CHAPTER 3

METHODOLOGY

Methodology refers to more than a simple set of methods; rather it refers to the rationale and the philosophical assumptions that underlie a particular study. This often refers to anything and everything that can be encapsulated for a discipline or a series of processes, activities and tasks. The process of methodology for this project is shown in Figure 3.1.

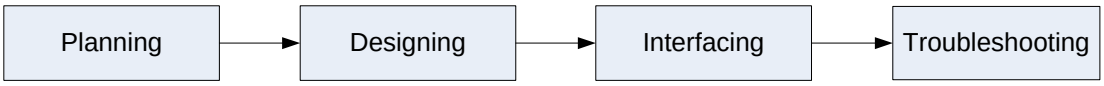

Figure 3.1: Overall Methodology Process

3.1 Planning

Planning is the (psychological) process of thinking about the activities required to create a desired future on some scale. This thought process is essential to the creation and refinement of a plan, or integration of it with other plans. The term is also used to describe the formal procedures used in an endeavor, such as the creation of documents, diagrams or meetings to discuss the important issues to be addressed, the objectives to be met and the strategy to be followed. Beyond this, planning has a different meaning depending on the political or economic context in which it is used.

3.2 Design

For the designing process, it is separated in two major processes. One is the designing hardware and the other one is designing software. The figure of the process is shown as in Figure 3.2.

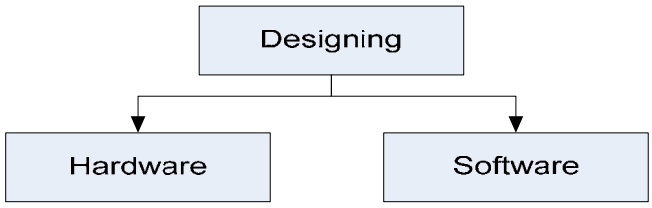

Figure 3.2: Block Diagram of Designing Process

3.2.1 Hardware Design

In designing hardware, there is a need to make sure all the process are followed the sequence of the flow in order to make sure all the process done finely. For the hardware in this project, it required to design all of the part of the block diagram as shown in Figure 3.3.

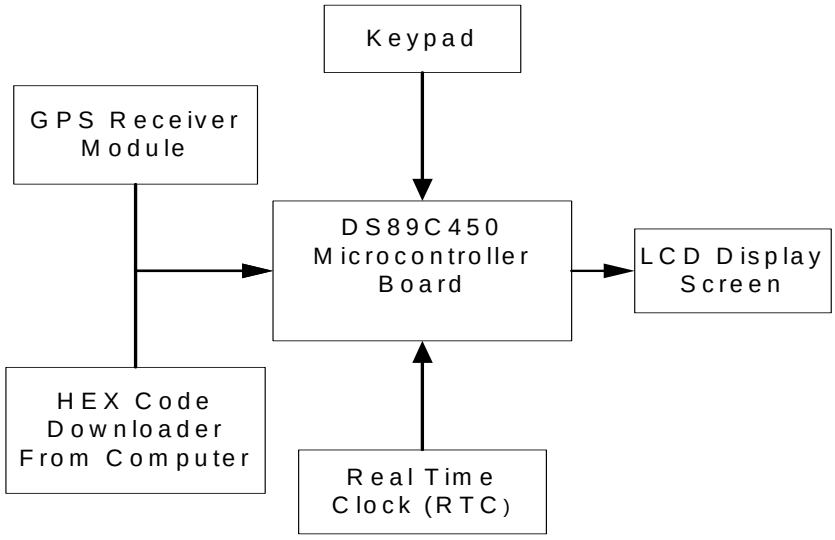

Figure 3.3: Block Diagram Overall System for Hardware

- *Microcontroller board*

  It is used to process data through information of longitude and latitude which is received from GPS Module. In this process the calculation of pray time system involved.

- *LCD Display*

  It is used to display pray time for the five time of prayer.

- *Keypad*

  It is used to select the pray time depends on the user needed.

- *Real Time Clock (RTC)*

  It used to give the real time too user like digital clock.

- *GPS Receiver Module (JP7)*

  It used to receive the data from satellite, contains the position of longitude and latitude.

- *HEX Code Downloader from Computer*

  It used to do the first calculation for the pray time using C programming.

3.2.1.1 Microcontroller Board Design

In designing DS89C450 microcontroller board, the ORCAD Released 9.1 software was used in order to create the circuit and the layout to make printed circuit Board (PCB). In order to design a circuit, there is a need to have the data sheet for the entire component that will be used to prevent from using the wrong footprint when creating the layout. The design starts using ORCAD Capture CIS as shown in Figure3.4. The design rules check will be execute to make sure the connection between each part do not have any error. When there was no error, the net-list was creating to make layout process.

After all the designing circuit processes are done, there is another step to be completed before PCB can be created. The layout process was done by using Layout Plus. Within this process, the component for the board is arranged manually depends on size of the board needed. After that, the board is auto-route to make the final design for the board. Then, the editing for the pad-stack, layers, footprint and obstacle was done to make sure the board is no error. The layout for the board is shown in the Figure 3.5a and 3.5b. That was the complete layout for double layer. The layout design will be sent to the Microcontroller Fabrication Laboratory for the process of making the PCB.

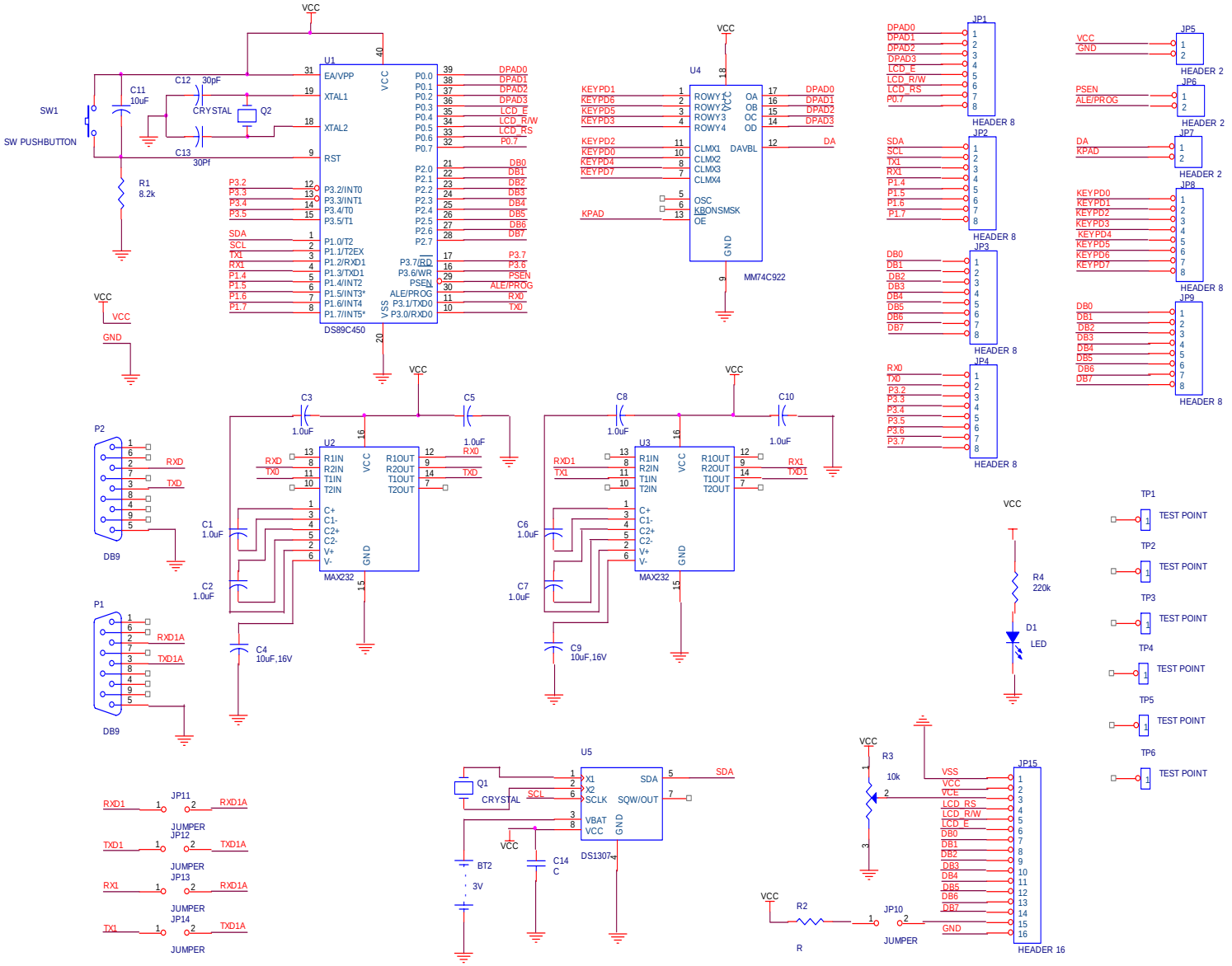

Figure 3.4: DS89C450 Circuit Designing Using ORCAD Capture CIS

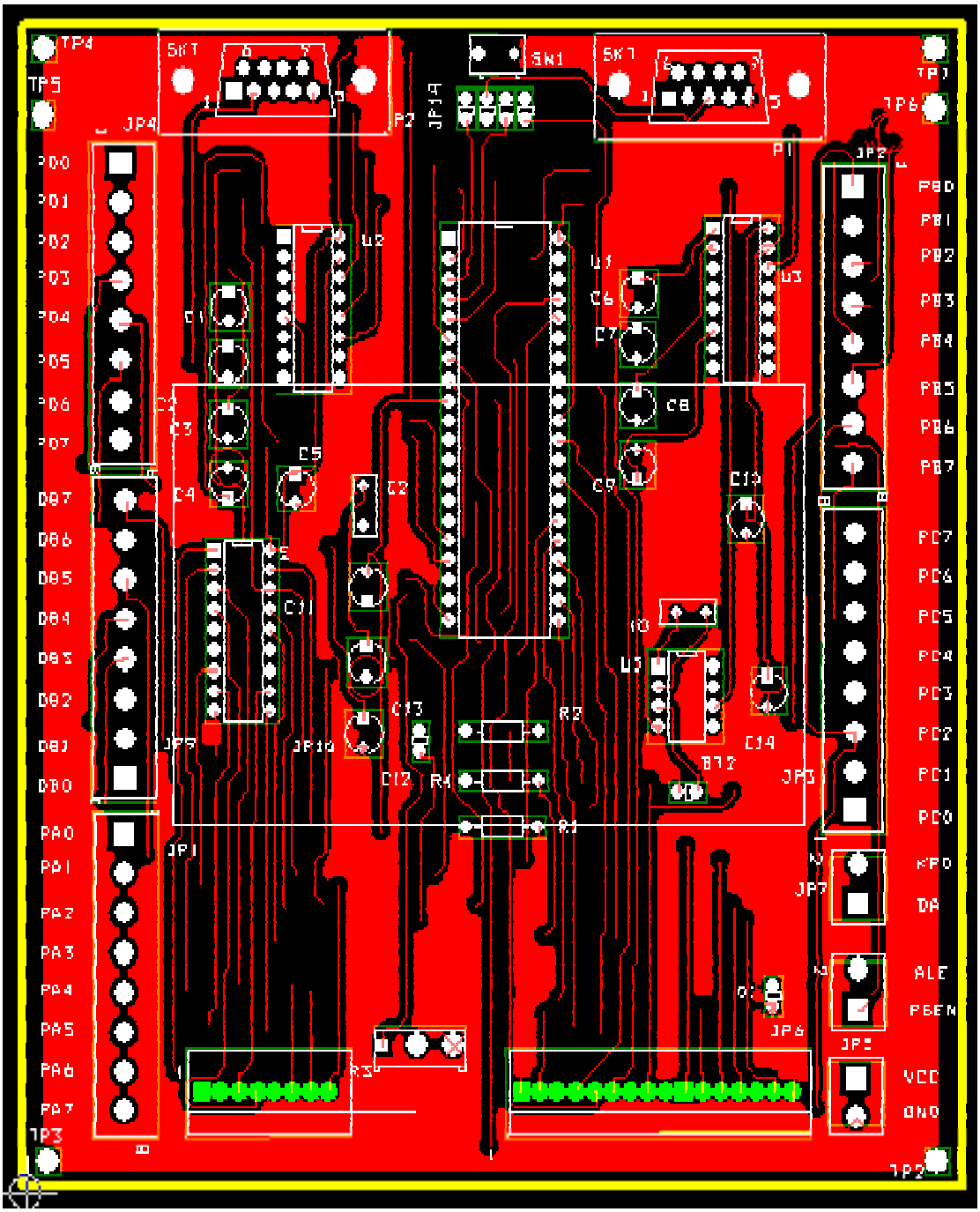

Figure 3.5a: Layout for the Bottom Layer

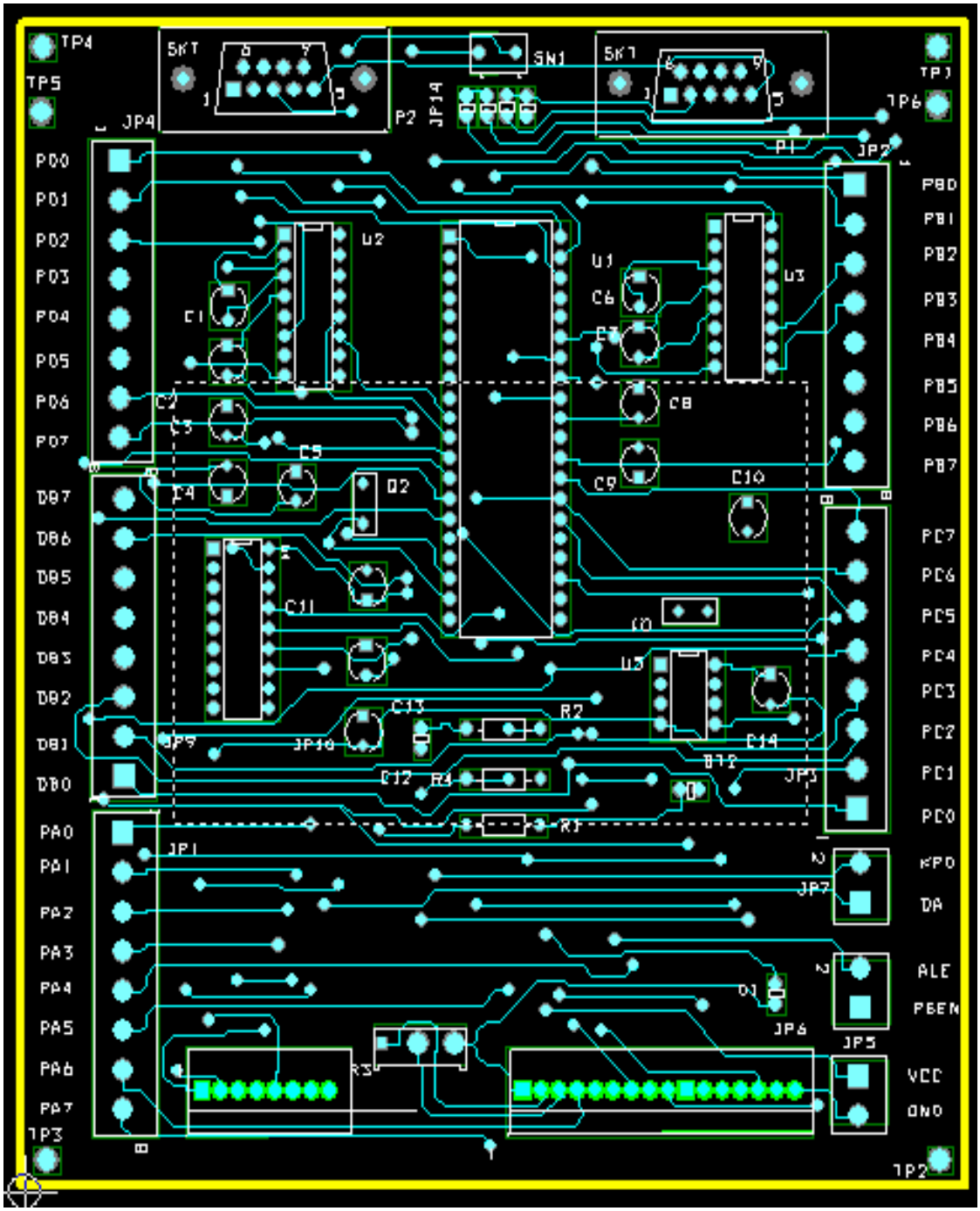

Figure 3.5b: Layout for the Top Layer

3.2.1.2 Microcontroller Architecture

Microprocessors are generally utilized for relatively high performance applications where cost and size are not critical selection criteria. Because microprocessor chips have their entire function dedicated to the Control Processor Unit (CPU) and thus have room for more circuitry to increase execution speed, they can achieve very high-levels of processing power. However, microprocessors require external memory and I/O hardware. Microprocessor chips are used in desktop PCs and workstations where software compatibility, performance, generality, and flexibility are important. By contrast, microcontroller chips are usually designed to minimize the total chip count and cost by incorporating memory and I/O on the chip. They are often "application specialized" at the expense of flexibility. In some cases, the microcontroller has enough resources on-chip that it is the only IC required for a product. Examples of a single-chip application include the key fob used to arm a security system, a toaster, or hand-held games. The hardware interfaces of both devices have much in common, and those of the microcontrollers are generally a simplified subset of the microprocessor. The primary design goals for each type of chip can be summarized this way:

- Microprocessors are most flexible
- Microcontrollers are most compact

There are also differences in the basic CPU architectures used, and these tend to reflect the application. Microprocessor based machines usually have a von Neumann architecture with a single memory for both programs and data to allow maximum flexibility in allocation of memory. Microcontroller chips, on the other hand, frequently embody the Harvard architecture, which has separate memories for programs and data. Figure 3.6a and Figure 3.6b illustrates this difference.

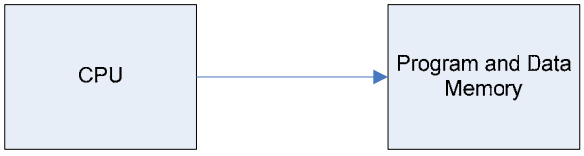

Figure 3.6 a: The von Neumann Architecture [10]

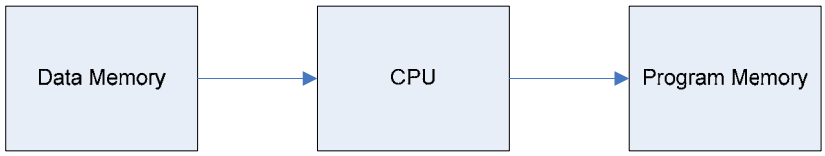

Figure 3.6 b: The Harvard Architecture Block Diagram [10].

The advantage of the Harvard architecture has for embedded applications is due to the two types of memory used in embedded systems. A fixed program and constants can be stored in non-volatile ROM memory while working variable data storage can reside in volatile RAM. Volatile memory loses its contents when power is removed, but non volatile ROM memory always maintains its contents even after power is removed. The Harvard architecture also has the potential advantage of a separate interface allowing twice the memory transfer rate by allowing instruction fetches to occur in parallel with data transfers. Unfortunately, in most Harvard architecture machines, the memory is connected to the CPU using a bus that limits the parallelism to a single bus. A typical embedded computer consists of the CPU, memory and I/O.

The generic DS89C450 offer the highest performance available in 8051-compatible microcontroller, which contains two separate buses for both program and data. It is based on an 8 bit central processing unit with an 8 bit Accumulator and another 8 bit B register as main processing blocks. Other portions of the architecture include few 8 bit and 16 bit registers and 8 bit memory locations. Each DS89C450 device has some amount of data RAM built in the device for internal processing. This area is used for stack operations and temporary storage of data. This base architecture is supported with on-chip peripheral functions like I/O ports, timers/counters, versatile serial communication port. So it is clear that this DS89C450 architecture was designed to cater many real time embedded needs. The following list gives the features of the DS89C450 architecture:

- One Clock per -Machine Cycle.
- DC to 33 MHz Operation.
- 64K Program Memory address space.
- 64K Data Memory address space.

• 16kB/32kB/64kB Flash Memory.

• 4 Bi-directional, 8-bit I/O ports.

• Three 16 bit timer/counters.

• Full Duplex UART.

• 13 interrupt sources (6 external) with 5 interrupts priority levels.

About the non-mentioning of memory space meant for the program storage, it is the most important part of any embedded controller. Intel delivered all these microcontrollers (8051) with user's program fused inside the device. The memory portion was mapped at the lower end of the Program Memory area. But, after getting devices, customers could not change any thing in their program code, which was already made available inside during device fabrication.

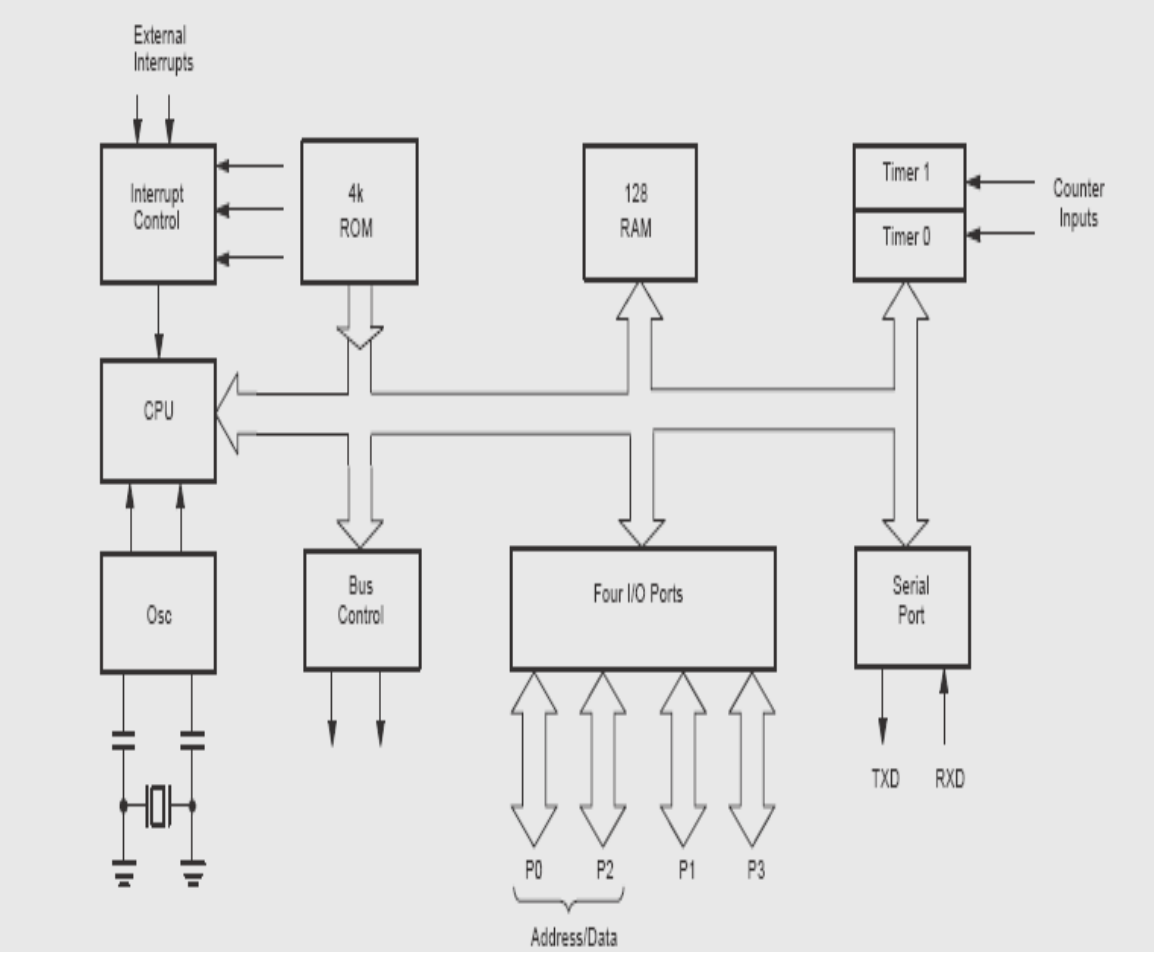

Figure 3.7: Functional Diagram [10]

3.2.1.3 Central Processing Unit (CPU)

The CPU is the brain of the microcontrollers reading user's programs and executing the expected task as per instructions stored there in. Its primary elements are an 8 bit Arithmetic Logic Unit (ALU), Accumulator (Acc), few more 8 bit registers, B register. Stack Pointer (SP), Program Status Word (PSW) and 16 bit registers. Program Counter (PC) and Data Pointer Register (DPTR). The ALU performs arithmetic and logic functions on 8 bit input variables. Arithmetic operations include basic addition, subtraction, multiplication and division. Logical operations are AND, OR, Exclusive OR as well as rotate, clear, complement and etc. Apart from all the above, ALU is responsible in conditional branching decisions, and provides a temporary place in data transfer operations within the device. B register is mainly used in multiply and divide operations. During execution, B register either keeps one of the two inputs or then retains a portion of the result. For other instructions, it can be used as another general purpose register. Program Status Word keeps the current status of the ALU in different bits.

Stack Pointer (SP) is an 8 bit register. This pointer keeps track of memory space where the important register information is stored when the program flow gets into executing a subroutine. The stack portion may be placed in any where in the on-chip RAM. But normally SP is initialized to 07H after a device reset and grows up from the location 08H. The Stack Pointer is automatically incremented or decremented for all PUSH or POP instructions and for all subroutine calls and returns. Program Counter (PC) is the 16 bit register giving address of next instruction to be executed during program execution and it always points to the Program Memory space. Data Pointer (DPTR) is another 16 bit addressing register that can be used to fetch any 8 bit data from the data memory space. When it is not being used for this purpose, it can be used as two eight bit registers [10].

3.2.1.4 Input/Output Ports

The DS89C450's I/O port structure is extremely versatile and flexible. The device has 32 I/O pins configured as four eight bit parallel ports (PO, P1, P2 and P3). Each pin can be used as an input or as an output under the software control. These I/O pins can be accessed directly by memory instructions during program execution to get required flexibility. These port lines can be operated in different modes and all the pins can be made to do many different tasks apart from their regular I/O function executions. Instructions, which access external memory, use port PO as a multiplexed address/data bus.

At the beginning of an external memory cycle, low order 8 bits of the address bus are output on PO. The same pins transfer data byte at the later stage of the instruction execution. Also, any instruction that accesses external Program Memory will output the higher order byte on P2 during read cycle. Remaining ports, P1 and P3 are available for standard I/O functions. But all the 8 lines of P3 support special functions: Two external interrupt lines, two counter inputs, serial port's two data lines and two timing control strobe lines are designed to use P3 port lines. When users do not use these special functions, they can use corresponding port lines as a standard I/O. Even within a single port, I/O operations may be combined in many ways. Different pins can be configured as input or outputs independent of each other or the same pin can be used as an input or as output at different times. It can comfortably combine I/O operations and special operations for Port 3 lines.

3.2.1.5 Timers/Counters

DS89C450 has three 16 bit Timers/Counters capable of working in different modes. Each consists of a 'High' byte and a 'Low' byte which can be accessed under software. There is a mode control register and a control register to configure these timers/counters in number of ways. Use software to get longer delays.

3.2.1.6 Serial Port

Each DS89C450 provides two UARTs that are controlled and access by SFRs. Each UART has an address is used to read and write the value contain in the UART. The same address is used for both read and writes operations and the read and writes operations are distinguished by the instruction. Its own SFR control register controls each UART.

3.2.1.7 Memory Organization

There are three distinct memory areas in the DS89C450, scratched pad registers, program memory and data memory. The registers are located on-chip but the program and data memory spaces can be on-chip, off-chip or both. The DS89C450 have 16kB/32kB/64kB of on-chip program memory, respectively, implemented in flash memory and also have 1kB of on-chip data memory space that can be configured as program space using PRAME bit in the ROMSIZE feature. The DS89C450 uses a memory-addressing scheme that separates program memory from data memory. The program and data segments can be overlapped since they are accessed in different manners.

If the maximum address on-chip or data memory is exceeded, the DS89C450performs an external memory access using the expanded memory bus. The PSEN signal goes active low to serve as a chip enable or output enable when performing a code fetch from external memory program. MOVX instructions activate the RD or WR signal for external MOVX data memory access. The program memory ROMSIZE feature allows the DS89C450 to act as a boot loader for an external memory. It is also enables the use of the overlapping external program spaces. The lower 128

bytes of on-chip flash memory, if ROMSIZE is greater than 0 are used to store rest and interrupt vectors. 256 bytes on-chip RAM serve as a register area and program stack, which are separated from the data memory.

3.2.1.8 Common Memory Space

The DS89C450's Data Memory may not be used for program storage. So it means that it can not execute instructions out of this Data Memory. But, there is a way to have a single block of off-chip memory acting as both Program and Data Memory. By gating together both memory read controls (RD and PSEN) using an AND gate, a common memory read control signal can be generated. In this arrangement, both memory spaces are tied together and total accessible memory is reduced from 128 Kbytes to 64 Kbytes. The 8031 can read and write into this common memory block and it can be used as Program and Data Memory. Users can use this arrangement during program development and debugging phase. Without taking Microcontroller off the socket to program its internal ROM (EPROM/Flash ROM), this common memory can be use for frequent program storage and code modifications.

3.2.1.9 Interrupts

The 8031 has five interrupt sources; one from the serial port when a transmission or reception operation is executed; two from the timers when overflow occurs and two come from the two input pins INTO, INT1. Each interrupt may be independently enabled or disabled to allow polling on same sources and each may be classified as high or low priority. A high priority source can override a low priority service routine. These options are selected by interrupt enable and priority control registers, IE and IP. When an interrupt is activated, then the program flow completes the execution of the current instruction and jumps to a particular program location where it finds the interrupt service routine. After finishing the interrupt service routine, the program flows return to back to the original place. The Program Memory address, 0003H is allotted to the first interrupt and next seven bytes can be used to do any task associated with that interrupt [10].

3.2.1.10 LCD Screen

The DS89C450 board's LCD port provides the 14 signals needed for standard character based LCD modules. This backlight display plugs directly into the 8051 development board. It displays 20 characters by 2 lines. Characters are black against a green background. It is shown in Figure 3.8.

Figure 3.8: LCD 20x2 Lines

This LCD includes a green LED backlight, which allows the characters to be viewed without ambient light. In normal room light, the characters are visible without the backlight. A resistor is included for current limiting to the backlight. The LCD then connected to the development board as shown in Figure 3.9. The pins configuration is shown in the Table 3.1.

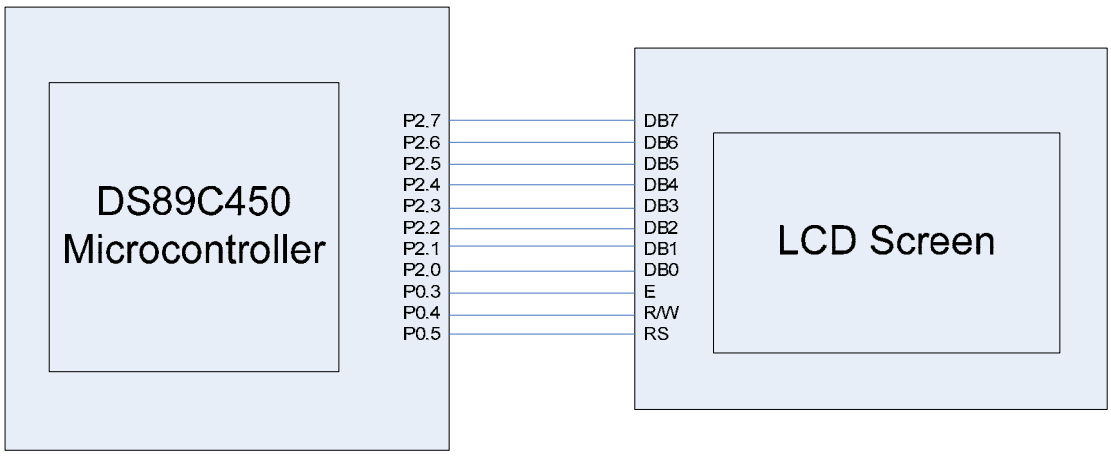

Figure 3.9: Pins Connection from Microcontroller to LCD Screen

Table 3.1: Configuration Number of Pins [12]

Pins Number	Abbreviation	Function of Pins
1	Vss	GND
2	Vdd	+3v or +5V
3	Vo	Contrast Adjustment
4	RS	H/L Register select line
5	R/W	H/L Read/Write signal
6	E	H to L Enable signal
7	DB0	H/L Data bus line
8	DB1	H/L Data bus line
9	DB2	H/L Data bus line
10	DB3	H/L Data bus line
11	DB4	H/L Data bus line
12	DB5	H/L Data bus line
13	DB6	H/L Data bus line
14	DB7	H/L Data bus line
15	A/Vee	4.2 For LED/Negative voltage output
16	K	Power supply for B/L 0V

3.2.1.11 Keypad

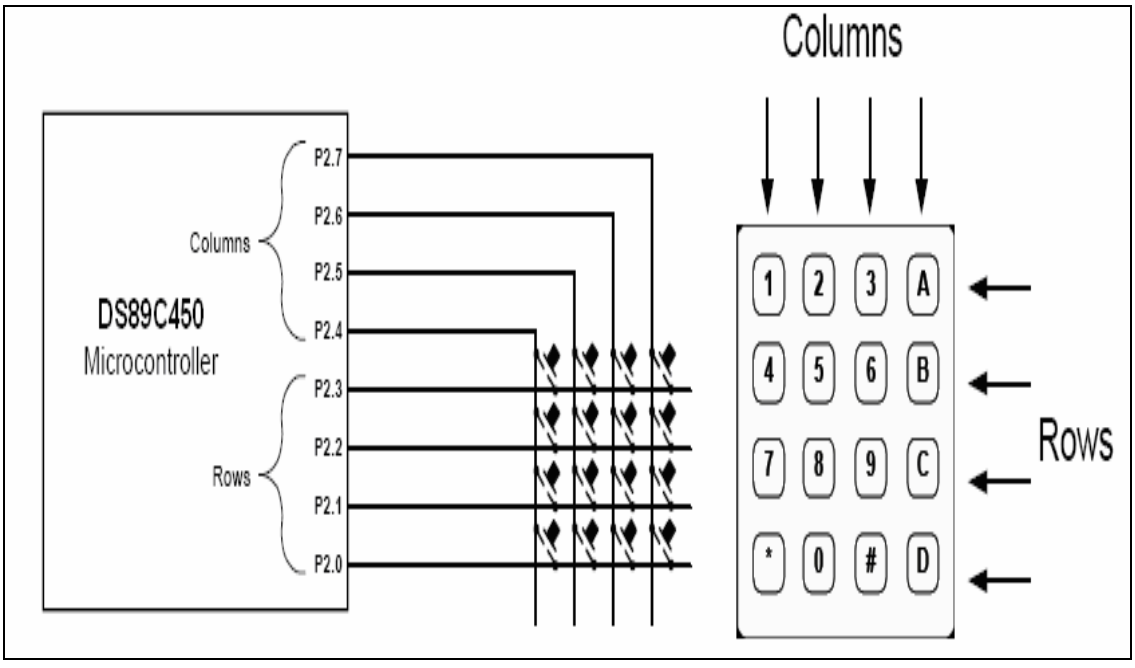

Figure 3.10: Connection between Keypad and Microcontroller Directly

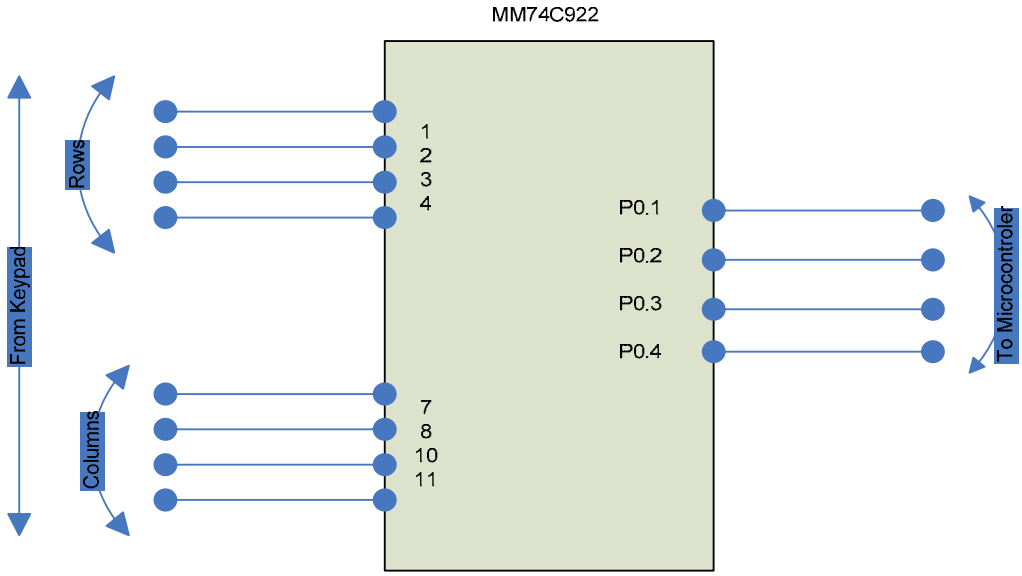

Figure 3.11: Connection between Keypad and Microcontroller Using MM74C922 [12]

The DS89C450 board also connected with keypad as shown as Figure 3.10. It used to make user comfortable to choose the pray time by pressing the code that was provided. It cannot be direct connecting from the microcontroller. It has to connect to the MM74C922 as shown in Figure 3.11 to encode the signal from keypad before it face the microcontroller. The chip scans the keypad waiting for a key press. When a button is pressed pin 11 of the MM74C922 goes high. Then a 4 bit binary number corresponding to the button press is sent to pins 14-17 of the IC. The programming straight forward use the button command to read the state of pin 11 of the MM74C922. When its high read the states of pin 14 to 17 and save it. The 4 bit binary numbers need to be converted to a decimal number. Then the bunch is used if statements to do for each pin of the keypad. The two capacitors are for scanning and debouncing times. Pin 5 is for scan and pin 6 is the debounce.

3.2.1.12 Real Time Clock DS1307 (RTC)

DS1307 is one device that used widely in microcontroller applications as shown in Figure 3.12 It provides Serial Real Time Clock is a low–power, full BCD clock/calendar plus 56 bytes of nonvolatile SRAM. Address and data are transferred serially via the 2 wire bidirectional bus. The clock/calendar provides seconds, minutes, hours, day, date, month and year information. The end of the month date is automatically adjusted for months with less than 31 days, including corrections for leap year. The clock operates in either the 24 hour or 12 hour format with AM/PM indicator. The DS1307 has a built–in power sense circuit which detects power failures and automatically switches to the battery supply.

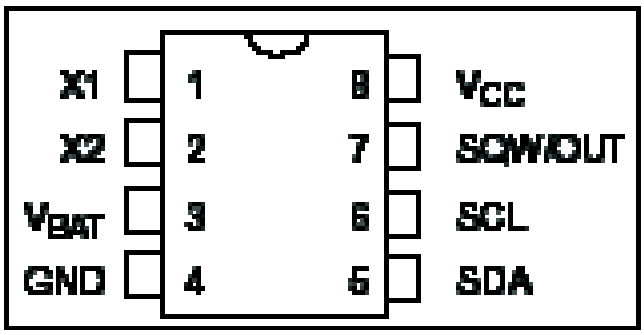

Figure 3.12: DS1307 Pin Out

It is importance to have some knowledge about the IC when there is a need to interface with DS1307 because the DS1307 operates as a slave device on the serial bus. The bus configuration is shown in Figure 3.13. Access is obtained by implementing a START condition and providing a device identification code followed by a register address (see Figure 3.14). Subsequent registers can be accessed sequentially until a STOP condition is executed. The START and STOP conditions are generated using the low level drives, SEND_START and SEND_STOP found in the attached microcontroller code. Also the subroutines SEND_BYTE and READ_BYTE provide the 2 wire handshaking required for writing and reading 8 bit words to and from the DS1307.

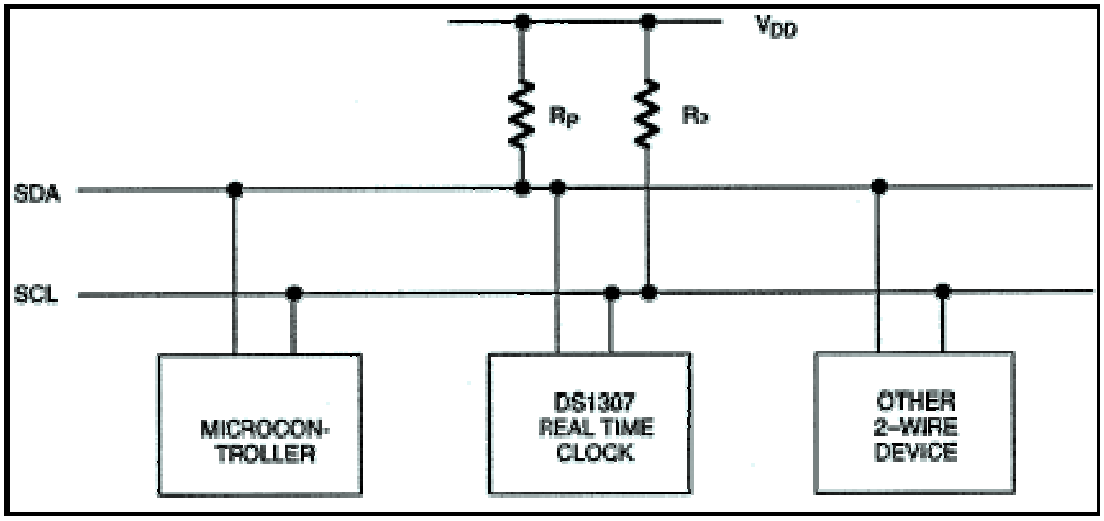

Figure 3.13: DS1307 Bus Configuration

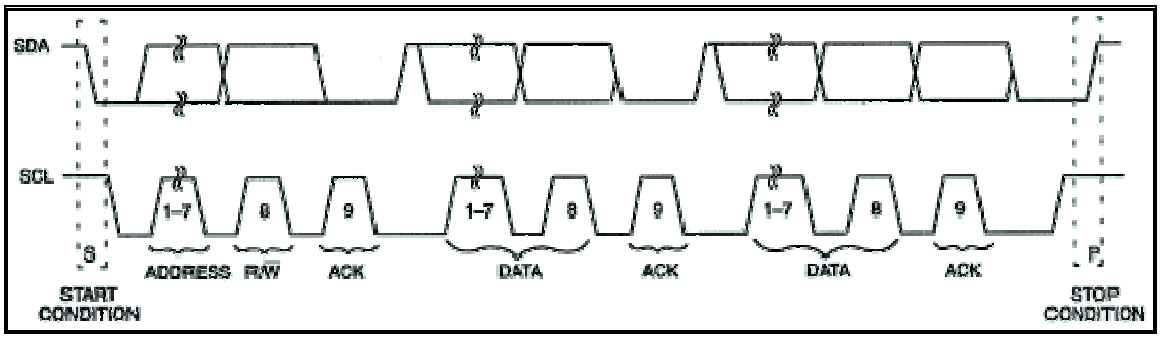

Figure 3.14: Data Transfer on 2 Wire Serial Buses

- *Start data transfer:*

A change in the state of the data line from high to low, while the clock line is high, defines a START condition.

- *Stop data transfer:*

A change in the state of the data line from low to high, while the clock line is high, defines the STOP condition.

- *Data valid*:

The state of the data line represents valid data when, after a START condition, the data line is stable for the duration of the high period of the clock signal. The data on the line must be changed during the low period of the clock signal. There is one clock pulse per bit of data. Each data transfer is initiated with a START condition and terminated with a STOP condition. The number of data bytes transferred between the START and the STOP conditions is not limited, and is determined by the master device. The information is transferred byte–wise and each receiver acknowledges with a ninth bit.

- *Acknowledgement*

Each receiving device, when addressed, is obliged to generate the acknowledges after the reception of each byte. The master device must generate an extra clock pulse which is associated with this acknowledge bit. A device that acknowledges must pull down the SDA line during the acknowledge clock pulse in such a way that the SDA line is stable low during the high period of the acknowledge related clock pulse. Of course, setup and hold times must be taken into account. A master must signal an end of data to the slave by not generating an acknowledge bit on the last byte that has been clocked out of the slave. In this case, the slave must leave the data line high to enable the master to generate the STOP condition. Detail refer to Appendix B

3.2.1.13 GPS Receiver JP7 Module

The GPS Receiver Module continuously tracks all satellites in view, thus providing accurate satellite position data. The highly integrated digital receiver uses the SiRFstarII-Low Power chipset Operate the GPS receiver with an antenna connected to it and with no obstruction between the receiver and the satellite. The antenna socket should not be shorted as this would render the GPS receiver dysfunctional. The receiver can not be used with the damaged antenna. The antenna used is FALCOM ANT-4 (active). The SiRFstarII is architecture shown in Figure 3.15. It was builds on the high-performance SiRFstarI core, adding an acquisition accelerator, differential GPS processor, multi-path mitigation hardware and satellite-tracking engine. Detail refers to Appendix F

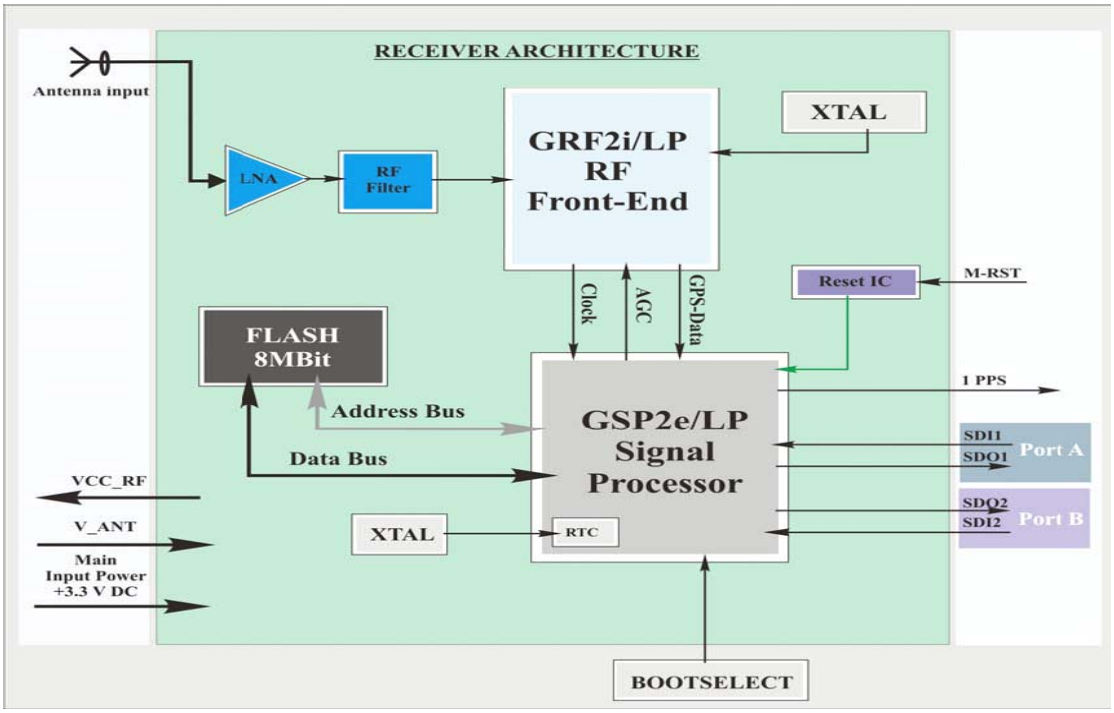

Figure 3.15: Receiver Architecture of the GPS Receiver JP7 Module

3.2.2 Software

A software development process is a structure imposed on the development of a software product. Software is a consisting of programs which is enables a computer to perform specific tasks, as opposed to the physical components of the system (hardware). This includes all application software examples a word processor, which enables a user to perform a task and system software such as an operating system, which enables other software to run properly by interfacing with hardware and with other software or custom software made to user specifications. But for this project, stage of development software involved for designing the algorithm of calculation the prayer time for Muslims. This project use Franklin Proview32 as a compiler to test the program before it was burn in microcontroller. For the embedded language, C was written and used to implement the program first then the language can be converted.

3.2.2.1 Benefits of C in Embedded System

- *Not be overwhelmed by details.*
  8-bit microcontrollers are not just small; microcontrollers include only the logic needed to perform their restricted tasks, at the expense of programmer 'comfort'. Working with these limited resources through a C compiler helps to abstract the architecture and keep from miring down in op-code sequences and silicon bugs.

- *Learn the basics of portability.*
  Embedded applications are cost sensitive. There may be great incentive to change part (or even architectures) to reduce the per-unit cost. However, the cost of modifying assembly language code to allow a program written for one microcontroller to run on a different microcontroller may remove any incentive to make the change.

- *Spend more time on algorithm design and less time on implementation.*
  C is a high level language and will be able to program applications quickly and easily. C's breadth of expression is concise and powerful; therefore each line of code written in C can replace many lines in assembly language. Debugging and maintaining code written in C is much easier than in code written in assembly language.

3.2.2.2 Franklin Overview

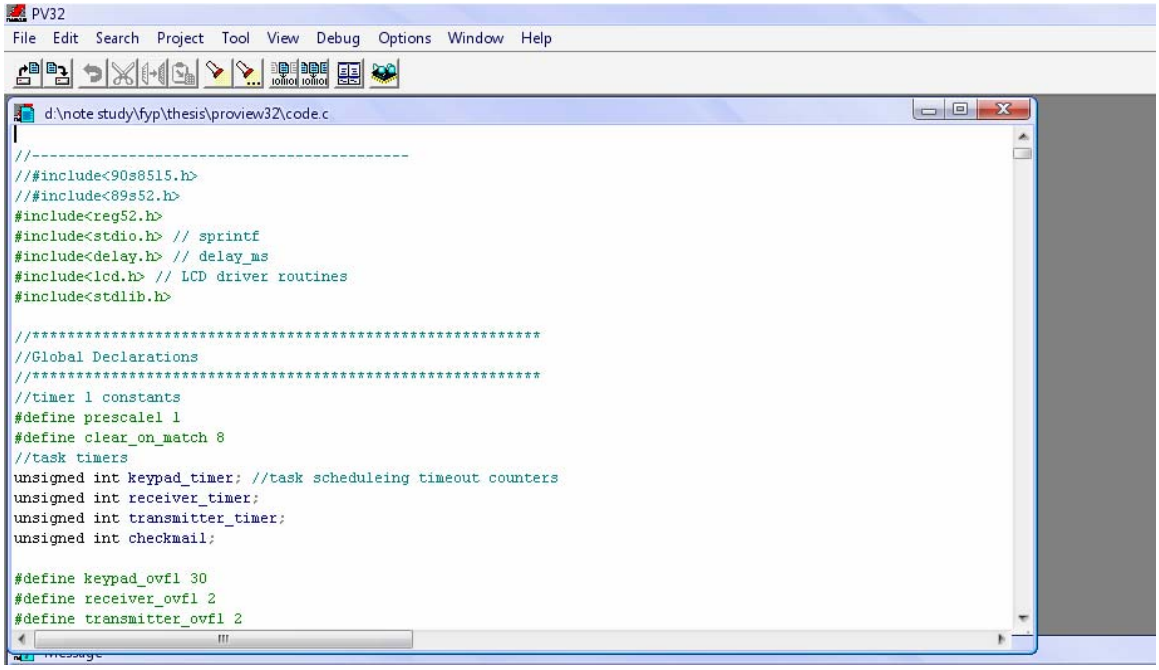

Figure 3.16: ProView32 Software

Proview32 is a fully featured and Integrated Development Environment that provides smooth seamless access to all the tools in the professional developer's arsenal. From editing to debugging, ProView32 can manage all aspects of product development for any member of 8051 member.

ProView32 is based on a fast multi-document editor designed to meet the specific needs of the programmer's art. The various method, menus, commands, and shortcuts are all fully compliant with the Microsoft specifications for Win95 and Win NT. Classic commands, such a sting search and block actions, have been added to provide specific and advanced features for the professional developer. A customizable color syntax highlighting editor is used to indicate specific syntax elements as they appear in the source file such as key words, comments, identifiers, operators, and so on. Whether it is C or assembler, the syntax highlighting is keyed to the intrinsic file type. This permits the professional user to quickly and easily identify those parts of code responsible for syntactic errors, for instance, an unclosed comment. Other helpful

commands, such as “search for matching delimiter” make ProView32 a powerful easily used tool specifically to provide your embedded solution.

3.2.2.3 Project Manager

The project manager creates a link between the various file that comprise a project and the tools necessary to create that project. A project is dedicated to a target 8051, 251 or XA and each of the project’s file are associated, by their file type to the appropriate translating tool. C and assembler files are translated by the compiler and assembler respectively. The linker manages object and library files and output format conversion as necessary.

In addition, the user may define that other custom tools be used too, even DOS application. Option for all tools is globally defined for the course of the project. As specific needs dictate, it is a simple matter to alter any files individual attributes as necessary to achieve your goals.

The project/Make command directs the integrated “make” utility to build or rebuild the target program for the current project. Each source file will be translated by its associated tool if any of its dependencies are found to be out of date. Dependency analysis even directly or indirectly included files is automatic.

3.2.2.4 Flow Chart for the Algorithm

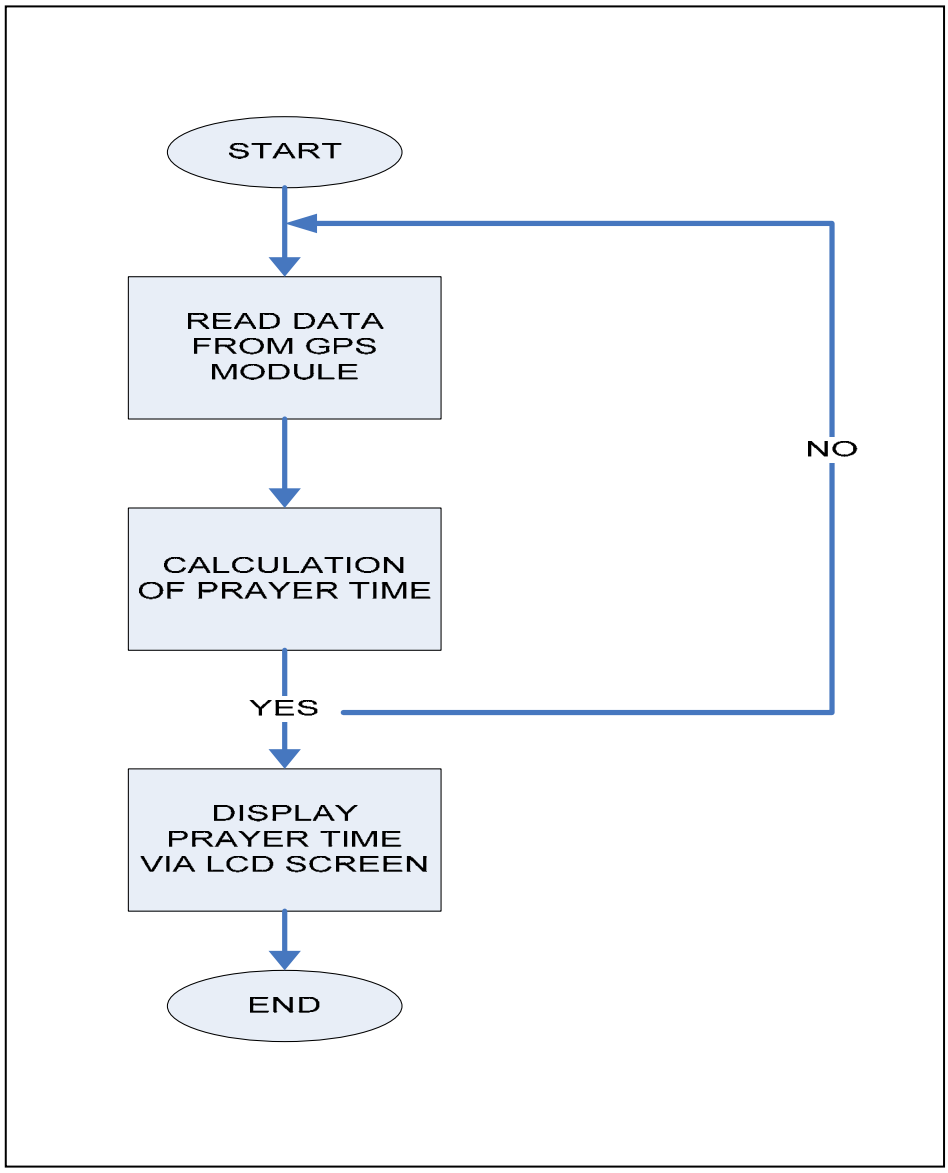

Figure 3.17: Overall Flow Chart System

3.2.2.5 Microcontroller Software

Franklin is used to compile the sample LED blinking to test the microcontroller board. As the result, the LED was blinking and this confirms that the microcontroller is working.

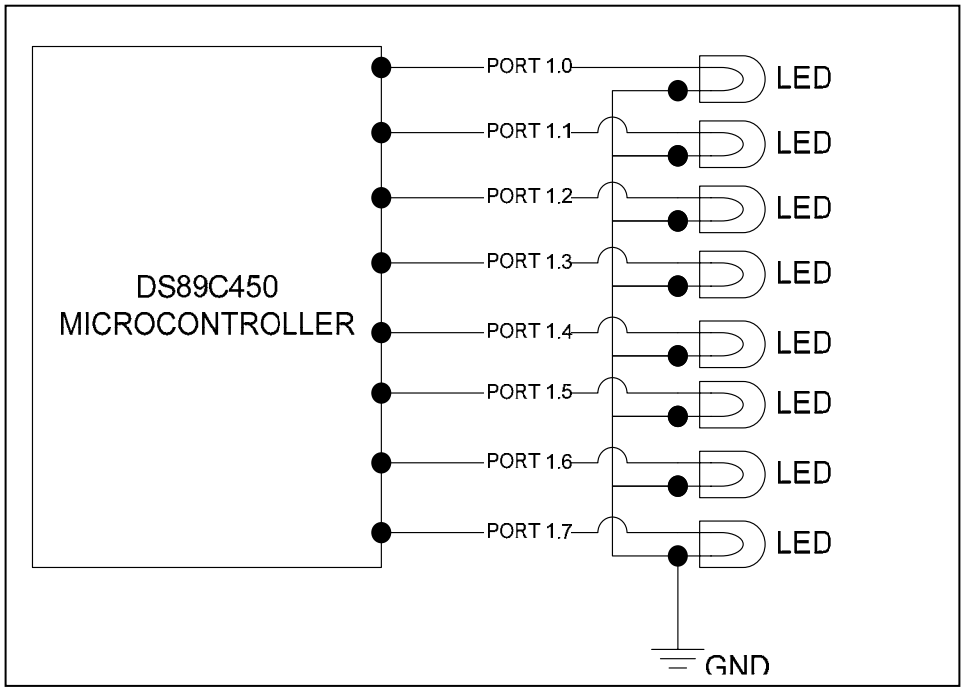

Figure 3.18: Microcontroller Connection with LED [12]

Microcontroller Test Program

In order to test the microcontroller, the C programming below was used to make sure the output ports were function well.

```
#include <reg51.h>
#include <stdio.h>
viod DELAY_HARDWARE_50ms (void){

//configure timer 0 as a 16-bit//
TMOD&=0x0F;                    //clear all T0 bits (T1 left unchanged)//
TMOD|=0x01;                    //set required T0 bits (T1 left unchanged)//
 ET0=0;                        //no interrupt//
/*values for 50 ms delay*/
TH0=0x3C;                      //timer 0 initial value (high byte)//
TL0=0xB0;                      //timer 0 initial value (low byte)//
```

```
TF0=0;                          //clear overflow flag//
TR0=1;                          //start timer 0//
TR0=0;                          //start timer 0//
}
const num[ ]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x40};
void main (void){
        unsigned char j;
        unsigned char d;
        P1=0;                   //initial ZERO to P1//
        while (1){
        for(j=0;j<8;j++){
                P1=num[j];
                 DELAY_HARDWARE_50ms( );
                 }
                        for(d=0;d<2;d++){
                          DELAY_HARDWARE_50ms( );
                          }
                                for(j=0;j<8;j--){
                                  P1=num[j];
                                  DELAY_HARDWARE_50ms( );
                                }
                        }
                }
        }
```

3.2.2.6 RTC Software

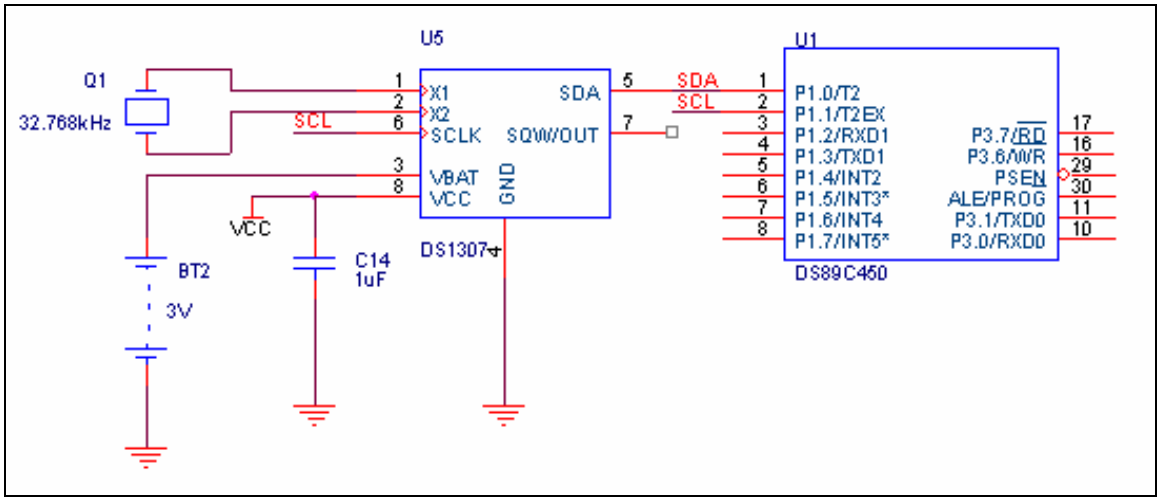

Figure 3.19: DS1307 Connection with Microcontroller DS89C450

RTC Test Program

For the clock and date, program as shown below was used to test the DS1307 was function as expected.

```
#include "reg.51h"
#define DS1307_SDA_ P1.0
#define DS1307_SCL_ P1.1
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_A0, rcv=PIN_A1)
#use i2c(master, sda=DS1307_SDA, scl=DS1307_SCL)
#fuses NOWDT,XT, NOPUT, NOPROTECT
#include "ds1307.c"
void main(void)
{
  int8 sec,min,hour,day,date,month,year;
  delay_ms(50);
  init_ds1307();                          // initial DS1307
  sec=read_ds1307(0);
  write_ds1307(0,sec & 0x7F);             // enable oscillator(bit 7 =0)
  while(true)
        {
              sec=read_ds1307(0);               // read second
              min=read_ds1307(1);               // read minute
               hour=read_ds1307(2);             // read hour
              day=read_ds1307(3);               // read day
              date=read_ds1307(4);              // read date
               month=read_ds1307(5);            // read month
              year=read_ds1307(6);              // read year
              putc(0x0c);
              printf("Time : %02X:%02X:%02X\r\n",hour,min,sec);
              printf("Day  : %02X\r\n",day);
               printf("Date : %02X/%02X/20%02X\r\n",date,month,year);
              delay_ms(500);
```

```
        }
}

// initial DS1307

void init_DS1307()
{
  output_float(DS1307_SCL_P1.0);
  output_float(DS1307_SDA_P1.1);
}


// write data one byte to DS1307

void write_DS1307(byte address, BYTE data)
{
  short int status;
  i2c_start();
  i2c_write(0xd0);
  i2c_write(address);
  i2c_write(data);
  i2c_stop();
  i2c_start();
  status=i2c_write(0xd0);
  while(status==1)
  {
    i2c_start();
    status=i2c_write(0xd0);
  }
}

// read data one byte from DS1307

BYTE read_DS1307(byte address)
```

```
{
  BYTE data;
  i2c_start();
  i2c_write(0xd0);
  i2c_write(address);
  i2c_start();
  i2c_write(0xd1);
  data=i2c_read(0);
  i2c_stop();
  return(data);
}
```

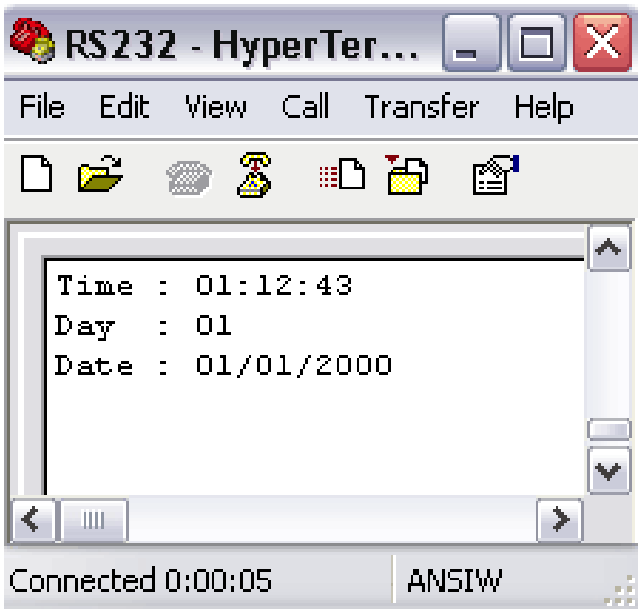

Figure 3.20: Result for the RTC IC: DS1307

3.2.2.7 LCD Software

LCD Test Program

The program below has shown the C language that was used to test the LCD whether it can work.

```
// LCD_interface.C //
#include <reg51.h>
#include <stdio.h>
#include <intrins.h>

sbit PIN=P2^0;                  // I/O   Pin for P2.0//
sbit PIN=P2^1;                  // I/O   Pin for P2.1//
sbit PIN=P2^2;                  // I/O   Pin for P2.2//



#define LINE1_ADDR 0x00;        // Start of line 1 in the DD RAM //
#define LINE2_ADDR 0x40         // Start of line 2 in the DD RAM //

// Display Command //

#define CLEAR_DISPLAY 0x01      //Clear display & set data addr to  0//
#define DD_RAM PTR 0x80         // Addr display Data RAM pointer //
#define DISP_INIT 0x38          // 8 bits 2 lines //
#define INC_MODE 0x06           // Display data RAM pointer increment
                                after write  //


// ReadInstrReg; Read from Instruction Register of LCD Display Device //

static unsigned char ReadInstrReg (void) {
unsigned char Instr;

P0=0xff;                        //DATA PORT is input //
P20_PIN=0;                      //select instruction register //
P21_PIN=1;                      //read operation //
P22_PIN=1;                      //give operation start signal //
_nop_( );_nop_( );              //wait //
Instr=P0;                       //read instruction //
P22_PIN=0;
return (Instr);
}
```

```
//WriteInstrReg; Write to Instruction Register of LCD Display Device//

static void WriteInstrReg (unsigned char Instr) {
P20_PIN=0;                          //select instruction register //
P21_PIN=0;                          //write operation //
P22_PIN=1;                          //give operation start signal //
_nop_( );_nop_( );                  //wait//
P0=Instr;                           //write instruction //
P0=0xff;                            //DATA PORT is input [prevent I/O port
                                      from damage] //

}

// ReadDataReg; Read from Data Register of LCD Display Device//

static unsigned char ReadDataReg (void) {
unsigned char val;
P0=0xff;                            //DATA PORT is input //
P20_PIN=1;                          //select data register //
P21_PIN=1;                          //read operation //
P22_PIN=1;                          //give operation start signal //
_nop_( );_nop_( );                  //wait //
val=P0;                             //read instruction //
P22_PIN=0;
return (val);
}

endif




// WriteDataReg; Write Data Register to LCD Display //

static void WriteDataReg(unsigned char val) {

P20_PIN=1;                          //select data register //
P21_PIN=0;                          //write operation //
P22_PIN=1;                          //give operation start signal //
_nop_( );_nop_( );                  //wait //
P0=val;                             //write value  //
P22_PIN=0;
P0=0xff;                            //DATA PORT is input [prevent I/O port
                                      from damage] //

}

char putchar (char c) {
```

```
unsigned char line;
if(c= =’\n’) {
        line = ReadInstrReg ();
                if (line & LINE2_ADDR);
                WriteInstrReg(LINE1_ADDR | DD_RAM_PTR);
 }

else {
        WriteInstrReg(LINE2_ADDR | DD_RAM_PTR);
}
}
        else{
        WriteDataReg (c);
}
return (c);
}

void wait (unsigned int time){
int i;
                for (i=0; i<time; i++){
        }
}

//Clear_Display: Write0x20 to Display RAM//

static void Clear_Display (void);
unsigned char c;

        for (c=’A’ ; c<’Z’; c++) {
                putchar (c);
        }
                printf(“DATE: \n”);


        Clear_Display( );
        for (c=0; c<10; c++) {
                printf(“ TIME:%bd\n’,c)
        }
        while (1);
        }
```

3.2.2.8 Keypad Software

Keypad Test Program

In order to test the keypad, the program below was used.

```
#include<stdio.h>
#include<stdlib.h>
#include<dos.h>
#include<conio.h>
#include<string.h>

void KeyPress(int);

void main(void)
{
int Baseaddr;
int Port1, Port2, Port3;
int Cntrl;
int Reading;

clrscr();
Port1 = Baseaddr;
Port2 = Baseaddr + 1;
Port3 = Baseaddr + 2;
Cntrl = Baseaddr + 3;

outportb(Cntrl, 155); /* configure all three ports as input*/
gotoxy(1,5);
cprintf("Press a key on the keypad to display value.\n");
gotoxy(1,13);
printf("To quit, press any key on the keyboard.\n");
Reading=0;
        while(!kbhit()) {
                do {
                        Reading = inportb(Port1)
                                } while ( (Reading >= 128) )
                                        KeyPress(Port1)
                                }
                        }
```

```
void KeyPress(int Port1) {
int index;
char Selection;
index = inportb(Port1) % 128;
switch (index) {
        case 0 : Selection = '.'; break;
        case 1 : Selection = 'B'; break;
        case 2 : Selection = '<'; break;
        case 3 : Selection = '>'; break;
        case 4 : Selection = 'E'; break;
        case 5 : Selection = '3'; break;
        case 6 : Selection = '6'; break;
        case 7 : Selection = '9'; break;
        case 8 : Selection = '0'; break;
        case 9 : Selection = '2'; break;
         case 10: Selection = '5'; break;
        case 11: Selection = '8'; break;
        case 12: Selection = 'C'; break;
        case 13: Selection = '1'; break;
        case 14: Selection = '4'; break;
        case 15: Selection = '7'; break;
        default: Selection = 'E'; break;
        }
gotoxy(1,8);
        printf("The %c key was just pushed.", Selection);
}
```