Shri Ramdeobaba College of Engineering and Management, Nagpur (MS)

(An Autonomous Institution Permanently affiliated to Rashtrasant Tukadoji Maharaj Nagpur University)

An ISO 9001:2015 Certified Institution. NAAC Certified 'A' Grade

Department of Electrical Engineering

Laboratory Manual

Microcontroller Laboratory

EEP 353

(V Semester Electrical)

Program Outcomes (UG)

- PO1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals to the solution of engineering problems.
- PO2. **Problem analysis:** Identify, formulate, review literature, and analyze complex engineering problems using first principles of mathematics, natural sciences, and engineering sciences.
- PO3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public safety, societal and environmental considerations.
- PO4. **Conduct problem investigations:** Use research-based knowledge including experimentation, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- PO5. **Modern tool usage:** Select, and apply appropriate techniques, resources, and modern engineering and IT tools for analyzing the engineering activities with an understanding of the limitations.
- PO6. **The engineer, industry and society:** Apply contextual knowledge to assess industrial, societal and safety related issues and understand consequent relevance to the professional engineering practice.
- PO7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- PO8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- PO9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- PO10. **Communication:** Communicate effectively on complex engineering activities such as, being able to understand and write effective reports, make effective presentations, and give and receive clear instructions.
- PO11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team in multidisciplinary environments.
- PO12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.
-

Programme Specific Outcomes

- PSO1. Analyze and design electrical networks, machines, control systems, power systems, power converters and evaluate the performance.
- PSO2. Understand and develop electrical systems considering energy efficiency, power scenario, environmental issues and industry applications.

Course Details

Course Name: Microcontroller Laboratory

Course Code : EEP353

Lab Hours : Two /Week

Credits : 0ne

Prerequisite: Digital Electronics,

Curse Objective:

The objectives of this laboratory course are to prepare students for Microcontroller programming, introduce the open source/proprietary development environment and make them acquainted with microcontroller development board.

Course Outcomes:

At the completion of this course, students will be able to:

CO1 **Use** open source or proprietary development environment and microcontroller development board for Microcontroller programming.
CO2 **Implement** control algorithm using suitable programming language.
CO3 **Set up** the circuit on microcontroller development board for testing of program.
CO4 **Debug** the program to make it working.
CO5 **Design** a small application based on microcontroller

Evaluation Scheme:

Internal Evaluation : 25 Marks	External Evaluation: 25 Marks
Continuous evaluation: 15 • Program writing • Program execution • Timely submission **Journal writing: 04** **Viva voce : 06**	**Program writing : 8M** **Program execution :7M** **Viva Voce : 10M**

Ref Books/Resources:
1. Laboratory manual
2. Open source development tool guide
3. Product Datasheet

CO and PO Mapping

Course Outcomes	PO and PSOs													
At the completion of this course, students will be able to:	**PO 1**	**PO 2**	**PO 3**	**PO 4**	**PO 5**	**PO 6**	**PO 7**	**PO 8**	**PO 9**	**PO 10**	**PO 11**	**PO 12**	**PSO 1**	**PSO 2**
	Engineering knowledge	**Problem analysis**	**Design/development of solutions**	**Conduct problem investigations**	**Modern tool usage**	**The engineer, industry and society**	**Environment and sustainability**	**Ethics**	**Individual and team work**	**Communication**	**Project management and finance**	**Life-long learning Life-long learning**	**Analyze and design Analyze and design**	**Understand and develop**
CO1 Use *open source or proprietary development environment and microcontroller development board for Microcontroller programming.*					**S**									
CO2 Implement *control algorithm using suitable programming language.*			**S**											**S**
CO3 Set up *the circuit on microcontroller development board for testing of program.*			**S**											
CO4 Debug *the program to make it working.*				**S**										**S**
CO5 Design *a small application based on microcontroller*			**S**						**M**					**S**
S: Strong ; M: Medium, W: Weak														

Microcontroller Laboratory Assignments			
Sr. No	Assignment	Details	Concept Covered
1.	LED Interfacing	This Assignment demonstrates simple LEDs Blinking operation. The LEDs will continuously toggle in periodic interval thus creating Blinking effect	I/O PORTs, Delay function,
2.	LED and Switches interfacing	This assignment demonstrates LED and Switch interfacing operation. The program will continuously scan the keys and respective LEDs will turn on when a switch is pressed,	I/O PORTs and LED and Switches interfacing
3.	Seven Segment Display(SSD) interfacing	This assignment demonstrates Multiplexed Seven Segment Display interfacing using BCD to 7segment Decoder 74LS47. BCD Inputs are connected to lower nibble of PORT and 2 Selects pins are used for selecting Seven Segment Display. A counter from 0 to 99 will run on two seven segment displays	Multiplexed Seven Segment Display(SSD) interfacing
4.	RELAY and BUZZER interfacing	This assignment demonstrates simple RELAY and BUZZER ON/OFF operation. Two switches are used to ON and OFF the RELAY and BUZZER respectively.	Electromechanical relay control and Buzzer Indication.
5.	LCD 16x2 interfacing	This assignment demonstrates the 16x2 LCD operation in 4 bit mode. Data and commands will be given to LCD in 4 bit mode and text will be displayed in both the lines.	LCD Interfacing
6	Analog Sensor interfacing using ADC with LCD 16x2	This Assignment demonstrates ADC 10 bit mode operation with LCD 16x2 in 4 bit mode. A potentiometer output is connected to pin ADC2(PA2) of ADC Port whose analog value will be converted into digital value and will be displayed on LCD Rotate the POT with proper tool to see changes when output voltage of pot changed	ADC Interface
7	Timer Interfacing	This Assignment demonstrates the Timer initialization in different modes and use of Timer for generation of PWM signals	Timer interface for delay generation Timer interface for PWM signal generation

8	Serial Communication with PC using UART	This Assignment demonstrates simple Serial communication operation with PC. Microcontroller will implement a serial communication protocol which is already implemented in PC. Controller will read a input from PC and in response send some characters to PC in order to check the communication is successfully established. Open any terminal software and type in any key you should get "OK" as a response from microcontroller.	Serial communication protocols

Rubrics

Evaluation Parameter		Rubric 1 Average (20 to 50% marks)	Rubric 2 Good (50 to 80% marks)	Rubric 3 Very Good (80 to 100% marks)
Continuous evaluation	Program writing: 6 Marks	Unable to implement program logic and implement using programming language	Can implement the control logic but Able to follow the experimental procedures and mathematical calculations. But unable to evaluate and interpret properly.	Able to follow experimental procedures and mathematical calculations. Also able to present and interpret properly and co-relate the practical with the theory.
	Program execution 6 Marks	Unable to use the open source IDE and develop the programs	Able to use the IDE but lags in developing use it effectively.	Able to select use the open source IDE and develop and simulate the program correctly.
	Timely submission 3 Marks	Completes the task taking additional time Not punctual in class	Completes the task taking additional time, Punctual in class	Complete the task in time Punctual in class
Journal writing : 4 marks		Report writing is not systematic, bad presentation	Systematic report, Lacks good presentation	Systematic report writing with good presentation
Viva voce : 6 Marks		Not communicate properly the technical terms related with course Majority of wrong answers	Not communicate properly the technical terms related with course Majority of right answers	Communicate properly the technical terms related with course Majority of right answers

Laboratory Assignments Details

Assignment 1: LED interfacing

Platform: AVR ATMEGA 32 Development Board
Micro controller: Atmega32

Concept Covered: I/O PORTs and LED

Description: This experiment demonstrates simple LEDs Blinking operation. The LEDs will continuously toggle every sec thus creating Blinking effect.

Hardware Setup: Output LEDs: PORTB

Connect the 8 pin 1 to 1 connector cable between PORTB and LEDs ARRAY Header in one to one pattern.

```
*****************************************************************************
#define F_CPU                                   16000000

#include <avr/io.h>                //      inclusion for defination of Registers
#include <util/delay.h>            //      inclusion for delay functions


int main() {                       //      Starting point of programs
        DDRB = 0xff;               //      make pins of PORTB as Output pins

        while(1)                           // indefinite loop
        {
                PORTB ^= 0xff;             // EX-OR the bits of PORTB in order to create the
toggle effect
                _delay_ms(1000);   // call function give delay of 1000 mili second ie. 1 sec
        }
}
```

Assignment 2: LED and SWITCHS interfacing

Platform: AVR ATMEGA16-32 Development Board
Microcontroller: Atmega32

Concept Covered: I/O PORTs and LED and SWITCHS interfacing

Description: This experiment demonstrates simple LED and SWITCHS interfacing operation. 4 LEDs are connected to lower nibble(PB0,PB1,PB2,PB3) of PORTB and 4 Switches are connected to upper nibble nibble(PB4,PB5,PB6,PB7) of PORTB. The respective LEDs will turn on when a switch is pressed

Hardware Setup:

Output LEDs:	PORTB(PB0,PB1,PB2,PB3)
Input Switches:	PORTB(PB4,PB5,PB6,PB7)

Connect the 4 pin 1 to 1 connector cable between PORTB and LEDs ARRAY Header in one to one pattern.

PB0--- LED1
PB1--- LED2
PB2--- LED3
PB3--- LED4

Connect the 4 pin 1 to 1 connector cable between PORTB and Switches Header in one to one pattern.

PB4--- SW1
PB5--- SW2
PB6--- SW3
PB7--- SW4

```
*****************************************************************************
*defineF_CPU                        16000000

*include <avr/io.h>          //      inclusion for definition Registers
*include <util/delay.h>      //      inclusion for delay functions


int main() {                        //      Starting point of programs
        DDRB = 0x0f;                //      Make pins of PORTB(PB0,PB1,PB2,PB3) as Output
Pins and PORTB(PB4,PB5,PB6,PB7) as Input Pins
        PORTB = 0xf0;               //      Keep LEDs Off and Enable pull-ups for Switches

        while(1)                            // indefinite loop
        {
                if(bit_is_clear(PINB,PB4)) {// Scan PB4 for Switch 1
                        PORTB |= 0x01;              // If Pressed turn ON LED1
                }
                else if(bit_is_clear(PINB,PB5)) {// Scan PB5 for Switch 2
                        PORTB |= 0x02;                      // If Pressed turn ON LED2
                }
```

```
                    else if(bit_is_clear(PINB,PB6)) {// Scan PB6 for Switch 3
                    PORTB |= 0x04;                          // If Pressed turn ON LED3
                    }
                    else if(bit_is_clear(PINB,PB7)) {       // Scan PB7 for Switch 4
                         PORTB |= 0x08;                     // If Pressed turn ON LED4
                    }
                    else {
                         PORTB = 0xf0;// If no Switch is pressed keep LEDS OFF
                    }
          }
}
```

Assignment 3: Seven Segment Display (SSD) interfacing

Platform: AVR ATMEGA16-32 Development Board
Microcontroller: Atmega32

Concept Covered: Multiplexed Seven Segment Display(SSD) interfacing

Description: This experiment demonstrates Multiplexed Seven Segment Display interfacing using BCD to 7segment Decoder 74LS47. BCD Inputs are connected to lower nibble of PORTD(PD0,PD1,PD2,PD3) and
2 Selects pins for selecting Seven Segment Display are connected to PD4 and PD5 of PORTD. A counter from 0 to 99 will run on two seven segment displays

Hardware Setup:

BCD Inputs:	PORTD(PD0,PD1,PD2,PD3)
Select Pins:	PORTD(PD4,PD5)

Connect the 6 pin 1 to 1 connector cable between PORTD and Seven Segment Displays Header in one to one pattern.

PD0--- D0
PD1--- D1
PD2--- D2
PD3--- D3
PD4--- SSD1
PD5--- SSD2

Refer board manual for more detailed description.

Note: Please check the connections properly before performing any experiments.

```
*****************************************************************************
*****
#define F_CPU                                                   16000000

#include <avr/io.h>                                  //      inclusion for defination Registers
#include <util/delay.h>                              //      inclusion for delay functions
#include <compat/deprecated.h>             //      inclusion for bit wise operations(sbi and cbi)

#define SSD_DDR              DDRD                    //      Macros for SSD data direction register
#define SSD_PORT     PORTD                 //      Macros for SSD port register
#define SSD1         PD4                             //      Macros for SSD1 select pin
#define SSD2         PD5                             //      Macros for SSD2 select pin

int main() {                                         //      Starting point of programs

        SSD_DDR |= 0x0F;                             //      Make Port pins as Output
        PORTD &= ~(0X0F);                            //      Clear pins initially

        sbi(SSD_DDR,SSD1);                           //      Set pin to select SSD1 as output
        sbi(SSD_DDR,SSD2);                           //      Set pin to select SSD2 as output
        sbi(SSD_PORT,SSD1);                    //      Keep both SSDs OFF initially
        sbi(SSD_PORT,SSD2);                    //  SSD will be selected when Data is to be
written

        while(1)                                       // indefinite loop
        {
                for(int i = 0;i < 99;i++) {     // Set a for loop to count from 0 to 99

                        for(int j = 0;j < 25;j++) {     // Set a for loop from 0 to 25 this done to
hold counter value for sometime

                        cbi(SSD_PORT,SSD1);                    // Select SSD1 to turn On
                        sbi(SSD_PORT,SSD2);                    // SSD2 will be OFF
                        SSD_PORT = (SSD_PORT & 0xf0) | (i/10);// extract Unit's digit and Send
Value on port to display on SSD1
                        _delay_ms(5);                          // provide a small delay for
digit to visible

                        sbi(SSD_PORT,SSD1);                    // SSD1 will be OFF
                        cbi(SSD_PORT,SSD2);                    // Select SSD2 to turn On
                        SSD_PORT = (SSD_PORT & 0xf0) | (i%10);// extract ten's digit and Send
Value on port to display on SSD2
                        _delay_ms(5);                          // provide a small delay for
digit to visible

                        }
                }
        }
}
```

Assignment 4: RELAY and BUZZER interfacing

Platform: AVR ATMEGA-32 Development Board
Microcontroller: Atmega32

Concept Covered: RELAY and BUZZER interfacing

Description: This experiment demonstrates simple RELAY and BUZZER ON/OFF operation. RELAY and BUZZER are connected to PC0,PC1 of PORTC and 2 Switches are connected to PD0,PD1 of PORTD. The respective Device will turn on when a switch is pressed else OFF condition.

Hardware Setup:

RELAYS and BUZZER:	PORTC(PC0,PC1)
Input Switches:	PORTD(PD0,PD1)

Connect the 3 pin 1 to 1 connector cable between PORTC and RELAY and BUZZER Header in one to one pattern.

PC0--- RELAY1
PC1--- BUZZER

Connect the 3 pin 1 to 1 connector cable between PORTD and Switches Header in one to one pattern.

PD0--- SW1
PD1--- SW2

Refer board manual for more detailed description.
Note: Please check the connections properly before performing any experiments.

```
*******************************************************************************
#define F_CPU                                       16000000

#include <avr/io.h>                 //      inclusion for definition of Registers
#include <util/delay.h>                     //      inclusion for delay functions
#include <compat/deprecated.h>      //      inclusion for bitwise functions like sbi and cbi

#define SW_DDR          DDRD        //  Definition of port for switches
#define SW_PIN          PIND
#define SW_PORT         PORTD
#define SW1             PD0
#define SW2             PD1

#define DEVICE_DDR DDRC             //  Definition of port for relays and  buzzer
#define DEVICE_PORT     PORTC
#define RELAY           PC0
#define BUZZER          PC1


int main() {                                //  Starting point of program
```

```
        cbi(SW_DDR,SW1);                        // Declare pins as input pins for switches
        cbi(SW_DDR,SW2);

        sbi(SW_PORT,SW1);                       // Enable pull ups
        sbi(SW_PORT,SW2);

        sbi(DEVICE_DDR,RELAY);              // Declare pins as Output pins for relays and buzzer
        sbi(DEVICE_DDR,BUZZER);

        cbi(DEVICE_PORT,RELAY);             // Keeping all devices OFF initially
        cbi(DEVICE_PORT,BUZZER);

        while(1)                                            // indefinite loop
        {
                if(bit_is_clear(SW_PIN,SW1)) {                      //check if switch1 is pressed
                        sbi(DEVICE_PORT,RELAY);                     // if yes turn relay1 ON
                }
                else if(bit_is_clear(SW_PIN,SW2)) {  //check if switch2 is pressed
                        sbi(DEVICE_PORT,BUZZER);                    // if yes turn relay2 ON
                }
                else {
                        cbi(DEVICE_PORT,RELAY);// Keep them OFF
                        cbi(DEVICE_PORT,BUZZER);
                }
        }
}
```

Assignment 5: LCD 16x2 interfacing

Platform: AVR ATMEGA16-32 Development Board
Microcontroller: Atmega32

Concept Covered: Character LCD 16x2 in 4 bit mode

Description: This experiment demonstrates simple LCD 16x2 operation in 4 bit mode. Data will feed to LCD in 4 bit mode and text will be displayed in both the line

Hardware Setup: LCD port: PORTC

Connect the 6 pin 1 to 1 connector cable between PORTC and LCD Header in one to one pattern.

PC2---- RS
PC3---- EN
PC4----D4
PC5---- D5
PC6----D6
PC7----D7

R/W of LCD is grounded by connecting the jumper(R/W to GND) make sure it is connected unless one use R/W Pin in Program

Refer board manual for more detailed description.

Note: Please check the connections properly before performing any experiments.

```
#define        F_CPU                          16000000

#include <avr/io.h>              //     inclusion for defination Registers
#include <util/delay.h>          //     inclusion for delay functions

#define LCD_DDR      DDRC        // Macro to define LCD Data Port
#define LCD_PORT     PORTC

#define LCD_CTRL_DDR       DDRC  // Macro to define LCD Control pin Port
#define LCD_CTRL_PORT      PORTC
#define RS                            PC2
#define EN                            PC3

#define RS_HIGH()                LCD_CTRL_PORT |= (1 << RS) // RS High
#define RS_LOW()                 LCD_CTRL_PORT &= ~(1 << RS) // RS Low
#define EN_HIGH()                LCD_CTRL_PORT |= (1 << EN) // EN High
#define EN_LOW()                 LCD_CTRL_PORT &= ~(1 << EN) // EN Low

/* Functions prototypes */
void LCD_CMD(unsigned char cmd);
void LCD_DATA(unsigned char data);
void LCD_PULSE(void);
```

```
int main() {                                               //      Starting point of programs

        char str[] = "Hello LCD 16x2";              // String to display on LCD
        LCD_DDR |= 0xF0;                            // Make pins to be used as Outputs
        LCD_CTRL_DDR |= 1 << RS | 1 << EN;          // Make pins to be used as Outputs

        // LCD Reset Squence
        LCD_CMD(0x30);_delay_ms(1);
        LCD_CMD(0x30);_delay_ms(1);
        LCD_CMD(0x30);_delay_ms(1);

        LCD_CMD(0x02); // Command LCD for 4 Bit operations
        LCD_CMD(0x28); // to set lcd in 4 bit,2 lines,font 5x7
        LCD_CMD(0x0C); // to set display on,cuRSor off ,blinking off
        LCD_CMD(0x06); // cuRSor incremENt, no display shift
        LCD_CMD(0x01); // to clear lcd

        LCD_CMD(0x80+4); // Select Line1 and 4th coloumn
        LCD_DATA('*');                  // Send Characters to display on LCD
        LCD_DATA('*');
        LCD_DATA('A');
        LCD_DATA('V');
        LCD_DATA('R');
        LCD_DATA('*');
        LCD_DATA('*');

        LCD_CMD(0xC0+1);   // Select Line2 and 1st coloumn

        int i = 0;

        while(str[i] != 0) {
                LCD_DATA(str[i++]); // to Send String to be Displayed
        }

        while(1); // Halt
}

void LCD_CMD(unsigned char cmd)
{
        LCD_PORT = ((LCD_PORT & 0x0f) | cmd); // put command on lcd port
        RS_LOW(); // RS low
        LCD_PULSE();
        LCD_PORT = ((LCD_PORT & 0x0f) | (cmd<<4)); // put command on lcd port
        RS_LOW(); // RS low
        LCD_PULSE();
        _delay_ms(5);
}
void LCD_DATA(unsigned char data)
{
        LCD_PORT = ((LCD_PORT & 0x0f) | data); // put data on lcd port
        RS_HIGH(); // RS high
        LCD_PULSE();
        LCD_PORT = ((LCD_PORT & 0x0f) | (data<<4)); // put data on lcd port
        RS_HIGH(); // RS high
        LCD_PULSE();
        _delay_ms(5);
}
void LCD_PULSE(void)
{
        EN_HIGH(); // EN high
        _delay_us(100);  // give some delay
        EN_LOW(); // EN low
}
```

Assignment 6 : Analog Sensor interfacing using ADC with LCD 16x2

Platform: AVR ATMEGA16-32 Development Board
Microcontroller: Atmega32

Concept Covered: ADC in 10 bit resolution mode, LCD 16x2 in 4 bit mode

Description: This experiment demonstrates simple ADC 10 bit mode operation with LCD 16x2 in 4 bit mode. A potentiometer output is connected to pin ADC2 (PA2) of ADC Port whose analog value will be converted into digital value and will be displayed on LCD. Rotate the POT with proper tool to see changes when output voltage of pot changed

Hardware Setup:	LCD port:	PORTC
	POT :	ADC2 ie.PA2 of PORTA

Connect the 6 pin 1 to 1 connector cable between PORTC and LCD Header in one to one pattern.

PC2---- RS
PC3---- EN
PC4----D4
PC5---- D5
PC6----D6
PC7----D7

R/W of LCD is grounded by connecting the jumper(R/W to ground) make sure it is connected unless one use R/W Pin in Program Connect the jumper POT---ADC2 on Development Board in order to connect it to PA2 of ADC port

Refer board manual for more detailed description.

Note: Please check the connections properly before performing any experiments.
****

```
#define     F_CPU                       16000000

#include <avr/io.h>             //     inclusion for definition Registers
#include <util/delay.h>         //     inclusion for delay functions
#include <stdlib.h>             //     inclusion for library functions like iota

#define LCD_DDR     DDRC        // Macro to define LCD Data Port
#define LCD_PORT    PORTC

#define LCD_CTRL_DDR        DDRC  // Macro to define LCD Control pin Port
#define LCD_CTRL_PORT       PORTC
#define RS                  PC2
#define EN                  PC3

#define RS_HIGH()   LCD_CTRL_PORT |= (1 << RS)       // RS High
#define RS_LOW()    LCD_CTRL_PORT &= ~(1 << RS)     // RS Low
#define EN_HIGH()   LCD_CTRL_PORT |= (1 << EN)      // EN High
```

```
#define EN_LOW()    LCD_CTRL_PORT &= ~(1 << EN)    // EN Low


/* functions prototypes */
void LCD_CMD(unsigned char cmd);
void LCD_DATA(unsigned char data);
void LCD_PULSE(void);

int main() {                               // Starting point of program
int adc_result;                            // Variable to store the converted value
char str[] = "ADC Demo";                   // Welcome message
LCD_DDR |= 0xF0;                           // Define pins connected to LCD as output pins
LCD_CTRL_DDR |= 1 << RS | 1 << EN;         // LCD initialization

LCD_CMD(0x30);_delay_ms(1);
LCD_CMD(0x30);_delay_ms(1);
LCD_CMD(0x30);_delay_ms(1);

LCD_CMD(0x02); // Command LCD for 4 Bit operations
LCD_CMD(0x28); // to set lcd in 4 bit,2 lines,font 5x7
LCD_CMD(0x0C); // to set display on,cuRSor off ,blinking off
LCD_CMD(0x06); // cuRSor incremENt, no display shift
LCD_CMD(0x01); // to clear lcd

// ADC Initialize

ADMUX = (1 << REFS0); // reference voltage to internal 5v VCC and 10 bit resolution is selected

ADCSRA = (1 << ADPS2)|(1 << ADPS1)|(1 << ADPS0 | 1 << ADEN);// precaler of 128

        int i = 0;

        LCD_CMD(0x80+4);
        while(str[i] != 0) {
                LCD_DATA(str[i++]); // Display welcome message
        }

        while(1) { // indefinite loop

ADMUX|=2;// select Channel 2 ie ADC2(PA2)
ADCSRA |= (1 << ADSC);// Start conversion
while(!(ADCSRA & (1<<ADIF))); // Wait for conversion to complete
ADCSRA|=(1<<ADIF);// Clear the flag manually

adc_result = ADC; // copy the result in variable

itoa(adc_result,str,10); // Convert the integer value into string to display on LCD
int i = 0;
LCD_CMD(0xC0+6); // Select line2 and 6th coloumn
while(str[i] != 0) {
                    LCD_DATA(str[i++]); // display the value
                    }
                    LCD_DATA(' ');
                    // Now rotate the pot with proper tool to see the changes
          }
}

void LCD_CMD(unsigned char cmd)
{
          LCD_PORT = ((LCD_PORT & 0x0f) | cmd); // put command on lcd port
          RS_LOW(); // RS low
          LCD_PULSE();
          LCD_PORT = ((LCD_PORT & 0x0f) | (cmd<<4)); // put command on lcd port
          RS_LOW(); // RS low
          LCD_PULSE();
          _delay_ms(5);
}

void LCD_DATA(unsigned char data)
{
          LCD_PORT = ((LCD_PORT & 0x0f) | data); // put data on lcd port
          RS_HIGH(); // RS high
          LCD_PULSE();
          LCD_PORT = ((LCD_PORT & 0x0f) | (data<<4)); // put data on lcd port
          RS_HIGH(); // RS high
          LCD_PULSE();
          _delay_ms(5);
}

void LCD_PULSE(void)
{
          EN_HIGH(); // EN high
          _delay_us(100);  // give some delay
          EN_LOW(); // EN low
}
```

Assignment 7 : Serial Communication with PC using UART

Platform: AVR ATMEGA16-32 Development Board
Microcontroller: Atmega32

Concept Covered: Serial communication UART

Description: This experiment demonstrates simple Serial communication operation with PC. Microcontroller will implement a serial communication protocol which is already implemented in PC. Controller will read a input from PC and in response send some characters to PC in order to check the communication is successfully established .Open any terminal software and type in any key you should get "OK" as a response from microcontroller.

Hardware Setup:

UART: Connect RXD and TXD jumpers in order to connect microcontroller with RS232 Communication Setup connect the board with your PC with RS232 Cable a cables another end should be connected to the COM port of PC

Refer board manual for more detailed description.
Note: Please check the connections properly before performing any experiments.

```
***************************************************************************/
#define F_CPU          16000000

#include <avr/io.h>                        //      inclusion for defination of Registers
#include <util/delay.h>                    //      inclusion for delay functions

#define USART_BAUDRATE   9600   //      Define Baud Rate value
#define BAUD_PRESCALE    (((F_CPU / (USART_BAUDRATE * 16UL))) - 1) // Formula to
Calculate UBRR values

char UART_ReceiveByte(void);
void UART_SendByte(char data);
void UART_SendString(char* str);

int main()
{
        char ch;

UCSRB |= (1 << RXEN) | (1 << TXEN);   // Turn on the transmission and reception circuitry
UCSRC |= (1 << URSEL)|(1 << UCSZ1)|(1 << UCSZ0); // Use 8-bit character sizes

UBRRL = (uint8_t)BAUD_PRESCALE;// Load lower 8-bits of the baud rate value into the low byte of the UBRR register
UBRRH = (uint8_t)(BAUD_PRESCALE >> 8);
 // Load upper 8-bits of the baud rate value into the high byte of the UBRR register

        while(1)                           // indefinite loop
        {
                ch = UART_ReceiveByte();    // Read a input from PC
                UART_SendString("OK\r");    // Send "OK" to PC
        }
}

char UART_ReceiveByte(void)
{
  //Wait untill a data is available

  while(!(UCSRA & (1<<RXC)));

  //Now USART has got data from host
  //and is available is buffer

  return UDR;
}
void UART_SendByte(char data)
{
  //Wait until the transmitter is ready

  while(!(UCSRA & (1<<UDRE)));

  //Now write the data to USART buffer

  UDR=data;
}


void UART_SendString(char* str)
{
        while(*str != '\0')
        {
                UART_SendByte(*str++);
        }
}
```

Assignment 8 : Real Time Clock using DS1307 and LCD 16x2

Platform: AVR ATMEGA16-32 Development Board
Microcontroller: Atmega32

Concept Covered: TWI/I2C protocol, RTC , LCD 16x2

Description: This experiment demonstrates simple Digital Clock and Calendar Demo. Microcontroller will implement a i2c- a two wire communication protocol which will communicate with Real time clock IC DS1307 and collects the date and time information therby displaying it on LCD 16x2.

Hardware Setup:
TWI/I2c Pins:
Connect PC0(SCL) and PC1(SDA) of PORTC from microcontroller to SCL and SDA header on Board

LCD Port: PORTC
Connect the 6 pin 1 to 1 connector cable between PORTC and LCD Header in one to one pattern.

PC2---- RS
PC3---- EN
PC4----D4
PC5---- D5
PC6----D6
PC7----D7

R/W of LCD is grounded by connecting the jumper(R/W to GND) make sure it is connected unless one use R/W Pin in Program

Refer board manual for more detailed description.
Note: Please check the connections properly before performing any experiments.

```
*************************************************************************/
#define F_CPU          16000000

#include <avr/io.h>                    //      inclusion for defination Registers
#include <util/delay.h>                //      inclusion for delay functions
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "LCD4Bit.h"           //      include LCD Library to use LCD functions

#define ACK            1
#define N_ACK          0

/* functions prototypes */
void TWI_Init(void);
void TWI_SendStart(void);
void TWI_SendSlaveData(uint8_t data);
uint8_t TWI_ReadSlaveData(uint8_t last);
void TWI_SendStop(void);
void RTC_Init(void);
void RTCSetTimeDate(void);
void RTCGetTimeDate(void);
uint8_t BinaryToBcd(uint8_t x);
uint8_t BcdToBinary(uint8_t x);

// structure to hold values for time and date
struct TimenDate
{
        int sec;
        int min;
        int hour;
        int day;
        int date;
        int month;
        int year;
};

struct TimenDate td;


int main(void) { //      Starting Point of Program

        char str[16];
        LCD_Init();                // Initialize LCD
        RTC_Init();                // Initialize DS1307

        /*td.date = 4;
        td.month = 8;
        td.year = 14;
        td.hour = 15;
        td.min = 26;
        td.sec = 0;
        RTCSetTimeDate();*/

        while(1)
        {
                RTCGetTimeDate();    // Read time and date from DS1307

                // Display them on LCD
                sprintf(str,"%02d/%02d/%02d",td.date,td.month,td.year);
                LCD_String(str,LINE1);

                sprintf(str,"%02d:%02d:%02d",td.hour,td.min,td.sec);
                LCD_String(str,LINE2);
        }
}

//#################################################
/*                          Functions implementing i2c/TWI                  */

void TWI_Init(void)
{
        TWBR=10;
        TWSR &= ~(1<<TWPS1);
        TWSR &= ~(1<<TWPS0);
}

void TWI_SendStart(void)
{
        TWCR = 1 << TWINT | 1 << TWSTA | 1 << TWEN;
        while(!(TWCR & (1 << TWINT)));
}

void TWI_SendSlaveData(uint8_t data)
{
        TWDR = data;
        TWCR = 1 << TWINT | 1 << TWEN;
        while(!(TWCR & (1 << TWINT)));
}

uint8_t TWI_ReadSlaveData(uint8_t last)
{
        char ch;

        if(!last)
        {
                TWCR = 1 << TWINT | 1 << TWEN;
        }
        else
        {
                TWCR = 1 << TWINT | 1 << TWEN | 1 << TWEA;
        }

        while(!(TWCR & (1 << TWINT)));

        ch = TWDR;

        return ch;
}

void TWI_SendStop(void)
{
        TWCR = 1 << TWINT | 1 << TWSTO | 1 << TWEN;
        while(!(TWCR & (1<<TWSTO)));
}

//#######################################################
/*                          Functions to communicate with DS1307                */

void RTC_Init(void)
{
        TWI_Init();
}

void RTCSetTimeDate(void)
{
        TWI_SendStart();
        TWI_SendSlaveData(0xD0);
        TWI_SendSlaveData(0x00);
        TWI_SendSlaveData(0x7F & BinaryToBcd(td.sec));
        TWI_SendSlaveData(0x7F & BinaryToBcd(td.min));
        TWI_SendSlaveData(0x3F & BinaryToBcd(td.hour));
        TWI_SendSlaveData(0x07 & BinaryToBcd(td.day));
        TWI_SendSlaveData(0x3F & BinaryToBcd(td.date));
        TWI_SendSlaveData(0x1F & BinaryToBcd(td.month));
        TWI_SendSlaveData(0xFF & BinaryToBcd(td.year));
        TWI_SendStop();
}

void RTCGetTimeDate(void)
{
        TWI_SendStart();
        TWI_SendSlaveData(0xD0);
        TWI_SendSlaveData(0x00);
        TWI_SendStart();
        TWI_SendSlaveData(0xD1);
        td.sec = (0xFF & BcdToBinary(TWI_ReadSlaveData(ACK)));
        td.min = (0x7F & BcdToBinary(TWI_ReadSlaveData(ACK)));
        td.hour = (0x3F & BcdToBinary(TWI_ReadSlaveData(ACK)));
        td.day = (0x07 & BcdToBinary(TWI_ReadSlaveData(ACK)));
        td.date = (0x3F & BcdToBinary(TWI_ReadSlaveData(ACK)));
        td.month = (0x1F & BcdToBinary(TWI_ReadSlaveData(ACK)));
        td.year = (0xFF & BcdToBinary(TWI_ReadSlaveData(N_ACK)));
        TWI_SendStop();
}

uint8_t BinaryToBcd(uint8_t x)
{
        uint8_t y;
        y = ((x/10 << 4) | (x % 10));
```

```
	return y;
}

uint8_t BcdToBinary(uint8_t x)
{
	uint8_t y;
	y = ((((x & 0xF0)>>4) *10 ) + (x  & 0x0f));
	return y;
}
```

AVR ATMEGA16-32 Development Board

Manual

Chapter 1

1.0 Board Description:

AVR Atmega16-32 Development Board is a powerful development board which can be used with any of two AVR microcontrollers Atmega16 or Atmega32 from Atmel. The board comes with Atmega16 AVR microcontroller installed in DIP package.

AVR Atmega16 is a high performance, low power 8-bit microcontroller from Atmel featuring 16K Bytes of Flash, 512 Bytes EEPROM, 1K Byte Internal SRAM, Two 8-bit Timer/Counters, One 16-bit Timer/Counter, Four PWM Channels, 8-channel 10-bit ADC, 32 Programmable I/O Lines and many more.

In order to explore its powerful features and start developing embedded applications the board is incorporated with 8 LED's, 4 Push button switch, 4x4 Matrix Keypad, 16x2 Character LCD, 2 Seven segment Displays, 3 Analog Sensors (LM35, LDR, Potentiometer), Dual DC Motor Drivers, Relays, Buzzer, Stepper Motor Driver, RS-232 interface for Serial Communication with PC, TWI interface for Real Time Clock and 32kb of External EEPROM.

The Board is extremely useful for learning embedded systems using AVR microcontroller and developing varieties of embedded application. Any customization is possible because of its design which allows you to use any of the microcontroller port anywhere in the application. For the sake of simplicity some peripherals are connected to the port directly using two pin jumpers which are easily removable is someone wants to use those pin for some other purpose.

The board operates at 7V to 15V DC power supply or battery and precaution must be taken before connecting any power source.

1. Please check the power source is within recommended range and then only connect it to the board.
2. Cross check the polarities before connecting batteries.
3. Do not remove any component unless you are sure how to put it back.
4. Please perform experiment under expert's supervision or one with knowledge of electronics.
5. Input power supply should not exceed 15 V DC.
6. Read the user manuals completely before start using this product

Chapter 2

2.0 Specifications:

- Microcontroller: ATMEGA16 with 16 MHz crystal (Also supports ATMEGA32)
- Single side high quality PCB.
- Required Power Supply: 7 to 15V DC
- Switches: Reset, Power
- 10 pin FRC and 6 pin straight connectors for In System Programming (ISP)
- On board LCD 16x2 interface
- 4x4 Matrix Keypad interface
- RS232 serial interface along with the MCUs UART jumpers
- 3 on board analog sensors 8 (LM35, LDR, POT) and channels for sensors interfacing
- RTC interface with 3V Li battery
- On board External EEPROM
- Dual Motor driving interface with L293D
- On board 2 Relays
- Eight on board LEDs and 4 Switches
- A Buzzer interfacing
- Stepper Motor interface
- 2 multiplexed 7 Segment Display with BCD to 7 segment decoder chip
- Software and Application examples in WinAVR are provided in the documentation CD

2.1 Kit Contains:

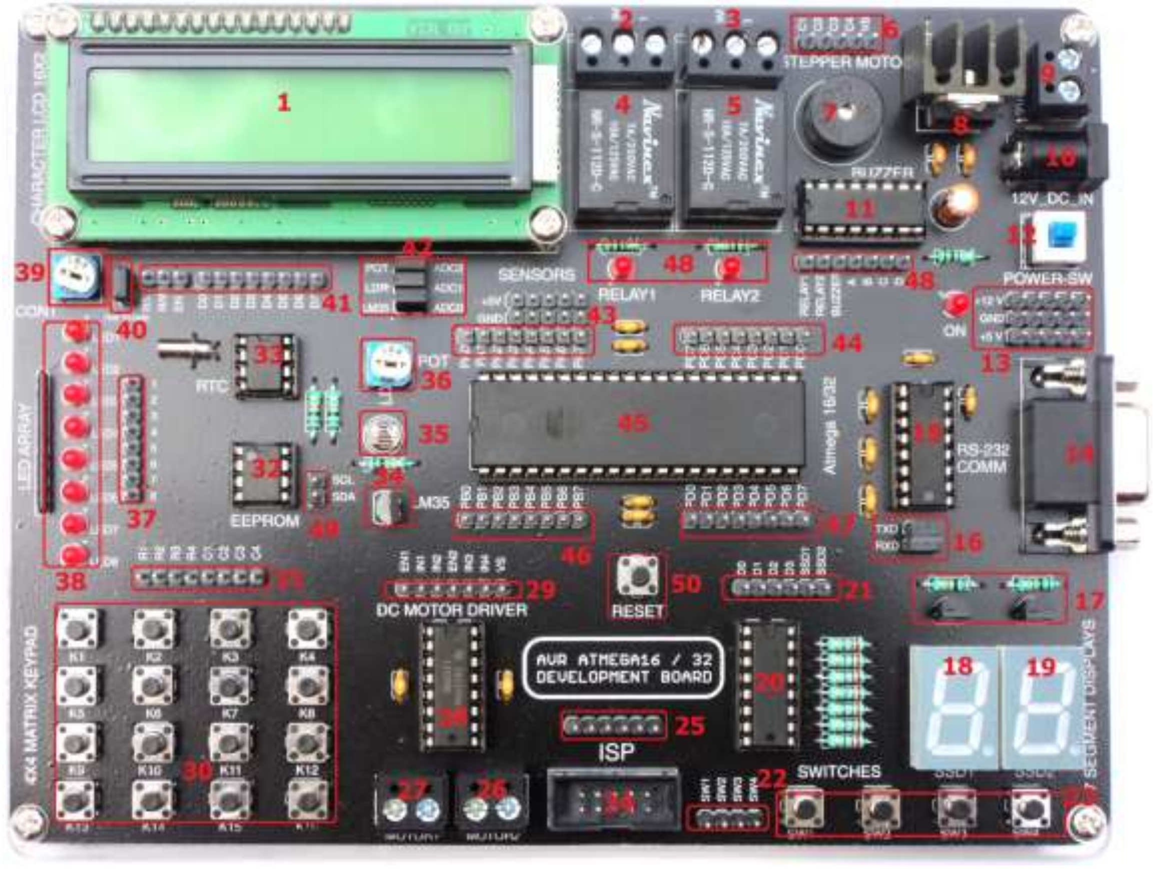
- 1- ATMEGA16-32 Development Board
- 1- 12 DC 1 A Power Supply
- 1- USB 2.0 Cable
- 1- DB9 Serial Cable
- 25 - 1 to 1 jumper cables
- 1- Documentation CD

2.2 Documentation CD contain following Application examples:

- LED interfacing
- IO interfacing
- 7 Segment display interfacing
- LCD interfacing
- 4x4 Matrix Keypad interfacing
- ADC sensor interfacing demo with LCD
- Relay buzzer interfacing using ULN2003 interface
- L293D demo
- UART Serial Communication example
- RTC demo
- External EEEROM interfacing

Chapter 3

Board Overview:

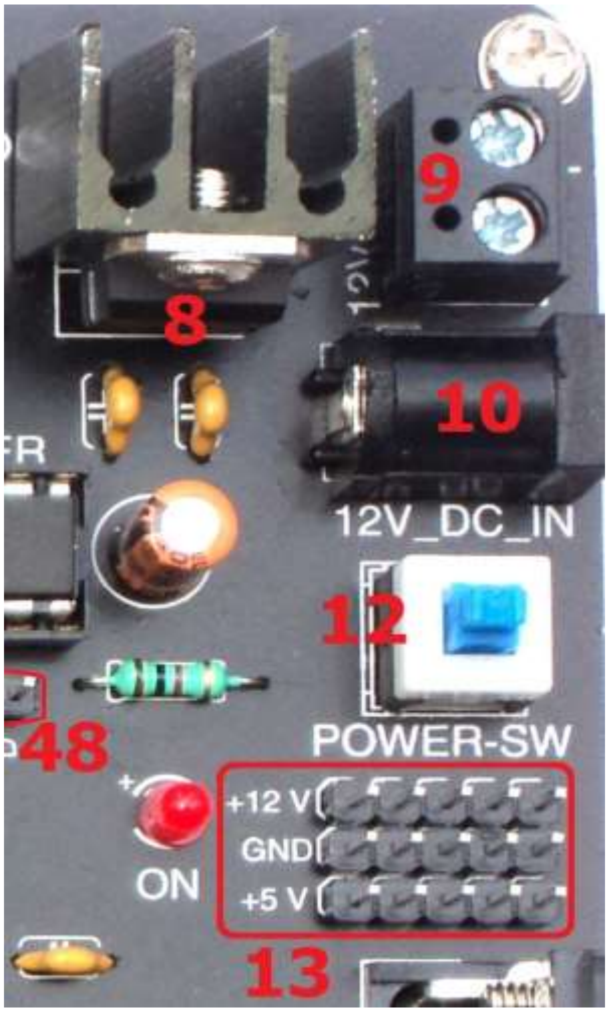


1. LCD 16x2
2. Relay 1 Pin out Connector
3. Relay 2 Pin out Connector
4. 12 V DC SPDT Relay1
5. 12 V DC SPDT Relay2
6. Pin out for Stepper Motor
7. Buzzer
8. 7805 with Heat sink
9. 2 Pin Battery Connector
10. DC Jack
11. IC ULN2003
12. Power On/Off Switch
13. Power Pin Headers
14. DB9 connector
15. IC Max232
16. RXD TXD Pin Selection Jumpers
17. 7 Segment Selection Transistors
18. 7 Segment Display 1
19. 7 Segment Display 2
20. IC 74LS47
21. 7 Segment Interfacing Header
22. Switches Interfacing Header
23. 4 Push Button Switches
24. 10 Pin FRC Connector for ISP
25. 6 Pin Straight header for ISP
26. 2 Pin out for DC motor 2
27. 2 Pin out for DC motor 1
28. IC L293D
29. DC Motor Driver Interfacing Header
30. 4x4 Matrix Keypad
31. 4x4 Matrix Interfacing Header
32. IC 24C32 32 Kb EEPROM
33. IC DS1307 RTC
34. IC LM35 Temperature Sensor
35. LDR
36. Potentiometer
37. LED Interfacing Header
38. 8 LED Array
39. LCD Contrast Adjust POT
40. LCD R/W to GND Jumper
41. LCD Interfacing Header
42. Analog Selection Jumpers
43. PORTA Pin out Header
44. PORTC Pin out Header
45. IC Atmega16 (MCU)
46. PORTB Pin out Header
47. PORTD Pin out Header
48. ULN2003 Header
49. SCL and SDA Header
50. Reset Switch

Chapter 4

4. Hardware Description: In this chapter various components mounted on the board and their principal of operation is explained in detail. It is very important that you go through this chapter before start using the ATMEGA16-32 Development Board

4.0 Power Supply Section:



The power supply section consists of 7805 voltage regulator IC with heat sink and filtering capacitors, which can supply 1 Amp current at regulated 5v.

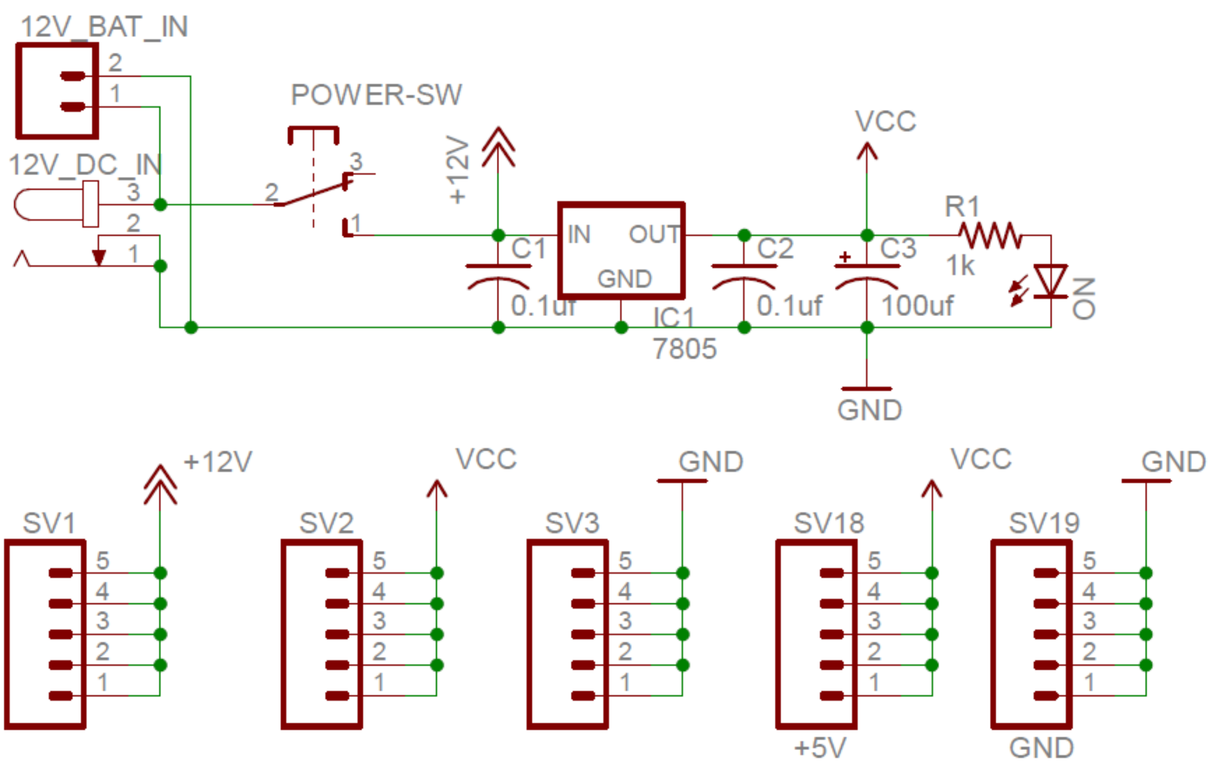
You can connect any DC adapter at DC jack which can supply 9V to 15V DC and at least 1 Amp of current. A standard adapter of 12V DC and 1 Amp supply is recommended for best results.

Instead of DC Adapter External Battery can also be connected to the through 2 pin screw terminal. Again battery voltage should not exceed 15V. Precaution should be taken while connecting + and – terminal to the connector.

After you connect the supply board can be turned on by pressing POWER-SW, ON led indicates that the board is turned ON as it glows if supply is fine.

There are +12v, GND and +5v pin out provided if required to use in any case with external circuit or on board.

4.1 Schematic of Power Supply Section:

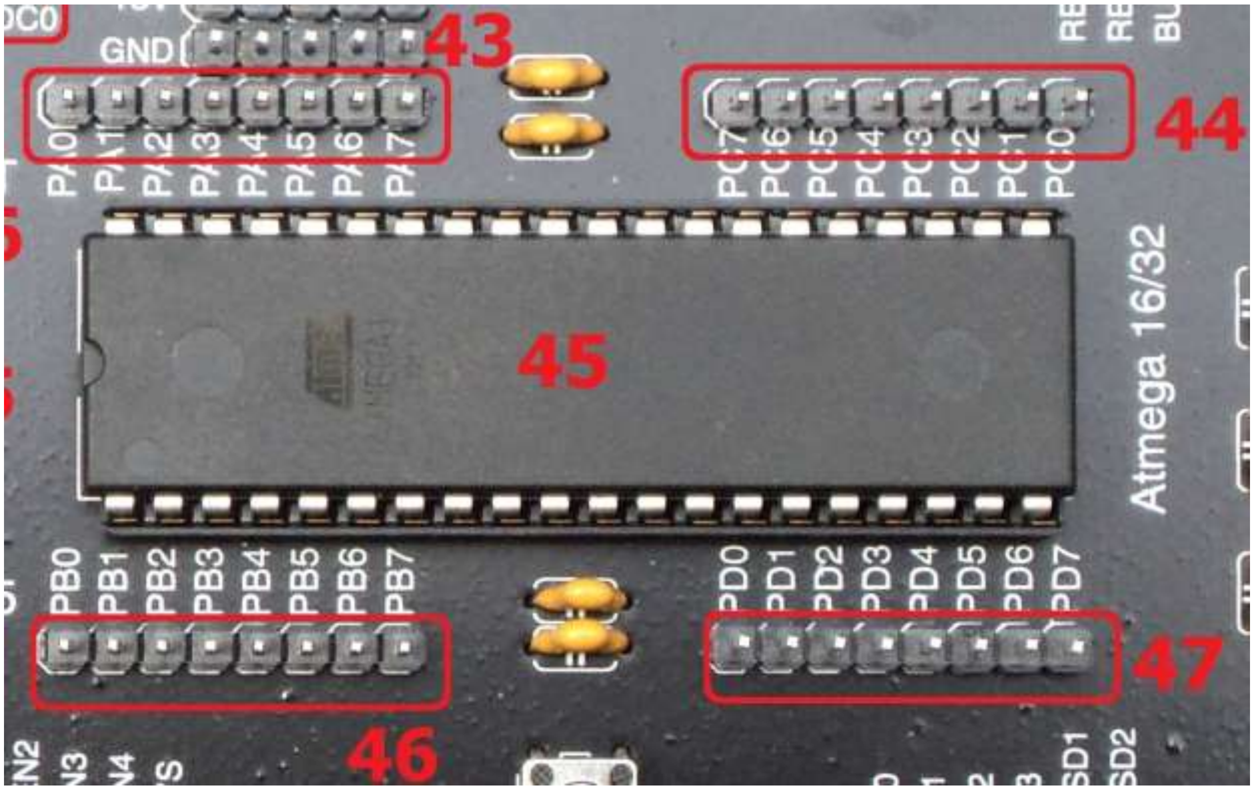


Chapter 5

5.0 Microcontroller I/O and Programming Section:

Microcontroller I/O Section:

Microcontroller I/O section consists Atmega16 along with some filtering capacitors for smooth voltage supply, a 16 MHz crystal oscillator for clock source and I/O pins headers

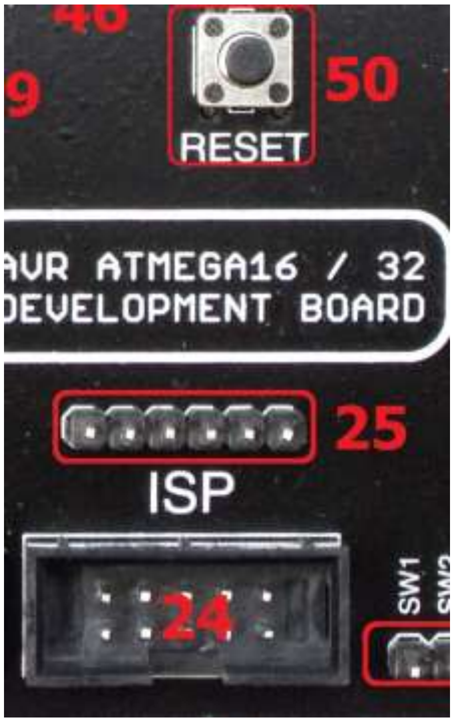


All 4 ports of Atmega16 are brought out in order to connect them any peripheral on board through 8 pin straight header

46-PORTB
47- PORTD
48- PORTC
43- PORTD

Programming Section:

This section consists of a reset switch and In Systems Programming connectors.



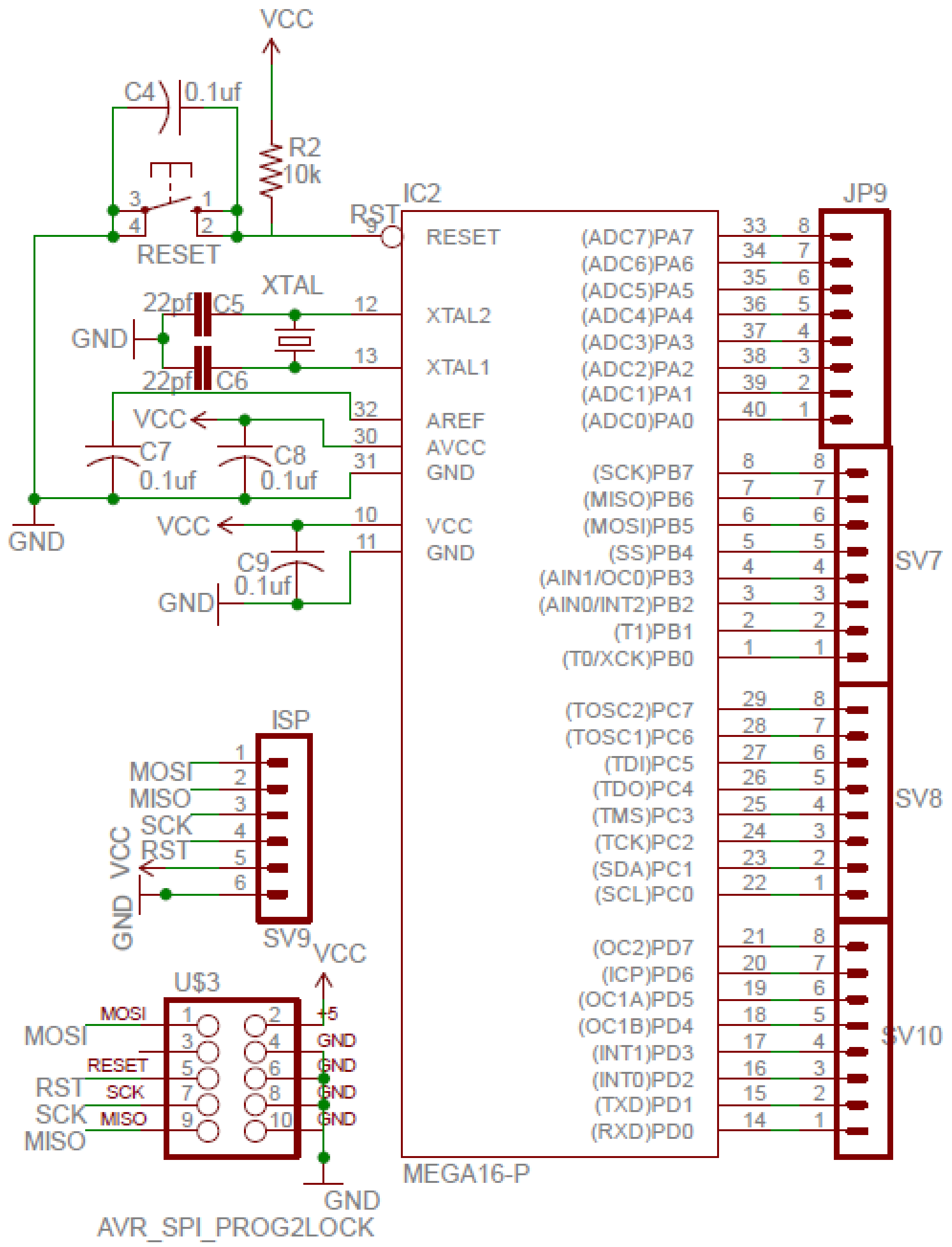
By pressing reset switch microcontroller jumps to the starting location of application memory. Reset is used when one wants to restart the program in application memory from initials position.

For programming Atmega16 two type of connectors are provided and programming can be done by using any of the two which basically suites your programmer.

No need to press reset switch while programming microcontroller as the programmer automatically resets the microcontroller while programming.

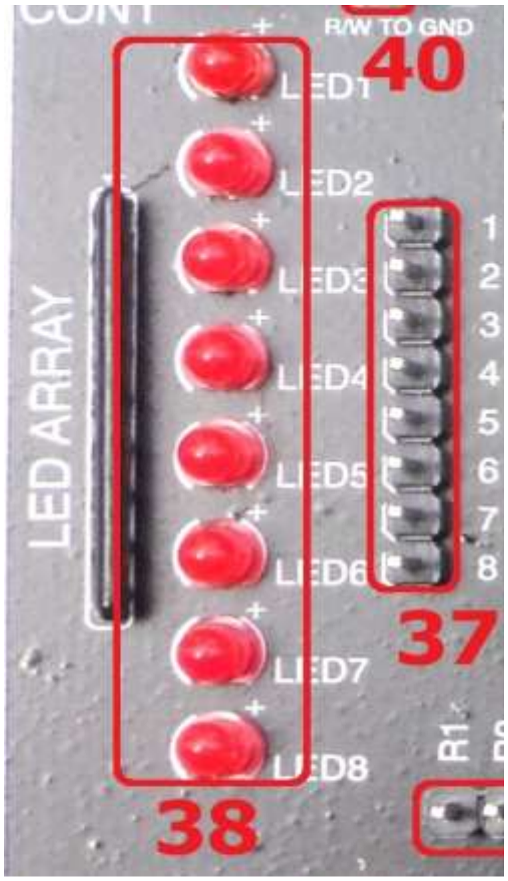
Please read getting started application note for more details on programming.

5.1 Circuit Schematic for microcontroller and programming section:



Chapter 6

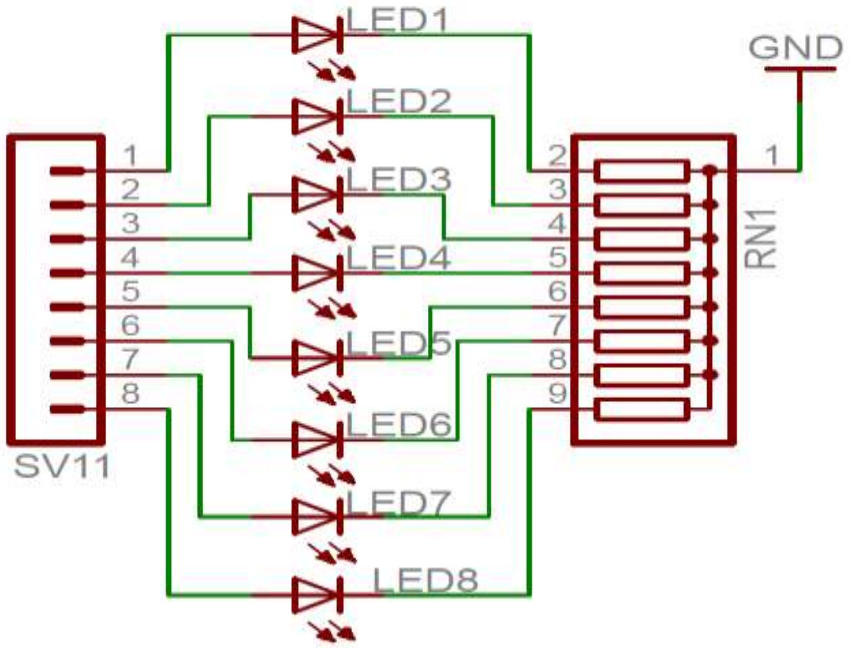
6.0 LED Array Section:



LED array section consists of high intensity 8 Red LEDs connect with 8 pin straight male header which can be connected to any port using 1 to 1 jumper wires

In order to turn on this particular LEDs one needs to provide logic 1 i.e. 5v on the I/O pin with which it is connected.

6.1 LED Array Schematic:



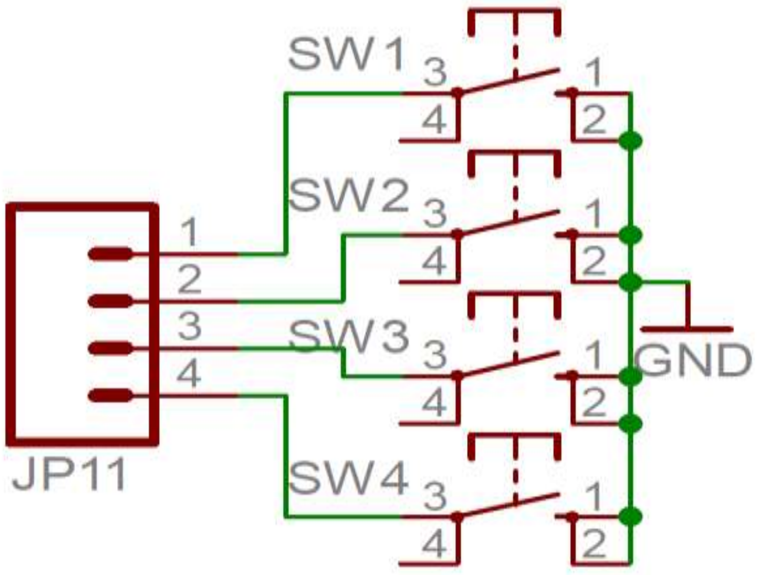
Chapter 7

7.0 Push Button Switches Section:

This section consists of 4 Push button switches are which can be accessed through 4 pin header and can be connected to any I/O pin through jumper wires

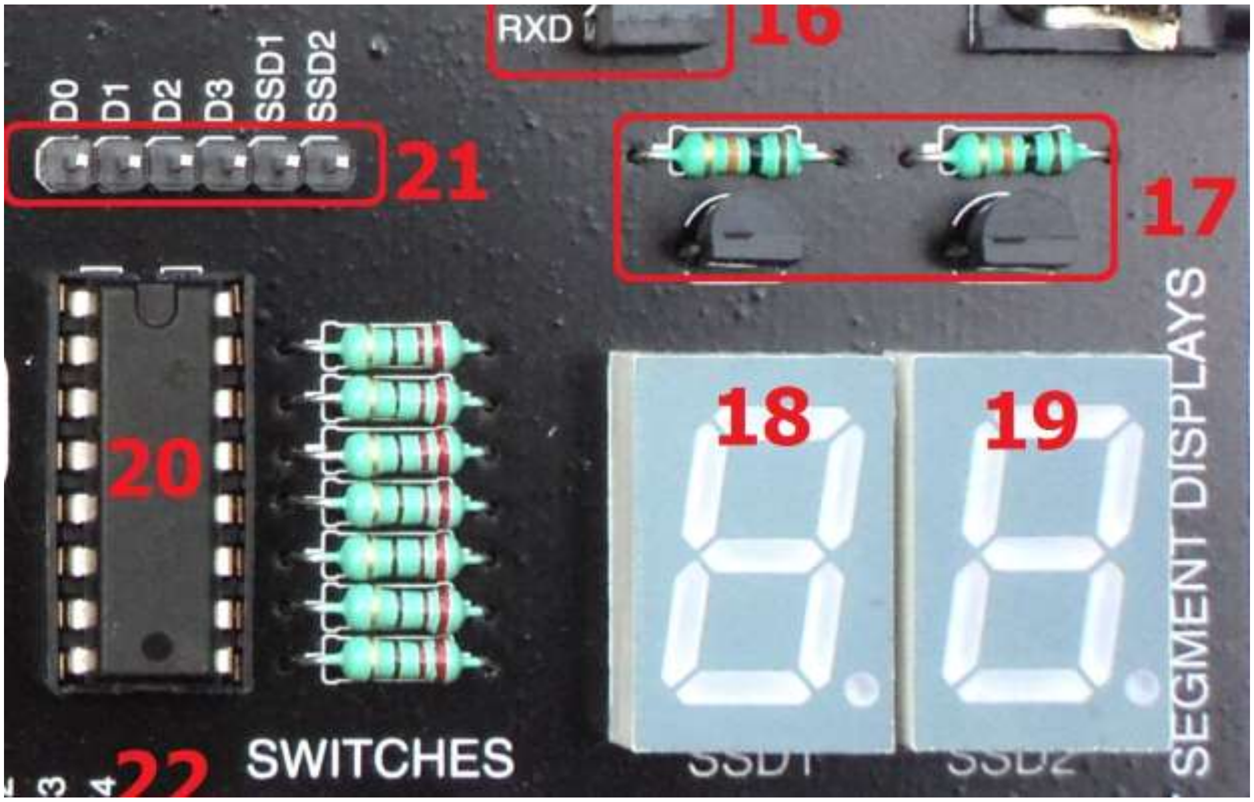
Switches are connected in such a way that if a switch is presses the particular header pin related to that particular switch will be grounded, which will create a logic 0 i.e. GND on that particular header pin.

7.1 Schematic for Push Button Switches Section:



Chapter 8

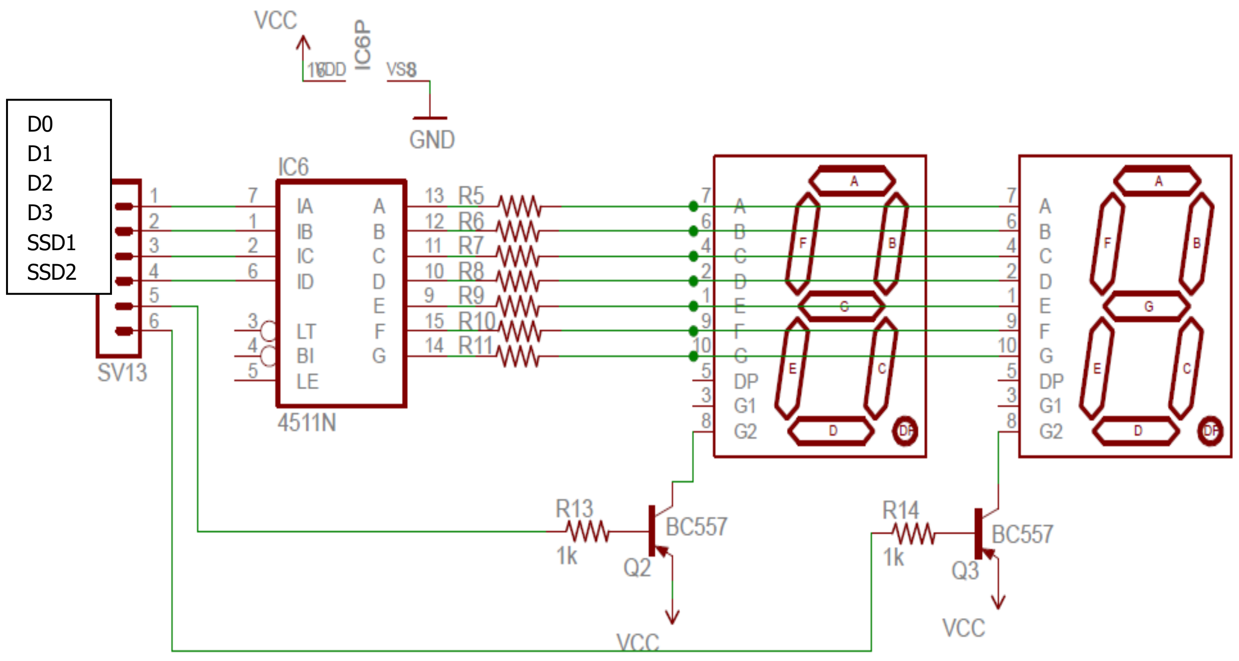
8.0 7 Segment Display Section:



This section consists of two common anode multiplexed 7 segment displays and a BCD to 7 segment decoder chip 74LS47 along with the interfacing header. For more details on working for 74LS47 please read the datasheet of 74LS47.

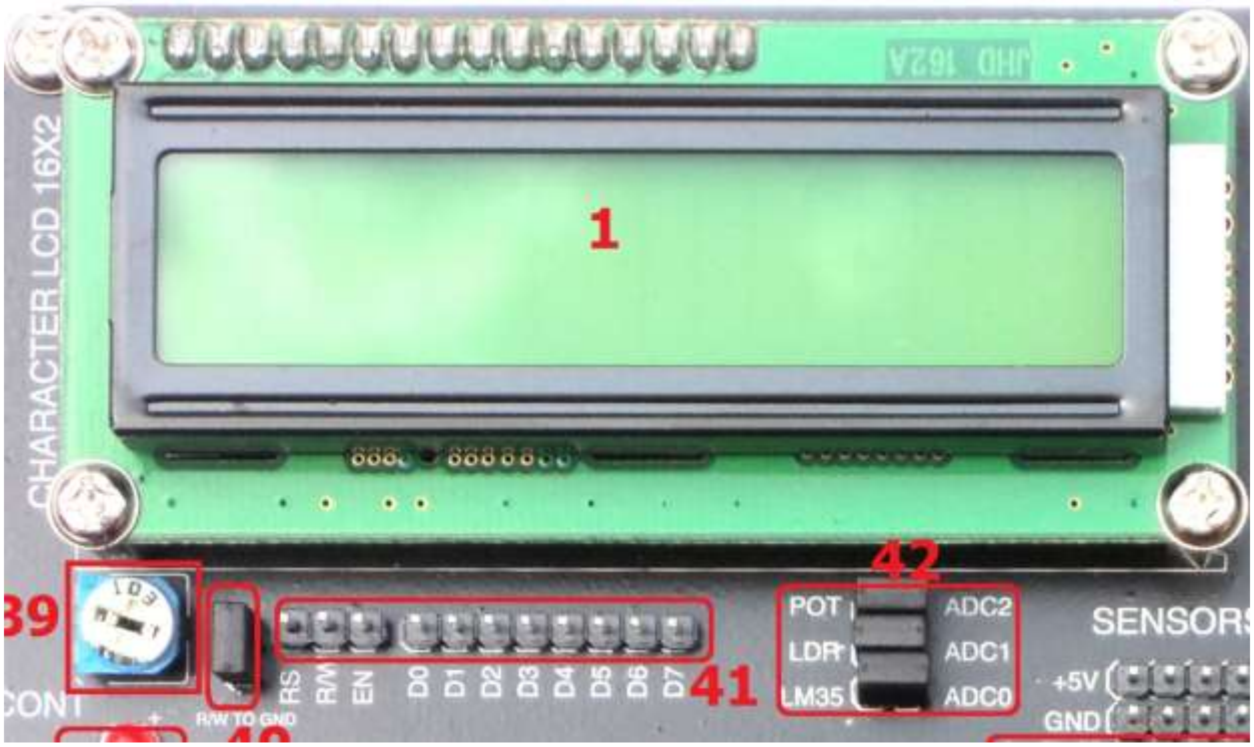
The interfacing header allows you to input BCD Code through pins D0 D1 D2 D3
As the displays are multiplexed the data pins are common so in order to select a particular display in which data is to be shown one need to select it through SSD1 an SSD3 Select pins. These select pins are connected to anode terminals of each display through PNP transistors. Thus by giving logic 0 on is pin will activate the transistor there by activating the 7 segment display.

8.1 7 Segment Display Section Schematic:



Chapter 9

9.0 Character LCD 16 x 2 Sections:

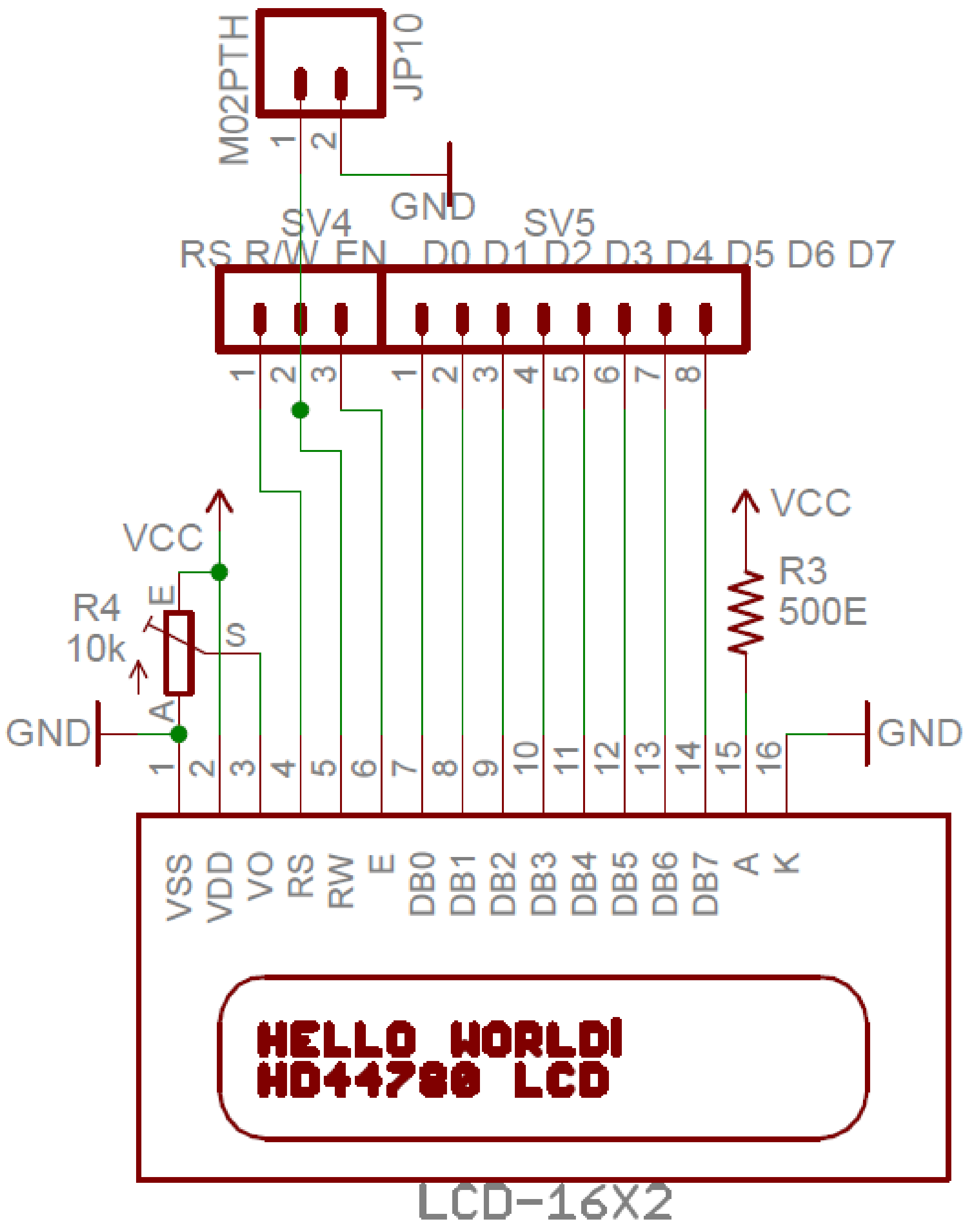


This section consists of LCD 16x2 and its interfacing Header along with the LCD contrast adjustment POT through which intensity on characters can be adjusted.

The interfacing header consists of control pins of LCD RS, RW, EN and data pins D0 to D7.

Most of the programmers ground the R/W pin unless it is used in programming so we did it using an R/W to ground jumper. It will connect to ground as long as jumper is connected and we can use LCD with only 2 control pins RS and EN. Thus we can reduce the pins need to interface LCD with microcontroller.

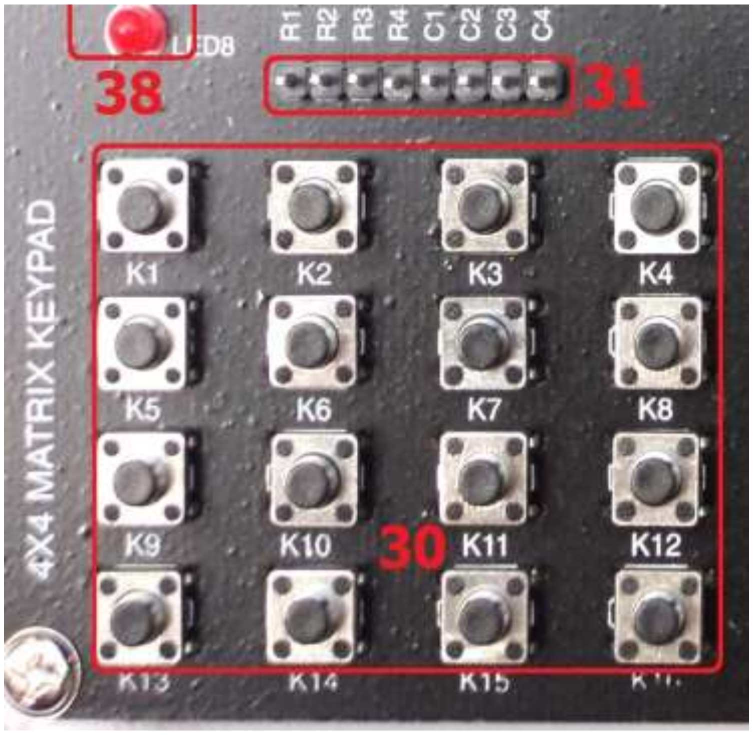
9.1 Character LCD 16 x 2 Sections Schematic:



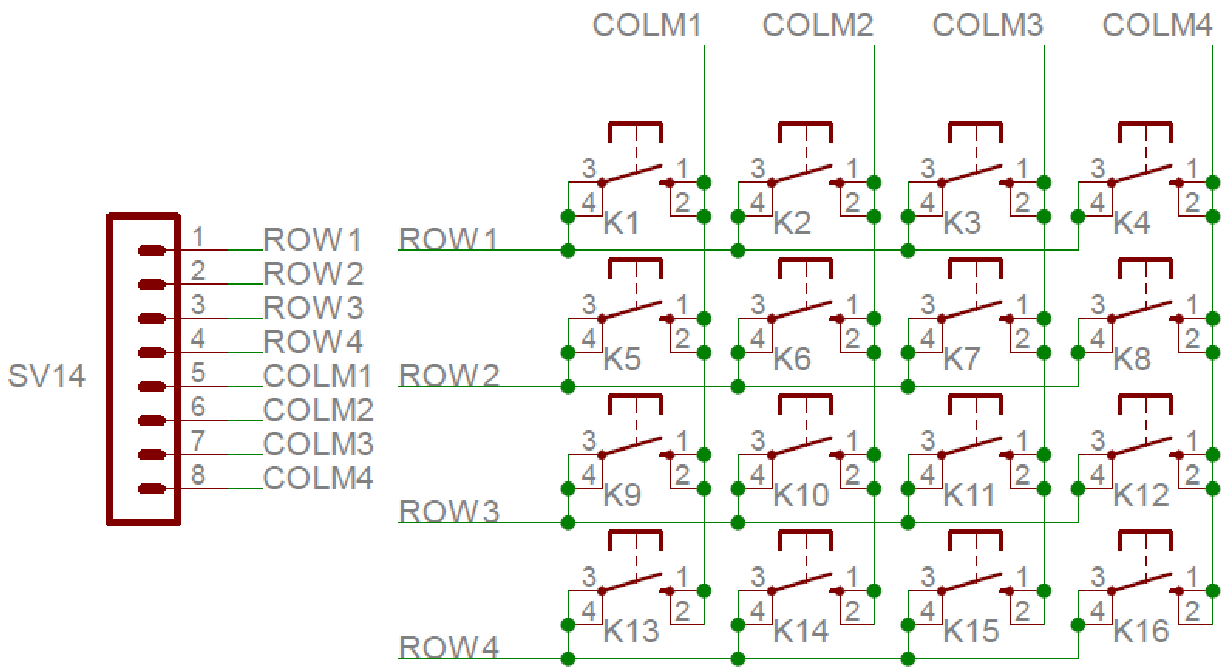
Chapter 10

10.0 4 x 4 Matrix Keypad Section:

This Section consists of 4x4 matrix keypad arranged in 4 row and 4 column sequence and an 8 pin keypad interfacing header which can be connected to microcontroller port in order to add keypad feature into the application.



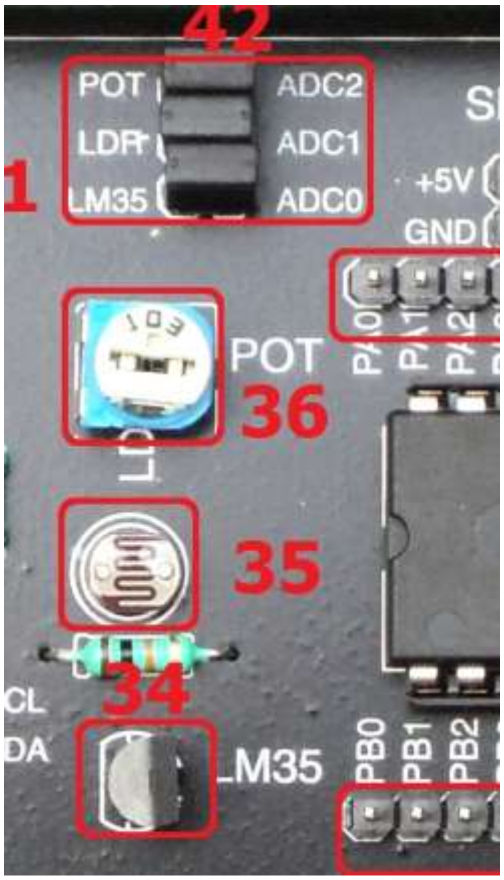
10.1 4 x 4 Matrix Keypad Section Schematic:



Chapter 11

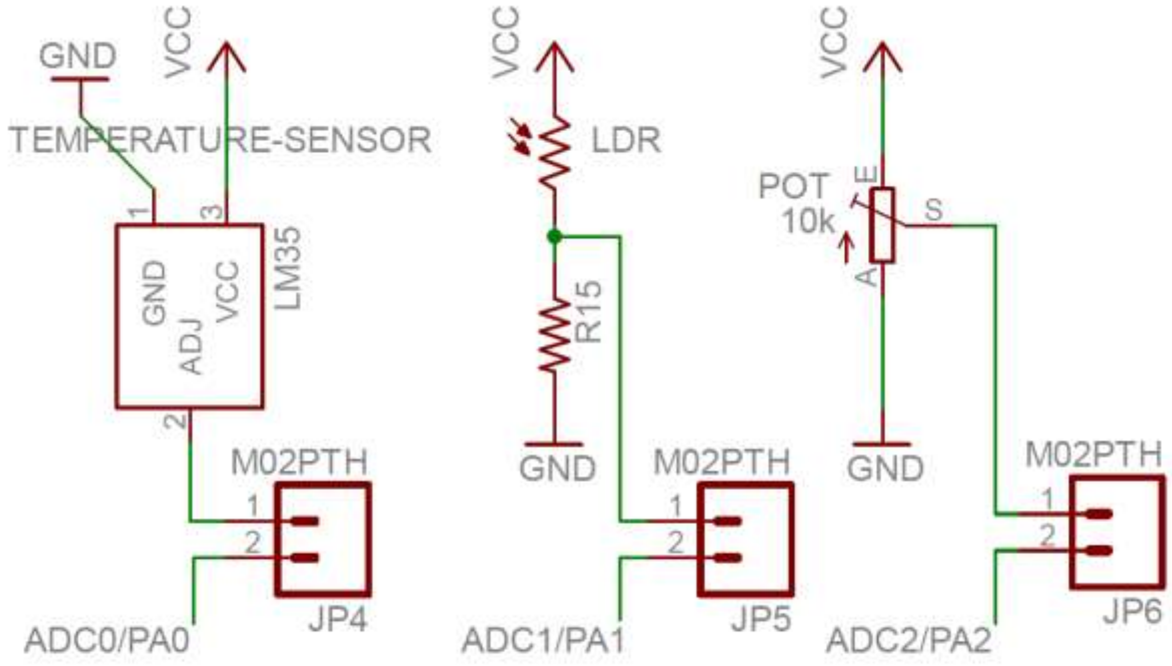
11.0 Analog Sensors Section:

This section consists of analog sensors connected to the ADC port of microcontroller along with the selection jumper. The section jumper allows you make a choice whether to connect the sensor to the ADC pin or if one wants to use that pin for some other purpose the jumper needs to be remove in order to set that pin free to use with other circuit.



36 is POT connected to ADC2 Pin through jumper
35 is LDR connected to ADC1 Pin through jumper
34 is LM35 connected to ADC0 Pin through jumper

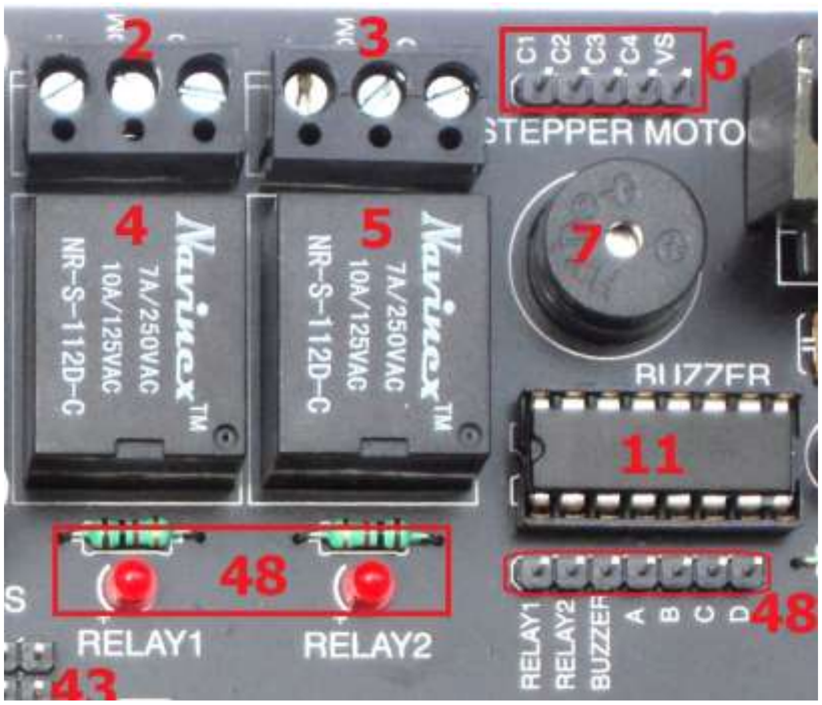
11.1 Analog Sensors Section Schematic:



Chapter 12

12.0 Relays and Buzzer Section:

This section consists of 2 relays and a buzzer connected at the output of ULN2003 Driver IC. In addition to this a stepper motor can be interface with the stepper motor interfacing header (6).
All this peripherals are controlled through the ULN2003 Driver IC header (48).

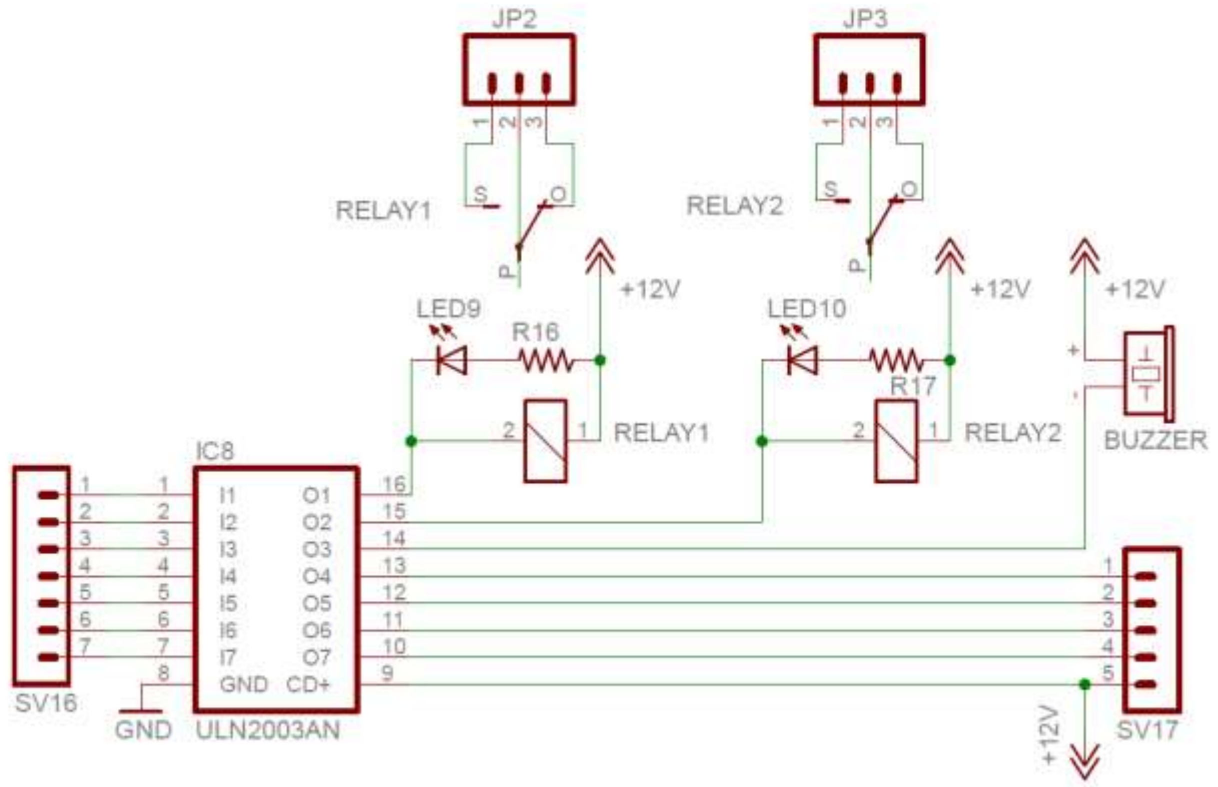


In order to turn on Relay1 one needs to provide logic 1 at the RELAY1 pin of the IC header (48) and logic 0 to turn it OFF. Similarly Relay2 and buzzer can be turned ON or OFF. Devices (AC / DC) can be controlled using relays through 3 Pin screw terminal connectors of specific relay.

Pins A, B, C, and D of the IC header (48) controls the stepper motor if connected. One needs to provide a proper sequence needed for stepper motor to rotate on this pins and a stepper motor can be driven through stepper motor header (6).
Connect C1 to Coil 1 of Stepper Motor
Connect C2 to Coil 2 of Stepper Motor
Connect C3 to Coil 3 of Stepper Motor
Connect C4 to Coil 4 of Stepper Motor
Connect VS to Supply Pin of Stepper Motor

VS here Provide +12V to drive DC Stepper Motor at 12V Voltage.

12.1 Relays and Buzzer Section Schematic:

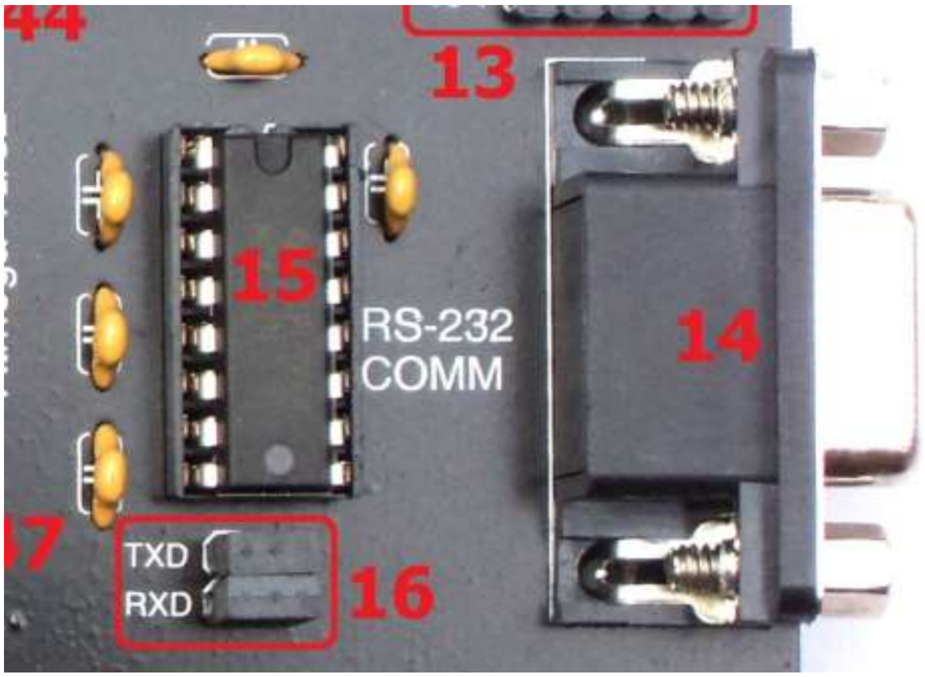


Chapter 13

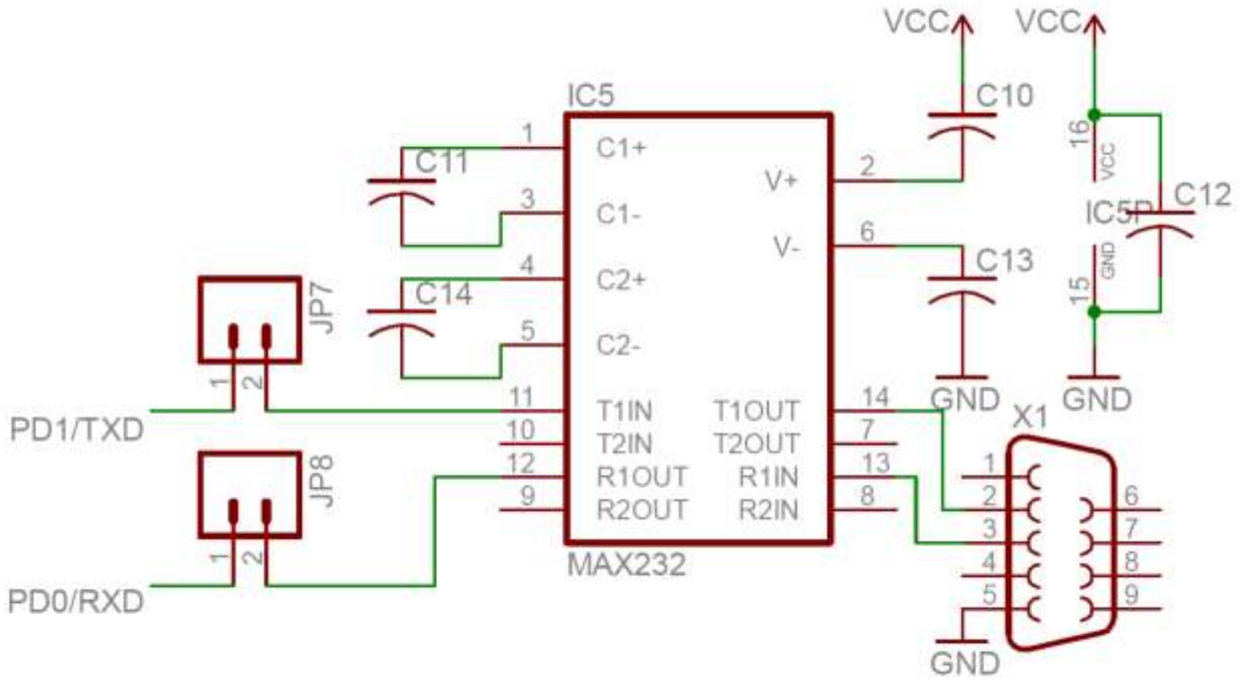
13.0 UART Serial Communication Section:

This section consists of an RS-232 communication circuit made of MAX232 IC (15) and a 9 pin connector (14). A serial communication with PC can be established by connecting PC COM port to this connector using a serial RS-232 standard cable.

In order to connect this section with microcontroller the two serial communication pins of microcontroller can be connected through RXD and TXD jumpers (16).
Removing this jumper will disconnect connection between microcontroller and RS-232 communication section.



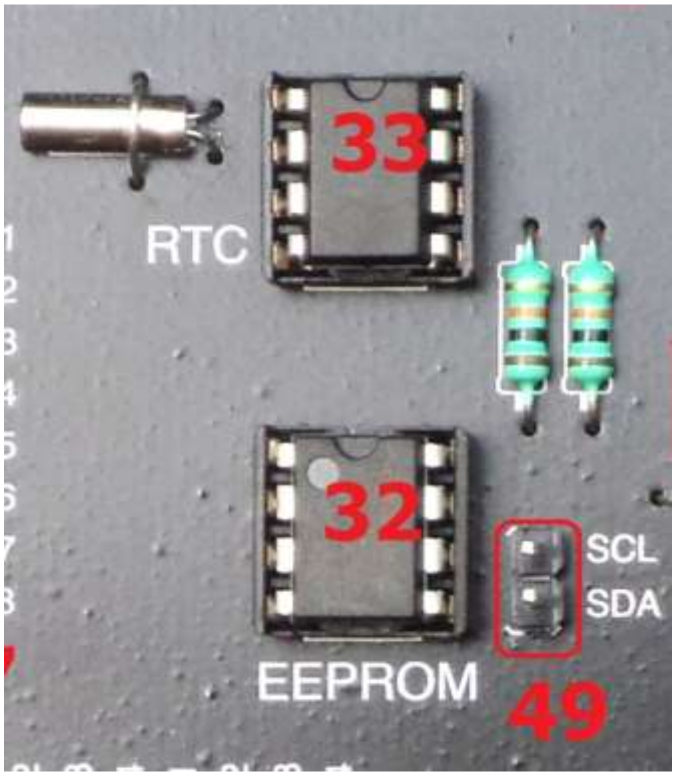
13.1 UART Serial Communication Section Schematic:



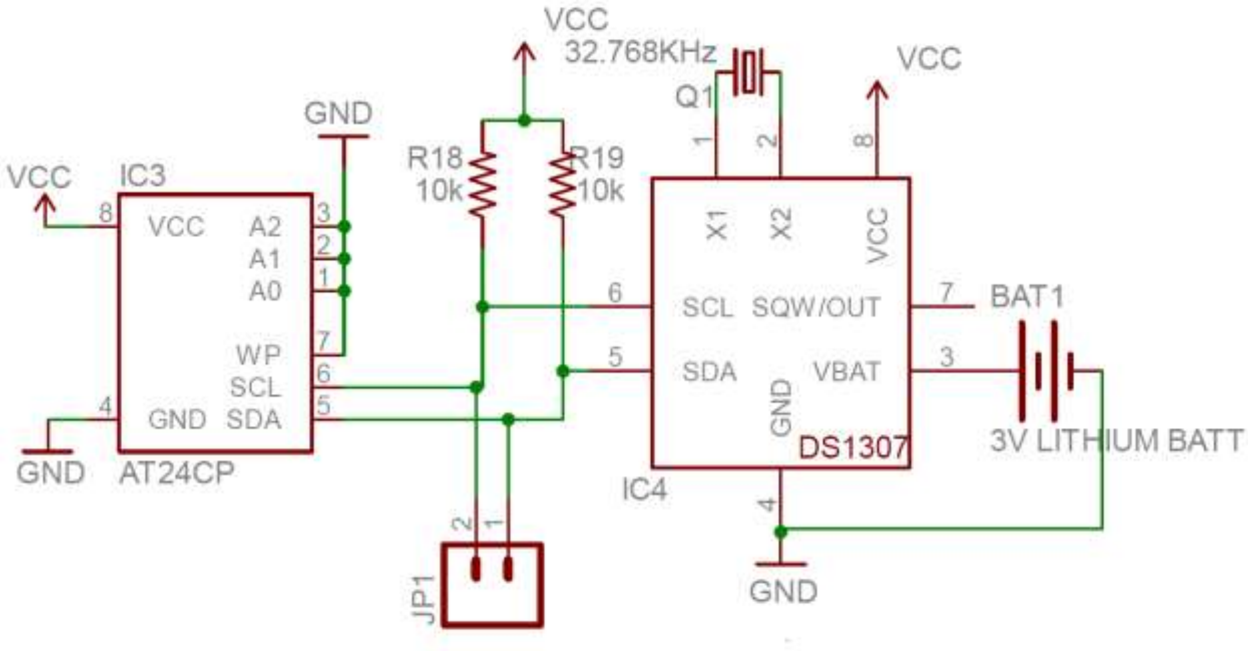
Chapter 14

14.0 RTC and EEPROM Section:

This section consists of IC DS1307 (RTC) and IC 24c32 (EEPROM Memory) connected in TWI or I2C interface. The 2 pin header (49) provides SCL and SDA pin common to this IC. In order to interface these ICs with microcontroller SCL of this 2 pin header needs to be connected to PC0 (SCL) and SDA to PC1 (SDA) of microcontroller.



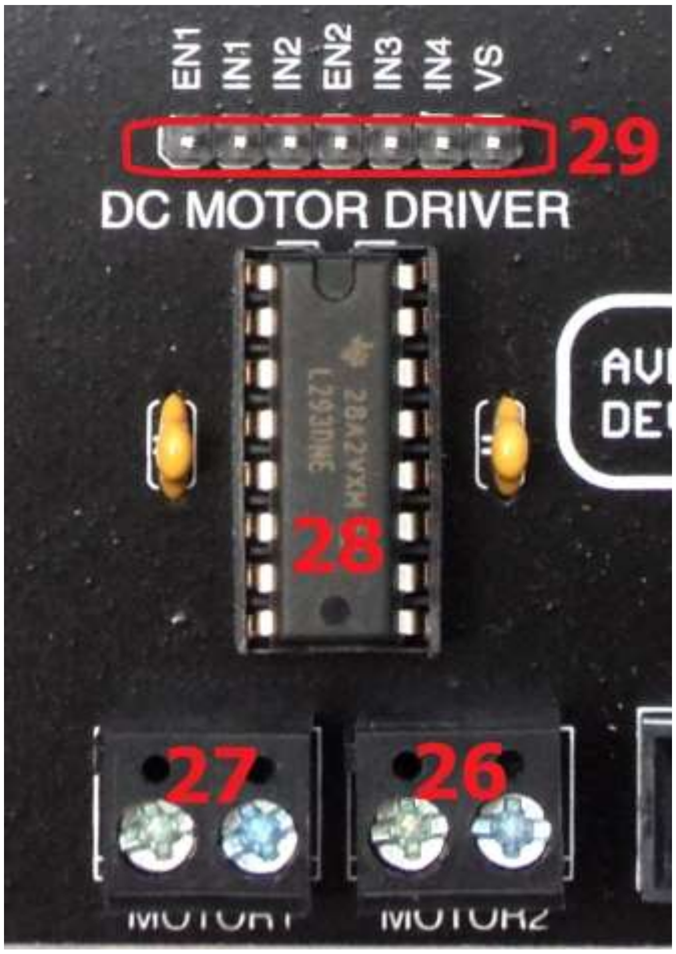
14.1 RTC and EEPROM Section Schematic:



Chapter 15

15.0 DC motor Driver Section:

This section consists of a motor driver IC L293D along with the interfacing header and two 2 pin screw terminal connectors to connect motors.
Two motors can be connected and controlled through this circuit. The header provides you full control over IC. One needs to give proper logics to the pins in order to driving dc motors in clockwise and counterclockwise direction.



EN1 logic 1(+5v) on this pin will enable IN1 and IN2 to control Motor 1

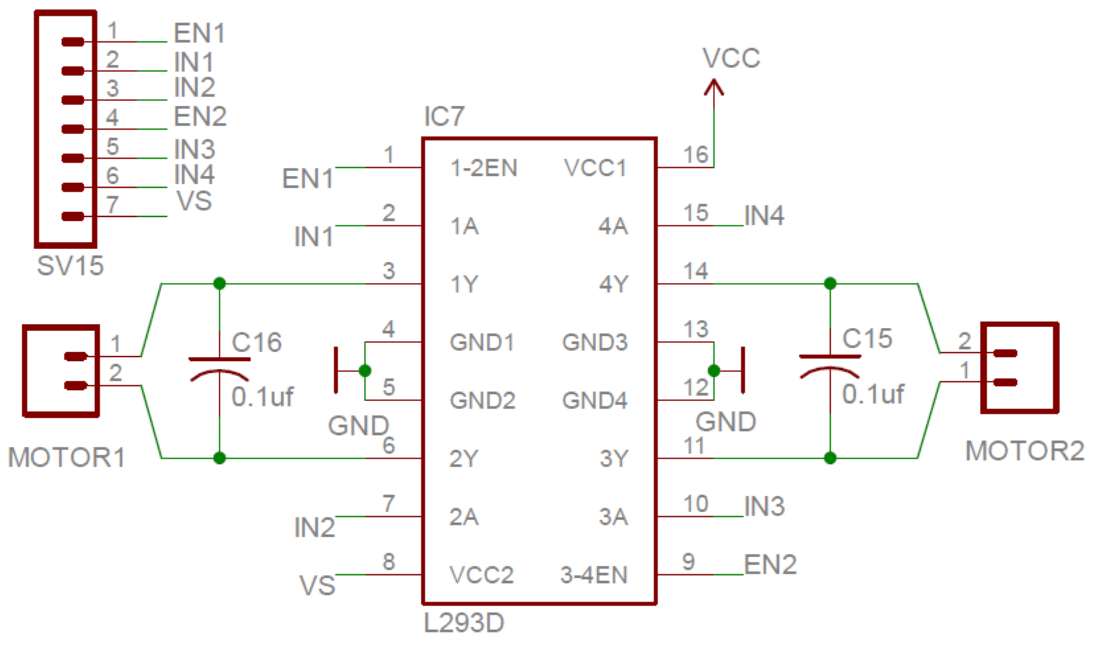
IN1 and IN2 together decides the direction of Motor1

Similarly,
EN2 logic 1(+5v) on this pin will enable IN3 and IN4 to control Motor 2

IN3 and IN4 together decides the direction of Motor2

VS pin gives you flexibility to select the voltage at which motor needs to operate. Board provides power outputs of +5V and +12V, VS can be connected to any one of this power pins.

15.1 DC motor Driver Section Schematic:



Using AVR Studio7 for Programing in C/ Assembly Language

1.1 The AVR Studio 7 Integrated Development Environment (IDE)

Atmel Studio 7 is a software development environment developed by Atmel. It provides single development platform for Atmel's 8- bits, 32-bits and ARM Cortex-M families of AVR microcontrollers. Some Features of Atmel Studio 7:

- It is a full software development environment with an editor, simulator, programmer, etc.
- It comes with its own integrated **open sources C compiler** the "**AVR GNU C Compiler (GCC)"**. As such you do not need a third party C compiler.
- It provides a single environment to develop programs for 8-bits, 32-bits and ARM Cortex-M AVR series of microcontrollers.
- It also integrates fully Q-Touch studio.
- Provides support for several programmers including the STK500, AVR Dragon, JTAG ICE etc.

1.2 To Create a Project for developing assembly language programs

Install AVR Studio 7 before continuing the following steps.

https://www.microchip.com/mplab/avr-support/atmel-studio-7

Step 1:

To create assembly project first start Atmel Studio 7 by going to the start menu on your PC, select Atmel AVR Tools then Atmel Studio 7. See the splash screen for Atmel Studio 7, as shown figure below, this indicates that Atmel Studio 7 is starting up. It will take few seconds as the software is bulky. After Atmel Studio 7 starts the Atmel Studio 7 Start Page will appear as shown in the figure below.

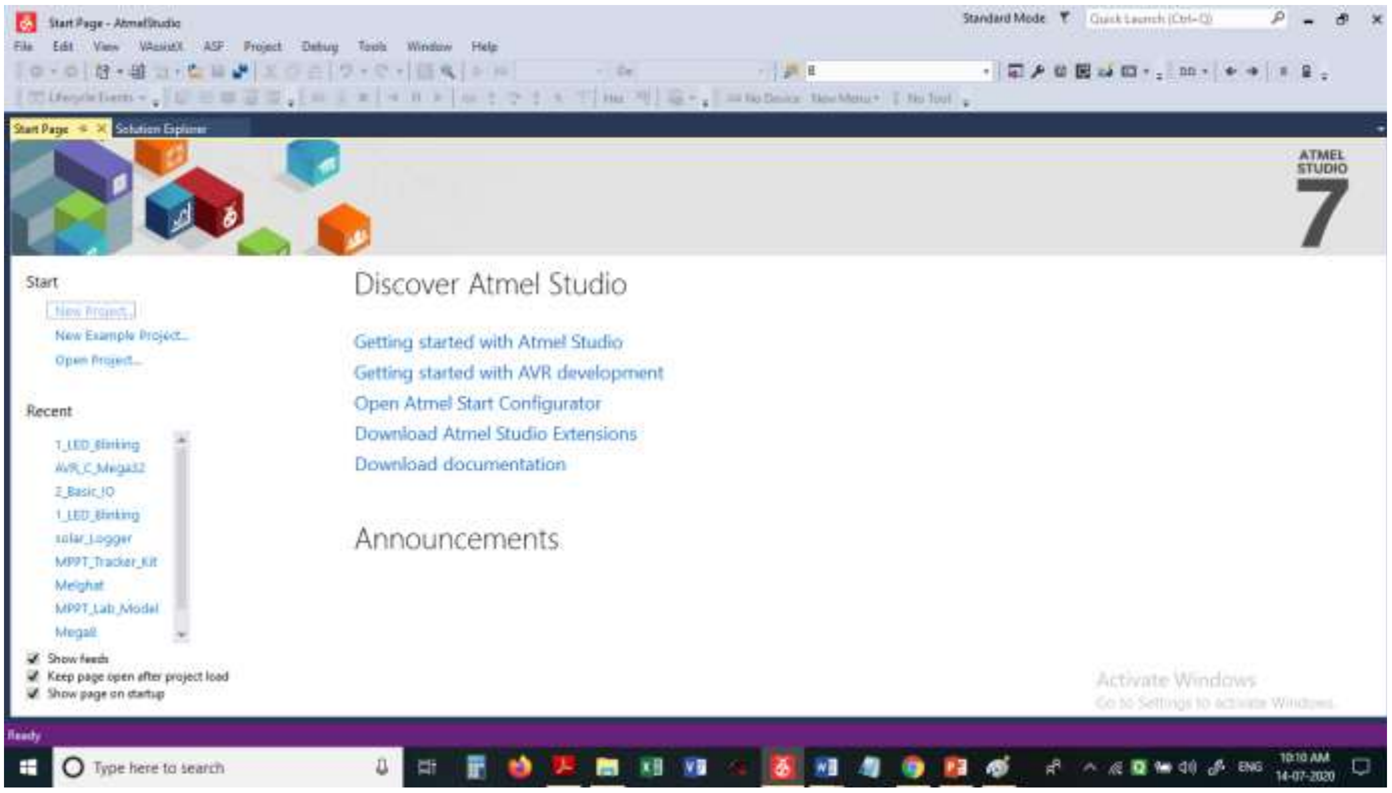
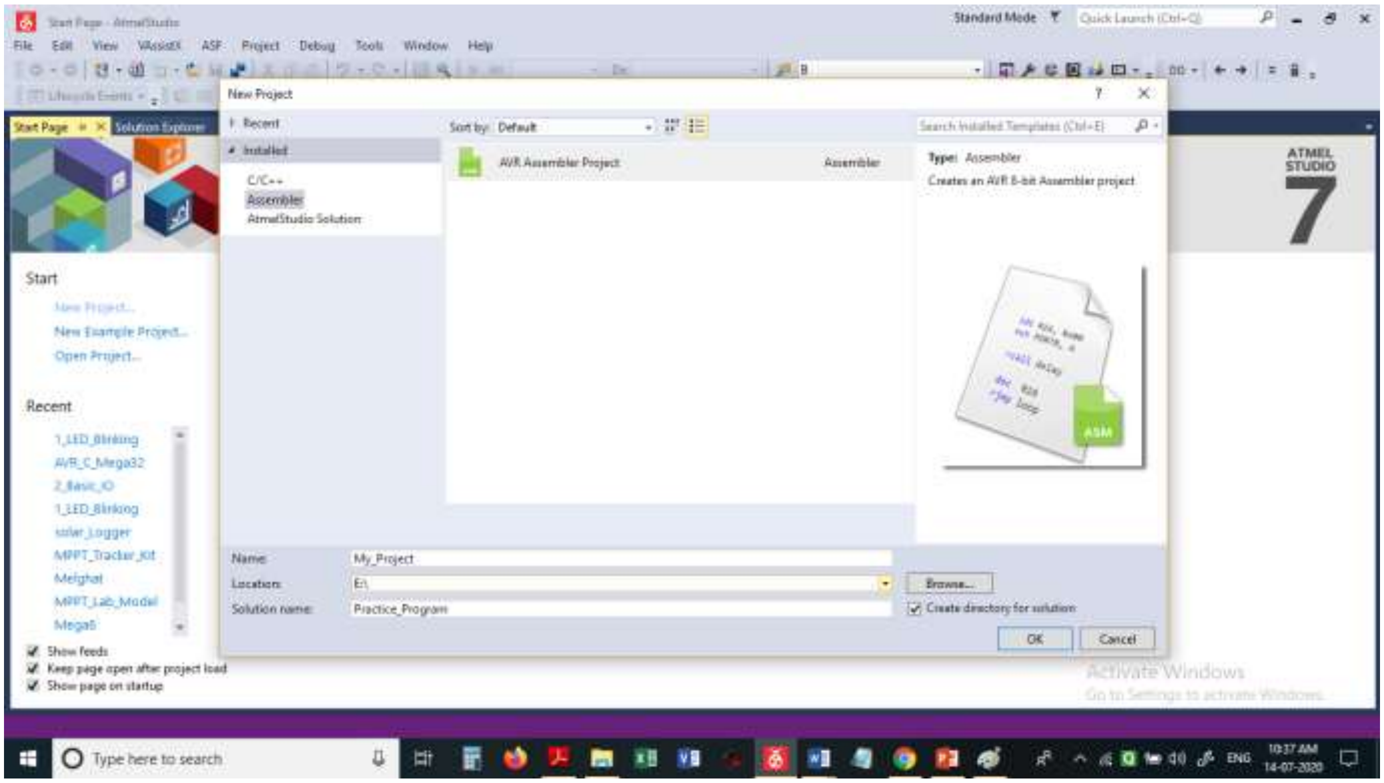


Fig.1: Initial Screen shot

Step:2
Click on New Project pointed to by the blue arrow in the Figure-1.Then select, Assembler, Give project name as “My Project” and Solution Name as “Practice Program”. Following window will appear:



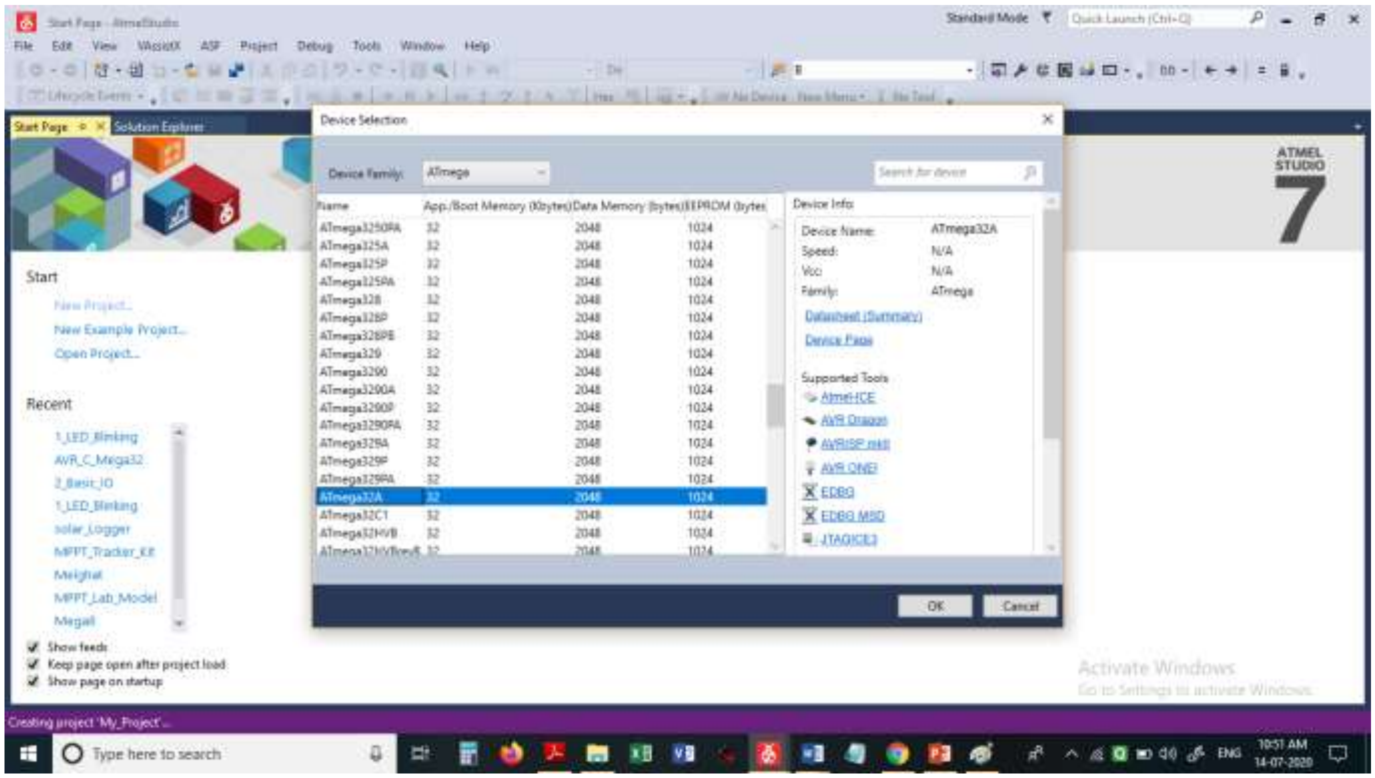
Name: My_Project
Location: E:\
Solution name: Practice_Program

Fig.2: Initial Screen shot

A folder “Practice Program” will be created in the “ E” drive of your PC . This folder will have another folder “My_Project”, where in all your programs will be stored.

Step B4:

The window below is the device selection screen for Atmel Studio 7. Scroll down and select the microcontroller you will be using(ATMEGA32A in for this Lab work).


Device Selection

Device Family: ATmega

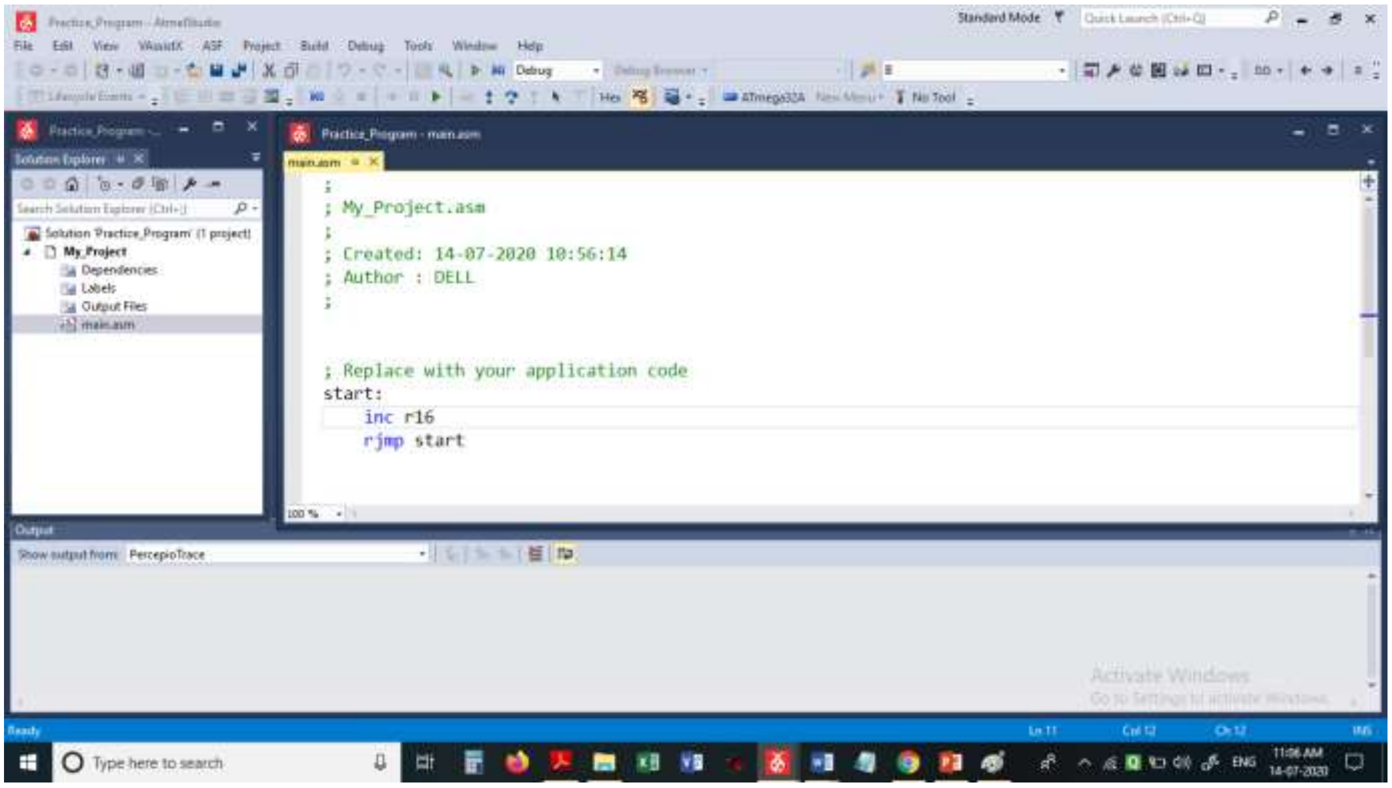
Name	App./Boot Memory (Kbytes)	Data Memory (bytes)	EEPROM (bytes)
ATmega3250PA	32	2048	1024
ATmega325A	32	2048	1024
ATmega325P	32	2048	1024
ATmega325PA	32	2048	1024
ATmega328	32	2048	1024
ATmega328P	32	2048	1024
ATmega328PB	32	2048	1024
ATmega329	32	2048	1024
ATmega3290	32	2048	1024
ATmega3290A	32	2048	1024
ATmega3290P	32	2048	1024
ATmega3290PA	32	2048	1024
ATmega329A	32	2048	1024
ATmega329P	32	2048	1024
ATmega329PA	32	2048	1024
ATmega32A	32	2048	1024
ATmega32C1	32	2048	1024
ATmega32HVB	32	2048	1024
ATmega32HVBrevB	32	2048	1024

Device Info:

Device Name:	ATmega32A
Speed:	N/A
Vcc:	N/A
Family:	ATmega

Fig.3 : Device selection window

Step4:
Press OK. Following screen will appear, where in solution name is "Practice_Program" and Project name is "My Project", with main.asm is a file where you can store your programs.
Click on Window-Float or Dock option and get the suitable window view.


Practice_Program - AtmelStudio
File Edit View VAssistX ASF Project Build Debug Tools Window Help
Solution Explorer
Solution 'Practice_Program' (1 project)
My_Project
Dependencies
Labels
Output Files
main.asm
Output
Show output from: PercepioTrace

```
;
; My_Project.asm
;
; Created: 14-07-2020 10:56:14
; Author : DELL
;


; Replace with your application code
start:
    inc r16
    rjmp start
```

Fig.4 : Editor Window

This is the Atmel Studio 7 editor window, where you can type your assembly program. The editor opens with a pre-defined program structure. Programmer can use the same or overwrite by his own programs.
Creating a new file:
With "Right-Click" on My_Project, Add-→New Item--→Assembler File, you can create a new assemble file. Give the proper name and store it in "My_Project" folder.

Alternately
With Rename option on right-click, you can rename the main.asm file also.

Step 5:
Storing a program using Editor Window:
Type the program given in Figure, on the Atmel Studio 7 editor.
This program has Explanations on the program are given in the comments fields. Do this exercise by typing the program own your own and Please do not cut and paste.

Step 6:
Closing/Opening an Existing Project.
Whenever it need be, Programmer can always edit existing project that has been created, and closed. Also Project can be closed and opened again. Do it as a self-learning exercise.

Debugging:
Debugging is a process of finding logical errors in a program. Also, the debugging can be used to keep the track of flow of a program and monitor changes of Register, memory location or variables after the execution of Instructions.
At the beginner level, debugging can also be used to understand and confirm the operation of instructions, subroutines, statement or functions in a program.
As the status of Registers and Memories can be checked after every instructions, user can identify the part of program where the logic has been not been properly implemented.
The debugging can be done either by using a software called **"SIMULATOR"** or a Hardware called **"EMULATOR"**. The debugging using Simulator software is referred as Simulation while that using Emulator is the Emulation. Simulation speed much less as compared to Emulation.
There is no standard method of debugging. Normally following methods are used for debugging of a program:

- Single Step by Step, using Step Over or Step Into operation.
- Set breakpoint(s) at location where we want to inspect the input or output of the program. Then we run (or simulate) until the program reach thebreakpoint.
- Step Over an instruction or function (or procedure) and inspect the input or output of the program's instruction or function (or procedure).
- Step Into the function (or procedure) and inspect the input or output of the program's function (or procedure).