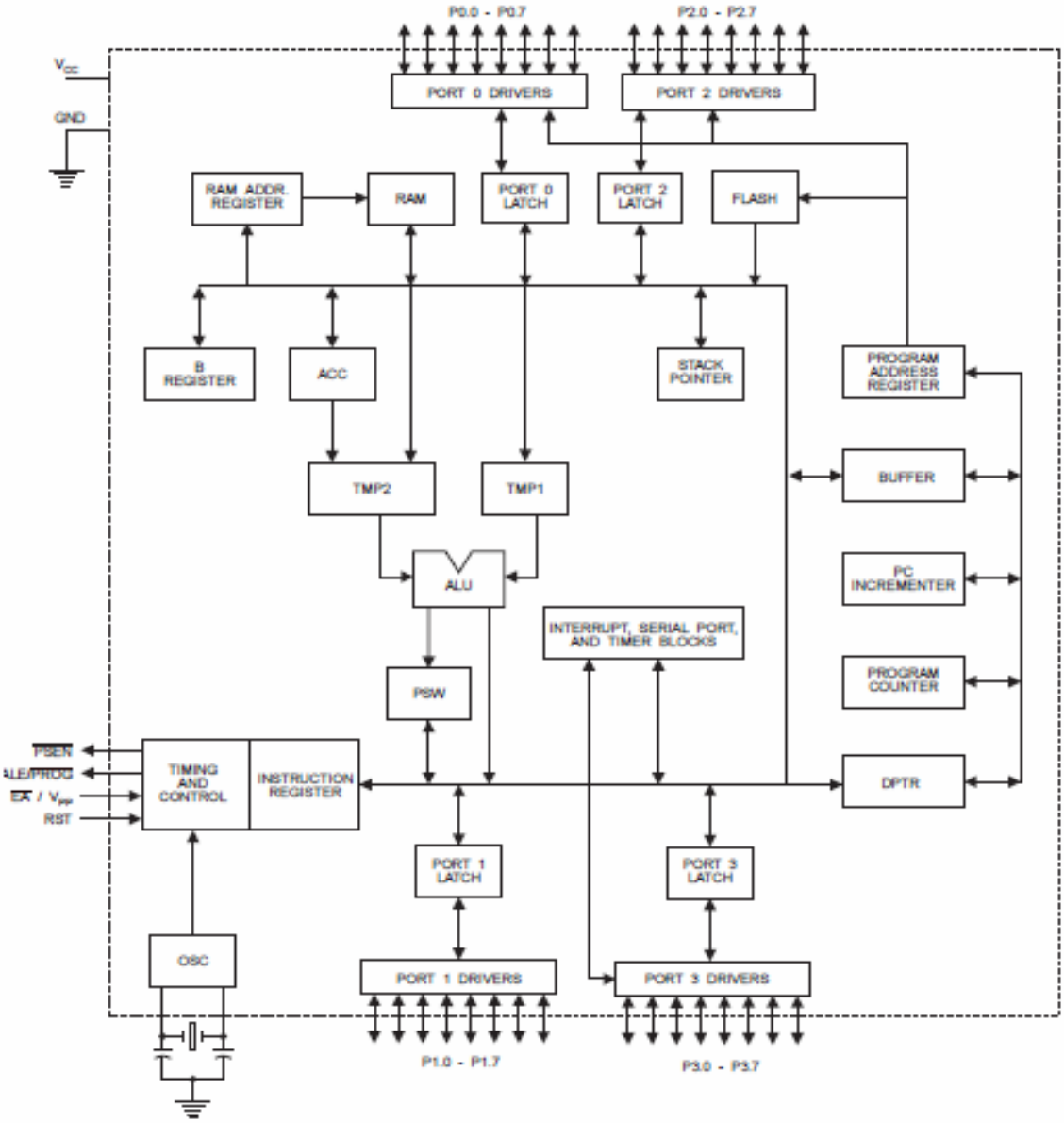
AT89C51

Not Recommended for New Designs. Use AT89S51.

Rev. 0265G–02/00

Appendix A

Block Diagram



Appendix A

The AT89C51 provides the following standard features: 4K bytes of Flash, 128 bytes of RAM, 32 I/O lines, two 16-bit timer/counters, a five vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator and clock circuitry. In addition, the AT89C51 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port and interrupt system to continue functioning. The Power-down Mode saves the RAM contents but freezes the oscillator disabling all other chip functions until the next hardware reset.

Pin Description

VCC

Supply voltage.

GND

Ground.

Port 0

Port 0 is an 8-bit open-drain bi-directional I/O port. As an output port, each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high-impedance inputs.

Port 0 may also be configured to be the multiplexed low-order address/data bus during accesses to external program and data memory. In this mode P0 has internal pullups.

Port 0 also receives the code bytes during Flash programming, and outputs the code bytes during program verification. External pullups are required during program verification.

Port 1

Port 1 is an 8-bit bi-directional I/O port with internal pullups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Port 1 also receives the low-order address bytes during Flash programming and verification.

Port 2

Port 2 is an 8-bit bi-directional I/O port with internal pullups. The Port 2 output buffers can sink/source four TTL inputs. When 1s are written to Port 2 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @ DPTR). In this application, it uses strong internal pullups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @ RI), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits and some control signals during Flash programming and verification.

Port 3

Port 3 is an 8-bit bi-directional I/O port with internal pullups. The Port 3 output buffers can sink/source four TTL inputs. When 1s are written to Port 3 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (I_{IL}) because of the pullups.

Port 3 also serves the functions of various special features of the AT89C51 as listed below:

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

Port 3 also receives some control signals for Flash programming and verification.

RST

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/$\overline{PROG}$

Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input ($\overline{PROG}$) during Flash programming.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE

Appendix A

pulse is skipped during each access to external Data Memory.

If desired, ALE operation can be disabled by setting bit 0 of SFR location 8EH. With the bit set, ALE is active only during a MOVX or MOVC instruction. Otherwise, the pin is weakly pulled high. Setting the ALE-disable bit has no effect if the microcontroller is in external execution mode.

PSEN

Program Store Enable is the read strobe to external program memory.

When the AT89C51 is executing code from external program memory, PSEN is activated twice each machine cycle, except that two PSEN activations are skipped during each access to external data memory.

EA/VPP

External Access Enable. EA must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, EA will be internally latched on reset.

EA should be strapped to V_{CC} for internal program executions.

This pin also receives the 12-volt programming enable voltage (V_{pp}) during Flash programming, for parts that require 12-volt V_{pp}.

XTAL1

Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

XTAL2

Output from the inverting oscillator amplifier.

Oscillator Characteristics

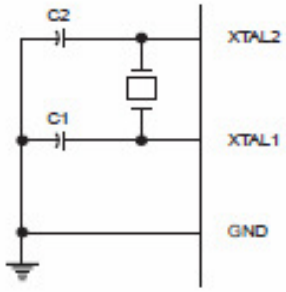
XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 1. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external clock source, XTAL2 should be left unconnected while XTAL1 is driven as shown in Figure 2. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum voltage high and low time specifications must be observed.

Idle Mode

In idle mode, the CPU puts itself to sleep while all the on-chip peripherals remain active. The mode is invoked by software. The content of the on-chip RAM and all the special functions registers remain unchanged during this mode. The idle mode can be terminated by any enabled interrupt or by a hardware reset.

It should be noted that when idle is terminated by a hard ware reset, the device normally resumes program execution, from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write to a port pin when idle is terminated by reset, the instruction following the one that invokes idle should not be one that writes to a port pin or to external memory.

Figure 1. Oscillator Connections



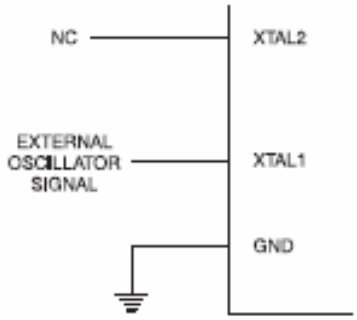
Note: C1, C2 = 30 pF ± 10 pF for Crystals
= 40 pF ± 10 pF for Ceramic Resonators

Status of External Pins During Idle and Power-down Modes

Mode	Program Memory	ALE	PSEN	PORT0	PORT1	PORT2	PORT3
Idle	Internal	1	1	Data	Data	Data	Data
Idle	External	1	1	Float	Data	Address	Data
Power-down	Internal	0	0	Data	Data	Data	Data
Power-down	External	0	0	Float	Data	Data	Data

Appendix A

Figure 2. External Clock Drive Configuration



Power-down Mode

In the power-down mode, the oscillator is stopped, and the instruction that invokes power-down is the last instruction executed. The on-chip RAM and Special Function Registers retain their values until the power-down mode is terminated. The only exit from power-down is a hardware reset. Reset redefines the SFRs but does not change the on-chip RAM. The reset should not be activated before V_{CC} is restored to its normal operating level and must be held active long enough to allow the oscillator to restart and stabilize.

Program Memory Lock Bits

On the chip are three lock bits which can be left unprogrammed (U) or can be programmed (P) to obtain the additional features listed in the table below.

When lock bit 1 is programmed, the logic level at the $\overline{EA}$ pin is sampled and latched during reset. If the device is powered up without a reset, the latch initializes to a random value, and holds that value until reset is activated. It is necessary that the latched value of $\overline{EA}$ be in agreement with the current logic level at that pin in order for the device to function properly.

Lock Bit Protection Modes

Program Lock Bits				
	LB1	LB2	LB3	Protection Type
1	U	U	U	No program lock features
2	P	U	U	MOVC instructions executed from external program memory are disabled from fetching code bytes from internal memory, $\overline{EA}$ is sampled and latched on reset, and further programming of the Flash is disabled
3	P	P	U	Same as mode 2, also verify is disabled
4	P	P	P	Same as mode 3, also external execution is disabled

Appendix A

Programming the Flash

The AT89C51 is normally shipped with the on-chip Flash memory array in the erased state (that is, contents = FFH) and ready to be programmed. The programming interface accepts either a high-voltage (12-volt) or a low-voltage (V_{CC}) program enable signal. The low-voltage programming mode provides a convenient way to program the AT89C51 inside the user's system, while the high-voltage programming mode is compatible with conventional third-party Flash or EPROM programmers.

The AT89C51 is shipped with either the high-voltage or low-voltage programming mode enabled. The respective top-side marking and device signature codes are listed in the following table.

	V_{PP} = 12V	V_{PP} = 5V
Top-side Mark	AT89C51 xxxx yyww	AT89C51 xxxx-5 yyww
Signature	(030H) = 1EH (031H) = 51H (032H) =F FH	(030H) = 1EH (031H) = 51H (032H) = 05H

The AT89C51 code memory array is programmed byte-by-byte in either programming mode. *To program any non-blank byte in the on-chip Flash Memory, the entire memory must be erased using the Chip Erase Mode.*

Programming Algorithm: Before programming the AT89C51, the address, data and control signals should be set up according to the Flash programming mode table and Figure 3 and Figure 4. To program the AT89C51, take the following steps.

1. Input the desired memory location on the address lines.
2. Input the appropriate data byte on the data lines.
3. Activate the correct combination of control signals.
4. Raise $\overline{EA}/V_{PP}$ to 12V for the high-voltage programming mode.
5. Pulse ALE/$\overline{PROG}$ once to program a byte in the Flash array or the lock bits. The byte-write cycle is self-timed and typically takes no more than 1.5 ms. Repeat steps 1 through 5, changing the address and data for the entire array or until the end of the object file is reached.

$\overline{Data}$ Polling: The AT89C51 features $\overline{Data}$ Polling to indicate the end of a write cycle. During a write cycle, an attempted read of the last byte written will result in the complement of the written datum on PO.7. Once the write cycle has been completed, true data are valid on all outputs, and the next cycle may begin. $\overline{Data}$ Polling may begin any time after a write cycle has been initiated.

Ready/$\overline{Busy}$: The progress of byte programming can also be monitored by the RDY/$\overline{BSY}$ output signal. P3.4 is pulled low after ALE goes high during programming to indicate BUSY. P3.4 is pulled high again when programming is done to indicate READY.

Program Verify: If lock bits LB1 and LB2 have not been programmed, the programmed code data can be read back via the address and data lines for verification. The lock bits cannot be verified directly. Verification of the lock bits is achieved by observing that their features are enabled.

Chip Erase: The entire Flash array is erased electrically by using the proper combination of control signals and by holding ALE/$\overline{PROG}$ low for 10 ms. The code array is written with all "1"s. The chip erase operation must be executed before the code memory can be re-programmed.

Reading the Signature Bytes: The signature bytes are read by the same procedure as a normal verification of locations 030H, 031H, and 032H, except that P3.6 and P3.7 must be pulled to a logic low. The values returned are as follows.

(030H) = 1EH indicates manufactured by Atmel
(031H) = 51H indicates 89C51
(032H) = FFH indicates 12V programming
(032H) = 05H indicates 5V programming

Programming Interface

Every code byte in the Flash array can be written and the entire array can be erased by using the appropriate combination of control signals. The write operation cycle is self-timed and once initiated, will automatically time itself to completion.

All major programming vendors offer worldwide support for the Atmel microcontroller series. Please contact your local programming vendor for the appropriate software revision.

REFERENCES

1. http://www.mcuexamples.com/Projects/Antenna/TV-Antenna.php
2. https://www.google.com.pk/
3. http://www.datasheetarchive.com/
4. http://www.DatasheetCatalog.com
5. http://www.atmel.com
6. http://pcbheaven.com/wikipages/How_Key_Matrices_Works/
7. http://www.doctronics.co.uk/4511.htm