CHAPTER - I

INTRODUCTION

1.1 ABOUT PROGRAMMER'S KEYBOARD

Designing and building of a keyboard with those feature which helps in speeding up the programmer or say to coder to do coding on the respective IDE editor window. The client software is standalone, a desktop application.

1.2 Embedded Application

An embedded system is a computer system designed for specific control functions within a larger system, often with real-time computing constraints. It is *embedded* as part of a complete device often including hardware and mechanical parts. By contrast, a general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs. Embedded systems control many devices in common use today. Embedded systems contain processing cores that are either microcontrollers or digital signal processors (DSP). A processor is an important unit in the embedded system hardware. It is the heart of the embedded system.

1.2.1 History

One of the first recognizably modern embedded systems was the Apollo Guidance Computer, developed by Charles Stark Draper at the MIT Instrumentation Laboratory. At the project's inception, the Apollo guidance computer was considered the riskiest item in the Apollo project as it employed the then newly developed monolithic integrated circuits to reduce the size and weight. An early mass-produced embedded system was the Autonetics D-17 guidance computer for the Minuteman missile, released in 1961. It was built from transistor logic and had a hard disk for main memory. When the Minuteman II went into production in 1966, the D-17 was replaced with a new computer that was the first high-volume use of integrated circuits. This program alone reduced prices on quad nand gate ICs from $1000/each to $3/each, permitting their use in commercial products. Since these early applications in the 1960s, embedded systems have come down in price and there has been a dramatic rise in processing power and functionality. The first microprocessor for example, the Intel 4004, was designed for calculators and other small systems but still

required many external memory and support chips. In 1978 National Engineering Manufacturers Association released a "standard" for programmable microcontrollers, including almost any computer-based controllers, such as single board computers, numerical, and event-based controllers. As the cost of microprocessors and microcontrollers fell it became feasible to replace expensive knob-based analog components such as potentiometers and variable capacitors with up/down buttons or knobs read out by a microprocessor even in some consumer products.

1.2.2 Characteristics

Embedded systems are designed to do some specific task, rather than be a general-purpose computer for multiple tasks. Some also have real-time performance constraints that must be met, for reasons such as safety and usability; others may have low or no performance requirements, allowing the system hardware to be simplified to reduce costs. Embedded systems are not always standalone devices. Many embedded systems consist of small, computerized parts within a larger device that serves a more general purpose. For example, the Gibson Robot Guitar features an embedded system for tuning the strings, but the overall purpose of the Robot Guitar is, of course, to play music. Similarly, an embedded system in an automobile provides a specific function as a subsystem of the car itself.

1.2.3 Processors in Embedded System

Embedded processors can be broken into two broad categories. Ordinary microprocessors (μP) use separate integrated circuits for memory and peripherals. Microcontrollers (μC) have many more peripherals on chip, reducing power consumption, size and cost. In contrast to the personal computer market, many different basic CPU architectures are used, since software is custom-developed for an application and is not a commodity product installed by the end user. Both Von Neumann as well as various degrees of Harvard architectures are used. RISC as well as non-RISC processors are found. Word lengths vary from 4-bit to 64-bits and beyond, although the most typical remain 8/16-bit. Most architectures come in a large number of different variants and shapes, many of which are also manufactured by several different companies.

1.3 Firmware

In electronic systems and computing, firmware is the combination of persistent memory and program code and data stored in it. Typical examples of devices containing firmware

are embedded systems (such as traffic lights, consumer appliances, and digital watches), computers, computer peripherals, mobile phones, and digital cameras. The firmware contained in these devices provides the control program for the device. Firmware is held in non-volatile memory devices such as ROM, EPROM, or flash memory. Changing the firmware of a device may rarely or never be done during its economic lifetime; some firmware memory devices are permanently installed and cannot be changed after manufacture.

1.4 Device Drive Software

In computing, a device driver is a computer program that operates or controls a particular type of device that is attached to a computer. A driver typically communicates with the device through the computer bus or communications subsystem to which the hardware connects. When a calling program invokes a routine in the driver, the driver issues commands to the device. Once the device sends data back to the driver, the driver may invoke routines in the original calling program. Drivers are hardware-dependent and operating-system-specific. They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface.

1.4.1 Purpose of device drive software

Programmers can write the higher-level application code independently of whatever specific hardware the end-user is using. Physical layers communicate with specific device instances. For example, a serial port needs to handle standard communication protocols such as XON/XOFF that are common for all serial port hardware. This would be managed by a serial port logical layer. However, the physical layer needs to communicate with a particular serial port chip. 16550 UART hardware differs from PL-011. The physical layer addresses these chip-specific variations.

1.4.4 Application

Drivers may interface with:

- printers
- video adapters
- Network cards
- Sound cards

- Local buses of various sorts—in particular, for bus mastering on modern systems
- Low-bandwidth I/O buses of various sorts (for pointing devices such as mice, keyboards, USB, etc.)
- Computer storage devices such as hard disk, CD-ROM, and floppy disk buses (ATA, SATA, SCSI)

1.5 Purpose of the Project

This project is partly based on embedded systems and rest of the part is software based. Special key sets are provided which are dedicated to do specific task. This project is been developed keeping the programmers as our audience. For now, this will be developed for windows platform. This project will not be developed as per IEEE, FCC (Federal Communication Commission) Electronics specification. This keyboard will have a USB interface to connect with your system. Software will be developed further for rendering this keyboard from your system. That software will reside on computer system.

1.6 Benefits of the Project

- This project has been specifically designed for the programmers to speed up their coding process.
- This project is useful for the programmer which works on the different platforms.

CHAPTER - II
LITERATURE REVIEW

2.1 COMPUTER KEYBOARD

In computing, a keyboard is a typewriter-style device, which uses an arrangement of buttons or keys, to act as mechanical levers or electronic switches. Following the decline of punch cards and paper tape, interaction via teleprinter-style keyboards became the main input device for computers.

A keyboard typically has characters engraved or printed on the keys and each press of a key typically corresponds to a single written symbol. However, to produce some symbols requires pressing and holding several keys simultaneously or in sequence. While most keyboard keys produce letters, numbers or signs (characters), other keys or simultaneous key presses can produce actions or computer commands.

2.1.1 Computer Keyboard History

While typewriters are the definitive ancestor of all key-based text entry devices, the computer keyboard as a device for electromechanical data entry and communication derives largely from the utility of two devices: teleprinter (or teletypes) and keypunches. It was through such devices that modern computer keyboards inherited their layouts.

As early as the 1870s, teleprinter-like devices were used to simultaneously type and transmit stock market text data from the keyboard across telegraph lines to stock ticker machines to be immediately copied and displayed onto ticker tape. The teleprinter, in its more contemporary form, was developed from 1903–1910 by American mechanical engineer Charles Krum and his son Howard, with early contributions by electrical engineer Frank Pearne. Earlier models were developed separately by individuals such as Royal Earl House and Frederick G. Creed.

2.1.2 Layout

2.1.2.1 Alphabetic

There are a number of different arrangements of alphabetic, numeric, and punctuation symbols on keys. These different keyboard layouts arise mainly because different people

need easy access to different symbols, either because they are inputting text in different languages, or because they need a specialized layout for mathematics, accounting, computer programming, or other purposes. The United States keyboard layout is used as default in the currently most popular operating systems: Windows, Mac OS X and Linux. The common QWERTY-based layout was designed early in the era of mechanical typewriters, so its ergonomics were compromised to allow for the mechanical limitations of the typewriter.

As the letter-keys were attached to levers that needed to move freely, inventor Christopher Sholes developed the QWERTY layout to reduce the likelihood of jamming. With the advent of computers, lever jams are no longer an issue, but nevertheless, QWERTY layouts were adopted for electronic keyboards because they were widely used. Alternative layouts such as the Dvorak Simplified Keyboard are not in widespread use.

The QWERTZ layout is widely used in Germany and much of Central Europe. The main difference between it and QWERTY is that Y and Z are swapped, and most special characters such as brackets are replaced by diacritical characters.

2.1.3 Key types

2.1.3.1 Alphanumeric

Alphabetical, numeric, and punctuation keys are used in the same fashion as a typewriter keyboard to enter their respective symbol into a word processing program, text editor, data spread sheet, or other program. Many of these keys will produce different symbols when modifier keys or shift keys are pressed. The alphabetic characters become uppercase when the shift key or Caps Lock key is depressed. The numeric characters become symbols or punctuation marks when the shift key is depressed. The alphabetical, numeric, and punctuation keys can also have other functions when they are pressed at the same time as some modifier keys.

The Space bar is a horizontal bar in the lowermost row, which is significantly wider than other keys. Like the alphanumeric characters, it is also descended from the mechanical typewriter. Its main purpose is to enter the space between words during typing. It is large enough so that a thumb from either hand can use it easily. Depending on the operating system, when the space bar is used with a modifier key such as the control key, it may have

functions such as resizing or closing the current window, half-spacing, or backspacing. In computer games and other applications the key has myriad uses in addition to its normal purpose in typing, such as jumping and adding marks to check boxes. In certain programs for playback of digital video, the space bar is used for pausing and resuming the playback.

2.1.4.2 Modifiers

Modifier keys are special keys that modify the normal action of another key, when the two are pressed in combination. For example, <Alt> + <F4> in Microsoft Windows will close the program in an active window. In contrast, pressing just <F4> will probably do nothing, unless assigned a specific function in a particular program. By themselves, modifier keys usually do nothing.

The most widely used modifier keys include the Control key, Shift key and the Alt key. The AltGr key is used to access additional symbols for keys that have three symbols printed on them. On the Macintosh and Apple keyboards, the modifier keys are the Option key and Command key, respectively. On MIT computer keyboards, the Meta key is used as a modifier and for Windows keyboards, there is a Windows key. Compact keyboard layouts often use a Fn key. "Dead keys" allow placement of a diacritic mark, such as an accent, on the following letter (e.g., the Compose key).

2.1.4 Technology

2.1.4.1 Key switches

In the first electronic keyboards in the early 1970s, the key switches were individual switches inserted into holes in metal frames. These keyboards cost from 80–120 US dollars and were used in mainframe data terminals. The most popular switch types were reed switches (contacts enclosed in a vacuum in a glass capsule, affected by a magnet mounted on the switch plunger – from Clare-Pendar in Post Falls Idaho, which became part of General Instrument, which used reed switch capsules made by C.P. Clare Co. in Illinois; and Key Tronic Corporation of Spokane, Washington), Hall-effect switches (using a Hall-effect semiconductor where a current is generated by a passing magnet – from Micro switch [19] in Illinois, which became part of Honeywell), and inductive core switches (again, activated by a magnet – from Cortron,which was part of ITW/Illinois Tool Works).

2.1.4.2 Control processor

Computer keyboards include control circuitry to convert key presses into key codes (usually scancodes) that the computer's electronics can understand. The key switches are connected via the printed circuit board in an electrical X-Y matrix where a voltage is provided sequentially to the Y lines and, when a key is depressed, detected sequentially by scanning the X lines.

The first computer keyboards were for mainframe computer data terminals and used discrete electronic parts. The first keyboard microprocessor was introduced in 1972 by General Instruments, but keyboards have been using the single-chip 8048 microcontroller variant since it became available in 1978. The keyboard switch matrix is wired to its inputs, it converts the keystrokes to key codes, and, for a detached keyboard, sends the codes down a serial cable (the keyboard cord) to the main processor on the computer motherboard. This serial keyboard cable communication is only bi-directional to the extent that the computer's electronics controls the illumination of the caps lock, num lock and scroll lock lights.

2.1.4.3 Connection types

There are several ways of connecting a keyboard to a system unit (more precisely, to its keyboard controller) using cables, including the standard AT connector commonly found on motherboards, which was eventually replaced by the PS/2 and the USB connection. Prior to the iMac line of systems, Apple used the proprietary Apple Desktop Bus for its keyboard connector.

Wireless keyboards have become popular for their increased user freedom. A wireless keyboard often includes a required combination transmitter and receiver unit that attaches to the computer's keyboard port. The wireless aspect is achieved either by radio frequency (RF) or by infrared (IR) signals sent and received from both the keyboard and the unit attached to the computer. A wireless keyboard may use an industry standard RF, called Bluetooth. With Bluetooth, the transceiver may be built into the computer. However, a wireless keyboard needs batteries to work and may pose a security problem due to the risk of data "eavesdropping" by hackers. Wireless solar keyboards charge their batteries from small solar panels using sunlight or standard artificial lighting. An early example of a consumer wireless keyboard is that of the Olivetti Envision.

2.1.5 Working of Keyboard

2.1.5.1 Functioning of a Computer Keyboard

In general, there are 80-110 keys in a computer keyboard. The keys may vary depending upon the brand and the type of operating system. Nevertheless, the shape, size and spacing of keys are almost same for all keyboards. Also the layout or arrangement of keys that represent letters, signs and symbols is same, which is referred to as QWERTY.

The working of a computer keyboard can be compared to a miniature computer. Inside the keyboard, there are metallic plate, circuit board (key matrix) and processor, which are responsible for transferring information from the keyboard to the computer.

2.1.5.2 Layout

Computer keyboards are an input device. They put the information a person types into a program on the computer. Most keyboards have 80 to 110 keys. The numbers and letters on the keyboard are displayed keycaps--these are the buttons that are pressed when a person types. The layout of the numbers and letters are the same on every keyboard and they are referred to as the QWERTY.

2.1.5.3 Communication

The keyboard connects to the computer via a five pin male plug or a PS/2 plug. Keyboards and computers work together in a bi-directional format. This means that they can each send information to one another. These bi-directional lines are the clock line coming from the keyboard and the data line coming from the computer. Both lines must be idle, or high in order for the keyboard to send data. The computer will send a signal to the keyboard through the clock line letting it know that the line is clear to send. If the line is not clear, the keyboard will hold the information until the line opens. When the line is low, the keyboard is waiting for a command from the computer. When the computer wants to send information to the keyboard, it brings the data and the clock line low. It does this to ensure that the keyboard does not send it a message at the same time.

2.2 Serial Communication

Whoever work with standard RS232 components from a .NET environment, and need real, embedded-like RS232 communication, realize in a short time that the component needs to be wrapped, or at least, additionally supported by custom software. I designed this tutorial because I couldn’t find something similar on the web. The code is an excerpt from a huge GPS tracking application, and it proved to work nicely. Just to mention, that there are other methods for RS232 communication such as using hand-shaking protocols, and hardware/software enabled pin control.

CHAPTER - III

PROBLEM IDENTIFICATION

3.1 Determining the Problem

Firstly, here we may define statements to define the problems which may be faced or which are standing before us. As we determine and frame the problems, many new topics will be discussed in detail. The upcoming points are the issue which stands as problem or say as challenge before us.

3.1.1 Monitoring Keyboard & Retrieving the Scan Code

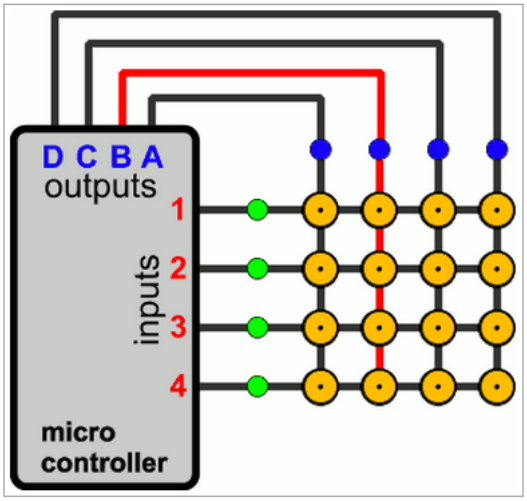


Figure 3.1 : Matrix Keypad connected to micro controller.

Keyboard is the basic input device, so far available for human to interact with computer and with any digital system. The case here that we have to manage to provide keys with each having special functioning. Then after, we have to keep an eye at any instance if any key is pressed, corresponding function should be done. So the task stands are:

- Determining Key states.
- Determining which key is pressed.
- Decide what kind of keypad should be used here.

3.1.2 Logic Circuit of Keyboard

This part is the main board, as an important part of the project too! This logic board shall be needed to monitor and read and send the data to computer containing the information about

the key and its associated attributes like key states. The main question or say the challenge is that is any logical control unit is required for this or it can be done with simple digital logic components!?

3.1.3 Plug-n-Play Interface

What will be the interfacing type of the keyboard which we will construct? By this, we mean that what standards or what plug type we shall provide to get our keyboard connected to computer. While deciding this, we have to consider the current technologies and also not to forget about the earlier technology too!

3.1.4 Keyboard Fabrication

This is one of the challenges which got risks too, in terms of money, time and resources. By this what we means is that for fabrication of the keyboard we have to do either at production environment where the things are been carried out under professionals guidance or by self just like DIY (Do It Yourself).

CHAPTER – IV

METHODOLOGY & WORKING

4.1 Proposed Method

As we have defined and also recognised our problems which are determined in nature, now we shall analyse the problems and get to many solutions possible. The solutions are described in forth coming points and in the project the solutions are accepted considering various issues and points and reasons too.

4.2 Hardware

We will be going into detail of each and every parts which are used in this project and also into the software part which makes this embedded system into live.

4.2.1 Schematics & Layouts

4.2.1.1Push Button

A **push-button** (also spelled **pushbutton**) or simply **button** is a simple switch mechanism for controlling some aspect of a machine or a process. Buttons are typically made out of hard material, usually plastic or metal. The surface is usually flat or shaped to accommodate the human finger or hand, so as to be easily depressed or pushed. Buttons are most often biased switches, though even many un-biased buttons (due to their physical nature) require a spring to return to their un-pushed state. Different people use different terms for the "pushing" of the button, such as **press**, **depress**, **mash**, and **punch**.

4.2.1.2 Crystals

The main terms used for crystal oscillators are explained as follows:

- **Operating temperature range:** Ambient temperature range within which the specified characteristics of a product can be achieved.
- **Frequency/temperature characteristics:**
  This means the upper and lower limits of the variation of output frequency when the ambient temperature is changed under

rated conditions excluding the temperature. However, the amount of change in temperature is dependent on the normal temperature (25±2 °C).

- **Frequency/voltage coefficient:**
  This means the upper and lower limits of the variation of output frequency when the power supply voltage is changed under rated conditions excluding the power supply voltage. However, the temperature when the power supply voltage is changed is a Normal temperature (25±2 °C) and the power supply voltage to be changed is a rated value.
- **Tristate (Standby function):**
  Any output circuit used for its output must be high impedance when output is OFF is installed.
- **Frequency adjustment range:**
  Frequency variation from nominal frequency shows a minimum value under rated conditions (normal temperature, rated voltage, and rated load).
- **Frequency control characteristic:**
  This shows the variable frequency amount when voltage within a specified range is applied to the frequency control input. When voltage is applied, temperature is normal temperature (25±2 °C) and power supply voltage is a rated value.
- **Overall frequency tolerance:**
  This shows the frequency stability when the relationship with temperature, power supply, load, etc. is not taken into consideration.

4.2.1.3 Resistor

A **resistor** is a passive two-terminal electrical component that implements electrical resistance as a circuit element. The current through a resistor is in direct proportion to the voltage across the resistor's terminals. This relationship is represented by Ohm's law.

$$I = \frac{V}{R}$$

4.2.1.4 Capacitor

A **capacitor** (originally known as **condenser**) is a passive two-terminal electrical component used to store energy in an electric field. The forms of practical capacitors vary widely, but all contain at least two electrical conductors separated by a dielectric (insulator); for example, one common construction consists of metal foils separated by a thin layer of insulating film. Capacitors are widely used as parts of electrical circuits in many common electrical devices.

4.2.1.5 MCU (Microcontroller unit)

A **microcontroller** (sometimes abbreviated **µC**, **uC** or **MCU**) is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. Neither program memory in the form of NOR flash or OTP ROM is also often included on chip, as well as a typically small amount of RAM. Microcontrollers are designed for embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications.

4.2.1.6 Printed Circuit Board

A **printed circuit board**, or **PCB**, is used to mechanically support and electrically connect electronic components using conductive pathways, tracks or signal traces etched from copper sheets laminated onto a non-conductive substrate. When the board has only copper tracks and features, and no circuit elements such as capacitors, resistors or active devices have been manufactured into the actual substrate of the board, it is more correctly referred to as **printed wiring board** (**PWB**) or **etched wiring board**.

4.2.1.7 Light Emitting Diode

A **light-emitting diode** (**LED**) is a semiconductor light source. LEDs are used as indicator lamps in many devices and are increasingly used for other lighting. Appearing as practical electronic components in 1962, early LEDs emitted low-intensity red light, but modern versions are available across the visible, ultraviolet, and infrared wavelengths, with very high brightness. When a light-emitting diode is forward-biased (switched on), electrons are able to recombine with electron holes within the device, releasing energy in the form of photons. This effect is called electroluminescence and the color of the light

(corresponding to the energy of the photon) is determined by the energy gap of the semiconductor.

4.2.1.8 Berg Pins

A **Berg connector** is a brand of electrical connector used in computer hardware. Berg connectors are manufactured by Berg Electronics Corporation of St. Louis, Missouri, a division of Framatome Connectors International. Berg connectors have a 2.54 mm (=.100 inch) pitch, pins are square (0.64 mm x 0.64 mm = approx. 0.025 x 0.025 inch), and usually come as single or double row connectors.

4.2.1.9 USB connectors

Universal Serial Bus (**USB**) is an industry standard developed in the mid-1990s that defines the cables, connectors and communications protocols used in a bus for connection, communication and power supply between computers and electronic devices. USB was designed to standardize the connection of computer peripherals (including keyboards, pointing devices, digital cameras, printers, portable media players, disk drives and network adapters) to personal computers, both to communicate and to supply electric power. It has become commonplace on other devices, such as smartphones, PDAs and video game consoles.

4.2.2 Circuitry Modules

The hardware is been sub-divided into various interfacing modules. This approach will help us in assembling them with an ease. Also, if some kind of problems occurs, we will just have to work out over that particular module from where the problem is occurring. In forth coming sections you will find the schematics and also the layouts in later sections. Our Project in broader sense has been bifurcated:

1. Keyboard Module.
2. Controller Board Module.

4.2.2.1 Controller Board Module

Within this module, we will take a deep look in sub modules one by one. Here we will start from the MCU module…

4.2.2.1.1 Microcontroller Module

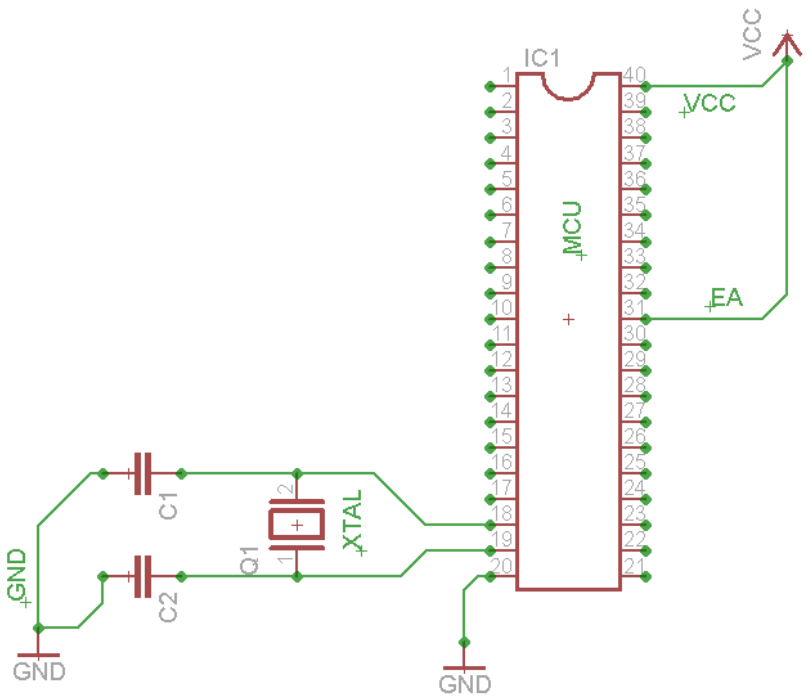


Figure 4.1: Crystal schematic for MCU enabled to use the on-chip program-data memory.

The figure which is shown above, is the basic bare bone circuit required for the 8051 MCU family to run and operate safely. For further information over this circuitry and other components which are connected to MCU, please refer to the datasheet of P89V51RD2FN/BN.

Components connected to MCU are:

- C1, C2 are capacitors, should be ceramic cap capacitors. Ranges from 12-22pF.
- XTAL- 0-40MHz is the range for this MCU. 11.0592MHz is used in our project.
- VCC- DC Current source voltage of 5V.
- GND- Ground signal.
- EA- To enable this MCU, according to the specification and directions mentioned in datasheet, we have to connect this pin to VCC for proper working.

- C1, C2 are capacitors which provided stability to oscillator and these values are recommended in the datasheet itself.
- Reset pin is been left and it will be connected to the USB to UART converter and that module will handle the Reset signal.

4.2.2.1.2 Connection of MCU with keyboard pins

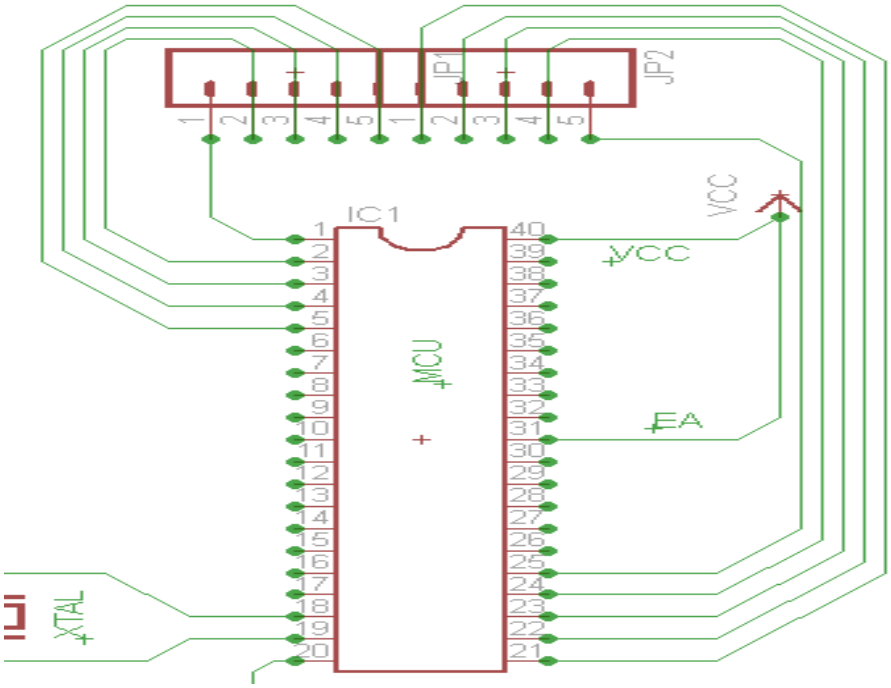


Figure 4.2 : Matrix Key pad connection with micro controller schematic.

The above schematic is extended, and the keyboard pin connection with MCU is shown. Under this, a list is there which describes this above sketch.

Description:

- Keyboard: 5x5 sized, means 25 keys on board.
- This, 5 input lines and 5 output lines. Hence, there are 10 lines which would be connected to MCU.
- From left to right, first 5 bits, or say first 5 lines are connected to MCU's lower bit valued of Port 1.
- Rest of the five bits, or say the last five lines are connected to MCU's lower bit values of Port 2.
- JP1, JP2 are the jumpers which are actually berg pins (female) which will hold the keyboard into it.

Next sub module coming is the USB interfacing module of this board. The power source, i.e. VCC comes from here only.

4.2.2.1.3USB interfacing

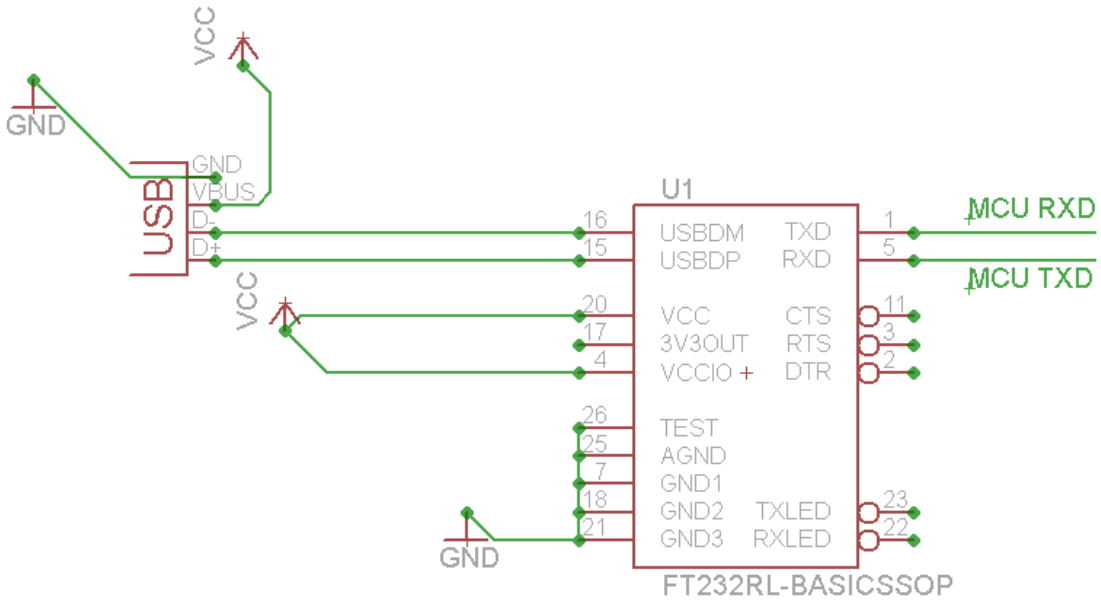


Figure 4.3: Schematic of USB-to-UART bridge converter. FT232RL.

The above schematic is the sketch of the USB to UART serial bridge IC which is in SSOP-28 package. This IC provides us the direct communication from computer to MCU. As MCU works on 5V, and computer works at higher volts than this, thus this IC acts as an interface between PC to MCU. But to fabricate this IC at this level was very difficult and it was a matter of luck. Due to its tiny kind of size we have to pick a readily available soldered breakout module of this IC for proper working. In the left, the snap of that IC is shown. This IC is manufactured by FTDI Ltd. For further information regarding the same IC, please go through the data sheet of FT232RL.

The breakout module which is been used in our project for USB to TTL converter, all the information regarding the same is been mentioned under this.

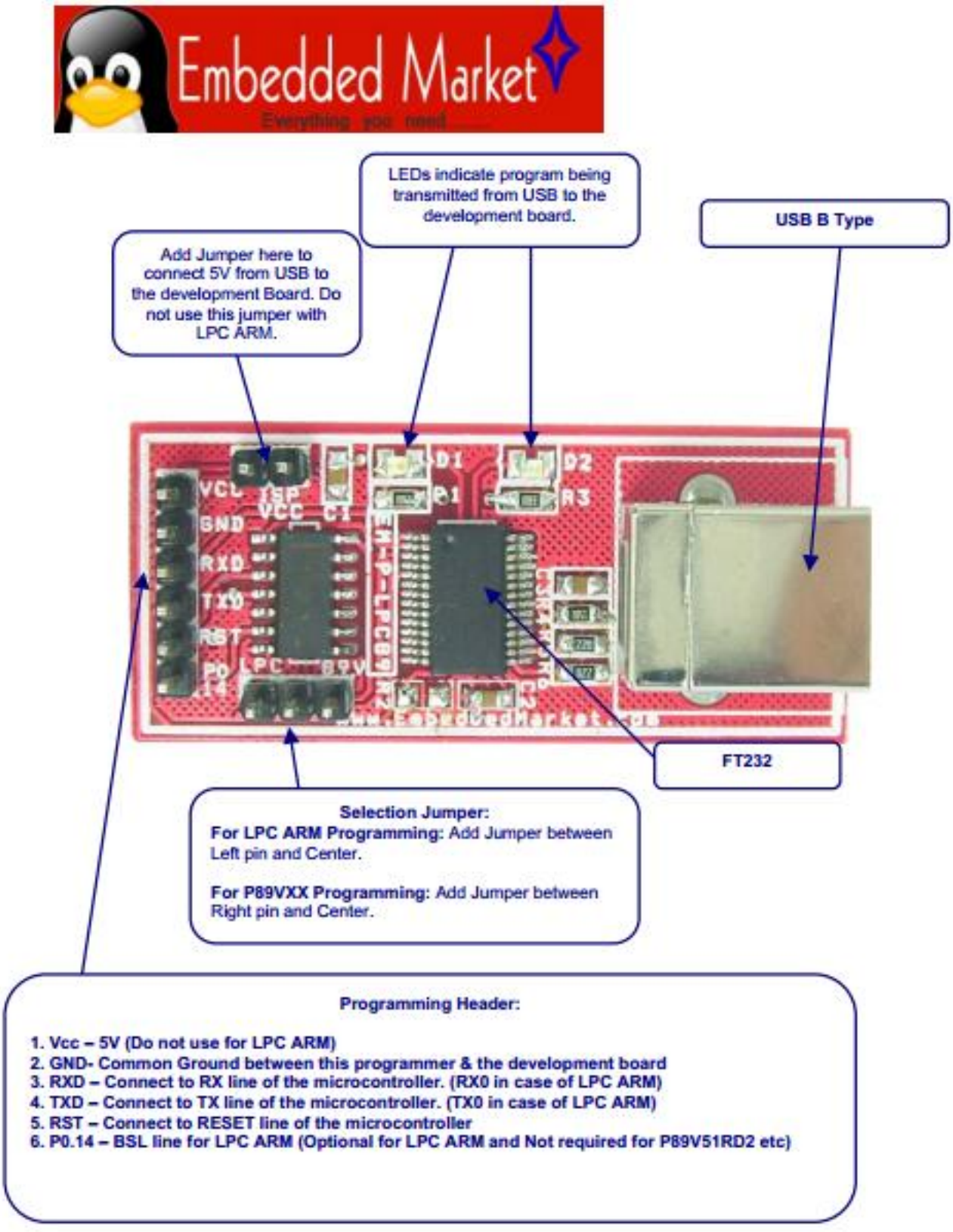


Figure 4.4 8051, ARM7 Programmer.

The above datasheet snap describes about the USB to TTL converter which is been used in our project for communication purpose between PC to MCU. And the pin descriptions are also labelled in the diagram. VCC is coming from this location, i.e. from USB port of PC.

4.2.2.1.4 Controller Board Module Layout

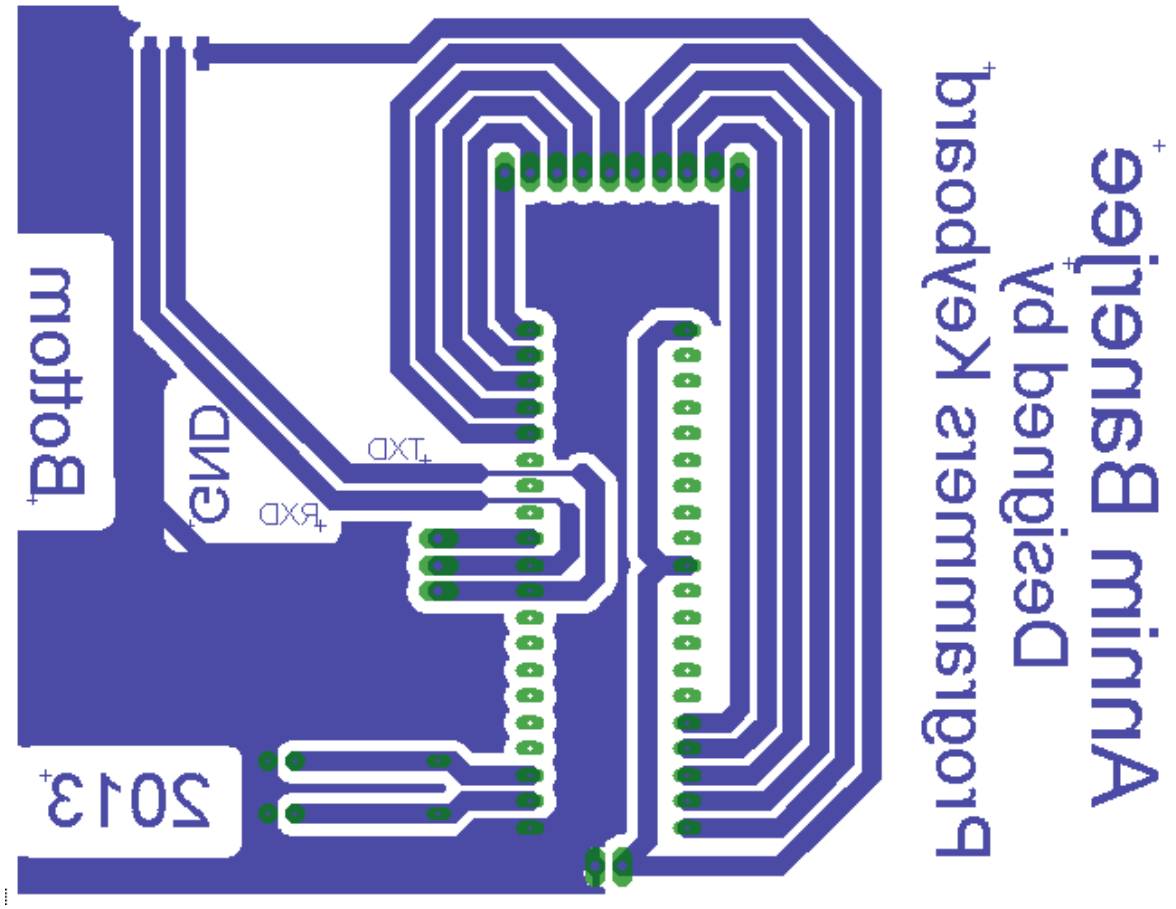


Figure 4.5: Circuit Layout of Controller board.

This is the circuit layout which will be used to fabricate the PCB board. This is the positive of the layout and already mirrored as this is the BOTTOM layer of the PCB. This layout is been created in Eagle CAD software.

4.2.2.2 Keyboard Module

Under this Module, we will be discussing the connection of the switches, aka Toggle switches and there arrangements on the board too.

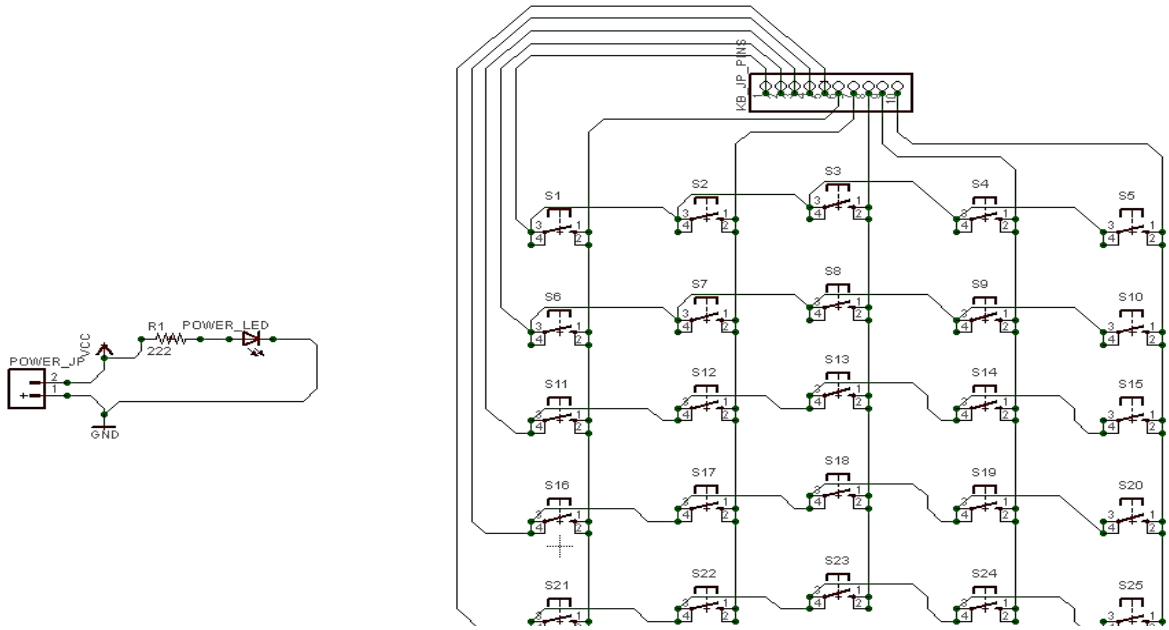


Figure 4.6: Schematic sketch of Matrix Keypad.

4.2.2.2.1 Matrix keyboard Schematic

The above sketch is the matrix arrangement of the switches, aka push button with 4 pins. Similarly, the 10 pin berg pin (male) connectors are provided so the keyboard will stand over the controller board. These 10 lines are directly connected to MCU. And MCU is continually monitoring the signal values as HIGH or LOW.

4.2.2.2.2 Keyboard Layout

Our keyboard is of Double Layer PCB board. This is designed in software, i.e. Eagle CAD software. (Left: Top layer, Right: Bottom Layer)

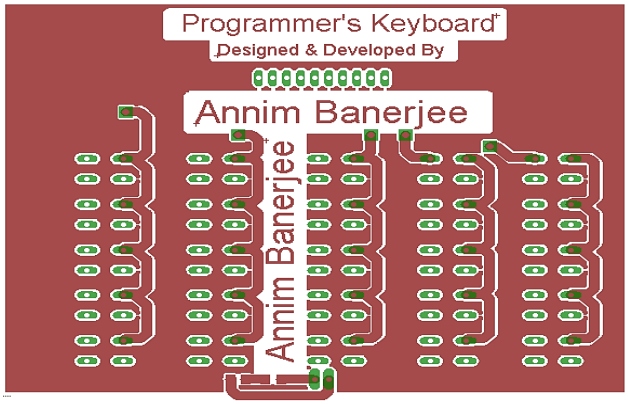


Figure 4.7: Layout of Matrix Keyboard: Top Layer

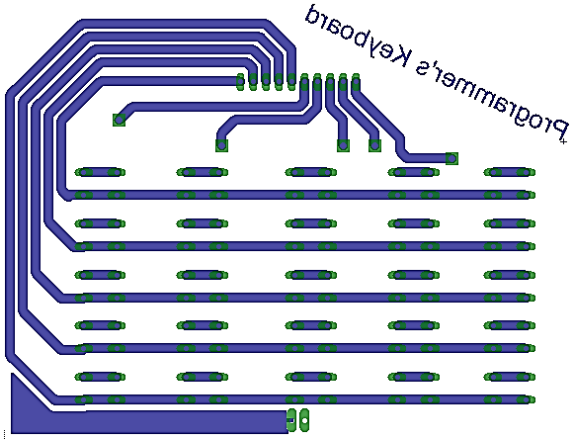


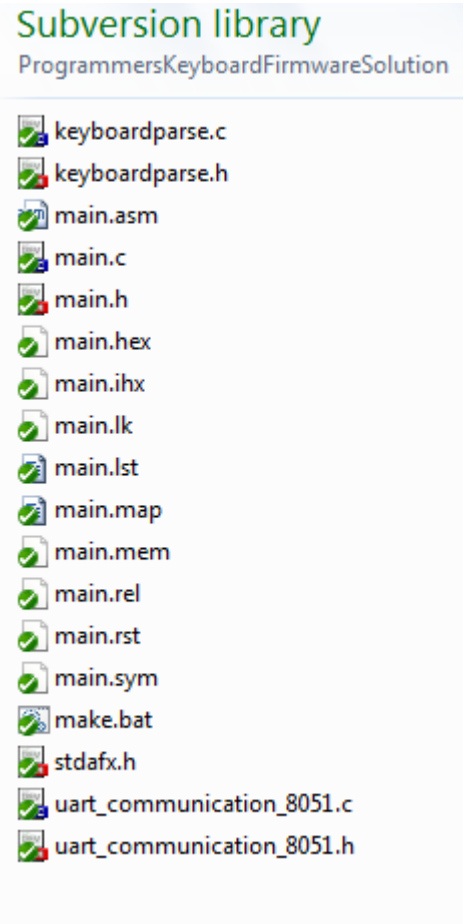
Figure 4.8: Layout of Matrix Keypad: Bottom Laver

4.3 Software

Now the working of software comes into the picture. Now, we will be discussing over the firmware code which will be running in the MCU, whose soul task is to monitor the key's states and send the information of the pressed key to PC.

4.3.1 Firmware

Taking about the firmware which has been written by us and has only one task is to send the information of the key's state to PC. The code is written in Embedded-C. Compiler used is SDCC (Small Device C Compiler). The Project file structure is shown below:



- uart_communication_8051.h: Header file for UART communication.
- Uart_communication_8051.c: Source file of the above declared prototypes.
- Keyboardparse.h: Header file for keyboard monitoring.
- Keyboardparse.c: Source file for the above declared prototypes.
- Main.h : A Header file.
- Main.c: The source file of main.h
- Stdafx.h: standard application header file which includes the necessary header files for 8051 MCU family.
- Main.hex: the hex file which is to be burned to MCU program memory. An intel hex file format.

- **Main.h**

  Content of main.h is shown in left.

```
//
//Project title:            Programmer's Keyboard
//File name:                main.h
//Content:                  The header file for main source code.
//Type:                     Document, Firmware source code.
//Author:                   Annim Banerjee.
//Project Type:             Major Project 2013.
//compiler used:            sdcc.
//MCU Type:                 8051
//MCU IC:                   NXP 89V51RD2
//

#ifndef MAIN_H
#define MAIN_H

void delayGenericSeconds(int);

#endif
```

- **Main.c**
  - Includes of this file shown in left.
  - The entry point of the program is shown below.

```
//
//Project title:            Programmer's Keyboard
//File name:                main.c
//Content:                  The main source code startup file.
//Type:                     Document, Firmware source code.
//Author:                   Annim Banerjee.
//Project Type:             Major Project 2013.
//compiler used:            sdcc.
//MCU Type:                 8051
//MCU IC:                   NXP 89V51RD2
//


//headers included here.
#include "stdafx.h"
#include "main.h"
#include "uart_communication_8051.c"
#include "keyboardparse.c"
```

```
//declaring global variables here
unsigned char cKeyPressed;

//entry point function code goes here...
void main()
{
    //start uart communication...
    init_uart();

    //start keypad...
    StartKeyboard();
    //program loop goes heree...
    while( 1 )      //an infinite loop.
    {
        cKeyPressed = getKeycode();
        if( cKeyPressed > 0 )
        {
            serial_send( TranslatedKeyCodeToAlphabets(cKeyPressed) );
            delayGenericSeconds( 500 );
        }
    }
}
```

- About the entry point:
  - There's a loop which continually get the key state.
  - That loop is an endless loop.
  - If key not pressed, do not send any information.
  - If key pressed, send information of the key pressed and which key is pressed.

- Before all, initiate the UART communication and also initiate the keyboard for its working.
- Keyboardparse.h
  - Picture shown below is the content of this header file. Few macros are defined and prototypes too are declared.

```
//
//Project title:            Programmer's Keyboard
//File name:                keyboardParse.h
//Content:                  prototypes
//Type:                     Document, Firmware source code.
//Author:                   Annim Banerjee.
//Project Type:             Major Project 2013.
//compiler used:            sdcc.
//MCU Type:                 8051
//MCU IC:                   NXP 89V51RD2
//

#ifndef KEYBOARDPARSE_H
#define KEYBOARDPARSE_H

//macros defined here...
#define COLUMN1     P2_0
#define COLUMN2     P2_1
#define COLUMN3     P2_2
#define COLUMN4     P2_3
#define COLUMN5     P2_4
#define KEYBOARD    P1

void StartKeyboard();       //initialize the keyboard by setting the required default settings.
unsigned char getKeycode(); //get the key code against the key pressed.
unsigned char TranslatedKeyCodeToNumeric( unsigned char );      //translated the keycode to numeric form from 0-9 range.
unsigned char TranslatedKeyCodeToAlphabets( unsigned char );        //translated the keycode to alphabetic form from A-Z;

#endif
```

- Keyboardparse.c

```
1   //
2   //Project title:            Programmer's Keyboard
3   //File name:                keyboardParse.C
4   //Content:                  definations
5   //Type:                     Document, Firmware source code.
6   //Author:                   Annim Banerjee.
7   //Project Type:             Major Project 2013.
8   //compiler used:            sdcc.
9   //MCU Type:                 8051
10  //MCU IC:                   NXP 89V51RD2
11  //
12
13  /*
14  #define COLUMN1     P2_0
15  #define COLUMN2     P2_1
16  #define COLUMN3     P2_2
17  #define COLUMN4     P2_3
18  #define COLUMN5     P2_4
19  #define KEYBOARD    P1
20
21  */
22
23  #include "keyboardparse.h"
24
25  void StartKeyboard()        //initialize the keyboard by setting the required default settings.
26  {
27      //make rows as o/p and columns as i/p
28      //column:   P2
29      //row:      P1;KEYBOARD
30      P2 &= 0x1f; //set first 5 bits HIGH; setting col as i/p.
31      KEYBOARD &= 0x00;   //set rows as o/p;
32  }
```

The above shown code snap showing the definition of the *StartKeyboard()* method. The comments are there which describes the code itself.

```
}
unsigned char getKeycode()  //get the key code against the key pressed.
{
    unsigned char chKey=0, cCount, cKeyValue = 1;
    for( cCount = 0; cCount < 5; cCount++)              //loop for 5 rows.
    {
        KEYBOARD &= ~(0x10>>cCount);                    //make rows low one by one...

        if(!COLUMN1)
        {
        if(!COLUMN2)
        {
        if(!COLUMN3)
        {
        if(!COLUMN4)
        {
        if(!COLUMN5)
        {
        cKeyValue += 5;                                 //incrrease value for next row scanning.
        KEYBOARD |= 0x10>>cCount;                       //make the rows HIGH again.
    }
    return FALSE;
}
```

The above shown code snap is the code where we keep looking for the key states and if any of the key get pressed, immediately a corresponding value is generated. The comments are there which gives a pretty clear description about the code.

```
unsigned char TranslatedKeyCodeToNumeric( unsigned char chCode)     //translated the keycode to numeric form from 0-9 range.
{
    unsigned char chKey;

    chKey = chCode + 47;

    return chKey;
}

unsigned char TranslatedKeyCodeToAlphabets( unsigned char chCode)       //translated the keycode to alphabetic form from A-Z;
{
    unsigned char chKey;

    chKey = chCode + 64;

    return chKey;
}
```

These are two functions which convert the alphabet to number and from number to alphabet. One of the function is been called in the main entry point and the translated value is been send to the PC end for further processing.

- UART communication header content

```
//
//Project title:            Programmer's Keyboard
//File name:                uart_communication_8051.h
//Content:                  prototypes
//Type:                     Document, Firmware source code.
//Author:                   Annim Banerjee.
//Project Type:             Major Project 2013.
//compiler used:            sdcc.
//MCU Type:                 8051
//MCU IC:                   NXP 89V51RD2
//

#ifndef UART_COMMUNICATION_8051_H
#define UART_COMMUNICATION_8051_H

void init_uart();                               //initialize the uart communication by setting up the environment in 8051.
unsigned char serial_read();                    //for serial read.
void serial_send( unsigned char );              //for serial send operation.

#endif
```

- UART communication source file content, with comments too.

```
//
//Project title:            Programmer's Keyboard
//File name:                uart_communication_8051.c
//Content:                  defination
//Type:                     Document, Firmware source code.
//Author:                   Annim Banerjee.
//Project Type:             Major Project 2013.
//compiler used:            sdcc.
//MCU Type:                 8051
//MCU IC:                   NXP 89V51RD2
//

//including headers here...
#include "uart_communication_8051.h"

void init_uart()
{
    TMOD = 0x20;    //set timer2 to 8 bit Auto-reload mode.
    SCON = 0x50;    //enable reception, set serial port mode to 8 bit UART.
    TH1 = 0xfd;     //set baud rate to 9600 for 11.0592MHz.
    TL1 = 0xfd;

    //start timer
    TR1 = 1;
}
```

- UART Read and Write function content with comments

```

unsigned char serial_read()
{
    while(!RI);     //wait for receive interrupt flag
    RI=0;           //now clear the flag if data received.
    return SBUF;
}

void serial_send( unsigned char cchText)
{

    SBUF=cchText;       //set new data to send now.
    while(!TI);         //wait for last data to send .
    TI=0;               //clear the flag.
}
```

- Stdafx.h: Contents with comments

```
//
//Project title:            Programmer's Keyboard
//File name:                stdafx.h
//Content:                  The standard application header file
//Type:                     Document, Firmware source code.
//Author:                   Annim Banerjee.
//Project Type:             Major Project 2013.
//compiler used:            sdcc.
//MCU Type:                 8051
//MCU IC:                   NXP 89V51RD2
//

#ifndef STDAFX_H
#define STDAFX_H


//include standard, essential header files here
#include <8051.h>



//define macros here...
#define TRUE    1
#define FALSE   0


#endif
```

- Makefile.bat

This file is been created for compiling the source code main.c and the compiler in result, produces the hex file. That hex file is then burnt on to MCU using *FlahsMagic* software provided by NXP Ltd.

```
cls
@echo off
echo Compiling your source code now...
sdcc main.c
if errorlevel 1 goto compileerror
echo compilation success.
echo Now packing ipx file to hex file
packihx < main.ihx > main.hex
if errorlevel 1 goto packerror
echo packing done!
dir main.*
goto exitpause
:compileerror
echo Error in compilation time.
goto exitpause
:packerror
echo error while packing file
goto exitpause
:exitpause
pause
```

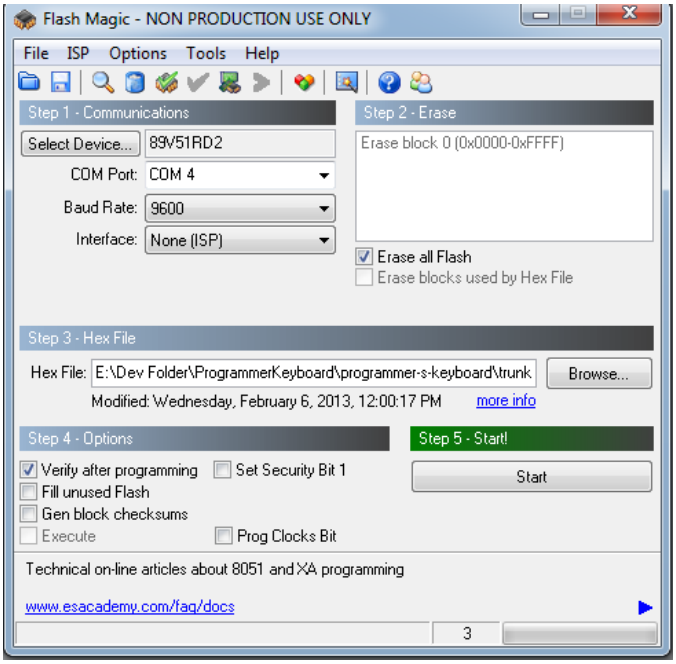


Figure 4.9: FlashMagic Application Window

This window is of *FlashMagic* software via which the hex file is been dumped in to MCU. There is a programmer used to port the hex file in to MCU and this is also been used for serial communication from MCU to PC. This device is been provided by Embedded Market.

4.3.2 Software on PC end

4.3.2.1 Introduction

"Programmer's Keyboard", named software is the software which will be sitting in PC front and it will be monitoring the keyboard's information as it comes up to the port. The purpose of this software is to read the data coming on com port and fetch up the key so pressed over the current working window.

This software is been developed in two different programming languages, i.e. in C++ and in C#. Aim after developing the same in two different languages was to gain experience and also see the prototyping experiences which one can see and observe the difference in levels of quality and dependency. In the next few pages, a detailed description of *"*Programmer's Keyboard*"* is been mentioned regarding the modules, coding and also about UI.

4.3.3 Requirements & Analysis

Gathering the requirement, which will be useful to design, develop and frame this software at its best. Below this, many point are mentioned which are then and there analysed too!

- What you need for establishing communication over a COM port?
    - The COM port name.
    - Baud Rate.
    - Parity.
    - Stop Bit.
    - IO writing APIs.
    - The Device Drivers.
- At which languages this keyboard will work for?
    - C programming language.
    - C++ programming language.
    - Java programming language.
    - C# programming language.
    - VB programming language.
- To connect to the device from UI, how?
    - A button will be enough to provide a way to interact for connecting to device and also to disconnect, as the COM port will close on disconnecting from *programmer's keyboard.*
- Controlling point on the go…
    - There should be some kind a way to change the currently working programming language while the application is running.
    - Application should be minimized or shouldn't appear in taskbar.
    - Hence, after minimization, the application should go in notification tray with the notifying icon form.
- Developing Technologies and Tools
    - Native development.
    - Visual C++ 2010 Express edition IDE.
    - Res Edit Resource editor.
    - For DOT.NET platform, Visual C# 2010 Express edition.
    - DOT.NET platform 3.5v.

In the next segment, a detailed discussion over how these requirements are been then designed and implemented.

4.3.4 Designing

The designing of UI, especially under windows platform is very easy now a day. But, developing the UI in native coding style is bit tedious job, not taking support of any kind of framework. Thus, to do this, we have to use ResEdit resource editor, with the help of this IDE kind of software; we can portray the UI of our application. This snap to the left, is the ResEdit Resource editor IDE, and the window show is our applications main window which will come up above all for connecting and disconnecting from our keyboard device. Once the resource script is been generated by this IDE, we can import the script to our VS2010 VC++ IDE which will be reflected in solution explorer as show to left, and compile the program to generate window.

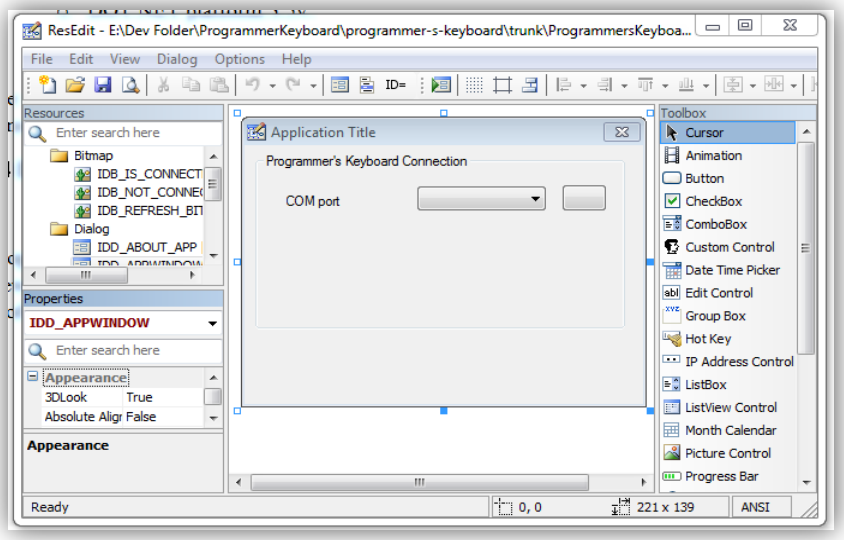


Figure 4.10: The ResEdit window

In DOT.NET platform, the UI designing is very fluid and convenient as there will be provided with many rich controls and also with drag-n-drop feature to design UI. For now, continuing with the native development, further more designing of UI is been done by hard coding too. This is been done by using the windows sdk, in which APIs are been provided to do such task either on runtime or at the initial phase of window creation. Snap to the left, is a dummy window created by using resource script and also few hard codes.

Programmer's keyboard application will be minimized to let the user work over, thus, there is a provision to put the application in notification tray. This feature is there in both the versions.

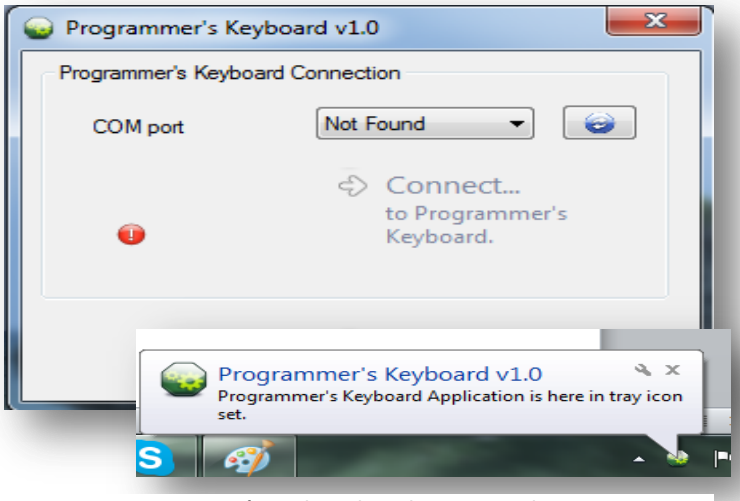


Figure4.11: Programmer's Keyboard Application window

4.3.5 Implementation

In this segment, after having done with designing, we will not discussing the coding portion, especially regarding developing the application under C programming language, i.e. native development.

- Considerations
    - Codes are quite similar in two versions.
    - UI is almost identical.

That's why, the explanation of the application, and how it's been implemented, is mentioned for once and goes with both the versions with little changes.

4.3.5.1 Solution File Structure

According to the Solution file structure so formed in project, according to that sequence we shall discuss.

- ProgrammersKeyboard.h: The header file having all the prototypes declared, like user defined functions and procedures, some global variables and MACROS too.
- resource.h: a header file specially created for having the MACROS as ID for the resources.

- stdafx.h: stands for “standard application framework”, this file contains the essential header files included for GUI programming in window platform.
- iconApp.ico: the applications logo/icon as resource.
- resource.rc: the only resource script file so generated from ResEdit resource editor IDE.
- ProgrammersKeyboard.cpp: The source file corresponding toProgrammersKeyboard.h header file, which contains the defination of all the functions and procedures like windowing and other event handlers too.
- stdafx.cpp: the source file corresponding to the stdafx.h header file.

Contents explanation of each file is been discussed in next segment.

4.3.5.1.1 ProgrammersKeyboard.h

The header file having all the prototypes declared, like user defined functions and procedures, some global variables and MACROS too. There is a list mentioned below gives you the description of each and every user defined function as decalred in this header file.

- WinMain function is not a user defined function; this is actually the **entry point** of the GUI based program. It always takes four parameters which are the new instance data type which OS provides to application when launched, next argument is also instance handle which represent any previous instance if running in memory already, third argument is command line argument and the fourth one is the option as integer value which specifies the window state at the time of launching the window. This function also returns an integer type data to OS after the application ends up, which is as an exit code.

- There are two functions that are prototypes of the CallBack procedures where all the event handlers corresponding to the window are been written in. These CallBack procedures takes four arguments and this is also not a user defined function, these format or say the function prototype is been designed and said mandatory by Microsoft itself and for further information regarding to these functions, please refer Microsoft APIs documentation. These functions take HWND as handle to window as first parameter, UINT i.e. unsigned integer as second argument which specifies the Window Message or say the intimation of the event, third and fourth one are the Wide Parameter and Low parameter which comes with the windows message giving

information as companion to it. WPARAM and LPARAM are 32 bit long data contains extra information many a times.

- Rest of the functions are user defined functions. *PerformEssentials* takes four parameters and by using that it executes few mandatory steps to put the window in front on screen with all basic settings. Like setting up the icon, title text, finding com port and putting it in combo box list as data and setting the buttons states for connecting and disconnecting, and putting image on button and other supplements task.
- PopulateCOMPort is another user defined function which takes same four arguments and fill up the combo box with the all available COM port on to the system. This function is thus used for refreshing purpose too!!!
- Other functions so named in such a way that the name itself can describe its purpose, what it is for!
- This is also a user defined function which has only one job is to keep seeking for any data on COM port and get that data to some kind of storage area.
- PressShiftKey, ReleaseShiftKey, HitEnterKey, HitTabKey, HitSpaceKey are some user defined functions, which simulates the event as key is pressed on immediate keyboard.
- **4.3.5.2 Control Flow of software**

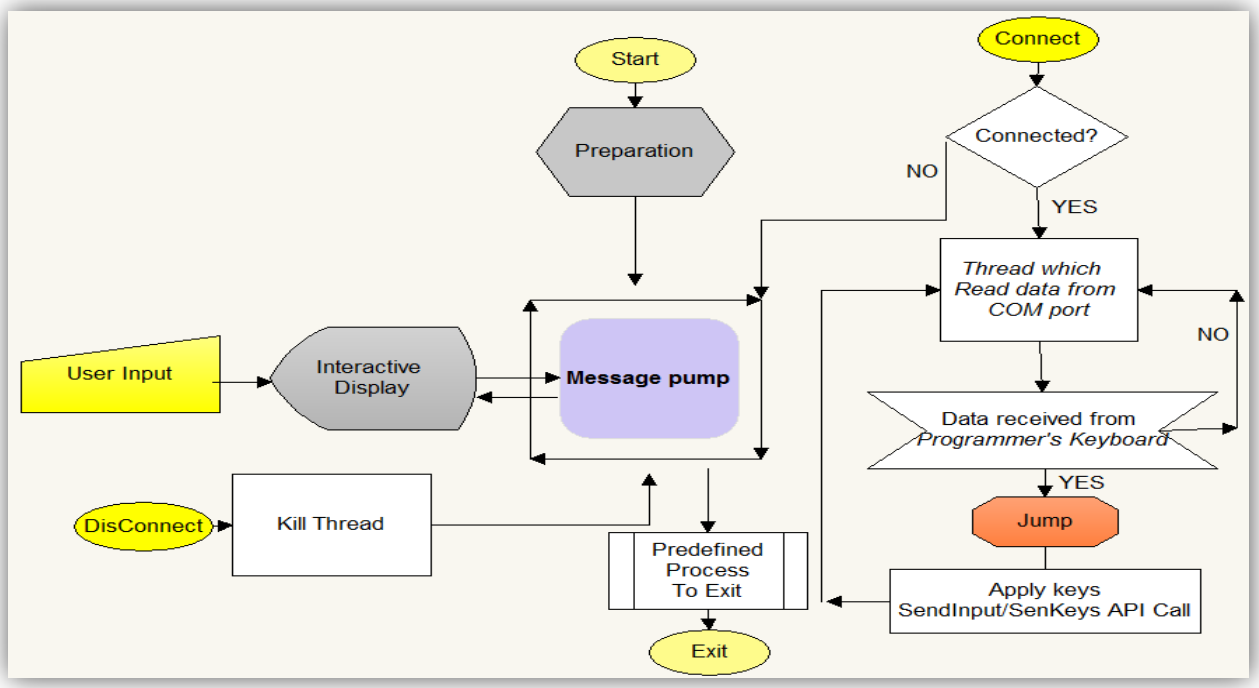


Figure 2.12 : Control Flow Diagram.

CHAPTER – V

RESULTS & DISCUSSION

5.1 Snapshots

In this segment you will find some of the snaps of the hardware and working of the software

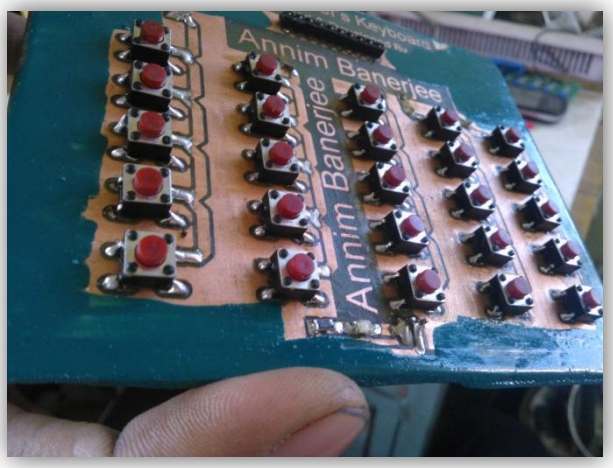


Figure5.1 : Keyboard: TOP Layer

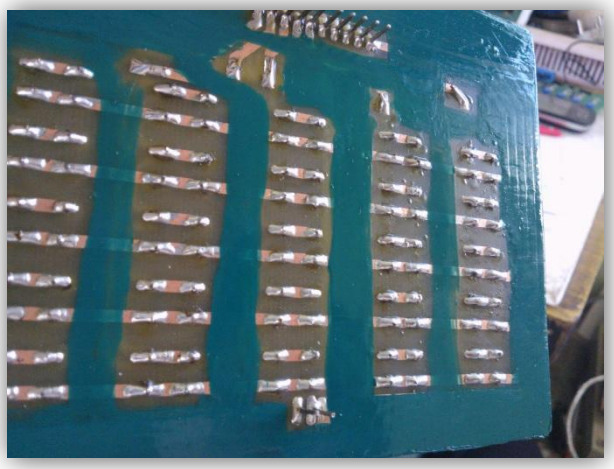


Figure5.2 : Keyboard: BOTTOM Layer

Figure 3.3: Controller Board Top Layer

which will be running on PC end. So, starting from hardware…

To the top, it is the TOP layer of keyboard part, of hardware; it's the BOTTOM layer of the same. After fabrication, it has been painted by fabric paint of green in colour.

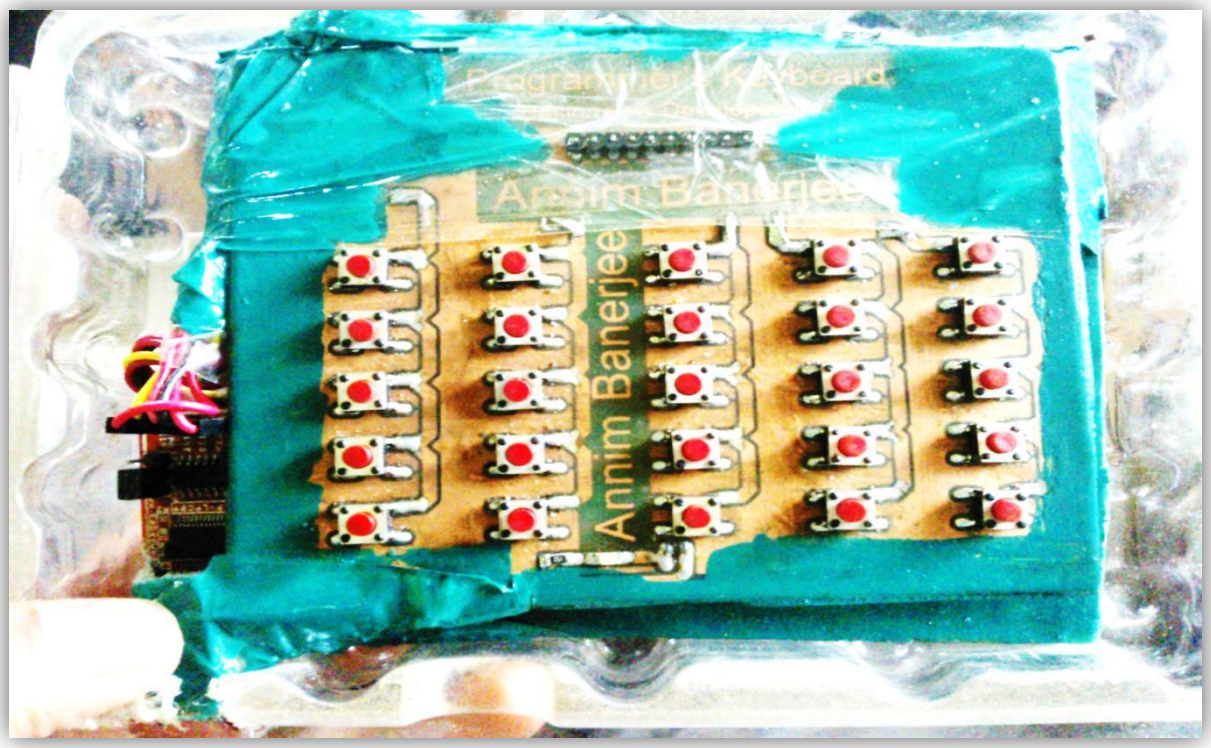


Figure5.4 : Assembled and packed: Keyboard

This snap, is the TOP layer of the controller board, of the hardware. This will be attached to the keyboard module, and keyboard module will sit on top of the controller board. The above image is a top view of our keyboard hardware which is been packed in a HDD case and been painted again with fabric paint for preventing the hardware from rusting. Further for putting a rigidity or say firmness, we have taped it with packaging tape. Some more snaps of the final assembled hardware are shown.

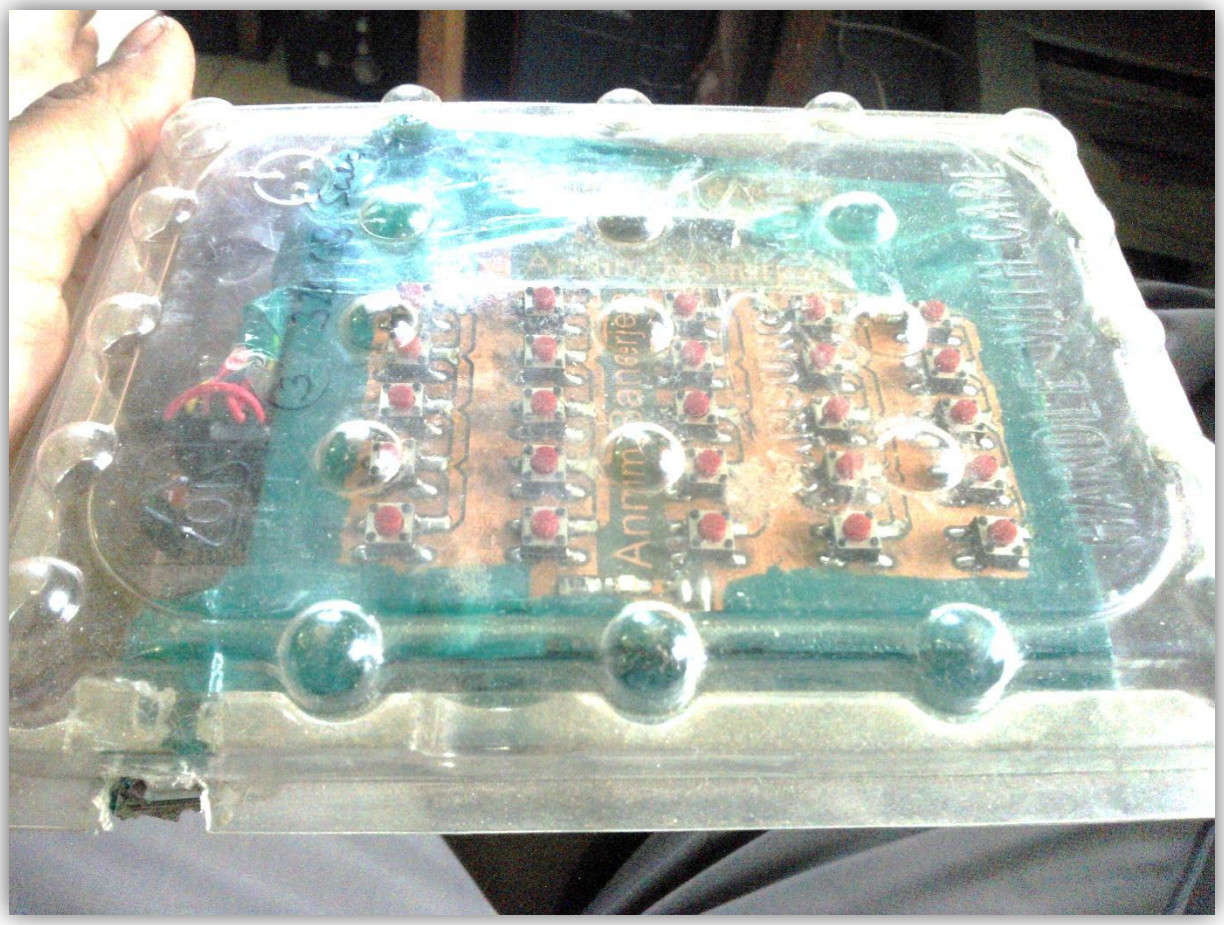


Figure5.5 : Packed keyboard

The hardware is then tested and verified as per the requirement is been created. You can assume that BBT test is been done and our project is been succeeded in that test. Further on,

Figure 5.6: Programmer's Keyboard window developed in DOT.NET

software side, you have already seen some snaps of software which is been created for this hardware, but few more snaps are shown below which are developed in Dot.NET platform for the same …

After when the device is connected and COM port is found, you can connect to device a on minimizing this window it will go to notification tray as notifying icon. And from there you can do you any changes by using the context menu options.

And when your work is done you can disconnect the device and can remove it safely.

CHAPTER – VI

CONCLUSION & SCOPE OF WORK

6.1 Conclusion

Putting all together, in this segment-

- A keyboard with USB interfacing for connectivity with computer.
- 25 keys with separate functionalities to each, as much as possible.
- Default maximum rate of communication between device and computer is at 9600 baud rate.
- Matrix key logic is been implemented to make keyboard.
- Hardware front-
    - Divided into two module-
        - Keyboard module.
            - Separate board which got all the keys mounted on it. And a power led too.
        - Controller module.
            - Board on which the controller is been planted with USB interfacing module too attached with it.
    - Firmware code-
        - Program written in Embedded C.
        - To burn code- Flash Magic is used, provided by NXP.
    - Controller used-
        - NXP P89V51RD2BN/FN
- Software front-
    - On PC front
        - Software is been developed in two versions-
            - One written in C language using Win APIs.
            - Other is written in C#, implemented on Dot.NET platform v3.5.
- Benefits
    - This keyboard puts most of the MRU type code snippets on editor window then and there on one push of switch.
    - Act as plugin to those IDEs which didn’t got an Auto Completion feature.

❖ **Scope of work**

- The USB module should be included in the circuitry of controller board with best suitable USB-to-Serial converter chip.
- Can be converted into a normal workable keyboard with 102, 104 keys.
- Can be converted into a controller for specific game.
- A good logic should be developed to replace the controller, i.e. MCU from the project and also UART communication can be re-designed by using very simple TTL logics. A small mechanism is been shown which might replace the MCU and also the USB-to-Serial converter chip, but this is an experimental, theoretical piece of information, shown in next segment…

◆ **Another possible approach**

- To replace MCU from board

  To replace MCU from board we have to go with TTL logics as for now this is one of the ways to go and deal with. For this we have to recall the conditions & inputs and outputs.

  We have-

  - 25 keys-25 input lines.
  - Take 5 bits to express 25 in binary form.
  - Hence, 5 output lines.

Thus, we have to deduce a kind of encoder logic which encodes 25 input lines to 455 output lines all together, iff one key is pressed at a time. We have to get a truth table for this to deduce an expression for our encoder logic.

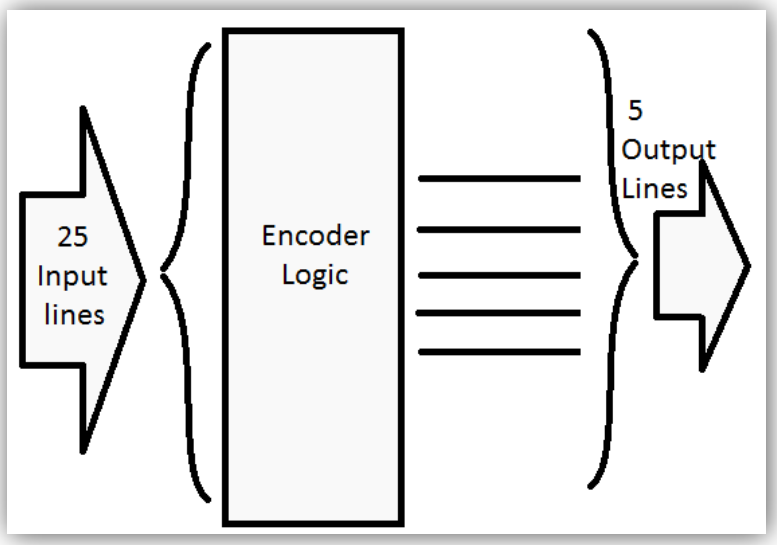


Figure 6.1: Block Diagram of encoder logic

In this logic diagram shown in previous page, there are many instances where many combinations in output lines will be not used as in 5 bit, at max value could be 32, but here only 25 values or say combinations will be considered as valid ones.

Consider input value variable names: A-Y

Consider output value variable names: O1, O2, O3, O4, O5.

Now let's construct the truth table for the above encoder logic and then after we shall deduce an expression from truth table.

Input Variables																									**Output Variables**					**Dec**
LSB																								**MSB**		**MSB**				**LSB**	
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**		**O1**	**O2**	**O3**	**O4**	**O5**	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	1	1
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	1	0	2
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	1	1	3
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	1	0	0	4
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	1	0	1	5
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	1	1	0	6
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	1	1	1	7
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	1	0	0	0	8
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	1	0	0	1	9
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	1	0	1	0	10
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	1	0	1	1	11
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0		0	1	1	0	0	12
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0		0	1	1	0	1	13
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0		0	1	1	1	0	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0		0	1	1	1	1	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0		1	0	0	0	0	16

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0		1	0	0	0	1	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0		1	0	0	1	0	18
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0		1	0	0	1	1	19
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0		1	0	1	0	0	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0		1	0	1	0	1	21
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0		1	0	1	1	0	22
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0		1	0	1	1	1	23
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0		1	1	0	0	0	24
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1		1	1	0	0	1	25

Table 6.1: Encoder logic Truth Table.

From the above truth table of 25 variables and 5 output variables, we shall deduce the Boolean expression one by one…

O1 = A+C+E+G+I+K+M+O+Q+S+U+W+Y.

O2= B+C+F+G+J+K+N+O+R+S+V+W.

O3= D+E+F+G+L+M+N+O+P+T+U+V+W.

O4= H+I+J+K+L+M+N+O+X+Y.

O5= P+Q+R+S+T+U+V+W+X+Y.

Where + means OR logic operation.

The truth table so deduced may or may not be the best optimal one. As this logic is getting more complex unlike it appears to. To calculate numbers of OR gates shall be required would be, 53 OR gates. Now if we pick a quad 2-OR gate IC, then around 14 OR gate chips are required. This may increase the cost of board and also consume space. This is because we are considering 25 keys here…

Now the data transmission between this logic to computer, this is a real challenge where a highly equipped machine has to communicate with our hardware. Available interfacing technologies are-

- USB v1.1, v2.0, v3.0, HS etc.
- Serial port as RS232, i.e. DB9 port with 9 pins.
- Serial Port as printer port, i.e. DB25 with 25 pins.

- **Protocol** to call for communication to happen-**using USB technology…**
  - We got one D+ pin and D- pin, one pin got data and other one is the mirror image of the same in negative mean.
  - Few more hardware required –
    - One 8-bit parallel in, serial out shift register chip-
      - This takes 5 input lines parallel and put the same data out in serial way.
      - Still 3 bits left, consider as *don't care*.
      - Requires a clock tick to push serially data out from chip towards computer port and get it register on computer's port buffer.
  - Case-
    - Computer peeks onto port buffer for any data except 0x00f.
    - Iff any button pressed-
      - For once and for first time, a high signal shall be issued over D+ line and negative of the same too over D-.
      - The point when the software program finds a change in port buffer, it then recognised as ***START*** of the communication.
      - XOR the computer's port buffer with 0x00f data.
      - For next 8 cycles…
        - From computer, D+ carries a dummy HIGH signal to give a clock to parallel-in-serial-out sift register and over same channel D+, one by one bit will be transferred in to computers port buffer by ORing the value with previously stored data in computer's port buffer.
        - The final value so obtained in the computer's port buffer will be in binary format indicating the n^{th} number of key is been pressed!
        - Immediately XOR the computer's port buffer with 0x00f for clearing the buffer.
      - Put a voltage inverter will put the mirror of D+ in D- BUS line for correct communication as per USB data communication protocol.
      - Place a fuse on D+ and D- line for eliminating noise and have strong strength of data in BUS lines.

- **Protocol** to call for communication to happen-**using Serial port DB9, RS232 technology…**
    - With this port, we got one more signal BUS line and that is DTR signal which stands for ***Detect-To-Ready*** signal which acts as ***RESET*** signal for any MCU or any digital system. NOTing to this signal will be better in terms of usage.
    - Case regarding this technology is quite similar to USB technology based theory-protocol. We shall send a DUMMY ***START*** signal to computer from DTR BUS line and over software end, program will constantly pulling LOW. If found HIGH, that indicates a START condition and rest goes the same. This DTR will be used to give a clock tick to parallel-in-serial-out shift register to get the data bits out from it.
    - The TXD line of DB9 pin shall be used to put data from keyboard to computer's port buffer.
    - A voltage doubler of +10V circuit will be required to put data which will be evaluated for validation purpose by the PC.

A practical implementation of this theory may leads to an opportunity for creating a good and new technique for communication between a simple digital IO boards to computer. Although, the theory and protocol so discussed above, require a recitation from an experts of this domain. This is purely experimental, information content. This information can be shared, changed, distributable on your own risk. There are lot more work left which turns as scope of work to do.