

MICROPROCESSORS AND MICROCONTROLLERS LAB

Manual



Department of
ELECTRICAL AND ELECTRONICS ENGINEERING

GOKARAJU RANGARAJU
Institute of Engineering and Technology
(Autonomous)



GOKARAJU RANGARAJU
Institute of Engineering and Technology
(Autonomous)

CERTIFICATE

*This is to certify that it is a bonafide record of practical work
done by Mr./Ms. _____,*

Reg No. _____ 1997 in the

***“MICROPROCESSORS AND MICROCONTROLLERS
LABORATORY” in I-Semester of III-year during 20__
to 20__.***

Internal Examiner
Signature

External Examiner
Signature

Head of the Department
Signature



GOKARAJU RANGARAJU

Institute of Engineering and Technology **(Autonomous)**

MICROPROCESSORS and MICROCONTROLLERS LAB

Course Objectives: -

The objective of this course is to provide the student:

- To introduce the basics of microprocessors and its applications.
- To provide in depth knowledge of 8051 Microcontrollers.
- To expertise working with programming.
- To impart the I/O interfacing concepts for developing real time systems.
- To encourage the students in building real time applications.

Course Outcomes: -

Students will be able to:

- Familiarize with the assembly level programming using 8086.
- Judge the difference between Assembly language and Embedded C Programming
- Design circuits for interfacing different modules to microcontrollers.
- Experiment 8051 with different types of communicating devices.
- Execute various programs which can resemble to the real time applications.



GOKARAJU RANGARAJU

Institute of Engineering and Technology **(Autonomous)**

MICROPROCESSORS and MICROCONTROLLERS LAB

List of Experiments

Task-1: Using 8086 Processor Kits and/or Assembler

Assembly Language Programs to 8086 to Perform

- Arithmetic, Logical, String Operations on 16 Bit and 32-Bit Data.
- Bit level Logical Operations, Rotate, Shift, Swap and Branch Operations.

Task-2: Using 8051 Microcontroller Kit

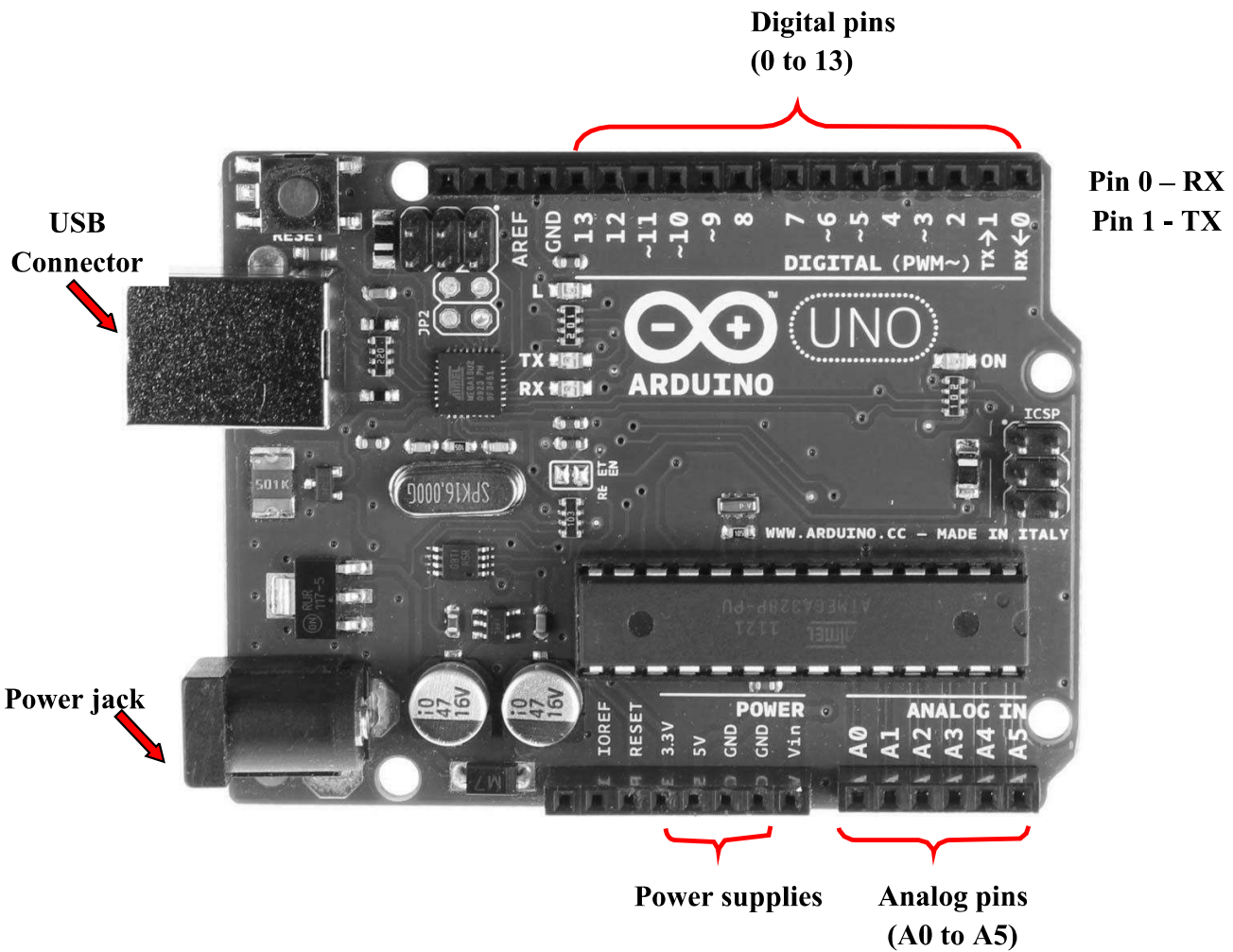
- LED's to 8051.
- Interfacing LCD to 8051.
- Interfacing Matrix Keypad to 8051.
- Interfacing DC Motor to 8051.

Task-3: Arduino Programming

- LEDs interfacing
- Switches and LED's interfacing
- 2*16 LCD
- Serial Communication
- Device control
- Reading sensors using ADC
- DC Motor control

Introduction to Arduino

The lab will be based on the Arduino Uno. The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.



Features of Uno board

| | |
|------------------------------------|---|
| Microcontroller | ATmega328 |
| Operating Voltage | 5V |
| Input Voltage (recommended) | 7-12V |
| Input Voltage (limits) | 6-20V |
| Digital I/O Pins | 14 (of which 6 provide PWM output) |
| Analog Input Pins | 6 |
| DC Current per I/O Pin | 40 mA |
| DC Current for 3.3V Pin | 50 mA |
| Flash Memory | 32 KB (ATmega328) of which 0.5 KB used by boot loader |
| SRAM | 2 KB (ATmega328) |
| EEPROM | 1 KB (ATmega328) |
| Clock Speed | 16 MHz |

Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V).
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Each of the 14 digital pins (pins 0 to 13) on the Uno can be used as an input or output, using [pinMode\(\)](#), [digitalWrite\(\)](#), and [digitalRead\(\)](#) functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) . In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the [analogWrite\(\)](#) function.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication using the [SPI library](#).
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though it is possible to change the upper end of their range using the AREF pin and the [analogReference\(\)](#) function.

Additionally, some pins have specialized functionality:

- **TWI: A4 or SDA pin and A5 or SCL pin.** Support TWI communication using the [Wire library](#).


ATmega168/328-Arduino Pin Mapping

Atmega168 Pin Mapping

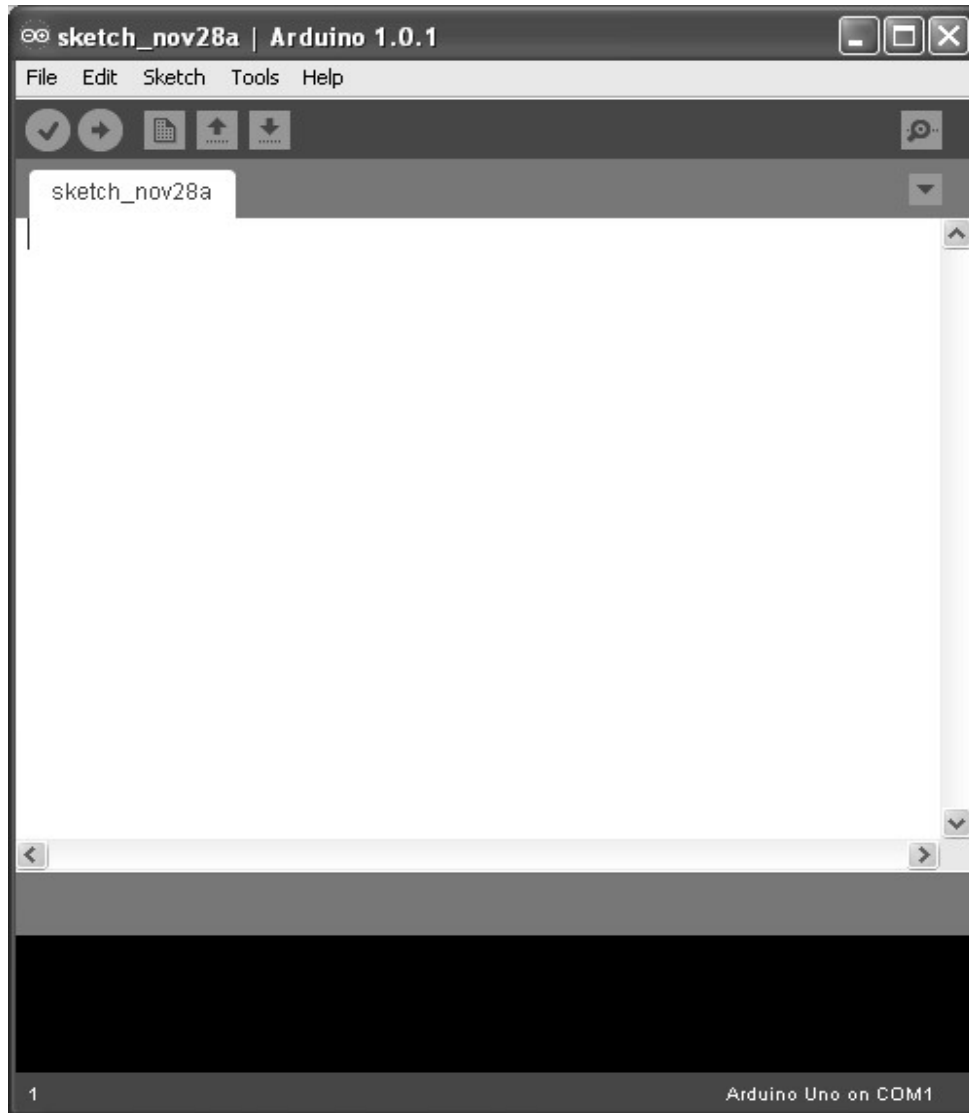
| Arduino function | | | | | Arduino function |
|---------------------|--------------------------|----|----|------------------------|----------------------|
| reset | (PCINT14/RESET) PC6 | 1 | 28 | PC5 (ADC5/SCL/PCINT13) | analog input 5 |
| digital pin 0 (RX) | (PCINT16/RXD) PD0 | 2 | 27 | PC4 (ADC4/SDA/PCINT12) | analog input 4 |
| digital pin 1 (TX) | (PCINT17/TXD) PD1 | 3 | 26 | PC3 (ADC3/PCINT11) | analog input 3 |
| digital pin 2 | (PCINT18/INT0) PD2 | 4 | 25 | PC2 (ADC2/PCINT10) | analog input 2 |
| digital pin 3 (PWM) | (PCINT19/OC2B/INT1) PD3 | 5 | 24 | PC1 (ADC1/PCINT9) | analog input 1 |
| digital pin 4 | (PCINT20/XCK/T0) PD4 | 6 | 23 | PC0 (ADC0/PCINT8) | analog input 0 |
| VCC | VCC | 7 | 22 | GND | GND |
| GND | GND | 8 | 21 | AREF | analog reference |
| crystal | (PCINT6/XTAL1/TOSC1) PB6 | 9 | 20 | AVCC | VCC |
| crystal | (PCINT7/XTAL2/TOSC2) PB7 | 10 | 19 | PB5 (SCK/PCINT5) | digital pin 13 |
| digital pin 5 (PWM) | (PCINT21/OC0B/T1) PD5 | 11 | 18 | PB4 (MISO/PCINT4) | digital pin 12 |
| digital pin 6 (PWM) | (PCINT22/OC0A/AIN0) PD6 | 12 | 17 | PB3 (MOSI/OC2A/PCINT3) | digital pin 11 (PWM) |
| digital pin 7 | (PCINT23/AIN1) PD7 | 13 | 16 | PB2 (SS/OC1B/PCINT2) | digital pin 10 (PWM) |
| digital pin 8 | (PCINT0/CLKO/ICP1) PB0 | 14 | 15 | PB1 (OC1A/PCINT1) | digital pin 9 (PWM) |

Digital Pins 11, 12 & 13 are used by the ICSP header for MISO, MOSI, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

Arduino Programming

Click on the Arduino executable which has the Arduino logo 

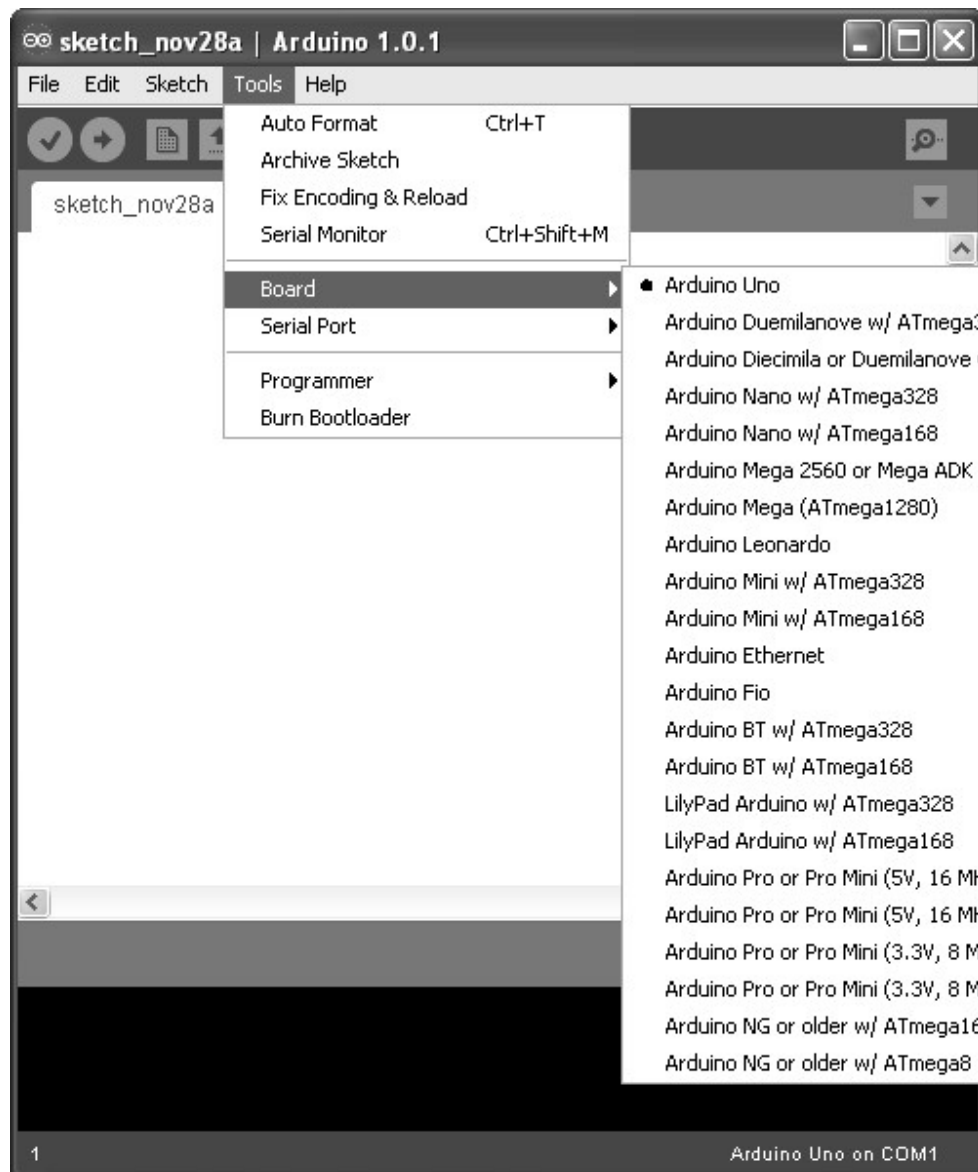
The following screen comes up:



The programs written for Arduino are called sketches. For the sketch to work on the Arduino Uno, there are two hardware related settings need to be done in the Arduino IDE –

- Board
- Serial Port

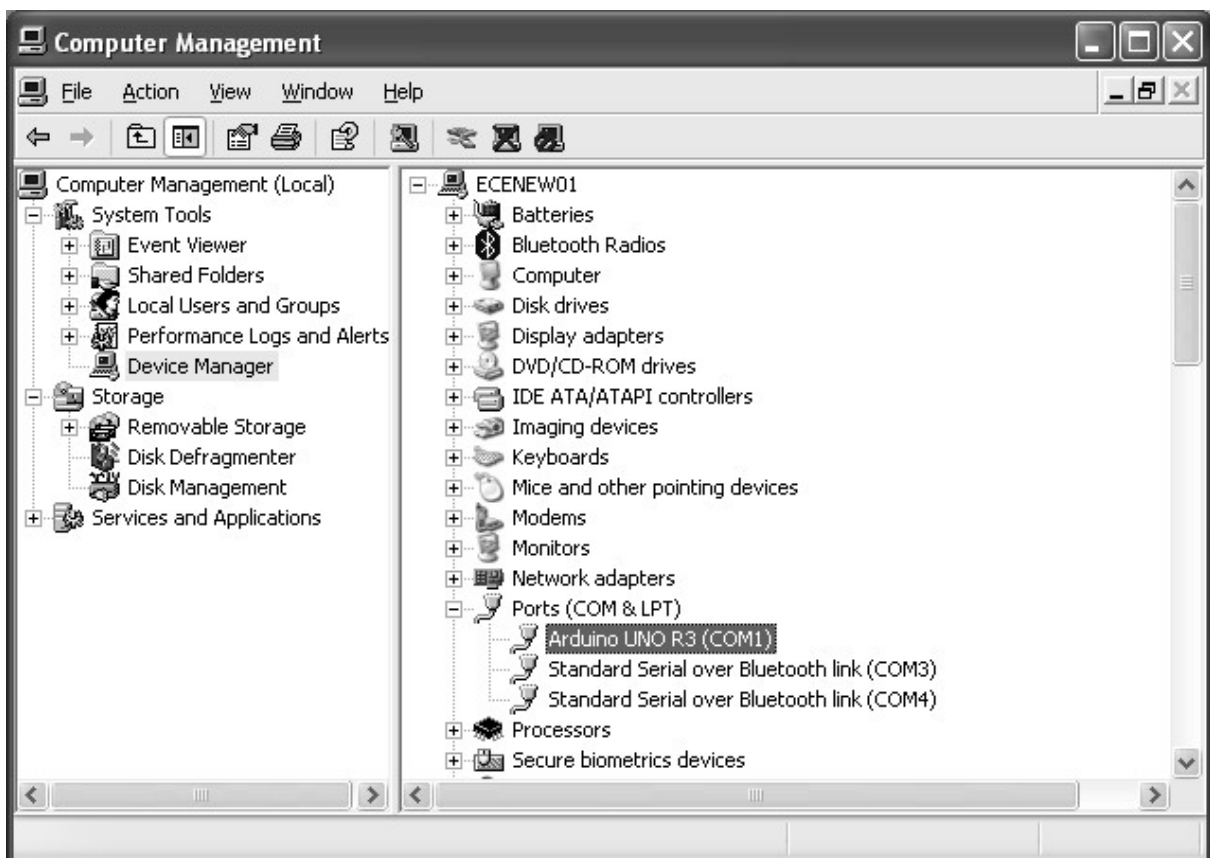
For selecting the board, go to the Tools tab and select Board. From the menu select Uno.



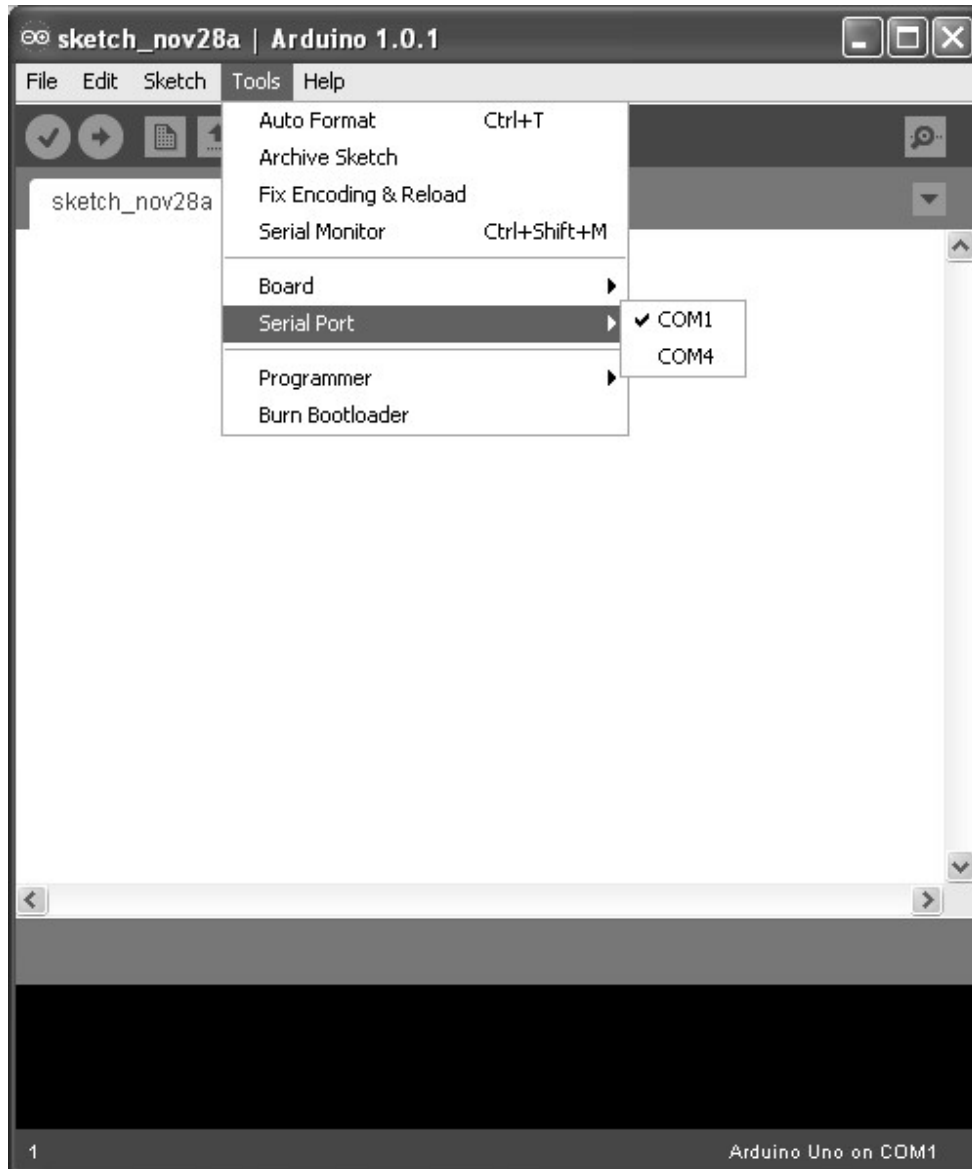
When you connect your Arduino Uno to the USB port of your laptop, it will be mapped as a serial port.

To know the serial port to which your Arduino is mapped, follow the following procedure:

- ▶ Right click on My Computer
- ▶ Select the Manage option
- ▶ In the pop up screen for Computer Management, select the Device Manager
- ▶ Expand the Ports item; the Arduino Uno will appear as one of the drop down items



In the Arduino IDE, select the Serial Port as the port to which the Arduino is mapped.



The basic structure of the Arduino sketch is fairly simple and has two required functions:

```
void setup()
{
    statements;
}
void loop()
{
    statements;
}
```

Where `setup()` is the preparation, `loop()` is the execution. Both functions are required for the program to work. The setup function should follow the declaration of any variables at the very beginning of the program. It is the first function to run in the program, is run only once, and is used to set pin Mode or initialize serial communication.

The loop function follows next and includes the code to be executed continuously – reading inputs, triggering outputs, etc. This function is the core of all Arduino programs and does the bulk of the work.

setup()

The `setup()` function is called once when your program starts. Use it to initialize pin modes, or begin serial. It must be included in a program even if there are no statements to run.

```
void setup()
{
    pinMode(pin, OUTPUT); // sets the 'pin' as output
}
```

loop()

After calling the `setup()` function, the `loop()` function does precisely what its name suggests, and loops consecutively, allowing the program to change, respond, and control the Arduino board.

```

void loop()
{
    digitalWrite(pin, HIGH); // turns 'pin' on
    delay(1000); // pauses for one second
    digitalWrite(pin, LOW); // turns 'pin' off
    delay(1000); // pauses for one second
}

```

pinMode(pin, mode)

Used in void setup() to configure a specified pin to behave either as an INPUT or an OUTPUT.

```
pinMode(pin, OUTPUT); // sets 'pin' to output
```

There are also convenient pullup resistors built into the Atmega chip that can be accessed from software. These built-in pullup resistors are accessed in the following manner:

```
pinMode(pin, INPUT); // set 'pin' to input
digitalWrite(pin, HIGH); // turn on pullup resistors
```

Pullup resistors would normally be used for connecting inputs like switches. Notice in the above example it does not convert pin to an output, it is merely a method for activating the internal pull-ups.

Pins configured as OUTPUT can provide 40 mA (milliamps) of current to other devices/circuits. This is enough current to brightly light up an LED (don't forget the series resistor), but not enough current to run most relays, solenoids, or motors.

Short circuits on Arduino pins and excessive current can damage or destroy the output pin, or damage the entire Atmega chip. It is often a good idea to connect an OUTPUT pin to an external device in series with a 470 Ω or 1K Ω resistor.

digitalRead(pin)

Reads the value from a specified digital pin with the result either HIGH or LOW. The pin can be specified as either a variable or constant (0-13).

```
value = digitalRead(Pin); // sets 'value' equal to the input pin
```

digitalWrite(pin, value)

Outputs either logic level HIGH or LOW at (turns on or off) a specified digital pin. The pin can be specified as either a variable or constant (0-13).

```
digitalWrite(pin, HIGH); // sets 'pin' to high
```

The following example reads a pushbutton connected to a digital input and turns on an LED connected to a digital output when the button has been pressed:

```
int led = 13; // connect LED to pin 13
int pin = 7; // connect pushbutton to pin 7
int value = 0; // variable to store the read value
void setup()
{
    pinMode(led, OUTPUT); // sets pin 13 as output
    pinMode(pin, INPUT); // sets pin 7 as input
}
void loop()
{
    value = digitalRead(pin); // sets 'value' equal to the input pin
    digitalWrite(led, value); // sets 'led' to the button's value
}
```

analogRead(pin)

Reads the value from a specified analog pin with a 10-bit resolution. This function only works on the analog in pins (0-5). The resulting integer values range from 0 to 1023.

```
value = analogRead(pin); // sets 'value' equal to 'pin'
```

Note: Analog pins unlike digital ones, do not need to be first declared as INPUT or OUTPUT.

analogWrite(pin, value)

Writes a pseudo-analog value using hardware enabled pulse width modulation (PWM) to an output pin marked PWM. On Uno, this function works on pins 3, 5, 6, 9, 10, and 11. The value can be specified as a variable or constant with a value from 0-255.

```
analogWrite(pin, value); // writes 'value' to analog 'pin'
```

A value of 0 generates a steady 0 volts output at the specified pin; a value of 255 generates a steady 5 volts output at the specified pin. For values in between 0 and 255, the pin rapidly alternates between 0 and 5 volts - the higher the value, the more often the pin is HIGH (5 volts). For example, a value of 64 will be 0 volts three-quarters of the time, and 5 volts one quarter of the time; a value of 128 will be at 0 half the time and 255 half the time; and a value of 192 will be 0 volts one quarter of the time and 5 volts three-quarters of the time.

Because this is a hardware function, the pin will generate a steady wave after a call to analogWrite in the background until the next call to analogWrite (or a call to digitalRead or digitalWrite on the same pin).

Note: Analog pins unlike digital ones do not need to be first declared as INPUT or OUTPUT.

The following example reads an analog value from an analog input pin, converts the value by dividing by 4, and outputs a PWM signal on a PWM pin:

```
int led = 10; // LED with 220 resistor on pin 10
int pin = A0; // potentiometer on analog pin 0
int value; // value for reading
void setup() {} // no setup needed
void loop()
{
  value = analogRead(pin); // sets 'value' equal to 'pin'
  value /= 4; // converts 0-1023 to 0-255
  analogWrite(led, value); // outputs PWM signal to led
}
```

delay(ms)

Pauses a program for the amount of time as specified in milliseconds, where 1000 equals 1 second.

```
delay(1000); // waits for one second
```

millis()

Returns the number of milliseconds since the Arduino board began running the current program as an unsigned long value.

```
value = millis(); // sets 'value' equal to millis()
```

Note: This number will overflow (reset back to zero), after approximately 9 hours.

Serial.begin(rate)

Opens serial port and sets the baud rate for serial data transmission. The typical baud rate for communicating with the computer is 9600 although other speeds are supported.

```
void setup()
{
  Serial.begin(9600); // opens serial port
} // sets data rate to 9600 bps
```

Note: When using serial communication, digital pins 0 (RX) and 1 (TX) cannot be used at the same time.

Serial.println(data)

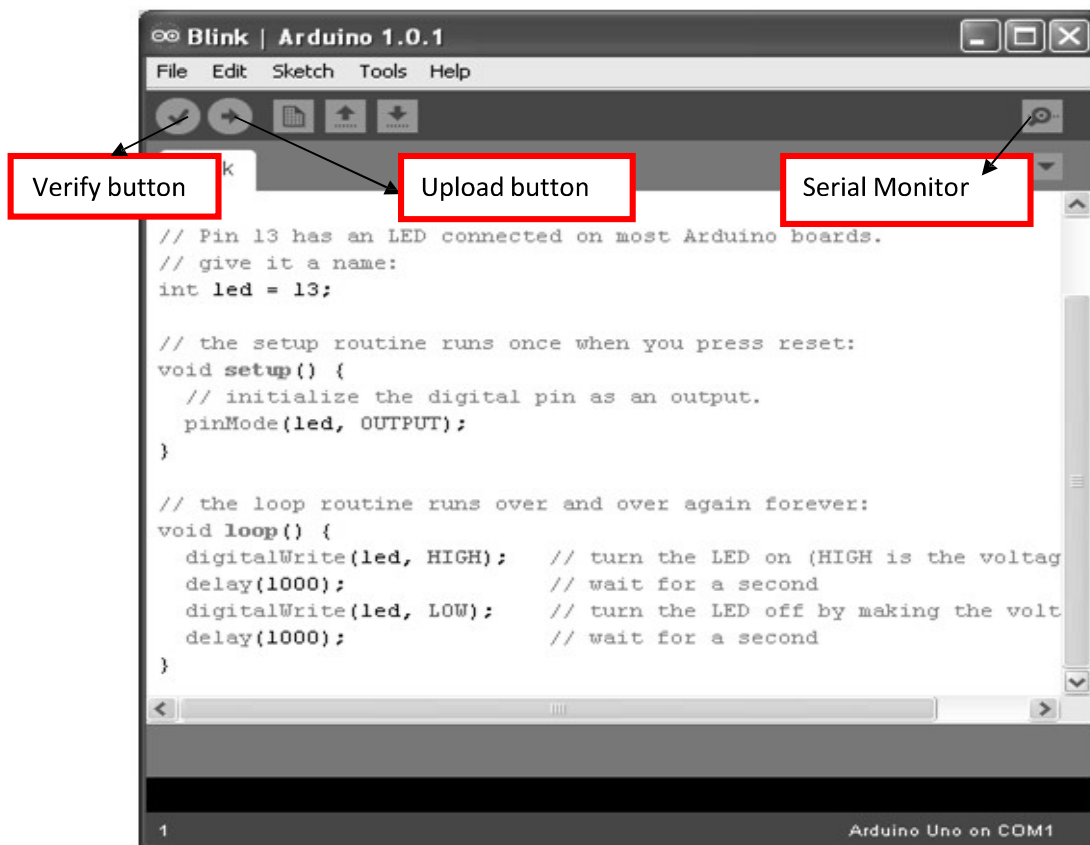
Prints data to the serial port, followed by an automatic carriage return and line feed. This command takes the same form as Serial.print(), but is easier for reading data on the Serial Monitor.

```
Serial.println(analogValue); // sends the value of
                               // 'analogValue'
```

Note: For more information on the various permutations of the Serial.println() and Serial.print() functions please refer to the Arduino website.

The following simple example takes a reading from analog pin0 and sends this data to the computer every 1 second.

```
void setup()
{
  Serial.begin(9600); // sets serial to 9600bps
}
void loop()
{
  Serial.println(analogRead(A0)); // sends analog value
  delay(1000); // pauses for 1 second
}
```



After entering your program, click on the Verify button for compilation. If there are errors, the line numbers of the errors are shown in the bottom window. Correct the errors. After successful verification, upload your program to the Arduino using the Upload button. A common cause for failure in uploading is that your Arduino is not connected to a different COM port than the one shown in the Arduino IDE.

LEDs and Switches

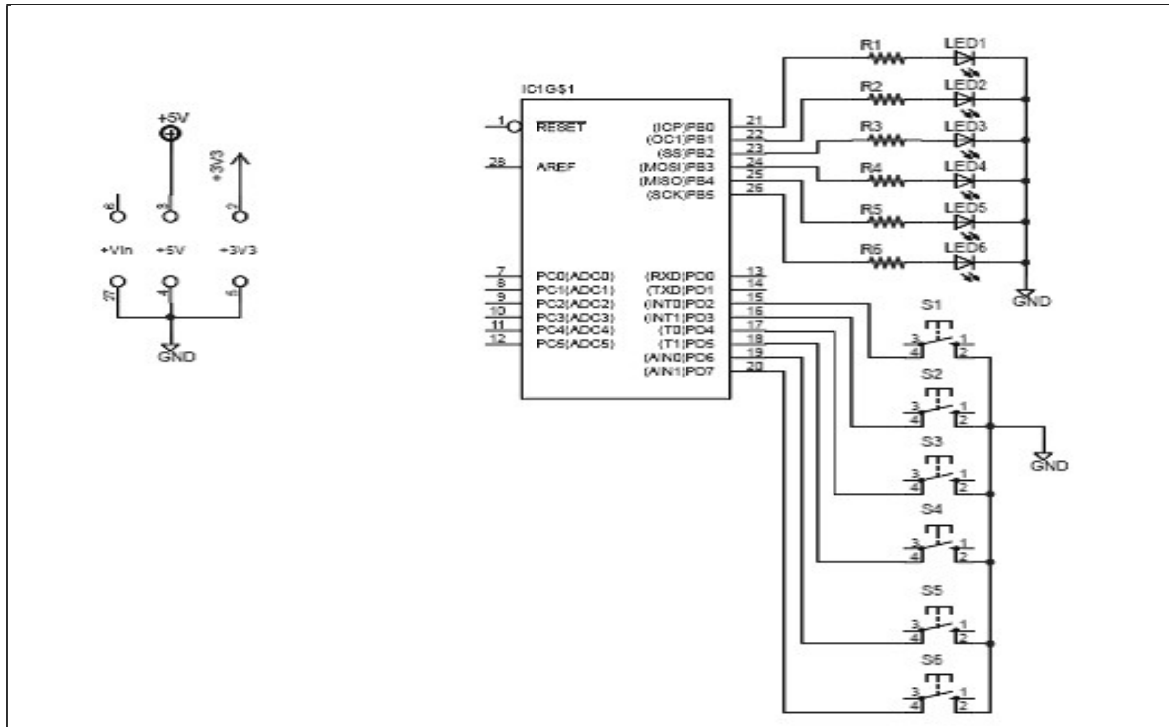
Introduction

The GRIET LED and switches shield has 6 LEDs and 6 switches. The cathodes of the LEDs are grounded and the anodes are connected to Arduino Uno pins through resistors. The switches when pressed will ground the connected Uno pins. The Uno pins connected to the switches should be made inputs with pull-ups enabled. The required 5V supply and ground for the shield are transferred from the Arduino Uno base board

Pin-out

| <i>LEDs and switches</i> | <i>Arduino Uno pins</i> | <i>ATmega328 pin</i> |
|--------------------------|-------------------------|----------------------|
| LED1 | 8 | PB0 |
| LED2 | 9 | PB1 |
| LED3 | 10 | PB2 |
| LED4 | 11 | PB3 |
| LED5 | 12 | PB4 |
| LED6 | 13 | PB5 |
| S1 | 2 | PD2 |
| S2 | 3 | PD3 |
| S3 | 4 | PD4 |
| S4 | 5 | PD5 |
| S5 | 6 | PD6 |
| S6 | 7 | PD7 |

Schematic



Bill of Materials

| Part | Value | Device | Package | Library | Sheet |
|-------------|----------------|--------------------|----------------|----------------|--------------|
| IC1 | Arduino Shield | ARDUINO2009_PIN_22 | ARDUINO2009_2 | arduino01 | 1 |
| LED1 | | LEDCHIPLED_1206 | CHIPLED_1206 | led | 1 |
| LED2 | | LEDCHIPLED_1206 | CHIPLED_1206 | led | 1 |
| LED3 | | LEDCHIPLED_1206 | CHIPLED_1206 | led | 1 |
| LED4 | | LEDCHIPLED_1206 | CHIPLED_1206 | led | 1 |
| LED5 | | LEDCHIPLED_1206 | CHIPLED_1206 | led | 1 |
| LED6 | | LEDCHIPLED_1206 | CHIPLED_1206 | led | 1 |
| R1 | 330R | R-US_R1206 | R1206 | rcl | 1 |
| R2 | 330R | R-US_R1206 | R1206 | rcl | 1 |
| R3 | 330R | R-US_R1206 | R1206 | rcl | 1 |
| R4 | 330R | R-US_R1206 | R1206 | rcl | 1 |
| R5 | 330R | R-US_R1206 | R1206 | rcl | 1 |
| R6 | 330R | R-US_R1206 | R1206 | rcl | 1 |
| S1 | 10XX Tactile | 10-XX | B3F-10XX | switch-omron | 1 |
| S1 | 10XX Tactile | 10-XX | B3F-10XX | switch-omron | 1 |
| S1 | 10XX Tactile | 10-XX | B3F-10XX | switch-omron | 1 |
| S1 | 10XX Tactile | 10-XX | B3F-10XX | switch-omron | 1 |
| S1 | 10XX Tactile | 10-XX | B3F-10XX | switch-omron | 1 |
| S1 | 10XX Tactile | 10-XX | B3F-10XX | switch-omron | 1 |

Arduino program

The program will read the status of one switch and use it to control an LED – LED is on when the switch is pressed and off when the switch is not pressed. LED1 and SW1 are used in the program

```
int LED1 = 8;
int S1 = 2;
void setup()
{
    pinMode(LED1,OUTPUT);
    pinMode(S1,INPUT_PULLUP); }

void loop()
{
    int status;
    status = digitalRead(S1);
    if(status == LOW)
        digitalWrite(LED1,HIGH);
    else
        digitalWrite(LED1,LOW);
}
```

Exercises

1. Write an Arduino program to control LED6 using switch SW6
2. Write an Arduino program to control the six LEDs using the six corresponding switches

LCD

Introduction

The GRIET LCD shield has the following resources

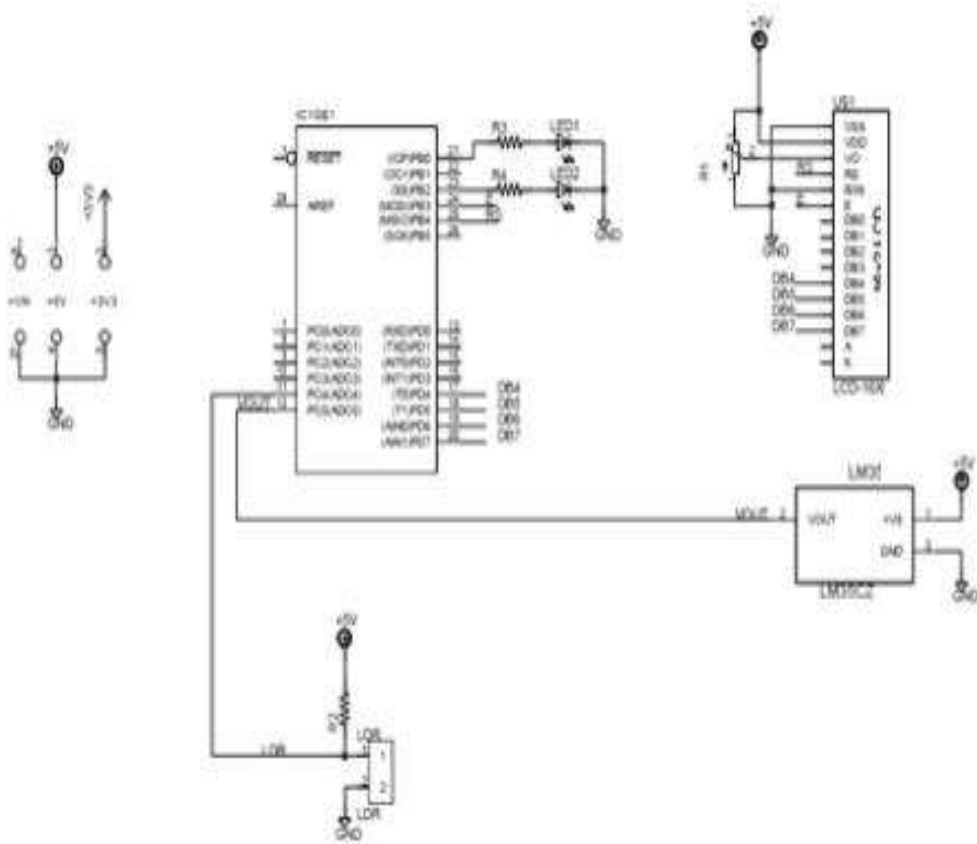
- 2x16 LCD
- LM35 temperature sensor
- LDR(Light Dependent Resistor)
- 2 LEDs

The 2x16 LCD uses the 4-bit interface. The RD/WR pin of the LCD is grounded so that write is permanently enabled. There is a potentiometer for adjusting the contrast. Adjust the pot till you see a strip of dark blocks in the first line of the LCD. The LM35 is connected to the A5 analog input pin of Uno. The LDR forms part of a potential divider circuit whose output is given to A4 analog input pin of Uno.

Pin-out

| <i>LCD, sensors & LEDs</i> | <i>Arduino Uno pins</i> | <i>ATmega328 pin</i> |
|--------------------------------|-------------------------|----------------------|
| LCD Enable | 11 | PB3 |
| LCD Register Select(RS) | 12 | PB4 |
| DB4 | 4 | PD4 |
| DB5 | 5 | PD5 |
| DB6 | 6 | PD6 |
| DB7 | 7 | PD7 |
| LM35 | A5 | PC5 |
| LDR | A4 | PC4 |
| LED1 | 8 | PB0 |
| LED2 | 10 | PB2 |

Schematic



Bill of Materials

| Part | Value | Device | Package | Library | Sheet |
|------|----------------|--------------------|------------------|-----------|-------|
| IC1 | Arduino Shield | ARDUINO2009_PIN_22 | ARDUINO2 09_2 | arduino01 | 1 |
| LCD | LCD-16X2 | LCD-16X2 | LCD-16X2 | SparkFun | 1 |
| LDR | LDR | LDR | LDR | BHolder | 1 |
| LED1 | | LEDCHIPLED 1206 | CHIPLED 1206 | led | 1 |
| LED2 | | LEDCHIPLED_1206 | CHIPLED_1206 | led | 1 |
| LM35 | LM35CZ | LM35CZ | TO127P254X495-3P | LM35 | 1 |
| R1 | 5K | R-TRIMM3296 | RTRIM3296W | rcl | 1 |
| R2 | | R-US 0204/7 | 0204/7 | rcl | 1 |
| R3 | 330R | R-US_R1204/7 | R1204/7 | rcl | 1 |
| R4 | 330R | R-US R1204/7 | R1204/7 | rcl | 1 |

Arduino program

The program is to write “Hello World” in the first line of the LCD. We use the LiquidCrystal.h library in the program.

```
#include <LiquidCrystal.h>

/* RS 12
   E 11
   D4 4
   D5 5
   D6 6
   D7 7 */
LiquidCrystal lcd(12,11,4,5,6,7);

void setup()
{
    lcd.begin(16,2);
    lcd.print(" hello, world");
}

void loop()
{
}
```

LCD library functions**LiquidCrystal()****Description**

Creates a variable of type LiquidCrystal. The display can be controlled using 4 or 8 data lines. If the former, omit the pin numbers for d0 to d3 and leave those lines unconnected. The RW pin can be tied to ground instead of connected to a pin on the Arduino; if so, omit it from this function's parameters.

Syntax

LiquidCrystal(rs, enable, d4, d5, d6, d7) LiquidCrystal(rs,
rw, enable, d4, d5, d6, d7) LiquidCrystal(rs, enable, d0, d1,
d2, d3, d4, d5, d6, d7) LiquidCrystal(rs, rw, enable, d0, d1,
d2, d3, d4, d5, d6, d7)

Parameters

rs: the number of the Arduino pin that is connected to the RS pin on the LCD

rw: the number of the Arduino pin that is connected to the RW pin on the LCD (*optional*)

enable: the number of the Arduino pin that is connected to the enable pin on the LCD

d0, d1, d2, d3, d4, d5, d6, d7: the numbers of the Arduino pins that are connected to the corresponding data pins on the LCD. d0, d1, d2, and d3 are optional; if omitted, the LCD will be controlled using only the four data lines (d4, d5, d6, d7).

begin()

Description

Specifies the dimensions (width and height) of the display.

Syntax

```
lcd.begin(cols, rows)
```

Parameters

lcd: a variable of type LiquidCrystal

cols: the number of columns that the display has

rows: the number of rows that the display has

setCursor()

Description

Position the LCD cursor; that is, set the location at which subsequent text written to the LCD will be displayed.

Syntax

```
lcd.setCursor(col, row)
```

Parameters

lcd: a variable of type LiquidCrystal

col: the column at which to position the cursor (with 0 being the first column)

row: the row at which to position the cursor (with 0 being the first row)

Exercise

1. Write an Arduino program to write your name in the first line of the LCD
2. Write an Arduino program to write your name in the first line of the LCD and your roll number in the second line

Serial communication

Introduction

The Arduino Uno board is capable of serial communication used for communication between the Arduino board and a computer or other devices. The Uno has a single serial port (also known as a UART or USART): **Serial**. It communicates on digital *pins* 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use *pins* 0 and 1 for digital input or output.

You can use the Arduino environment's built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to `begin()`.

Functions

`begin()`

Description

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200.

Syntax

```
Serial.begin(speed)
```

`println()`

Description

Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or `\r`) and a newline character (ASCII 10, or `\n`). This command takes the same forms as `Serial.print()`.

Syntax

```
Serial.println(val)
Serial.println(val, format)
```

Parameters

val: the value to print - any data type

format: specifies the number base (for integral data types) or number of decimal places (for floating point types)

Returns

byte

println() will return the number of bytes written, though reading that number is optional

Example:

```
void setup() {
  // open the serial port at 9600 bps
  Serial.begin(9600);
}
void loop() {
  println("Hello World");
  delay(1000);
}
```

Exercises

1. Write an Arduino program to write your name
2. Write an Arduino program to write your name in the first line and your roll number in the second line
3. Write an echo program to echo whatever is sent from the computer back to the computer

Reading sensors using internal ADC

Introduction

The GRIET LCD shield which has a temperature sensor and an ambient light sensor is used in this experiment. The temperature sensor is connected to pin A5 and the ambient light sensor is connected to pin A4 of Uno. To use the internal ADC, the function `AnalogRead()` is used.

Functions

`analogRead()`

Description

Reads the value from the specified analog pin. The Arduino Uno board contains a 6 channel, 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit.

It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

Syntax

```
analogRead(pin)
```

Parameters

pin: the number of the analog input pin to read from (A0 to A5 on Uno)

Returns

```
int (0 to 1023)
```


Arduino program

```
int val = 0;      // variable to store the value read

void setup()
{
  Serial.begin(9600);    // setup serial
}

void loop()
{
  val = analogRead(A5); // read the input pin
  Serial.println(val);   // debug value
}
```

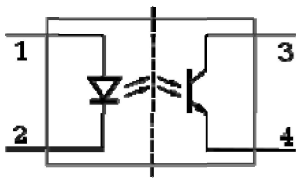
Exercise

1. Write an Arduino program to read the LM35 sensor, convert the value into temperature and display it on the serial monitor
2. Write an Arduino program to read the LM35 sensor, convert the value into temperature and display it on the LCD
3. Write an Arduino program to read the LDR sensor and display it on the serial monitor. Switch on LED1 when light is high and switch off the LED when light is low

Device Control

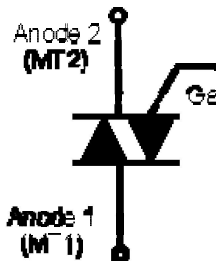
Introduction

The GRIET Triac shield is used for device control. Two devices can be controlled by the Triac shield. Each channel will have an Arduino pin connected to an opto-isolator and a triac. The main purpose of an opto-isolator is to prevent high voltages on one side of the circuit from damaging components on the other side.



Schematic diagram of an opto-isolator showing source of light (LED) on the left, dielectric barrier in the center, and sensor (phototransistor) on the right.

The triac is a three-terminal semiconductor device for controlling current. It is an ideal device to use for AC switching applications because it can control the current flow over both halves of an alternating cycle.

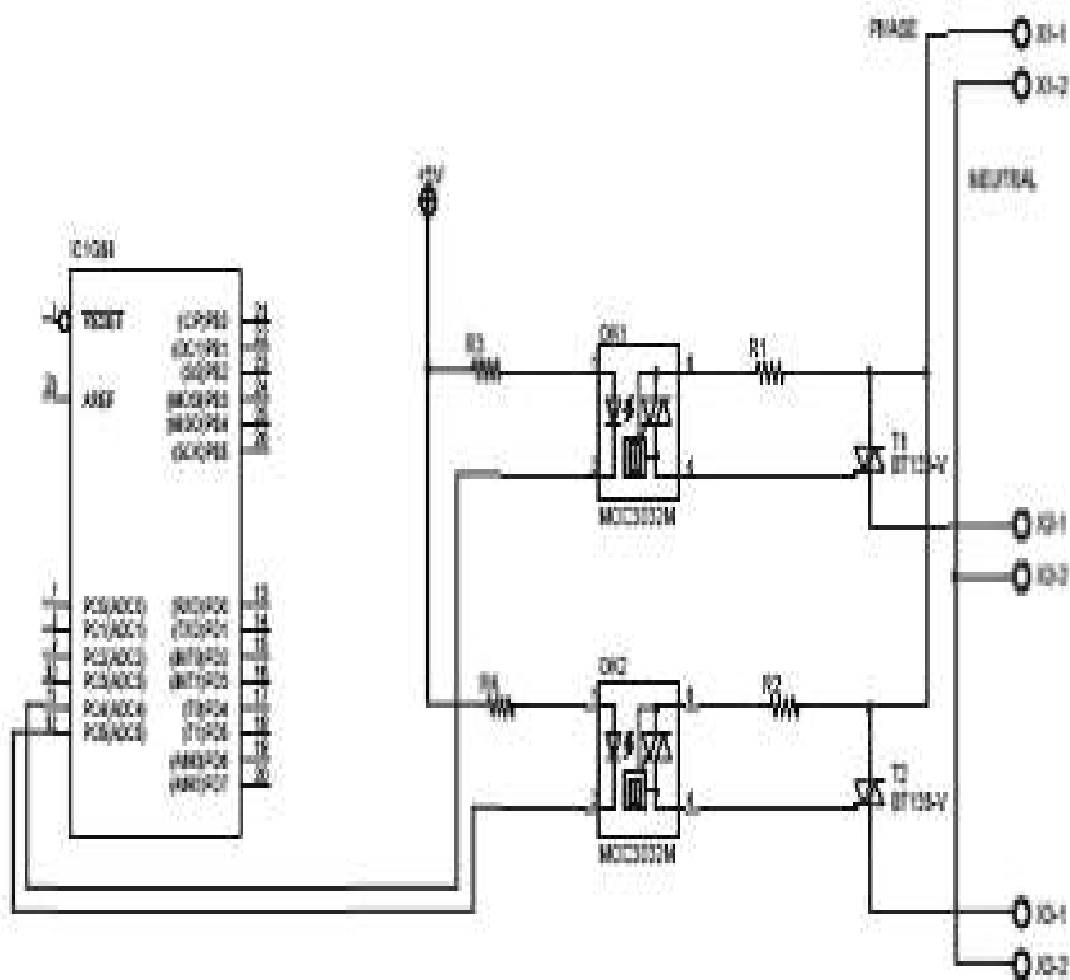


On the TRIAC symbol there are three terminals. These are the Gate and two other terminals. These other TRIAC terminals are often referred to as an "Anode" or "Main Terminal". As the TRIAC has two of these they are labelled either Anode 1 and Anode 2 or Main Terminal, MT1 and MT2.

Pin-out

| <i>Device</i> | <i>Arduino Uno pins</i> | <i>ATmega328 pin</i> |
|---------------|-------------------------|----------------------|
| Device | 4 | PC4 |
| Device2 | A5 | PC5 |

Schematic



Phase and neutral of 220V AC mains are connected to terminals X1-1 and X1-2 Two AC operated devices are connected to X2-1, X2-2 and X3-1,X3-2 respectively

Bill of Materials

| Part | Value | Device | Package | Library | Sheet |
|------|----------------|--------------------|---------------|-------------|-------|
| IC1 | Arduino Shield | ARDUINO2009_PIN_22 | ARDUINO2009_2 | arduino01 | 1 |
| OK1 | MOC3032M | MOC3032M | DIL06 | optocoupler | 1 |
| OK2 | MOC3032M | MOC3032M | DIL06 | optocoupler | 1 |
| R1 | 10K | R-US_0204/7 | 0204/7 | res | 1 |
| R2 | 10K | R-US_0204/7 | 0204/7 | res | 1 |
| R3 | 10K | R-US_R1204/7 | 0204/7 | res | 1 |
| R4 | 10K | R-US_R1204/7 | 0204/7 | res | 1 |
| T1 | BT138-V | BT138-V | TO220BV | triac | 1 |
| T2 | BT138-V | BT138-V | TO220BV | triac | 1 |
| X1 | | AK500/2 | AK500/2 | con-ptr500 | 1 |
| X2 | | AK500/2 | AK500/2 | con-ptr500 | 1 |
| X3 | | AK500/2 | AK500/2 | con-ptr500 | 1 |

Arduino program

```

void setup()
{
    pinMode(A4,OUTPUT);
    pinMode(A5,OUTPUT);
}
void setup()
{
    digitalWrite(A4,LOW);
    digitalWrite(A5,LOW);
    delay(1000);
    digitalWrite(A4,HIGH);
    digitalWrite(A5,HIGH);
    delay(1000);
}
    
```

Exercises

1. Write an Arduino program to take input from the serial port and perform the following:

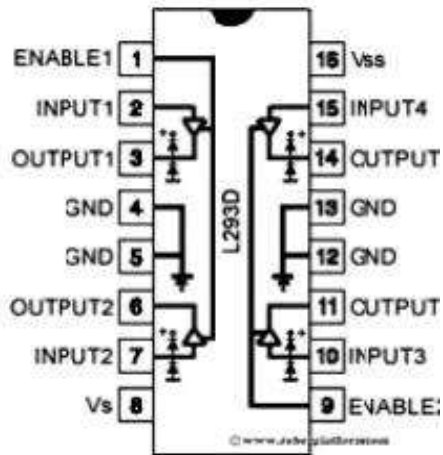
| Input character from serial port | Control action |
|---|-----------------------|
| 'A' | Device 1 ON |
| 'B' | Device 1 OFF |
| 'C' | Device 2 ON |
| 'D' | Device 2 OFF |

2. Write an Arduino program to read the LDR sensor in the LCD shield, apply a threshold and switch on an AC operated bulb using the Triac shield when the ambient light is low and turn off the bulb when the ambient light is high.

Motor control

Introduction

The GRIET Motor shield uses a dual H-bridge IC, L293D to control two 5V DC motors.



The Enable and Input pins are connecte to Arduino pins. The Enable pins are connected to PWM enabled pins while the Input pins are connected to digital pins.

Outputs 1 & 2 control one DC motor hile outputs 3 & 4 control the second C motor

TruthTable

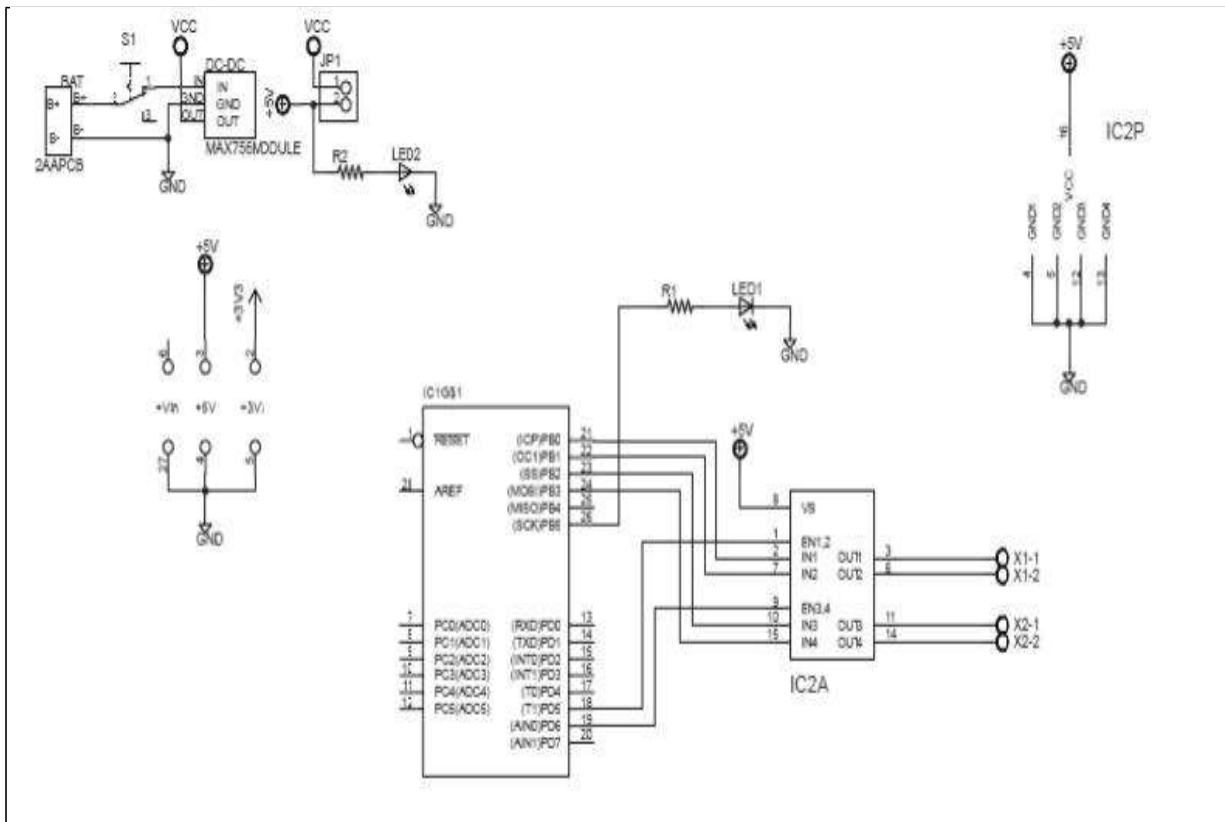
| Enable | Input1 | Input2 | Motor action |
|--------|--------|--------|----------------------------|
| H | H | L | Clock ise rotation |
| H | L | H | Counter-clockwise rotation |
| H | L | L | Stop |
| H | H | H | Stop |

The voltage applied to the Enable pin can be controlled using PWM to achieve speed control of the DC motor

Pin-out

| <i>L293D pins</i> | <i>Arduino Uno pins</i> | <i>ATmega328 pin</i> |
|-------------------|-------------------------|----------------------|
| IN1 | 8 | PB0 |
| IN2 | 9 | PB1 |
| EN1,2 | 5 | PD5 |
| IN3 | 10 | PB2 |
| IN4 | 11 | PB3 |
| EN3,4 | 6 | PD6 |

Schematic



Power supply

The power supply for the motor is derived from a DC-DC module. The input for the module can be 2 AA batteries in series, the output will be 5V. The module can supply up to 300mA of current.

Bill of Materials

| Part | Value | Device | Package | Library | Sheet |
|-------|----------------|--------------------|---------------|----------------|-------|
| BAT | 2AAPCB | 2AAPCB | 2AAPCB | BHolder | 1 |
| DC-DC | MAX756MODULE | MAX756MODULE | MAX756MODULE | BHolder | 1 |
| IC1 | Arduino Shield | ARDUINO2009_PIN_22 | ARDUINO2009_2 | arduino01 | 1 |
| LED1 | 1206Red | LEDCHIPLED_1206 | CHIPLED_1206 | led | 1 |
| LED2 | 1206Red | LEDCHIPLED_1206 | CHIPLED_1206 | led | 1 |
| R1 | 330R | R-US_R1206 | R1206 | rcl | 1 |
| R2 | 330R | R-US_R1206 | R1206 | rcl | 1 |
| S1 | | 255SB | 255SB | switch | 1 |
| X1 | | AK500/2 | AK500/2 | con- ptr500 | 1 |
| X2 | | AK500/2 | AK500/2 | con- ptr500 | 1 |

Arduino program

```

void setup()
{
    pinMode(5,OUTPUT);
    pinMode(6,OUTPUT);
    pinMode(8,OUTPUT);
    pinMode(9,OUTPUT);
    pinMode(10,OUTPUT);
    pinMode(11,OUTPUT);
    digitalWrite(5,HIGH);// Make EN1,2 HIGH
    digitalWrite (6,HIGH);// Make EN3,4 HIGH
}
void loop()
{
    // Clockwise rotation

```

```

digitalWrite (8,HIGH);
digitalWrite (9,LOW);
delay(3000);
// Stop
digitalWrite (8,LOW);
digitalWrite (9,LOW);
delay(1000);

// Anti-Clockwise rotation
digitalWrite (8,LOW);
digitalWrite (9,HIGH);
delay(3000);
// Stop
digitalWrite (8,LOW);
digitalWrite (9,LOW);
delay(1000);
}

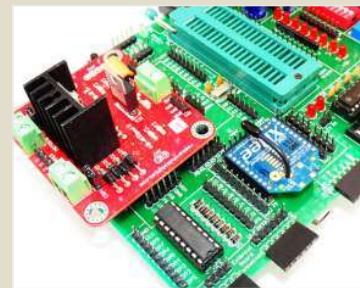
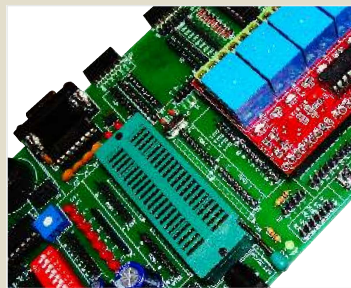
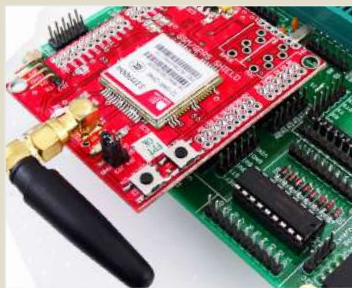
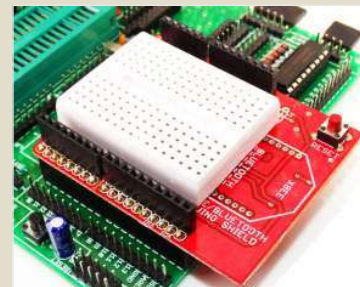
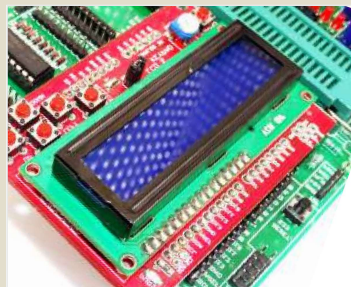
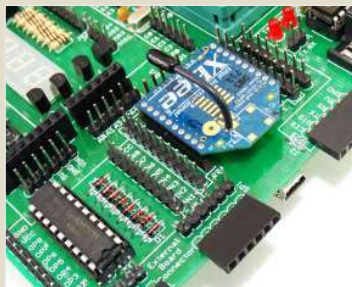
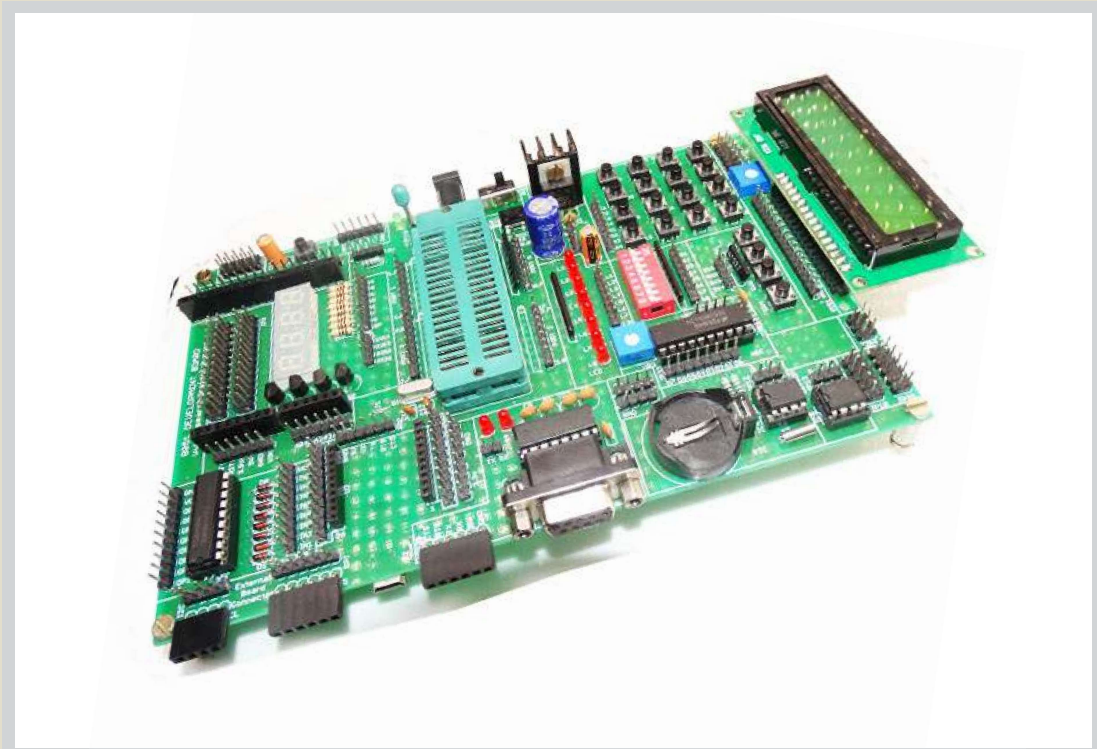
```

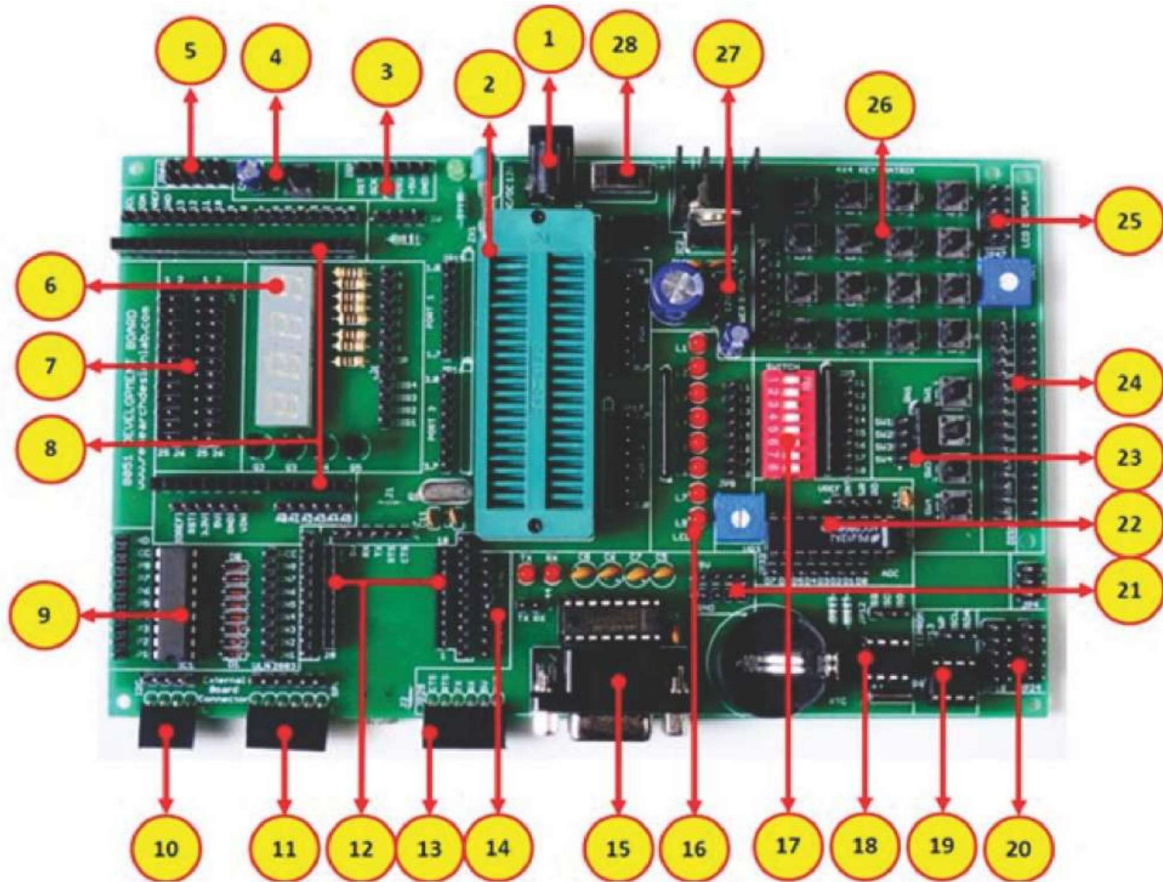
Exercises

1. Write an Arduino program to vary the speed of the motor from standstill to full speed in 5 seconds
2. Take the following commands from the Serial Monitor and control the motor as per table given below

| Command | Motor action |
|---------|-------------------------|
| 'S' | Stop |
| 'C' | Clockwise Rotation |
| 'A' | Anti-clockwise rotation |

Atmel Development Baord

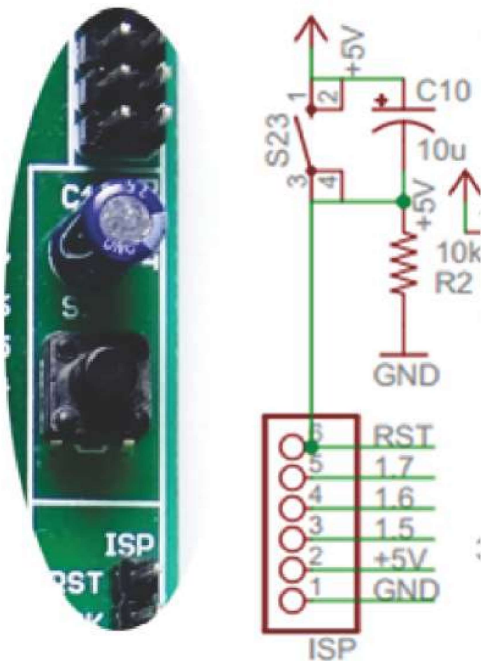




- | | |
|---|--|
| <ol style="list-style-type: none"> 1. Power supply, 5V-12V 2. 40 pin ZIF socket for IC mount. 3. ISP connector* 4. Reset 5. Node connector 6. 4x1 7 segment display 7. 26 pin raspberry connector 8. Arduino Shield footprint 9. ULN 2803 driver 10. I2C bus 11. SPI bus 12. XBEE footprint/XBEE Adaptor module 13. FT232 breakout board connector 14. DC 3.3V connectors | <ol style="list-style-type: none"> 15. DB-9 female connector 16. 8x1 LED's 17. 8 way DIP switch 18. RTC Module 19. EEPROM 20. 2x5x2 jumper node. 21. DC 5V connectors 22. Analog to Digital output 23. 4x1 keypad 24. 16x2 LCD connectors 25. Node connector 26. 4x4 Matrix Keypad 27. DC 12V connectors 28. Power ON switch |
|---|--|

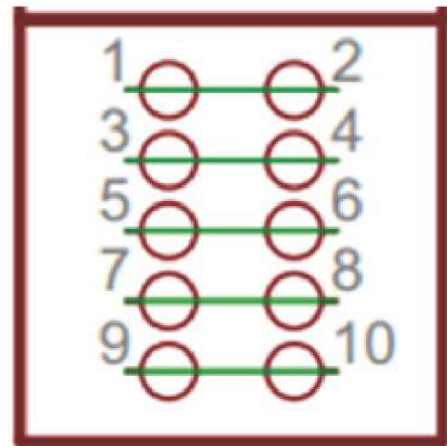
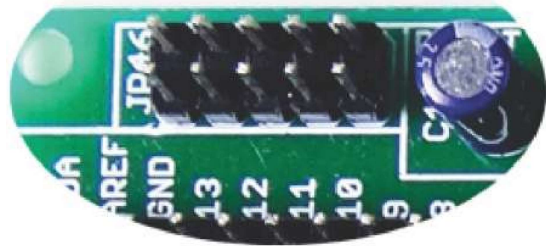
2. Reset

Resets your microcontroller by pressing s1
 NOTE: if you are following to on board program, the capacitor c1 should be desoldered and removed from the port. You also need to know, if you remove the capacitor the board has to reset manually by pressing the reset button s1 each time you burn a code.



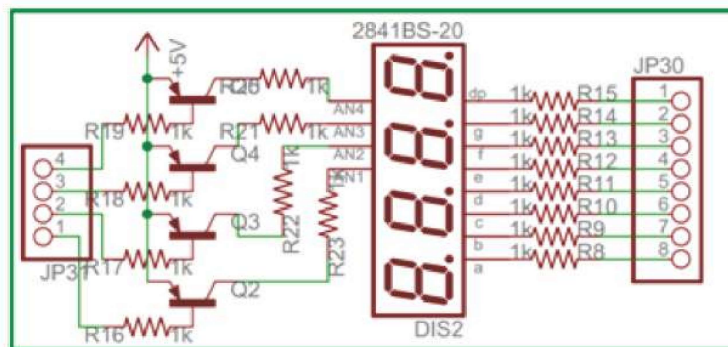
3. Node connector

Node connector is an additional on board connection extender or 1 connection IN and 1 connection out



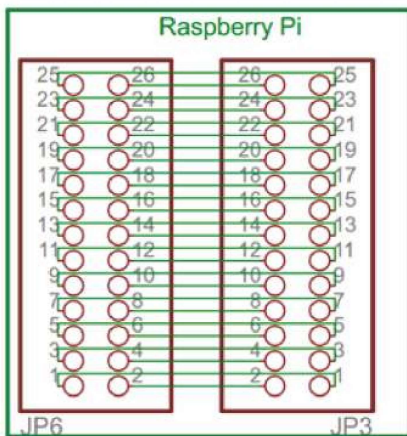
4. 4 digit 7 segment display

One seven segment digit consist of 7+1 LEDs which are arranged in a specific formation which can be used to represent digits from 0 to 9 and even some letters. One additional LED is used for marking the decimal dot, in case you want to write a decimal point in the desired segment.



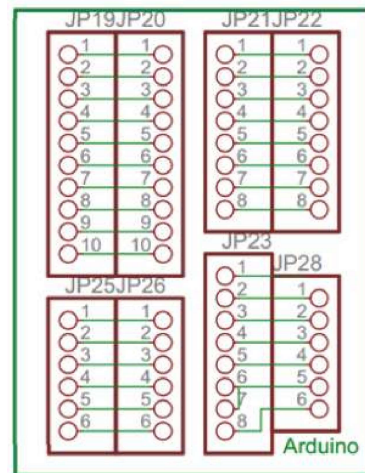
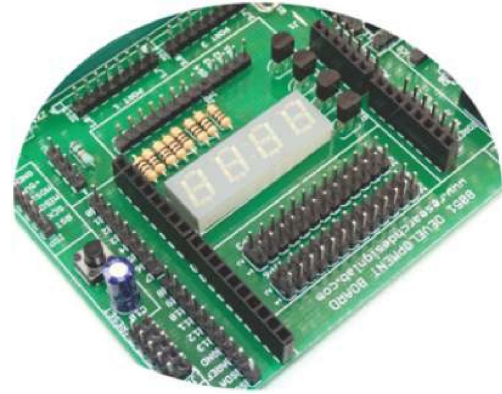
5. 26 pin raspberry connector

26 Pin raspberry connector is an easy way for making connections with raspberry pi with this development board.



6. Arduino Shield footprint

Arduino Shield footprint is provided in the board to mount different types of Arduino compatible shields on this development board.



7. ULN 2803 driver

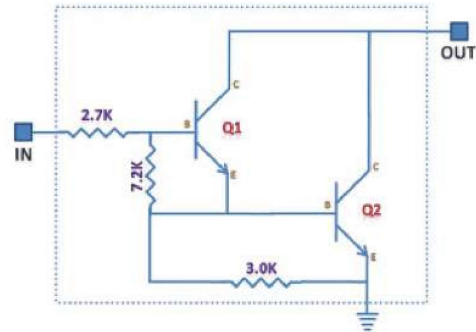
IC ULN2803 consists of octal high voltage, high current darlington transistor arrays. The eight NPN Darlington connected transistors in this family of arrays are ideally suited for interfacing between low logic level digital circuitry (such as TTL, CMOS or PMOS/NMOS) and the higher current/voltage requirements of lamps, relays, printer hammers or other similar loads for a broad range of computer, industrial, and consumer applications.

Features

- Eight Darlingtons with Common Emitter.
- Open-collector outputs.
- Free wheeling clamp diodes for transient suppression.
- Output Current to 500 mA.
- Output Voltage to 50 V.
- Inputs pinned opposite outputs to simplify board layout.

Working

The ULN 2803 IC consists of eight NPN Darlington connected transistors (often called a Darlington pair). Darlington pair consists of two bipolar transistors such that the current amplified by the first is amplified further by the second to get a high current gain β or h_{FE} . The figure shown below is one of the eight Darlington pairs of ULN 2803 IC.



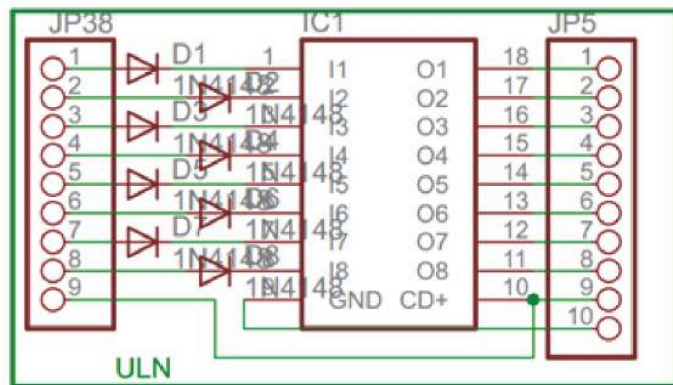
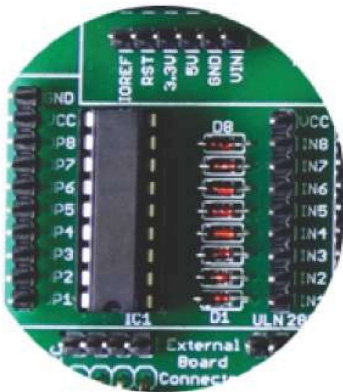
Now 2 cases arise:-

Case 1: When IN is 0 volts.

Q1 and Q2 both will not conduct as there is no base current provided to them. Thus, nothing will appear at the output (OUT).

Case 2: When IN is 5 volts.

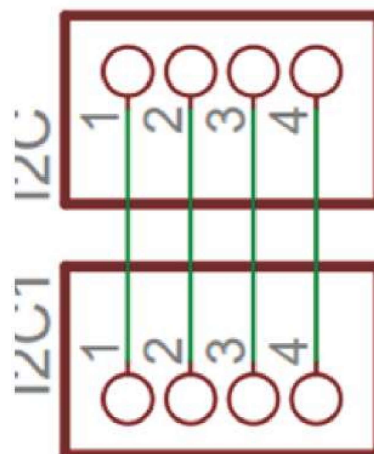
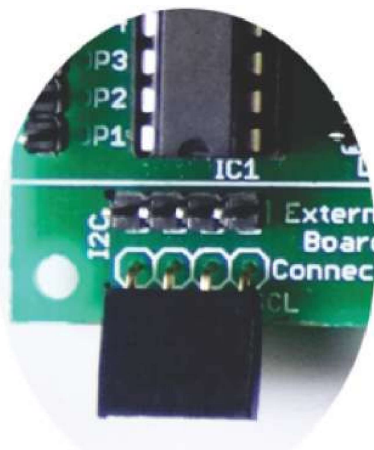
Input current will increase and both transistors Q1 and Q2 will begin to conduct. Now, input current of Q2 is combination of input current and emitter current of Q1, so Q2 will conduct more than Q1 resulting in higher current gain which is very much required to meet the higher current requirements of devices like motors, relays etc. Output current flows through Q2 providing a path (sink) to ground for the external circuit that the output is applied to. Thus, when a 5V input is applied to any of the input pins (1 to 8), output voltage at corresponding output pin (11 to 18) drops down to zero providing GND for the external circuit. Thus, the external circuit gets grounded at one end while it is provided +Vcc at its other end. So, the circuit gets completed and starts operating.



8. I2C bus

One IC that wants to talk to another must: (Protocol)

- 1) Wait until it sees no activity on the I2C bus. SDA and SCL are both high. The bus is 'free'.
- 2) Put a message on the bus that says 'its mine' - I have STARTED to use the bus. All other ICs then LISTEN to the bus data to see whether they might be the one who will be called up (addressed).
- 3) Provide on the CLOCK (SCL) wire a clock signal. It will be used by all the ICs as the reference time at which each bit of DATA on the data (SDA) wire will be correct (valid) and can be used. The data on the data wire (SDA) must be valid at the time the clock wire (SCL) switches from 'low' to 'high' voltage.
- 4) Put out in serial form the unique binary 'address'(name) of the IC that it wants to communicate with.
- 5) Put a message (one bit) on the bus telling whether it wants to SEND or RECEIVE data from the other chip. (The read/write wire is gone!)
- 6) Ask the other IC to ACKNOWLEDGE (using one bit) that it recognized its address and is ready to communicate.
- 7) After the other IC acknowledges all is OK, data can be transferred.
- 8) The first IC sends or receives as many 8-bit words of data as it wants. After every 8-bit data word the sending IC expects the receiving IC to acknowledge the transfer is going OK.
- 9) When all the data is finished the first chip must free up the bus and it does that by a special message called 'STOP'. It is just one bit of information transferred by a special 'wiggling' of the SDA/SCL wires of the bus.



9. SPI bus

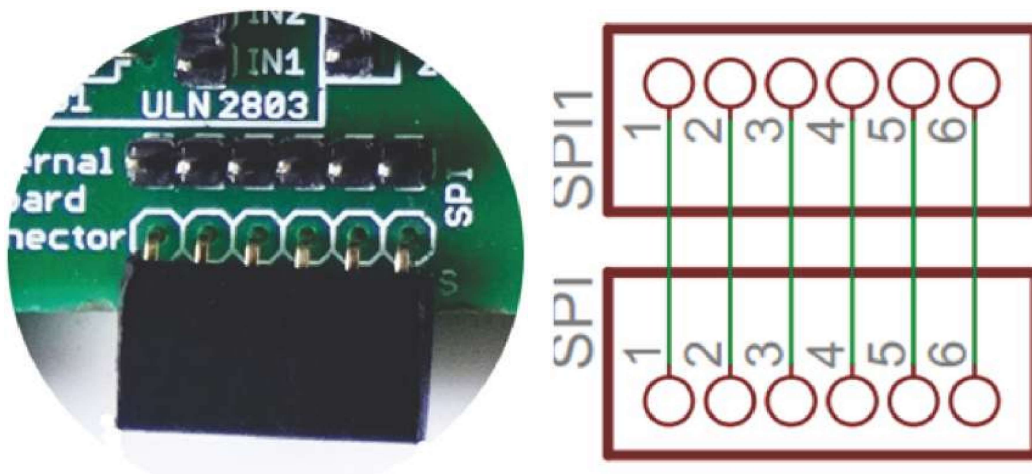
Serial to Peripheral Interface (SPI) is a hardware/firmware communications protocol developed by Motorola and later adopted by others in the industry. Microwire of National Semiconductor is same as SPI. Sometimes SPI is also called a "four wire" serial bus.

The Serial Peripheral Interface or SPI-bus is a simple 4-wire serial communications interface used by many microprocessor/microcontroller peripheral chips that enables the controllers and peripheral devices to communicate each other. Even though it is developed primarily for the communication between host processor and peripherals, a connection of two processors via SPI is just as well possible.

The SPI bus, which operates at full duplex (means, signals carrying data can go in both directions simultaneously), is a synchronous type data link setup with a Master / Slave interface and can support up to 1 megabaud or 10Mbps of speed. Both single-master and multi-master protocols are possible in SPI. But the multi-master bus is rarely used and look awkward, and are usually limited to a single slave.

The SPI Bus is usually used only on the PCB. There are many facts, which prevent us from using it outside the PCB area. The SPI Bus was designed to transfer data between various IC chips, at very high speeds. Due to this high-speed aspect, the bus lines cannot be too long, because their reactance increases too much, and the Bus becomes unusable. However, its possible to use the SPI Bus outside the PCB at low speeds, but this is not quite practical.

The peripherals can be a Real Time Clocks, converters like ADC and DAC, memory modules like EEPROM and FLASH, sensors like temperature sensors and pressure sensors, or some other devices like signal-mixer, potentiometer, LCD controller, UART, CAN controller, USB controller and amplifier.

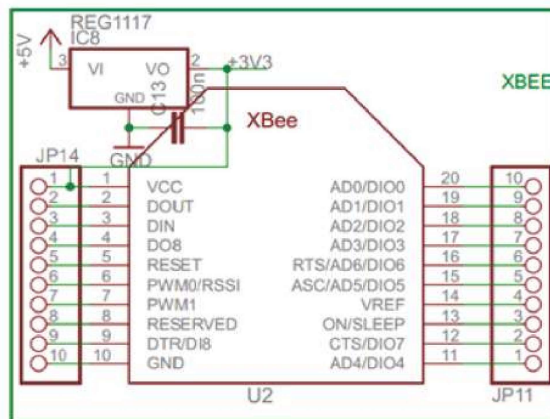
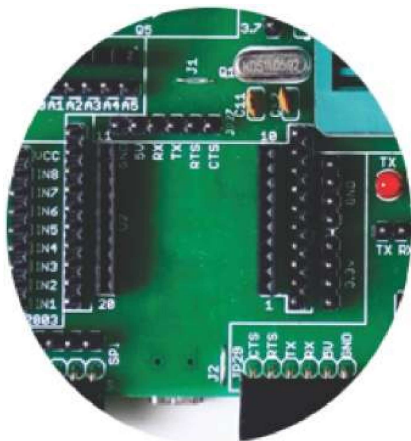


10. XBEE footprint/ XBEE Adaptor module

All XBeeZNet 2.5 modules can be identified by their unique 64-bit addresses or a user-configurable ASCII string identifier. The 64-bit address of a module can be read using the SH and SL commands. The ASCII string identifier is configured using the NI command.

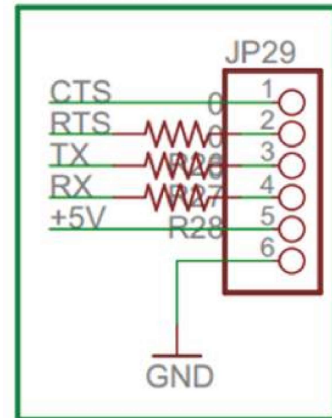
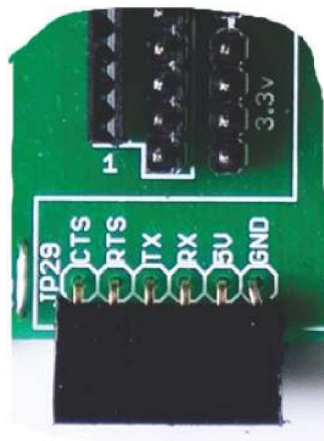
To transmit using device addressing, only the destination address must be configured. The destination address can be specified using either the destination device's 64-bit address or its NI-string. The XBee modules also support coordinator and broadcast addressing modes. Device addressing in the AT firmware is configured using the DL, DH, or DN commands. In the API firmware, the ZigBee Transmit Request API frame (0x10) can be used to specify destination addresses.

To address a node by its 64-bit address, the destination address must be set to match the 64-bit address of the remote. In the AT firmware, the DH and DL commands set the destination 64-bit address. In the API firmware, the destination 64-bit address is set in the ZigBee Transmit Request frame. ZigBee end devices rely on a parent (router or coordinator) to remain awake and receive any data packets destined for the end device. When the end device wakes from sleep, it sends a transmission (poll request) to its parent asking if the parent has received any RF data destined for the end device. The parent, upon receipt of the poll request, will send an RF response and the buffered data (if present). If the parent has no data for the end device, the end device may return to sleep, depending on its sleep mode configuration settings. The following figure demonstrates how the end device uses polling to receive RF data through its parent.



11. FT232 breakout board connector

A standard FT232 breakout board from researchdesignlab.com could be used to interface on these connectors, whose other end is connected to a USB.

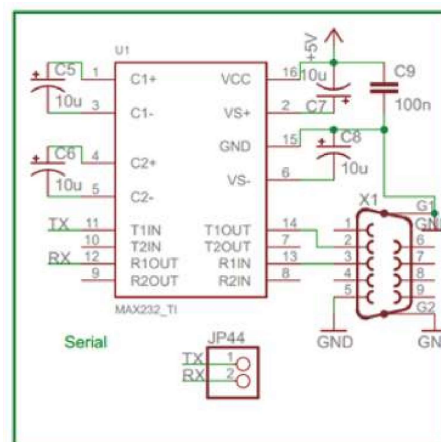
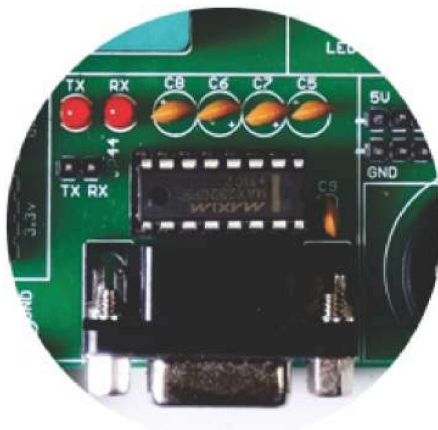


12. DC 3.3V connectors

These connectors provide on board 3.3V DC connections.

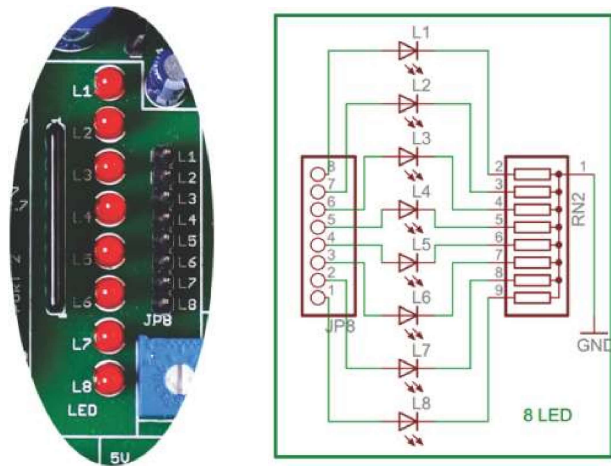
13. DB-9 female connector

RS-232 is a standard communication protocol for linking computer and its peripheral devices to allow serial data exchange. In simple terms RS232 defines the voltage for the path used for data exchange between the devices. It specifies common voltage and signal level, common pin wire configuration and minimum, amount of control signals.



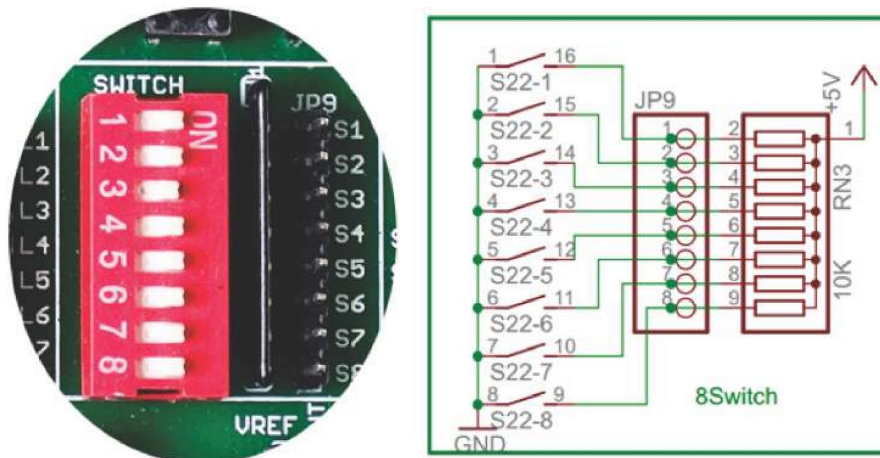
14. 8x1 LED's

LED's are used to indicate something, whether any pin is high or indicating the output for many purposes like indicating I/O status or program debugging running state. We have four led outputs on board which can be used by the programmer as per the requirement for testing and development.



15. 8 way DIP switch

DIP switches are an alternative to jumper blocks. Their main advantages are that they are quicker to change and there are no parts to lose.



16.RTC Module

The DS1307 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 a 2-wire bi-directional 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.

Operation

The DS1307 operates as a slave device on the serial bus. Access is obtained by implementing a START condition and providing a device identification code followed by a register address. Subsequent registers can be accessed sequentially until a STOP condition is executed. When VCC falls below $1.25 \times V_{BAT}$ the device terminates an access in progress and resets the device address counter. Inputs to the device will not be recognized at this time to prevent erroneous data from being written to the device from an out of tolerance system. When VCC falls below VBAT the device switches into a low current battery backup mode. Upon power up, the device switches from battery to VCC when VCC is greater than $V_{BAT} + 0.2V$ and recognizes inputs.

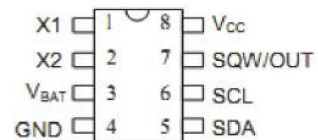
Features:

1. 56 byte nonvolatile RAM for data storage
2. 2-wire serial interface
3. Programmable square wave output signal
4. Automatic power-fail detect and switch circuitry
5. Consumes less than 500 nA in battery backup mode with oscillator running
6. Optional industrial temperature range $-40^{\circ}C$ to $+85^{\circ}C$
7. Available in 8-pin DIP or SOIC
8. Recognized by Underwriters Laboratory

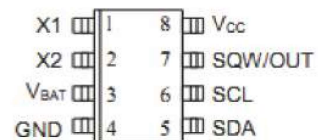
PIN DESCRIPTION

1. VCC - Primary Power Supply
2. X1, X2 - 32.768 kHz Crystal Connection
3. VBAT - +3V Battery Input
4. GND - Ground
5. SDA - Serial Data
6. SCL - Serial Clock
7. SQW/OUT - Square wave/Output Driver

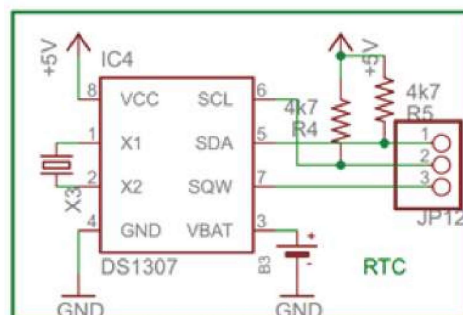
PIN ASSIGNMENT

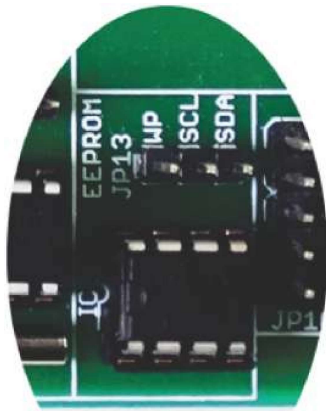


DS1307 8-Pin DIP (300 mil)



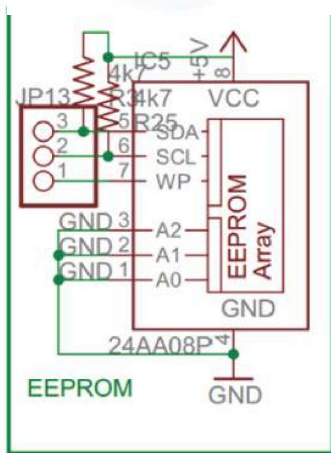
DS1307Z 8-Pin SOIC (150 mil)





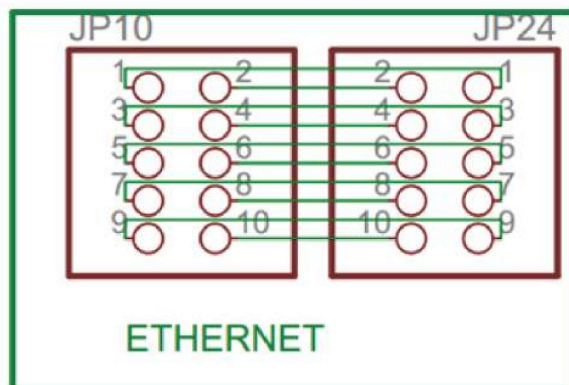
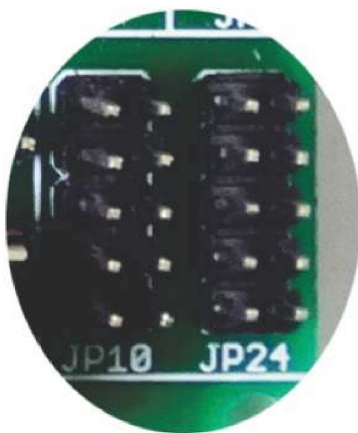
17. EEPROM

IC, EEPROM I2C 4K, 24C04, DIP8
 Memory Size: 4Kbit
 Memory Configuration: 512 x 8
 Interface Type: I2C, Serial
 Clock Frequency: 400kHz
 Supply Voltage Range: 2.5V to 5.5V
 Memory Case Style: DIP
 No. of Pins: 8
 Operating Temperature Range: -40°C to +85°C
 SVHC: No SVHC (19-Dec-2011)
 Base Number: 24
 Device Marking: M24C04
 IC Generic Number: 24C04
 Interface: I2C
 Interface Type: Serial, I2C
 Logic Function Number: 24C04
 Memory Configuration: 512 x 8
 Memory Size: 4Kbit
 Memory Type: EEPROM
 Memory Voltage Vcc: 2.5V
 Operating Temperature Max: +85°C
 Operating Temperature Min: -40°C
 Package / Case: DIP
 Supply Voltage Max: 5.5V
 Supply Voltage Min: 2.5V
 Termination Type: Through Hole
 Voltage Vcc: 2.5V



18. 2x5x2 jumper node

Node connector is an additional on board connection extender or 1 connection IN and 1 connection out





19. DC 5V connectors

These connectors provide on board 5V DC connections.

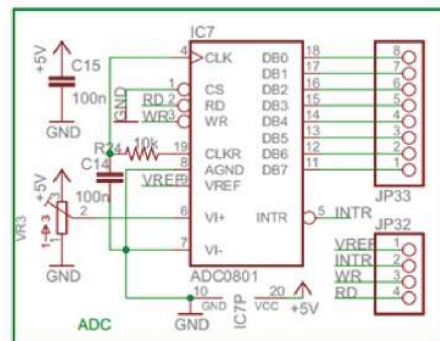
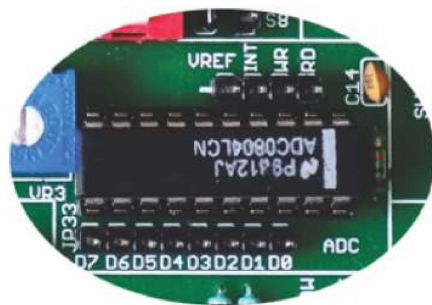
20. Analog to Digital output

The ADC0804 is CMOS 8-bit successive approximation A/D converters that use a differential potentiometric ladder similar to the 256R products. This converter is designed to allow operation with the NSC800 and INS8080 derivative control bus with TRI-STATE output latches directly driving the data bus. These A/Ds appear like memory locations or I/O ports to the microprocessor and no interfacing logic is needed.

Differential analog voltage inputs allow increasing the common-mode rejection and offsetting the analog zero input voltage value. In addition, the voltage reference input can be adjusted to allow encoding any smaller analog voltage span to the full 8 bits of resolution.

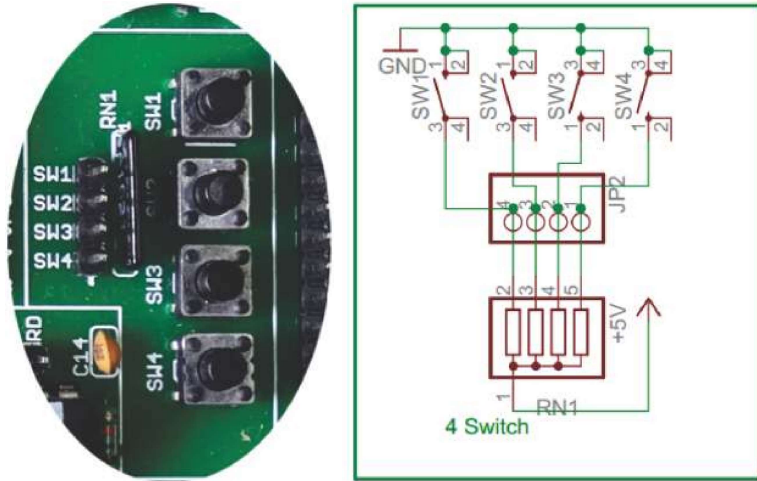
Features

- Compatible with 8080 μ P derivatives no interfacing logic needed - access time - 135 ns
- Easy interface to all microprocessors, or operates "stand-alone"
- Differential analog voltage inputs
- Differential analog voltage inputs
- Works with 2.5V (LM336) voltage reference
- On-chip clock generator
- 0V to 5V analog input voltage range with single 5V supply
- No zero adjust required
- 0.3x standard width 20-pin DIP package
- 20-pin molded chip carrier or small outline package
- Operates ratio metrically or with 5 VDC, 2.5 VDC, or analog span adjusted voltage reference



21. 4x1 keypad

Switches are mainly used to switch the controls of a module. We have four switches on board which can be used by the programmer as per the requirement for testing and development.



22. 16x2 LCD connectors

LCD screen consists of two lines with 16 characters each. Each character consists of 5x7 dot matrix. Contrast on display depends on the power supply voltage and whether messages are displayed in one or two lines. For that reason, variable voltage 0-V_{dd} is applied on pin marked as V_{ee}. Trimmer potentiometer is usually used for that purpose. Some versions of displays have built in backlight (blue or green diodes). When used during operating, a resistor for current limitation should be used (like with any LE diode). LCD Connection Depending on how many lines are used for connection to the microcontroller, there are 8-bit and 4-bit LCD modes. The appropriate mode is determined at the beginning of the process in a phase called “initialization”. In the first case, the data are transferred through outputs D0-D7 as it has been already explained. In case of 4-bit LED mode, for the sake of saving valuable I/O pins of the microcontroller, there are only 4 higher bits (D4-D7) used for communication, while other may be left unconnected.

Consequently, each data is sent to LCD in two steps: four higher bits are sent first (that normally would be sent through lines D4-D7), four lower bits are sent afterwards. With the help of initialization, LCD will correctly connect and interpret each data received. Besides, with regards to the fact that data are rarely read from LCD (data mainly are transferred from microcontroller to LCD) one more I/O pin may be saved by simple connecting R/W pin to the Ground. Such saving has its price. Even though message displaying will be normally performed, it will not be possible to read from busy flag since it is not possible to read from display.

Features:

1. Can display 224 different symbols.
2. Low power consumption.
3. 5x7 dot matrix format.
4. Powerful command set and user produced characters.

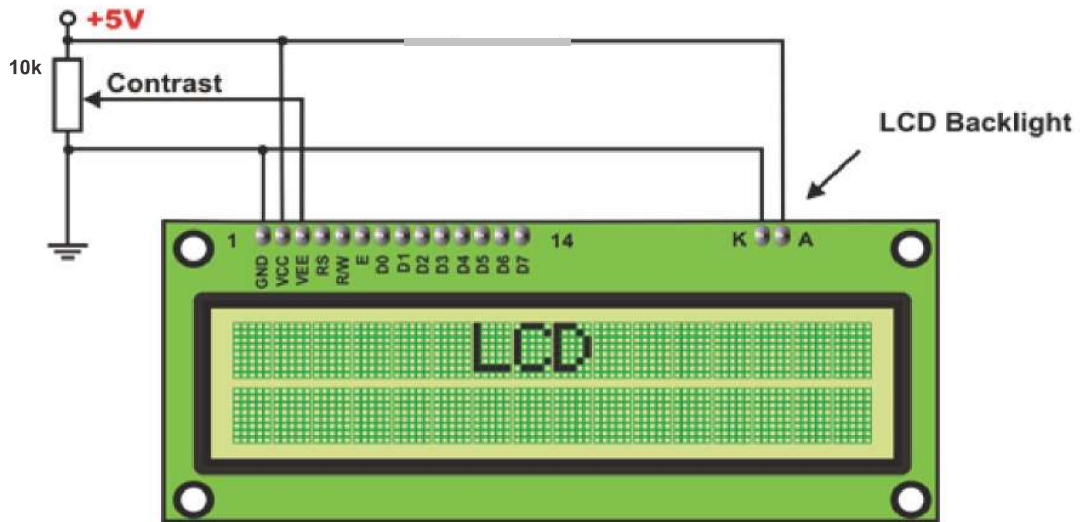
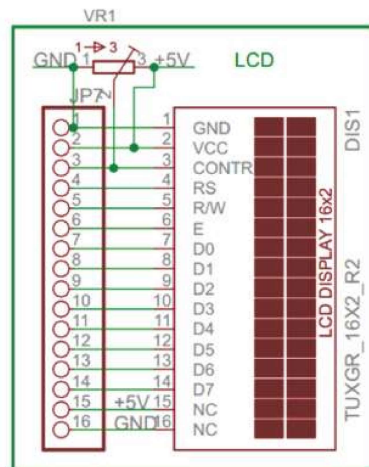
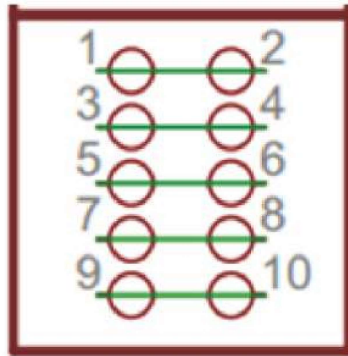


Fig: Circuit connections of LCD



PIN DESCRIPTION

1. Gnd:- Power supply ground
2. VCC:-+5v Power supply input
3. RS:- Reset pin
4. R/W:- Read/Write pin
5. En:-Enable pin
6. D0-D7:- Data lines



23. Node connector

Node connector is an additional on board connection extender or 1 connection IN and 1 connection out

24. 4x4 Matrix Keypad

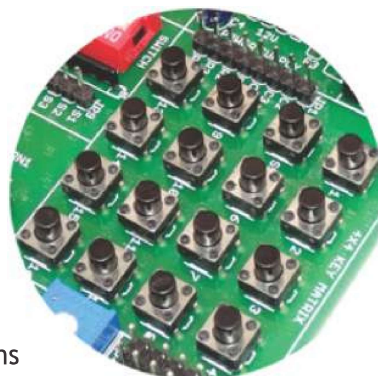
In a 4x4 matrix keypad eight Input/Output ports are used for interfacing with any microcontrollers. Rows are connected to Peripheral Input/Output (PIO) pins configured as output. Columns are connected to PIO pins configured as input with interrupts. In this configuration, four pull-up resistors must be added in order to apply a high level on the corresponding input pins as shown in below Figure. The corresponding hexadecimal value of the pressed key is sent on four LEDs.

WORKING

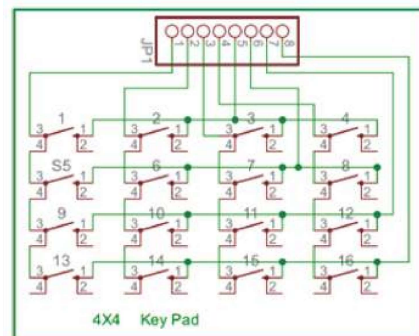
This Application Note describes programming techniques implemented on the AT91 ARM-based microcontroller for scanning a 4x4 Keyboard matrix usually found in both consumer and industrial applications for numeric data entry. AT91 Keyboard interface In this application, a 4x4 matrix keypad requiring eight Input/Output ports for interfacing is used as an example. Rows are connected to Peripheral Input/Output (PIO) pins configured as output. Columns are connected to PIO pins configured as input with interrupts. In this configuration, four pull-up resistors must be added in order to apply a high level on the corresponding input pins as shown in Figure 1. The corresponding hexadecimal value of the pressed key is sent on four LEDs.

FEATURES

1. Contact debouncing.
2. Easy to interface.
3. Interfaces to any microcontroller or microprocessor.
4. Data valid output signal for interrupt activation.



PIN DETAILS
 pin 1-4: R0-R3:- Rows
 pin 5-8: C0-C3:- Columns



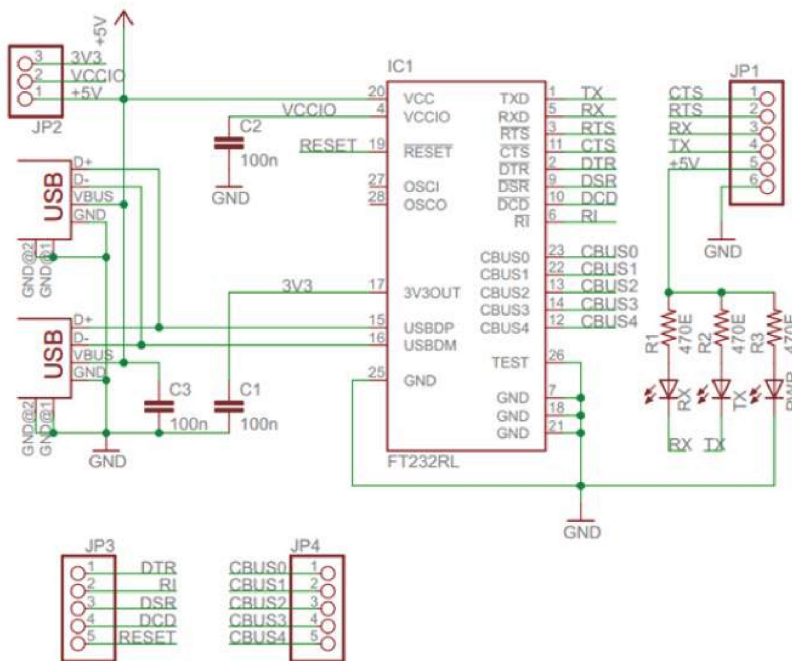


25. DC 12V connectors

These connectors provide on board 12V DC connections.

26. USB to serial converter (optional)

In modern computers you cannot find serial port (DB-9). But most of the basic controllers work with this protocol. To connect your system to such controllers we require USB to serial converters. This board has the facility to be connected directly to USB using a USB cable (A to B).



Programming Codes:

- | | |
|--|---|
| <ul style="list-style-type: none"> • LED BLINK http://researchdesignlab.com/8051-i/o-code • LCD http://researchdesignlab.com/8051-lcd-code • KEYPAD http://researchdesignlab.com/8051-keypad-code • UART http://researchdesignlab.com/8051-uart-code | <ul style="list-style-type: none"> • RTC http://researchdesignlab.com/8051-rtc-code • EEPROM http://researchdesignlab.com/8051-eprom-code • ADC http://researchdesignlab.com/8051-adc-code.html • 7 Segment Display http://researchdesignlab.com/7-segment-atmel-code.html |
|--|---|

LEDs Interfacing with 8051

Circuit Design:

The circuit mainly consists of AT89C51 microcontroller. AT89C51 belongs to the family of 8051 microcontroller. It is an 8-bit microcontroller. This microcontroller has 4KB of Flash Programmable and Erasable Read Only Memory and 128 bytes of RAM. This can be programmed and erased a maximum of 1000 times.

It has two 16-bit timers/counters. It supports USART communication protocol. It has 40 pins. There are four ports are designated as P0, P1, P2, and P3. Port P0 will not have internal pull-ups, while the other ports have internal pull-ups.

How to Control LEDs?

Light Emitting Diodes are the semi-conductor light sources. Commonly used LEDs will have a cut-off voltage of 1.7V and current of 10mA. When an LED is applied with its required voltage and current it glows with full intensity.

The Light Emitting Diode is like the normal PN diode, but it emits energy in the form of light. The color of light depends on the band gap of the semiconductor. The following figure shows “how an LED glows?”



Thus, LED is connected to the AT89C51 microcontroller with the help of a current limiting resistor. The value of this resistor is calculated using the following formula.

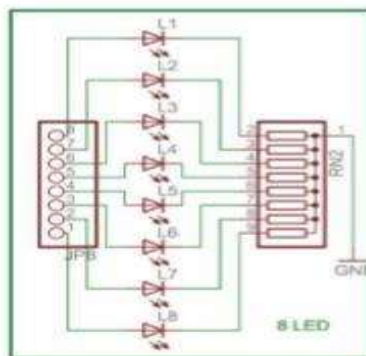
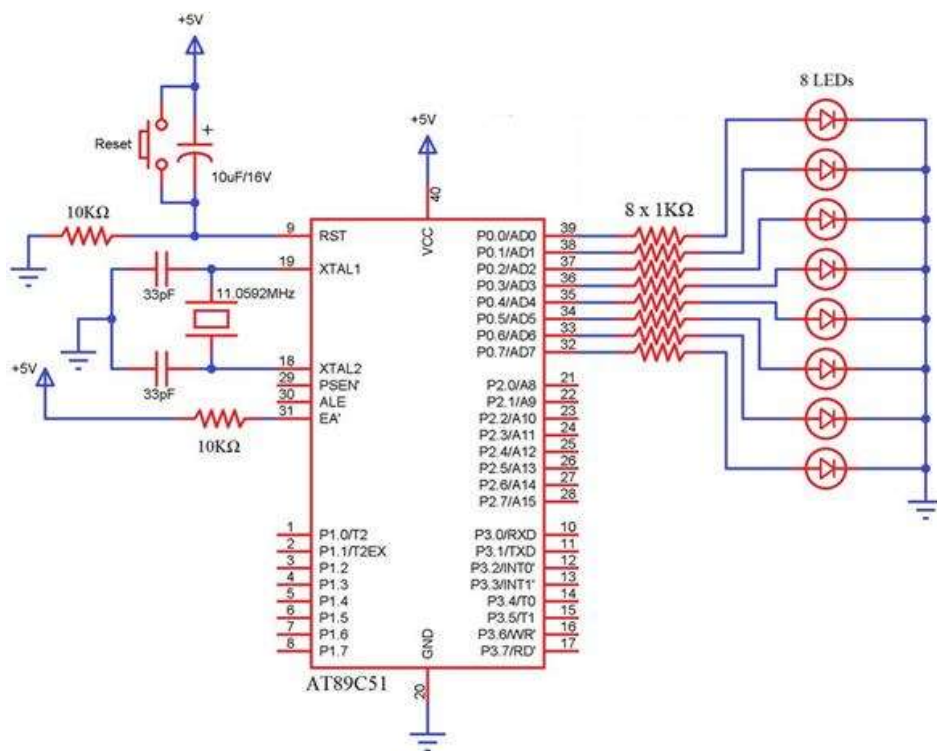
$$R = (V - 1.7) / 10\text{mA}, \text{ where } V \text{ is the input voltage.}$$

Generally, microcontrollers output a maximum voltage of 5V. Thus, the value of resistor calculated for this is 330 Ohms. This resistor can be connected to either the cathode or the anode of the LED.

Principle behind Interfacing LED with 8051

The main principle of this circuit is to interface LEDs to the 8051-family micro controller. Commonly, used LEDs will have voltage drop of 1.7v and current of 10mA to glow at full intensity. This is applied through the output pin of the micro controller.

Circuit Diagram:



Program:

```

*/
#include<reg52.h> //special function register declarations
//for the intended 8051 derivative
void delay(); // Function prototype declaration
sbit LED0=P2^0; //Define Port Pin P2.0 as LED0
sbit LED1=P2^1; //Define Port Pin P2.1 as LED1
sbit LED2=P2^2; //Define Port Pin P2.2 as LED2
sbit LED3=P2^3; //Define Port Pin P2.3 as LED3
sbit LED4=P2^4; //Define Port Pin P2.4 as LED4
sbit LED5=P2^5; //Define Port Pin P2.5 as LED5
sbit LED6=P2^6; //Define Port Pin P2.6 as LED6
sbit LED7=P2^7; //Define Port Pin P2.7 as LED7

void main() //Main Code
{
P2=0x00; //Set Port 2 all bits to 0
while(1) // Continuous loop
{
LED0=1; //Turn ON LED0
delay(); //Wait for a small delay
LED1=1; //Turn ON LED1
delay(); //Wait for a small delay
LED2=1; //Turn ON LED2
delay(); //Wait for a small delay
LED3=1; //Turn ON LED3
delay(); //Wait for a small delay
LED4=1; //Turn ON LED4
delay(); //Wait for a small delay
LED5=1; //Turn ON LED5
delay(); //Wait for a small delay
LED6=1; //Turn ON LED6
delay(); //Wait for a small delay
LED7=1; //Turn ON LED7
delay(); //Wait for a small delay
}
}

```

```
P2=0x00; //Turn OFF all LED's
delay(); //Wait for a small delay
}
}
void delay() // Delay Routine
{
unsigned int x=60000; // larger the value of x the more is the delay.
while (x--); // executes this statement until x decrements to 0;
}
```

LCD Interfacing with 8051

Display units are the most important output devices in embedded projects and electronics products. 16x2 LCD is one of the most used display unit. 16x2 LCD means that there are two rows in which 16 characters can be displayed per line, and each character takes 5X7 matrix space on LCD. In this tutorial we are going to connect 16X2 LCD module to the 8051 Microcontroller (AT89S52). Interfacing LCD with 8051 microcontrollers need to understand and connect 16 pins of LCD to the microcontroller.

16 pins of LCD module can divide it in five categories, Power Pins, contrast pin, Control Pins, Data pins and Backlight pins.

| Category | Pin NO. | Pin Name | Function |
|----------------|---------|------------|---|
| Power Pins | 1 | VSS | Ground Pin, connected to Ground |
| | 2 | VDD or Vcc | Voltage Pin +5V |
| Contrast Pin | 3 | V0 or VEE | Contrast Setting, connected to Vcc through a variable resistor. |
| Control Pins | 4 | RS | Register Select Pin, RS=0 Command mode, RS=1 Data mode |
| | 5 | RW | Read/ Write pin, RW=0 Write mode, RW=1 Read mode |
| | 6 | E | Enable, a high to low pulse need to enable the LCD |
| Data Pins | 7-14 | D0-D7 | Data Pins, Stores the Data to be displayed on LCD or the command instructions |
| Backlight Pins | 15 | LED+ or A | To power the Backlight +5V |
| | 16 | LED- or K | Backlight Ground |

Control Pins:

RS: RS is the register select pin. We need to set it to 1 if we are sending some data to be displayed on LCD. And we will set it to 0 if we are sending some command instruction like clear the screen (hex code 01).

RW: This is Read/write pin, we will set it to 0, if we are going to write some data on LCD. And set it to 1, if we are reading from LCD module. Generally, this is set to 0, because we do not have need to read data from LCD. Only one instruction “Get LCD status”, need to be read sometimes.

E: This pin is used to enable the module when a high to low pulse is given to it. A pulse of 450 ns should be given. That transition from HIGH to LOW makes the module ENABLE.

There are some preset command instructions in LCD which must have used them in program. To prepare the LCD (in lcd_init() function).

Some important command instructions are given below:

| Hex Code | Command to LCD Instruction Register |
|----------|--|
| 0F | LCD ON, cursor ON |
| 01 | Clear display screen |
| 02 | Return home |
| 04 | Decrement cursor (shift cursor to left) |
| 06 | Increment cursor (shift cursor to right) |
| 05 | Shift display right |
| 07 | Shift display left |
| 0E | Display ON, cursor blinking |
| 80 | Force cursor to beginning of first line |
| C0 | Force cursor to beginning of second line |
| 38 | 2 lines and 5×7 matrix |
| 83 | Cursor line 1 position 3 |
| 3C | Activate second line |
| 08 | Display OFF, cursor OFF |

| | |
|----|---------------------------------|
| C1 | Jump to second line, position 1 |
| OC | Display ON, cursor OFF |
| C1 | Jump to second line, position 1 |
| C2 | Jump to second line, position 2 |

LCD Initialization:

The steps that have to be done for initializing the LCD display is given below and these steps are common for almost all applications.

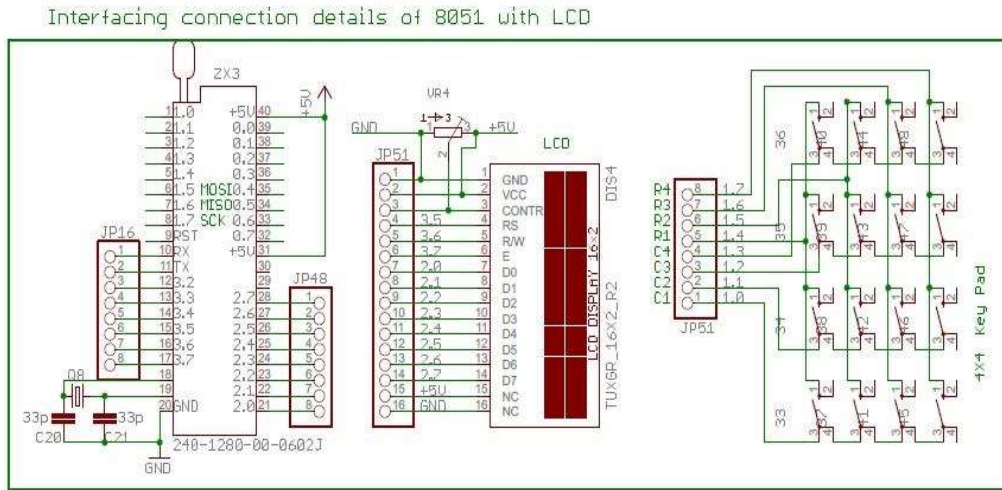
- Send 38H to the 8-bit data line for initialization
- Send 0FH for making LCD ON, cursor ON and cursor blinking ON.
- Send 06H for incrementing cursor position.
- Send 01H for clearing the display and return the cursor.

Sending Data to LCD:

The steps for sending data to the LCD module is given below. I have already said that the LCD module has pins namely RS, R/W and E. It is the logic state of these pins that make the module to determine whether a given data input is a command or data to be displayed.

- Make R/W low.
- Make RS=0 if data byte is a command and make RS=1 if the data byte is a data to be displayed.
- Place data byte on the data register.
- Pulse E from high to low.
- Repeat above steps for sending another data.

Circuit Diagram:



Program:

```

*/
#include<reg51.h>
#define LCD_PORT P2
sbit rs=P3^5;
sbit en=P3^7;
sbit D7=P2^7;
sbit rw=P3^6;

void busy();
void CMD_WRT(unsigned char);
void LCD_WRT(unsigned char *);
void DATA_WRT(unsigned char);
void DELAY();
void main()
{
unsigned char CMD[]={0x38,0x01,0x0f,0x06,0x80},TEMP1,i;
for(i=0;i<5;i++)
{
TEMP1=CMD[i];

```



```

        CMD_WRT(TEMP1);
    }
    DELAY();
        CMD_WRT(0X01);
    CMD_WRT(0X80);
    LCD_WRT("16X2 LCD DISPLAY");
    DELAY();
        DELAY();
        while(1);

}

void DELAY()
{
    unsigned int X=6000000;
    while(X--);
}

void busy()
{
    D7=1;
    rs=0;
    rw=1;
    while(D7!=0)
    {
        en=0;
        en=1;
    }
}

void CMD_WRT(unsigned char val)
{
    busy();
    LCD_PORT=val;
    rs=0;
    rw=0;
}

```

```
en=1;
en=0;
}

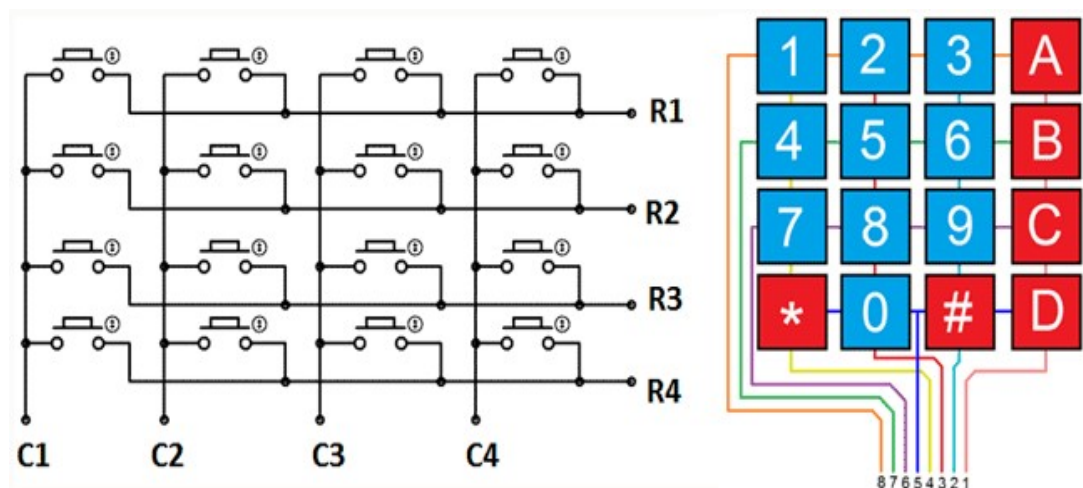
void LCD_WRT(unsigned char *string)
{
while(*string)
DATA_WRT(*string++);
}
void DATA_WRT(unsigned char ch)
{
busy();
LCD_PORT = ch;
rs=1;
rw=0;
en=1;
en=0;
}
```

KEYPAD Interfacing with 8051

Keypads are widely used input devices being used in various electronics and embedded projects. They are used to take inputs in the form of numbers and alphabets and feed the same into system for further processing.

4x4 Matrix Keypad:

Before we interface the keypad with microcontroller, first we need to understand how it works. Matrix keypad consists of set of Push buttons, which are interconnected. Like in our case we are using 4X4 matrix keypad, in which there are 4 push buttons in each of four rows. And the terminals of the push buttons are connected according to diagram. In first row, one terminal of all the 4 push buttons are connected together and another terminal of 4 push buttons are representing each of 4 columns, same goes for each row. So, we are getting 8 terminals to connect with a microcontroller.



First interface an LCD module to display the data which will be feed through KEYPAD.

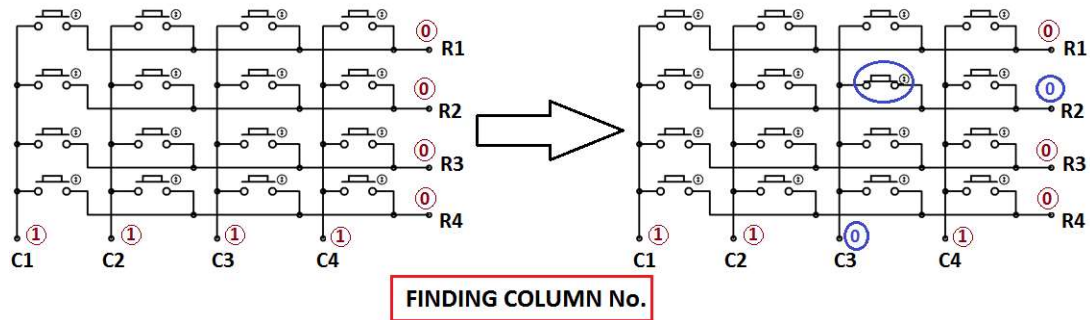
As shown in above circuit diagram, to interface Keypad, we need to connect 8 terminals of the keypad to any port (8 pins) of the microcontroller. Like we have connected keypad terminals to Port 1 of 8051. Whenever any button is pressed, we need to get the location of the button, means the corresponding ROW an COLUMN no. Once we get the location of the button, we can print the character accordingly.

Now the question is how to get the location of the pressed button?

Steps to get the location of the pressed button:

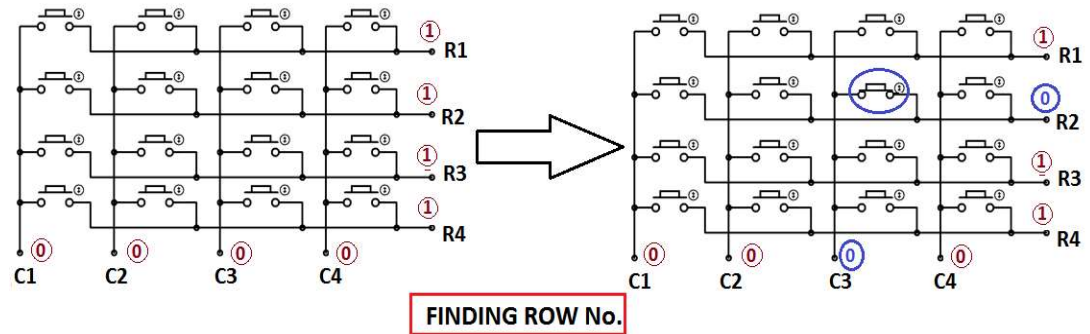
1. First we have made all the Rows to Logic level 0 and all the columns to Logic level 1.

2. Whenever we press a button, column and row corresponding to that button gets shorted and makes the corresponding column to logic level 0. Because that column becomes connected (shorted) to the row, which is at Logic level 0. So we get the column no. See main() function.



3. Now we need to find the Row no., so we have created four functions corresponding to each column. Like if any button of column one is pressed, we call function row_finder1(), to find the row no.

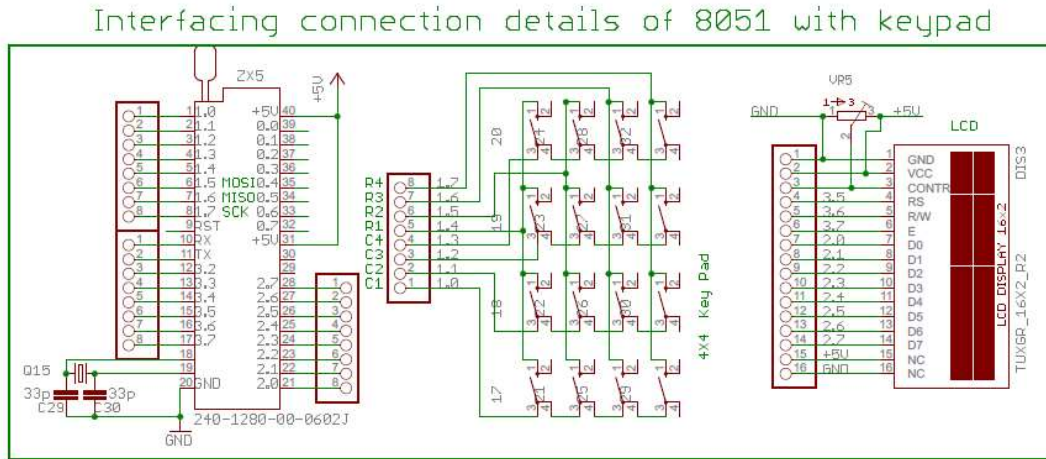
4. In row_finder1() function, we reversed the logic levels, means now all the Rows are 1 and columns are 0. Now row of the pressed button should be 0 because it has become connected (shorted) to the column whose button is pressed, and all the columns are at 0 logic. So we have scanned all rows for 0.



5. So whenever we find the Row at logic 0, means that is the row of pressed button. So now we have column no (got in step 2) and row no., and we can print no. of that button using lcd_data function.

Same procedure follows for every button press, and we are using while(1), to continuously check, whether button is pressed or not.

Circuit Diagram:



Program:

```

*/
#include<reg51.h>

#define KeyPort P1
#define LCD_PORT P2

sbit rs=P3^5;
sbit en=P3^7;
sbit D7=P2^7;
sbit rw=P3^6;

void busy();
void CMD_WRT(unsigned char);
void LCD_WRT(unsigned char *);
void DATA_WRT(unsigned char);
void DELAY();
void delay12(unsigned int time);
unsigned char check();

void main()
{
    unsigned char CMD[]={0x38,0x01,0x0f,0x06,0x80},TEMP1,i,j;
    unsigned char pass1[15];

    for(i=0;i<5;i++)
    {
        TEMP1=CMD[i];
        CMD_WRT(TEMP1);
    }
}
    
```

```

    }
    DELAY();
    CMD_WRT(0X01);
    CMD_WRT(0X80);
    LCD_WRT("4X4 KEYPAD");
    DELAY();

    while(1)
    {
    CMD_WRT(0X01);
    CMD_WRT(0X80);
    LCD_WRT("Enter the Key");
    DELAY();
    CMD_WRT(0XC0);
    for(j=0;j<15;j++)
    {
    pass1[j]=check();
    DATA_WRT(pass1[j]);
    DELAY();
    }
    }
}

void DELAY()
{
unsigned int X=6000000;
while(X--);
}
void busy()
{
D7=1;
rs=0;
rw=1;
while(D7!=0)
{
    en=0;
    en=1;
}
}

void CMD_WRT(unsigned char val)
{
busy();
LCD_PORT=val;
rs=0;
rw=0;
en=1;
en=0;
}

```

```

void LCD_WRT(unsigned char *string)
{
while(*string)
DATA_WRT(*string++);
}
void DATA_WRT(unsigned char ch)
{
busy();
LCD_PORT = ch;
rs=1;
rw=0;
en=1;
en=0;
}

unsigned char check()
{
unsigned char colloc=0,rowloc=0, colloc1,rowloc1;
unsigned char keypad[4][4]={'F','E','D','C',
'B','A','9','8',
'7','6','5','4',
'3','2','1','0'};

{

do
{
KeyPort = 0X0f;
do
{
colloc1 = KeyPort;
colloc1 &= 0x0F;
}while(colloc1 != 0x0F);

delay12(20);

do
{
colloc1=KeyPort;
colloc1 &= 0x0F;
}while(colloc1==0x0F);

delay12(20);

KeyPort = 0x0f;
colloc1=KeyPort;
colloc1 &= 0x0F;
}

```

```

if(colloc1==0x0E)
{
  colloc=0;
}
else if(colloc1==0x0D)
{
  colloc=1;
}
else if(colloc1==0x0B)
{
  colloc=2;
}
else if(colloc1==0x07)
{
  colloc=3;
}
KeyPort = 0xf0;
rowloc1 = KeyPort;
rowloc1 &= 0xf0;
if(rowloc1 == 0xE0)
{
  rowloc=0;
}

else if(rowloc1==0xD0)
{
  rowloc=1;
}
else if(rowloc1==0xB0)
{
  rowloc=2;
}
else if(rowloc1==0x70)
{
  rowloc=3;
}
// CMD_WRT(0XC0);
// DATA_WRT(keypad[rowloc][colloc]);

return(keypad[rowloc][colloc]);

}while(1);
}
}

void delay12(unsigned int time)
{
  unsigned int i;
  for(i=0;i<time;i++);
  for(i=0;i<10000;i++);
}

```


8 AND 16 BIT ARITHMETIC OPERATIONS

- a) Write an ALP program to perform 8 Bit arithmetic operations using MASM software and 8086.

AIM: -

To write an assembly language program for Addition of two 8-bit numbers.

COMPONENTS & EQUIPMENT REQUIRED: -

| S.No | Device | Range / Rating | Quantity (in No's) |
|------|--|----------------|--------------------|
| 1 | 8086 microprocessor kit/Win862 with PC | | 1 |
| 2 | Keyboard | | 1 |
| 3 | RPS | +5v | 1 |

PROGRAM FOR TWO 8-BIT NUMBERS:

A) ADDITION

- i) **Software**

```
MOV AL, 43
MOV BL, 11
ADD AL, BL
INT 03
```

- ii) **Hardware**

| MEMORY LOCATION | OP-CODE | LABLE | MNEMONIC OPERAND | COMMENTS |
|-----------------|---------|-------|------------------|----------|
| | | | MOV AL,43 | |
| | | | MOV BL,11 | |
| | | | ADD AL,BL | |
| | | | INT 3 | |

Observation Table

| Input | | Output | |
|----------|------|----------|------|
| Register | Data | Register | Data |
| AX | 4343 | AX | |
| BX | 1111 | | |

B) SUBTRACTION

8 Bit Subtraction

AIM: -

To write an assembly language program for subtraction of two 8-bit numbers.

COMPONENTS & EQUIPMENT REQUIRED: -

| S.No | Device | Range / Rating | Quantity (in No's) |
|------|--|----------------|--------------------|
| 1 | 8086 microprocessor kit/Win862 with PC | | 1 |
| 2 | Keyboard | | 1 |
| 3 | RPS | +5v | 1 |

PROGRAM:

i) **Software**

```
MOV AL, 43
MOV BL, 11
SUB AL, BL
INT 03
```

ii) **Hardware**

| MEMORY LOCATION | OP-CODE | LABLE | MNEMONIC OPERAND | COMMENTS |
|-----------------|---------|-------|---|----------|
| | | | MOV AL,43 MOV BL,11 SUB AL,BL INT 03 | |

Observation Table

| Input | | Output | |
|----------|------|----------|------|
| Register | Data | Register | Data |
| AX | 4343 | AX | 3232 |
| BX | 1111 | | |

C) MULTIPLICATION

8 Bit Multiplication

AIM: -

To write an assembly language program for multiplication of two 8-bit numbers.

COMPONENTS & EQUIPMENT REQUIRED: -

| S.No | Device | Range / Rating | Quantity (in No's) |
|------|---------------------------------------|----------------|--------------------|
| 1 | 8086 microprocessor kit/Win862with PC | | 1 |
| 2 | Keyboard | | 1 |
| 3 | RPS | +5v | 1 |

PROGRAM:

i) Software

```
MOV AL, 43
MOV BL, 11
MUL BL
INT 03
```

ii) Hardware

| MEMORY LOCATION | OP-CODE | LABEL | MNEMONIC OPERAND | COMMENTS |
|-----------------|---------|-------|---|----------|
| | | | MOV AL,43 MOV BL,11 MUL BL INT 3 | |

Observation Table

| Input | | Output | |
|----------|------|----------|------|
| Register | Data | Register | Data |
| AX | 4343 | AX | EA73 |
| BX | 1111 | DX | 047B |

D) DIVISION

i) 8 bit division

AIM:-

To write an assembly language program for division of two 8-bit numbers.

COMPONENTS & EQUIPMENT REQUIRED: -

| S.No | Device | Range / Rating | Quantity (in No's) |
|------|--|----------------|--------------------|
| 1 | 8086 microprocessor kit/Win862 with PC | | 1 |
| 2 | Keyboard | | 1 |
| 3 | RPS | +5v | 1 |

PROGRAM

i) Software

```
MOV AL, 10
MOV BL, 02
DIV BL
INT 03
```

ii) Hardware

| MEMORY LOCATION | OP-CODE | LABEL | MNEMONIC OPERAND | COMMENTS |
|-----------------|---------|-------|------------------|----------|
| | | | MOV AL,10 | |
| | | | MOV BL,02 | |
| | | | DIV BL | |
| | | | INT 3 | |

Observation Table

| Input | | Output | |
|----------|------|----------|------|
| Register | Data | Register | Data |
| AX | 4343 | AX | 0003 |
| BX | 1111 | DX | 03F2 |

RESULT:

- b) Write an ALP program to perform 16 Bit arithmetic operations using MASM software and 8086.

AIM: -

To write an assembly language program for addition of two 16-bit numbers.

COMPONENTS & EQUIPMENT REQUIRED: -

| S.No | Device | Range / Rating | Quantity (in No's) |
|------|--|----------------|--------------------|
| 1 | 8086 microprocessor kit/Win862 with PC | | 1 |
| 2 | Keyboard | | 1 |
| 3 | RPS | +5v | 1 |

ADDITION:

- i) **Software**

```
MOV AX, 4343
MOV BX, 1111
ADD AX, BX
INT 03
```

- ii) **Hardware**

| MEMORY LOCATION | OP-CODE | LABLE | MNEMONIC OPERAND | COMMENTS |
|-----------------|---------|-------|------------------|----------|
| | | | MOV AX,4343 | |
| | | | MOV BX,1111 | |
| | | | ADD AX,BX | |
| | | | INT 3 | |

Observation Table

| Input | | Output | |
|----------|------|----------|------|
| Register | Data | Register | Data |
| AX | 4343 | AX | |
| BX | 1111 | | |

SUBTRACTION:

AIM: -

To write an assembly language program for subtraction of two 16-bit numbers.

COMPONENTS & EQUIPMENT REQUIRED: -

| S.No | Device | Range / Rating | Quantity (in No's) |
|------|--|----------------|--------------------|
| 1 | 8086 microprocessor kit/Win862 with PC | | 1 |
| 2 | Keyboard | | 1 |
| 3 | RPS | +5v | 1 |

PROGRAM:

i) **Software**

```
MOV AX, 4343
MOV BX, 1111
SUB AX, BX
INT 03
```

ii) **Hardware**

| MEMORY LOCATION | OP-CODE | MNEMONIC OPERAND | COMMENTS |
|-----------------|---------|---|----------|
| | | MOV AX,4343 MOV BX,1111 SUB AX,BX INT 03 | |

Observation Table

| Input | | Output | |
|----------|------|----------|------|
| Register | Data | Register | Data |
| AX | 4343 | AX | 3232 |
| BX | 1111 | | |

C) MULTIPLICATION

AIM: -

To write an assembly language program for multiplication of two 16-bit numbers.

COMPONENTS & EQUIPMENT REQUIRED: -

| S.No | Device | Range / Rating | Quantity (in No's) |
|------|---------------------------------------|----------------|--------------------|
| 1 | 8086 microprocessor kit/Win862with PC | | 1 |
| 2 | Keyboard | | 1 |
| 3 | RPS | +5v | 1 |

PROGRAM:

i) Software

```
MOV AX, 4343
MOV BX, 1111
MUL BX
INT 03
```

ii) Hardware

| MEMORY LOCATION | OP-CODE | LABEL | MNEMONIC OPERAND | COMMENTS |
|-----------------|---------|-------|---|----------|
| | | | MOV AX,4343 MOV BX,1111 MUL BX INT 3 | |

Observation Table

| Input | | Output | |
|----------|------|----------|------|
| Register | Data | Register | Data |
| AX | 4343 | AX | EA73 |
| BX | 1111 | DX | 047B |

D) DIVISION

AIM:-

To write an assembly language program for division of two 16-bit numbers.

COMPONENTS & EQUIPMENT REQUIRED: -

| S.No | Device | Range / Rating | Quantity (in No's) |
|------|--|----------------|--------------------|
| 1 | 8086 microprocessor kit/Win862 with PC | | 1 |
| 2 | Keyboard | | 1 |
| 3 | RPS | +5v | 1 |

PROGRAM

```

i)      Software
        MOV AX, 0080
           MOV BX, 0008
           DIV BX
           INT 03
    
```

ii) **Hardware**

| MEMORY LOCATION | OP-CODE | LABEL | MNEMONIC OPERAND | COMMENTS |
|-----------------|---------|-------|------------------|----------|
| | | | MOV AX,0080 | |
| | | | MOV BX,0008 | |
| | | | DIV BX | |
| | | | INT 3 | |

Observation Table

| Input | | Output | |
|----------|------|----------|------|
| Register | Data | Register | Data |
| AX | 4343 | AX | 0003 |
| BX | 1111 | DX | 03F2 |

RESULT:

PRE LAB QUESTIONS:

1. How many bit 8086 microprocessor is?
2. What is the size of data bus of 8086?
3. What is the size of address bus of 8086?
4. What is the max memory addressing capacity of 8086?
5. Which are the basic parts of 8086?

POST LAB QUESTIONS:

1. How to move data from one register to other
2. To swapping the data what type register used
3. What are the advantages of maximum mode

PROGRAMS TO SORT NUMBERS

a) Write an ALP program to perform ascending order using 8086

AIM:-

Write an assembly language Program to sort the given numbers in ascending order

COMPONENTS & EQUIPMENT REQUIRED: -

| S.No | Device | Range / Rating | Quantity (in No's) |
|------|---|----------------|--------------------|
| 1 | 8086 microprocessor kit/win 862 with PC | | 1 |
| 2 | Keyboard | | 1 |
| 3 | RPS | +5v | 1 |

PROGRAM:

i) **Software**

```

MOV AX,0000H
MOV CH,0004H
DEC CH
UP1 : MOV CL, CH
      MOV SI, 2000H
UP:   MOV AL,[SI]
      INC SI
      CMP AL,[SI]
      JC DOWN
      XCHG AL,[SI]
      DEC SI
      MOV [SI], AL
      INC SI
DOWN: DEC CL
      JNZ UP
      DEC CH
      JNZ UP1
      INT 3

```

ii) Hardware

| MEMORY LOCATION | OP-CODE | LABEL | MNEMONIC OPERAND | COMMENTS |
|-----------------|---------|-------|------------------|----------|
| | | | MOV AX, 0000 | |
| | | UP1: | MOV CH, 0004 | |
| | | | DEC CH | |
| | | UP: | MOV CL, CH | |
| | | | MOV SI, 2000 | |
| | | | MOV AL, [SI] | |
| | | | INC SI | |
| | | | CMP AL, [SI] | |
| | | | JC DOWN | |
| | | DOWN: | XCHG AL, [SI] | |
| | | | DEC SI | |
| | | | MOV [SI], AL | |
| | | | INC SI | |
| | | | DEC CL | |
| | | | JNZ UP | |
| | | | DEC CH | |
| | | | JNZ UP1 | |
| | | | INT 03 | |

Observation Table:

| Input | | Output | |
|-----------------|------|-----------------|------|
| MEMORY LOCATION | Data | MEMORY LOCATION | Data |
| 2000 | | 2000 | |
| 2001 | | 2001 | |
| 2002 | | 2002 | |
| 2003 | | 2003 | |

RESULT:

b) Write an ALP program to perform descending order using 8086

AIM:-

Write an assembly language Program to sort the given numbers in descending order

COMPONENTS & EQUIPMENT REQUIRED: -

| S.No | Device | Range / Rating | Quantity (in No's) |
|------|--|----------------|--------------------|
| 2 | 8086 microprocessor kit/Win862 with PC | | 1 |
| | Keyboard | | 1 |
| 3 | RPS | +5v | 1 |

PROGRAM:

i) Software

```

MOV AX,0000
MOV CH,0004
DEC CH
UP1 : MOV CL, CH
      MOV SI, 2000
UP:   MOV AL,[SI]
      INC SI
      CMP AL,[SI]
      JNC DOWN
      XCHG AL,[SI]
      DEC SI
      MOV [SI],AL
      INC SI
DOWN: DEC CL
      JNZ UP
      DEC CH
      JNZ UP1
      INT 3
    
```

ii) Hardware

| MEMORY LOCATION | OP-CODE | LABEL | MNEMONIC OPERAND | COMMENTS |
|-----------------|---------|-------|------------------|----------|
| | | UP1: | MOV AX, 0000 | |
| | | | MOV CH, 0004 | |
| | | UP: | DEC CH | |
| | | | MOV CL,CH | |
| | | | MOV SI,2000 | |
| | | | MOV AL,[SI] | |
| | | | INC SI | |
| | | | CMP AL,[SI] | |
| | | DOWN: | JNC DOWN | |
| | | | XCHG AL,[SI] | |
| | | | DEC SI | |
| | | | MOV [SI],AL | |
| | | | INC SI | |
| | | | DEC CL | |
| | | | JNZ UP | |
| | | | DEC CH | |
| | | | JNZ UP1 | |
| | | | INT 3 | |

Observation Table

| Input | | Output | |
|-----------------|------|-----------------|------|
| MEMORY LOCATION | Data | MEMORY LOCATION | Data |
| 2000 | | 2000 | |
| 2001 | | 2001 | |
| 2002 | | 2002 | |
| 2003 | | 2003 | |

RESULT:

PRE LAB QUESTIONS:

1. What are the functions of BIU?
2. What are the functions of EU?
3. How many pin IC 8086 is?
4. What IC8086 is?
5. What is the size of instruction queue in 8086?

POST LAB QUESTIONS:

1. How clock signal is generated in 8086
2. What is the maximum internal clock frequency of 8086?
3. What is the need for Port

PROGRAM FOR STRING MANIPULATIONS OPERATIONS

- a) Write an ALP program to insert or delete a byte in the given string.

INSERT A BYTE IN A GIVEN STRING**AIM:-**

To write a ALP for insert a new byte in a given string

COMPONENTS & EQUIPMENT REQUIRED: -

| S.No | Device | Range / Rating | Quantity (in No's) |
|------|--------------------------------------|----------------|--------------------|
| 1 | 8086 microprocessor kit/MASM with PC | | 1 |
| 2 | Keyboard | | 1 |
| 3 | RPS | +5v | 1 |

PROGRAM:

- i) **Software**

```

ASSUME CS: CODE
CODE SEGMENT
    START:  MOV SI,2000H
            MOV DI,3000H
            MOV BX,5000H
            MOV CX,0005H
            CLD
    L1:     MOV AL,[SI]
            CMP AL,[BX]
            JZ L2
            MOVSB
            LOOP L1
            JMP L3
    L2:     MOVSB
            MOV BX,7000H
            MOV AL,[BX]
            MOV [DI],AL
            DEC CX
            INC DI
            REP MOVSB
    L3:     INT 3
CODE ENDS
END START

```

ii) **Hardware**

| MEMORY LOCATION | OP-CODE | LABEL | MNEMONIC OPERAND | COMMENTS |
|-----------------|---------|-------|------------------|----------|
| | | | MOV SI,2000 | |
| | | | MOV DI,3000 | |
| | | | MOV BX,5000 | |
| | | | MOV CX,0005 | |
| | | | CLD | |
| | | L1: | MOV AL,[SI] | |
| | | | CMP AL,[BX] | |
| | | | JZ L2 | |
| | | | MOVSB | |
| | | | LOOP L1 | |
| | | | JMP L3 | |
| | | L2: | MOVSB | |
| | | | MOV BX,7000 | |
| | | | MOV AL,[BX] | |
| | | | MOV [DI],AL | |
| | | | DEC CX | |
| | | | INC DI | |
| | | | REP | |
| | | | MOVSB | |
| | | | INT 3 | |
| | | L3: | | |

Observation Table

| Input | | Output | |
|-----------------|------|-----------------|------|
| MEMORY LOCATION | Data | MEMORY LOCATION | Data |
| 2000 | | 3000 | |
| 2001 | | 3001 | |
| 2002 | | 3002 | |
| 2003 | | 3003 | |
| 2004 | | 3004 | |
| 5000 | | 3005 | |
| 7000 | | | |

RESULT:

DELETE A BYTE IN A GIVEN STRING

AIM:-

To write a alp for delete a byte in a given string

COMPONENTS & EQUIPMENT REQUIRED: -

| S.No | Device | Range / Rating | Quantity (in No's) |
|------|--------------------------------------|----------------|--------------------|
| 1 | 8086 microprocessor kit/MASM with PC | | 1 |
| 2 | Keyboard | | 1 |
| 3 | RPS | +5v | 1 |

PROGRAM:

```

i)   Software
      ASSUME CS:CODE
      CODE SEGMENT
      START:    MOV SI,2000H
              MOV DI,3000H
              MOV BX,5000H
              MOV CX,0005H
              CLD
      L1:    MOV AL,[SI]
              CMP AL,[BX]
              JZ L2
              MOVSB
              LOOP L1
              JMP L3
      L2:    INC SI
              DEC CX
              REP MOVSB
      L3:    INT 03H
      CODE ENDS
      END START
  
```

ii) Hardware

| MEMORY LOCATION | OP-CODE | LABEL | MNEMONIC OPERAND | COMMENTS |
|-----------------|---------|-------|--|----------|
| | | L1: | MOV SI,2000 MOV DI,3000 MOV BX,5000 MOV CX,0005 CLD MOV AL,[SI] CMP AL,[BX] JZ L2 MOVSB LOOP L1 JMP L3 | |

| | | | | |
|--|--|-----|---|--|
| | | L2: | INC SI DEC CX REP MOVSB INT 3 | |
| | | L3: | | |

Observation Table

| Input | | output | |
|-----------------|------|-----------------|------|
| MEMORY LOCATION | Data | MEMORY LOCATION | Data |
| 2000 | | 3000 | |
| 2001 | | 3001 | |
| 2002 | | 3002 | |
| 2003 | | 3003 | |
| 2004 | | | |
| 5000 | | | |

RESULT:

PRE LAB QUESTIONS:

1. What do you mean by assembler directives?
2. What .model small stands for?
3. What is the supply requirement of 8086?
4. What is the relation between 8086 processor frequency & crystal Frequency?
5. Functions of Accumulator or AX register?

LAB ASSIGNMENT:

1. Write an alp for insert or delete a byte in a given string with SI memory location is 4000 and DI location is 6000?
2. Write an alp for moving or reversing the given string with the length of the string is 12?

POST LAB QUESTIONS:

1. Which interrupts are generally used for critical events?
2. Which Stack is used in 8086?
3. What is SIM and RIM instructions