INDEX

INTRODUCTION

Electrical motors are *electro-mechanical devices* which convert electrical energy to mechanical energy. They are inevitable parts of any industrial system. Practically, any electro-mechanical system is unimaginable without motors. So, understanding electric motors are of utmost importance for an engineering student. With the progress of engineering, the applications of motors are increasing day by day. A recent study shows that in USA alone there are *2 billion* induction motors in use. Hence, motors actually shape our modern lifestyle.

Among numerous kinds of motors, *Stepper Motor* is of great practical value. Commercially, stepper motors are used in *floppy disk drives*, *flatbed scanners*, *computer printers*, *plotters*, *slot machines*, *image scanners*, *compact disc drives*, *intelligent lighting* and many more devices. Computer-controlled stepper motors are a type of *motion-control positioning system*.

In the field of *lasers* and *optics* they are frequently used in precision positioning equipment such as linear actuators, linear stages, rotation stages and mirror mounts. Other uses are in *packaging machinery* and positioning of valve pilot stages for *fluid control systems*.

So one can conclude, understanding stepper motor and its operation gives a person greater insight in all those systems.

In our project required for *EEE 316*, we have investigated stepper motor and its control via computer. We have used micro-controller *ATMega32* to precisely control the stepper motor. ATMega32 receives data from computer via *serial communication link* and gives appropriate command output for controlling stepper motor. The stepper motor receives additional power from power source via *ULN2003*.

We have gone through many websites, several texts, related journals and papers in our study for the implementation of the project. We also took suggestions from our honorable instructor. Our project led us to the discovery of a new horizon of electrical engineering and ignited our passion for building electrical systems.

STEPPER MOTOR

About Stepper Motor

A *stepper motor* is a brushless DC electric motor that divides a full rotation into a number of *equal steps*. The motor's position can then be commanded to move and hold at one of these steps without any feedback sensor (an open-loop controller), as long as the motor is carefully sized to the application.

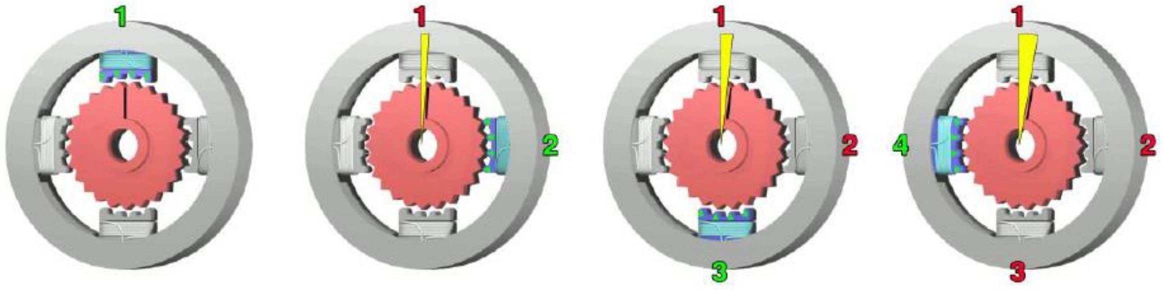


Figure 1: Animation of a simplified stepper motor (Unipolar)

- Frame 1
  The top electromagnet (1) is turned on, attracting the nearest teeth of the gear-shaped iron rotor. With the teeth aligned to electromagnet (1), they will be slightly offset from right electromagnet (2).

- Frame 2
  The top electromagnet (1) is turned off. The right electromagnet (2) is energized, pulling the teeth into alignment with it. This results in a rotation of 3.6° in this example.

- Frame 3
  The bottom electromagnet (3) is energized, thus, similarly another 3.6° rotation occurs.

- Frame 4
  The left electromagnet (4) is energized, rotating again by 3.6°. When the top electromagnet (1) is again enabled, the rotor will have rotated by one tooth position.

  Since there are 25 teeth, it will take 100 steps to make a full rotation in this example.

❒ Fundamentals of Operation

DC brush motors rotate continuously when voltage is applied to their terminals. On the contrary, stepper motors effectively have multiple *toothed* electromagnets arranged around a central gear-shaped piece of iron. The electromagnets are energized by an external control circuit, such as a microcontroller.

To make the motor shaft turn, *first*, one electromagnet is given power, which makes the gear's teeth magnetically attracted to the electromagnet's teeth. When the gear's teeth are aligned to the first electromagnet, they are slightly *offset* from the next electromagnet. So, when the next electromagnet is turned on and the first is turned off, the gear rotates slightly to align with the next one and from there, the process is repeated. Each of those slight rotations is called a ***step***. An integer number of steps make a full rotation. In this way, the motor can be turned by a precise angle.

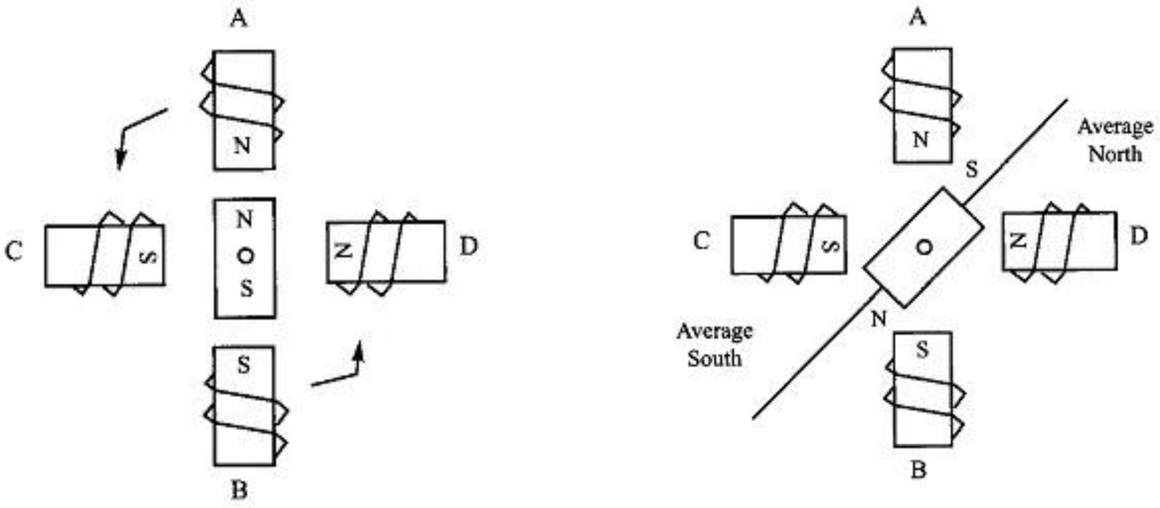


Figure 2: Operation of a stepper motor

❒ Types

Practically, there are four main types of stepper motors –

- Permanent Magnet Stepper Motor
- Variable Reluctance Stepper Motor
- Hybrid Synchronous Stepper Motor
- Lavet Type Stepping Motor

❒ Winding Arrangement

There are two basic winding arrangements for the coils in a two phase stepper motor — Bipolar and Unipolar. We have used unipolar motor in our project.

Two leads on each of the four coils of a stepper motor can be brought out in different ways…
— All eight leads can be taken out of the motor separately.
— Leads can be taken out by connecting A & C together and B & D together forming two coils.

If the coil ends are brought out as shown in *Figure 3*, then the motor is called a *bipolar* motor and if the wires are brought out as shown in *Figure 4* or *5*, with one or two *center tap(s)*, it is called a *unipolar* motor.

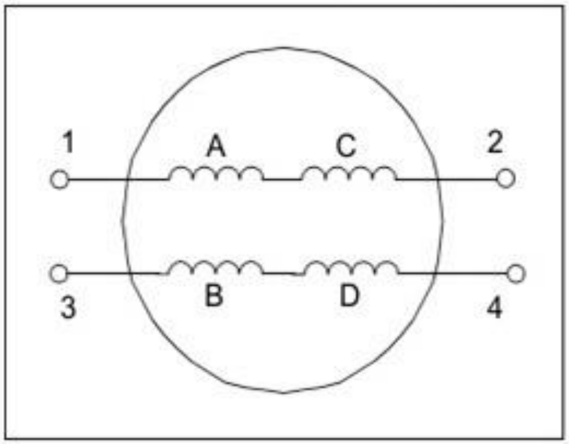


Figure 3: Bipolar Arrangement
(4-wire connection)

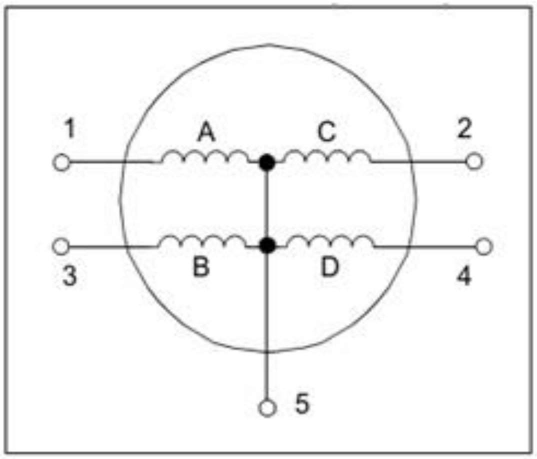


Figure 4: Unipolar Arrangement
(5-wire connection)

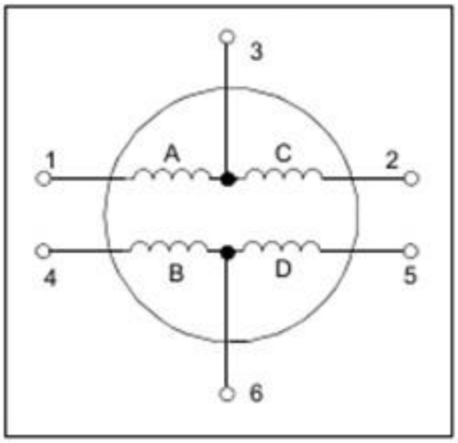


Figure 5: Unipolar Arrangement
(6-wire connection)

❒ Torque-Speed Characteristics

The speed of a stepper motor depends upon the rate at which the coils are turned on and off and is termed as the ***Step Rate***. The maximum step rate, as well as the maximum speed, depends upon the inductance of the stator coils.

Figure 6 shows the equivalent circuit of a stator winding and the relation between current rise and winding inductance. It takes a longer time to build the rated current in a winding with greater inductance compared to a winding with lesser inductance. So, when using a motor with higher winding inductance, sufficient time needs to be given for current to build up before the next step command is issued. If the time between two step commands is less than the current build-up time, it results in a ***slip***, i.e., *the motor misses a step*.

Unfortunately, the inductance of the winding is not well documented in most of the stepper motor data sheets. In general, for smaller motors, the inductance of the coil is much less than its resistance and the time constant is less. With a lower time constant, current rise in the coil will be faster, which enables a higher step rate. Using a *Resistance-Inductance (RL) Drive* can achieve a higher step rate in motors with higher inductance.

The best way to decide the maximum speed is by studying the *torque vs. step rate characteristics* of a particular stepper motor, when step rate is expressed in pulse per second or pps (*Figure 7*). ***Pull-in torque*** is the *maximum* load torque that the motor can start or stop instantaneously without miss-stepping. ***Maximum self-starting frequency*** is the maximum step-rate at which the motor can start instantaneously at no-load without miss-stepping. ***Pull-out torque*** is the torque available when the motor is continuously accelerated to the operating point. From the graph, it can be concluded that, for this particular motor, the maximum self-starting frequency is 200 pps, while at no-load, this motor can be accelerated up to 275 pps.

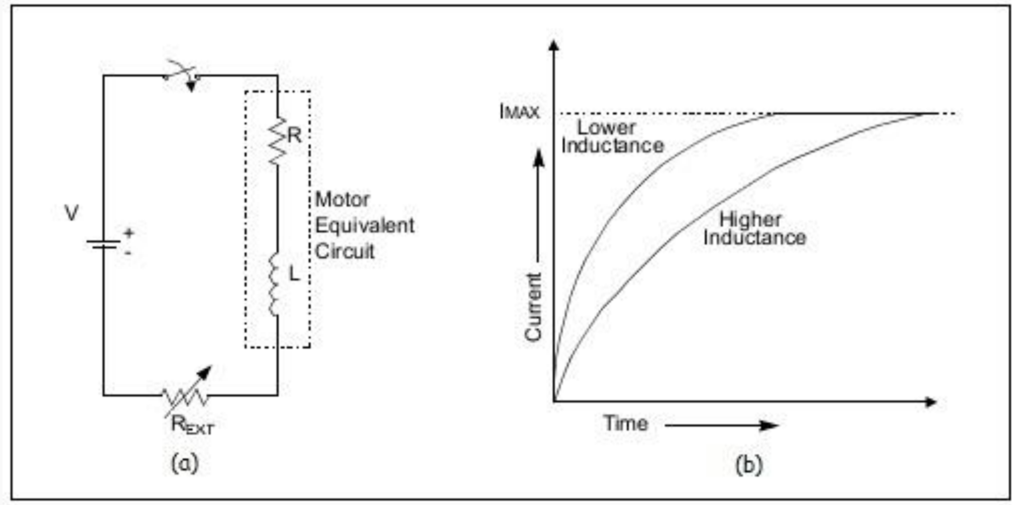


Figure 6: (a) Motor Equivalent Circuit
(b) Current Rise Rate in Stator Winding

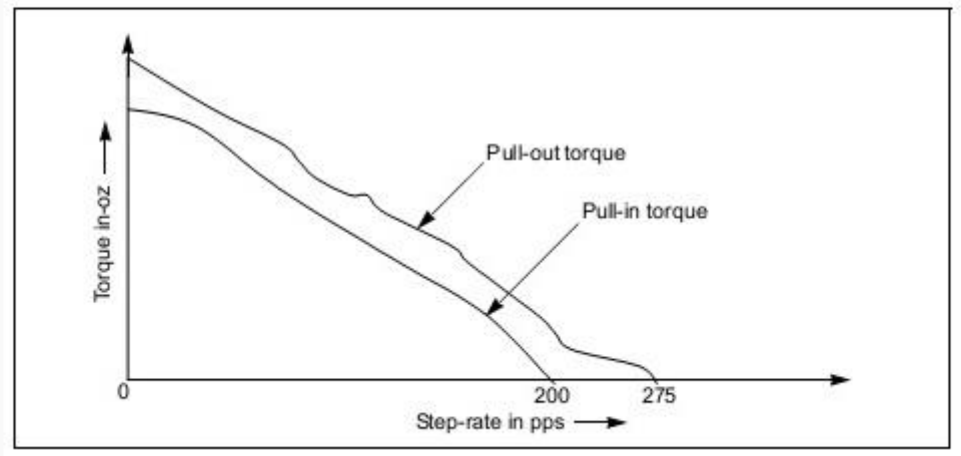


Figure 7: A typical Torque vs. Step Rate Characteristics Curve

❐ Ratings and Specifications

Stepper motors nameplates typically give only the winding current and occasionally the voltage and winding resistance. A stepper motor's low speed torque will vary directly with current. How quickly the torque falls off at faster speeds depends upon the winding inductance and the driving circuitry it is attached to, specially the driving voltage. Stepper motors should be sized according to published torque curve, which is specified by the manufacturer at particular driving voltages or using their own driving circuitry.

❐ Stepper Motor Systems

A Stepper Motor System, often combined with some type of user interface (Host Computer, PLC or Dumb Terminal), consists of three basic elements –

- Indexer
  The Indexer (or Controller) is a microprocessor capable of generating step pulses and direction signals for the driver. In addition, the indexer is typically required to perform many other sophisticated command functions.

- Driver
  The Driver (or Amplifier) converts the indexer command signals into the power necessary to energize the motor windings. There are numerous types of drivers, with different voltage and current ratings and construction technology. Not all drivers are suitable to run all motors, so when designing a Motion Control System, the driver selection process is critical.

- Stepper Motor
  The stepper motor is an electromagnetic device that converts digital pulses into mechanical shaft rotations. Advantages of step motors are low cost, high reliability, high torque at low speeds and a simple, rugged construction that operates at almost any environment. The main disadvantages in using a stepper motor is the resonance effect often exhibited at low speeds and decreasing torque with increasing speed.

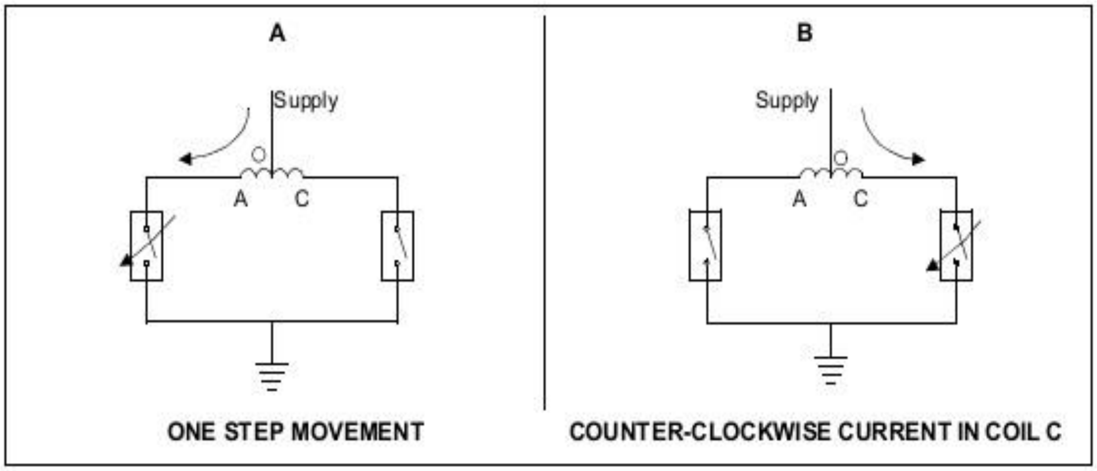
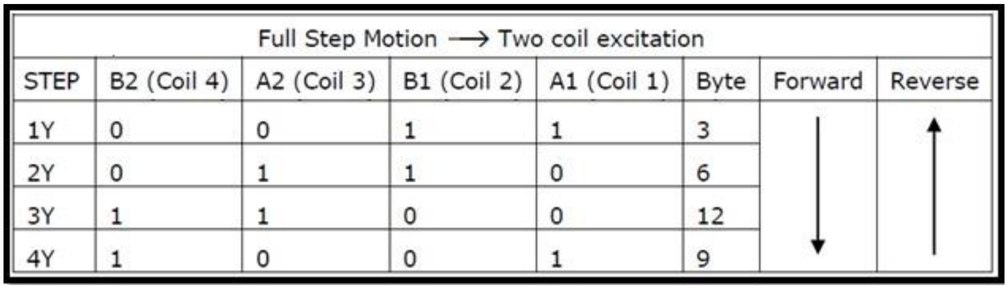


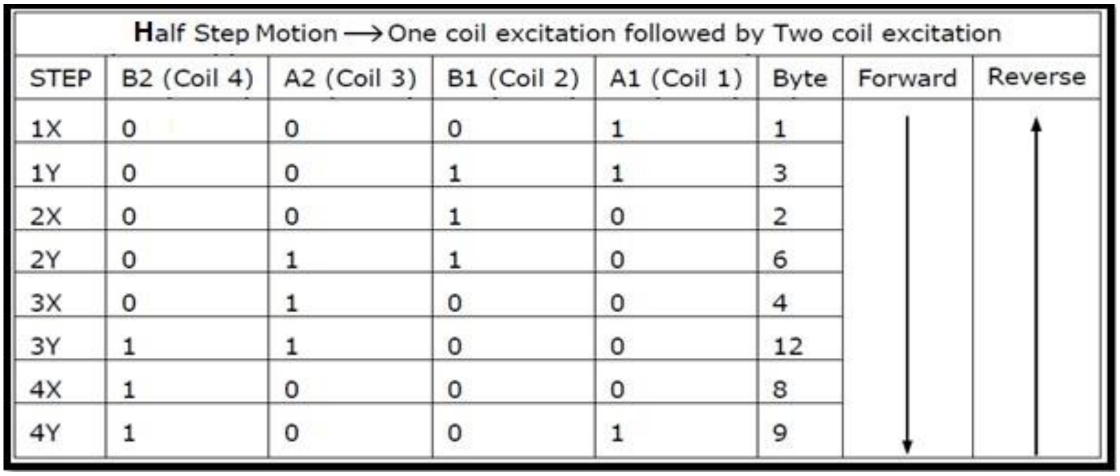
Figure 8: Simplified Drives for the Unipolar Motor



Full Step Motion ⟶ Two coil excitation

STEP	B2 (Coil 4)	A2 (Coil 3)	B1 (Coil 2)	A1 (Coil 1)	Byte	Forward	Reverse
1Y	0	0	1	1	3	↓	↑
2Y	0	1	1	0	6		
3Y	1	1	0	0	12		
4Y	1	0	0	1	9		

Table 1: Truth Table for Operation in Two Coil Excitation (Full Step Operation)



Half Step Motion ⟶ One coil excitation followed by Two coil excitation

STEP	B2 (Coil 4)	A2 (Coil 3)	B1 (Coil 2)	A1 (Coil 1)	Byte	Forward	Reverse
1X	0	0	0	1	1	↓	↑
1Y	0	0	1	1	3		
2X	0	0	1	0	2		
2Y	0	1	1	0	6		
3X	0	1	0	0	4		
3Y	1	1	0	0	12		
4X	1	0	0	0	8		
4Y	1	0	0	1	9		

Table 2: Truth Table for Operation in Wave Motion (Half Step Operation)

ATMega32

About ATMega32

The ATMega32 is a low-power CMOS 8 bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATMega32 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

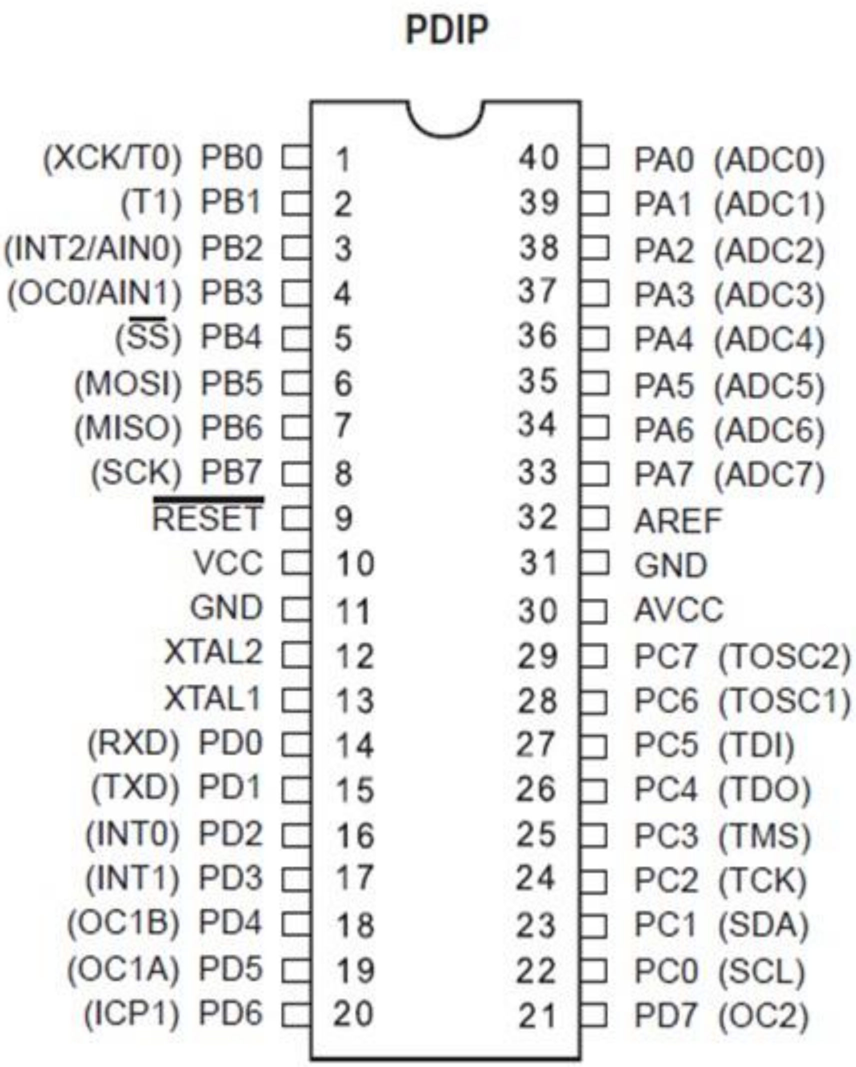


Figure 9: Pin Diagram of the Microcontroller IC ATMega32

❐ Pin Description

There are 40 pins in ATMega32 IC all having specific tasks…

- *VCC* ⇒ Digital Supply Voltage
- *GND* ⇒ Ground Potential
- *Port A (PA0 — PA7)*
  Port A serves either as the analog inputs to the A/D Converter or, as an 8 bit bidirectional I/O port. Port pins can provide internal pull-up resistors (selected for each bit). The Port A output buffers have symmetrical drive characteristics with both high sink and source capability. When pins PA0 to PA7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.
- *Port B (PB0 — PB7)*
  Port B is an 8 bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running. Port B also serves the functions of various special features.
- *Port C (PC0 — PC7)*
  Port C is an 8 bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high *sink* & *source* capability. As input, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running. If the JTAG interface is enabled, the pull-up resistors on pins PC5(TD I), PC3(TMS) and PC2(TCK) will be activated even if a reset occurs. The TD0 pin is tri-stated unless TAP states that shift out data are entered. Port C also serves the functions of the JTAG interface and other special features.
- *Port D (PD0 — PD7)*
  Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running. Port D also serves the functions of various special features.
- *RESET*
  Reset Input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. The minimum pulse length is given in the datasheet table. Shorter pulses are not guaranteed to generate a reset.
- *XTAL1*
  Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

- *XTAL2*
  Output from the inverting oscillator amplifier.

- *AVCC*
  AVCC is the supply voltage pin for Port A and the A/D Converter. It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter.

- *AREF*
  AREF is the analog reference pin for the A/D Converter.

USART

About USART

The full meaning of USART is — the *Universal Synchronous and Asynchronous Serial Receiver and Transmitter*. It is a highly flexible serial communication device. The main features are —

- Full Duplex Operation (Independent Serial Receive and Transmit Registers)
- Asynchronous or Synchronous Operation
- Master or Slave Clocked Synchronous Operation
- High Resolution Baud Rate Generator
- Supports Serial Frames with 5, 6, 7, 8 or 9 Data Bits and 1 or 2 Stop Bits
- Odd or Even Parity Generation and Parity Check Supported by Hardware
- Data Overrun Detection
- Framing Error Detection
- Noise Filtering includes False Start Bit Detection and Digital Low Pass Filter
- Three Separate Interrupts on TX Complete, TX Data Register Empty and RX Complete
- Multi-processor Communication Mode
- Double Speed Asynchronous Communication Mode

❒ Overview

A simplified block diagram of the USART transmitter is shown in Figure 11. CPU accessible I/O Registers and I/O pins are shown in bold.

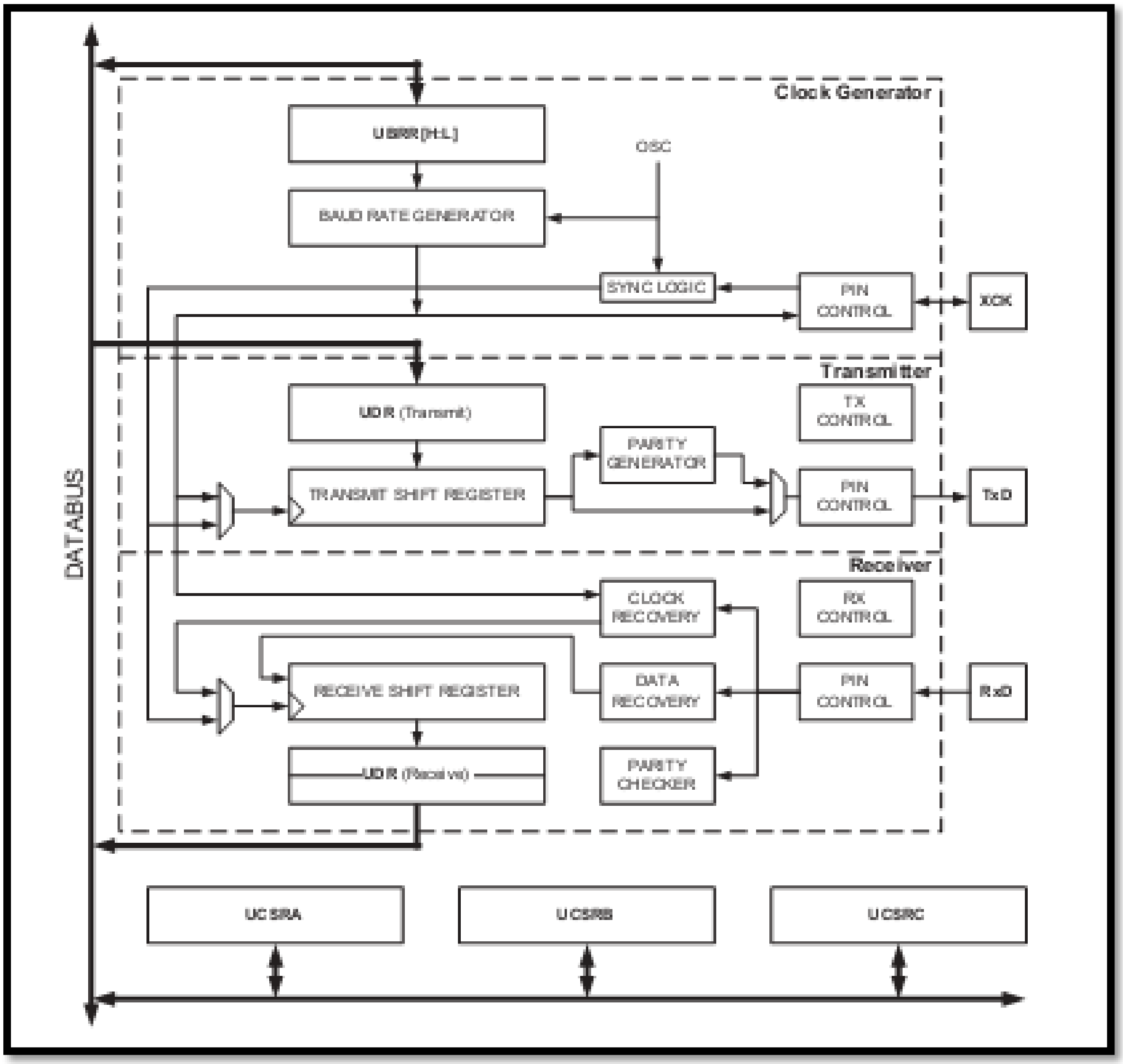


Figure 10: Simplified Internal Block Diagram of USART

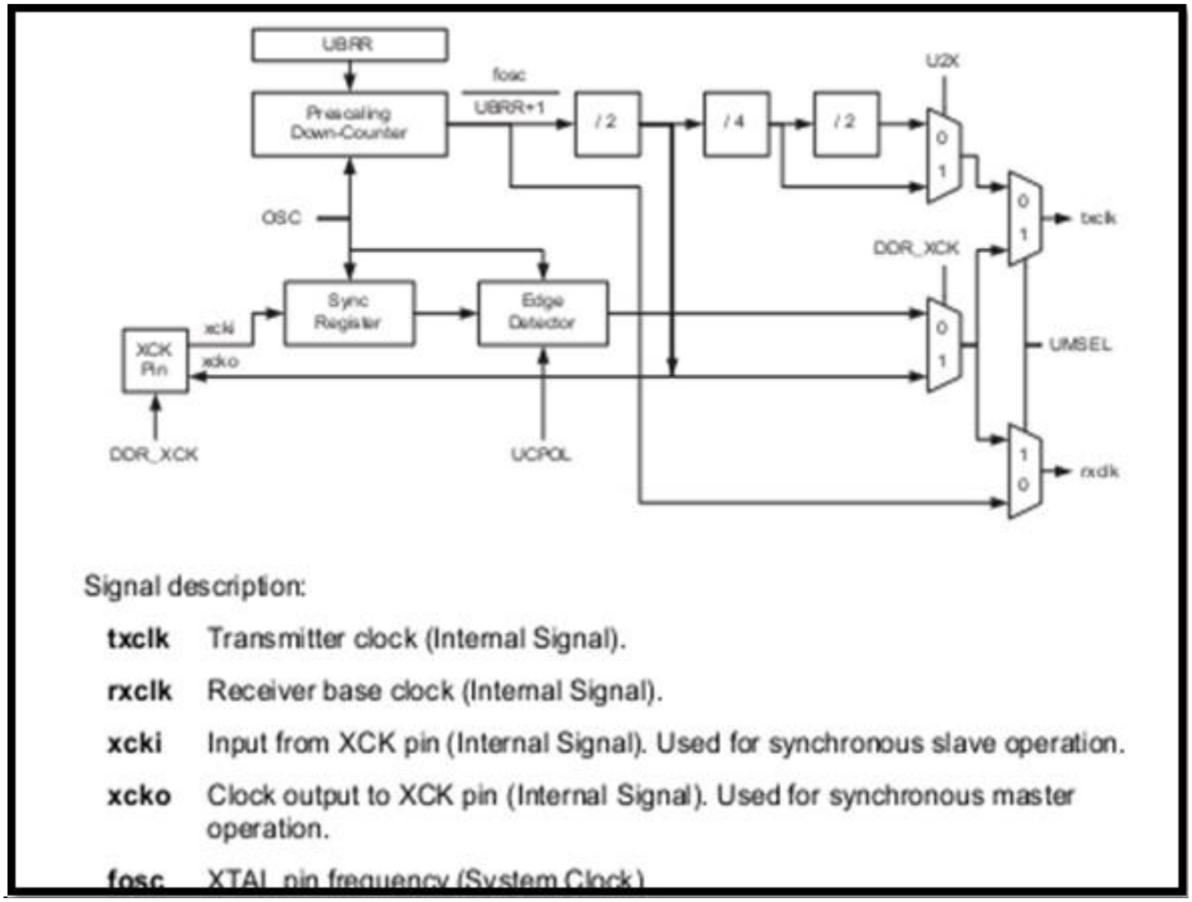
❒ Clock Generation

The clock generation logic generates the base clock for the Transmitter and Receiver. The USART supports four modes of clock operation —

- Normal Asynchronous Mode
- Double Speed Asynchronous Mode
- Master Synchronous Mode
- Slave Synchronous Mode.

The UMSEL bit in USART Control and Status Register C (UCSRC) selects between asynchronous and synchronous operation. Double Speed mode is controlled by the U2X found in the UCSRA Register. When using Synchronous mode (UMSEL = 1), the Data Direction Register for the XCK pin (DDR_XCK) controls whether the clock source is internal (Master mode) or external (Slave mode). The XCK pin is only active when using Synchronous mode.

Figure 11 shows a block diagram of the clock generation logic —



Signal description:

txclk	Transmitter clock (Internal Signal).
rxclk	Receiver base clock (Internal Signal).
xcki	Input from XCK pin (Internal Signal). Used for synchronous slave operation.
xcko	Clock output to XCK pin (Internal Signal). Used for synchronous master operation.
fosc	XTAL pin frequency (System Clock).

Figure 11: Clock Generation Logic Block Diagram

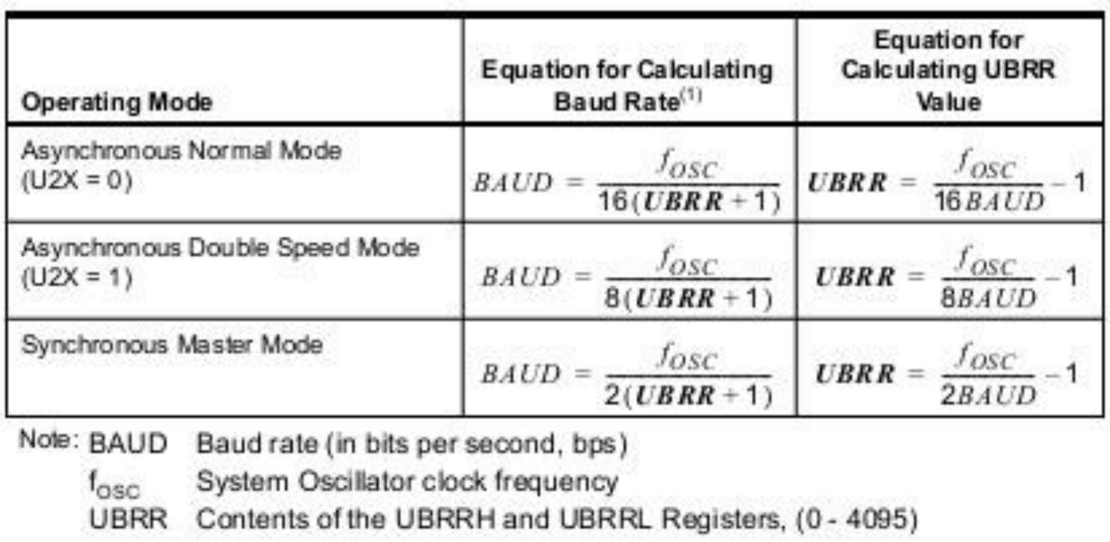
❒ Internal Clock Generation

The Baud Rate Generator

Internal clock generation is used for the asynchronous and the synchronous master modes of operation. The USART Baud Rate Register (UBRR) and the down-counter connected to it function as a programmable pre-scalar or baud rate generator.

The down-counter, running at system clock (f_{osc}), is loaded with the UBRR value each time the counter has counted down to zero or when the UBRRL Register is written. A clock is generated each time the counter reaches zero. This clock is the baud rate generator clock output $\left(= \mathbf{f_{osc}}/_{(\mathbf{UBRR}+1)}\right)$. The Transmitter divides the baud rate generator clock output by 2, 8 or 16 depending on mode. The baud rate generator output is used directly by the receiver's clock and data recovery units. However, the recovery units use a state machine that uses 2, 8 or 16 states depending on mode set by the state of the UMSEL, U2X and DDR_XCK bits.

Table 3 contains equations for calculating the baud rate (in bits/second) and for calculating the UBRR value for each mode of operation using an internally generated clock source.



Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \dfrac{f_{OSC}}{16(\boldsymbol{UBRR}+1)}$	$\boldsymbol{UBRR} = \dfrac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \dfrac{f_{OSC}}{8(\boldsymbol{UBRR}+1)}$	$\boldsymbol{UBRR} = \dfrac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \dfrac{f_{OSC}}{2(\boldsymbol{UBRR}+1)}$	$\boldsymbol{UBRR} = \dfrac{f_{OSC}}{2BAUD} - 1$

Note:		
	BAUD	Baud rate (in bits per second, bps)
	f_{OSC}	System Oscillator clock frequency
	UBRR	Contents of the UBRRH and UBRRL Registers, (0 - 4095)

Table 3: Equations for calculating Baud Rate & UBRR Values

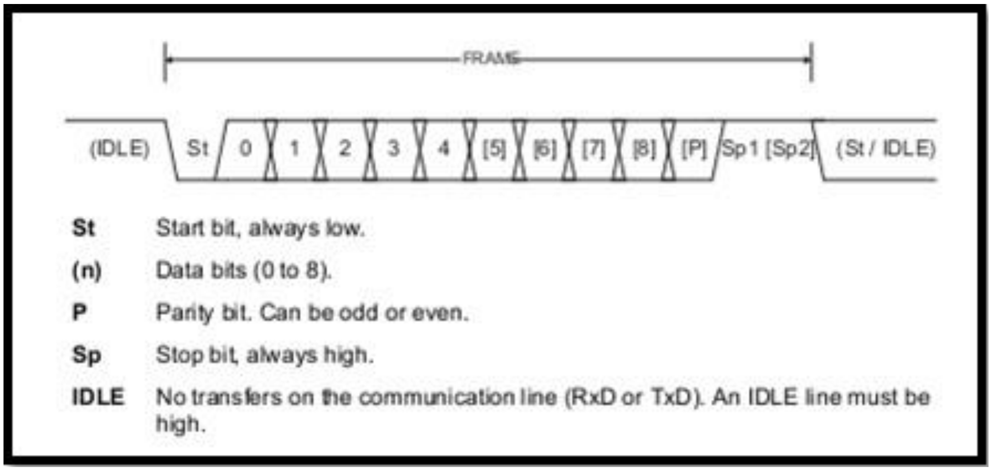
❐ Frame Formats

A serial frame is defined to be one character of data bits with synchronization bits (start and stop bits), and optionally a parity bit for error checking. The USART accepts all 30 combinations of the following as valid frame formats —

- 1 start bit
- 5, 6, 7, 8 or 9 data bits
- No even or odd parity bit
- 1 or 2 stop bits

A frame starts with the start bit followed by the least significant data bit. Then the next data bits, up to a total of nine, are succeeding, ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, it can be directly followed by a new frame, or the communication line can be set to an idle (high) state.

Figure 12 illustrates the possible combinations of the frame formats —



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high.

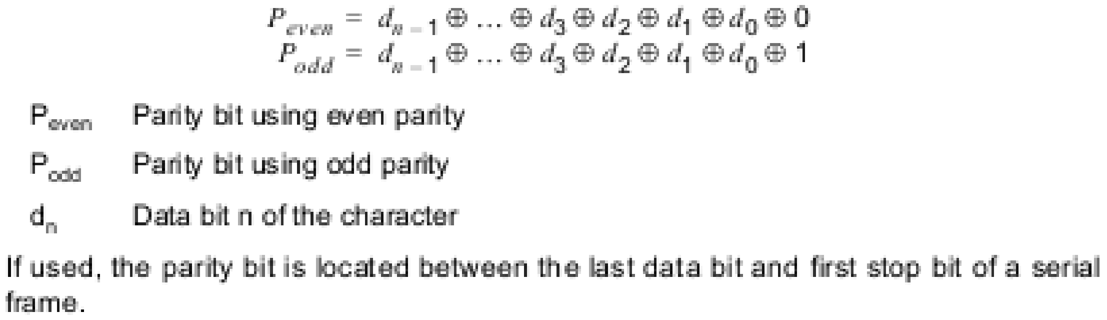
Figure 12: Frame Formats

The frame format used by the USART is set by the UCSZ2:0, UPM1:0, and USBS bits in UCSRB and UCSRC. The Receiver and Transmitter use the same settings. Note that, changing the settings of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter.

The USART Character Size (UCSZ2:0) bits select the number of data bits in the frame. The USART Parity mode (UPM1:0) bits enable and set the type of parity bit. The selection between one or two stop bits is done by the USART Stop Bit Select (USBS) bit. The receiver ignores the second stop bit. An FE (Frame Error) will therefore only be detected in the cases where the first stop bit is zero.

❐ Parity Bit Calculation

The parity bit is calculated by doing an Exclusive OR (XOR) of all the data bits. If odd parity is used, the result of the exclusive or is inverted. The relation between the parity bit and data bits is as follows —



$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

P_{even}	Parity bit using even parity
P_{odd}	Parity bit using odd parity
d_n	Data bit n of the character

If used, the parity bit is located between the last data bit and first stop bit of a serial frame.

In the presentation slide, detailed analysis of AVR serial programming has been discussed.

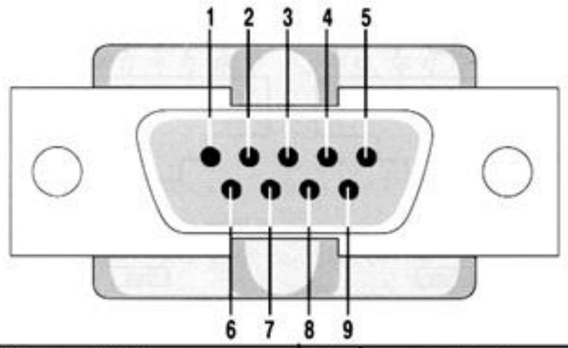
RS–232 AND MAX232

In our project, we have used RS–232 connectors and MAX232 chips to connect the PC with microcontroller.

The RS–232 is the most common Serial I/O interfacing standards now-a-days. In RS–232 standard, logic ‘1’ is represented by 3 to 25 Volts whereas, logic ‘0’ is represented by +3 to +25 Volts. For this reason, MAX232 chip was connected to convert this voltage to TTL level.

ATMega32 has two pins to serially transmit and receive data, which are named TX and RX respectively (pin no. 14 and 15).

Connection diagram of the RS–232 connector along with the Max232 chip is given below —



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Figure 13: Pin Diagram of the RS–232 Connector

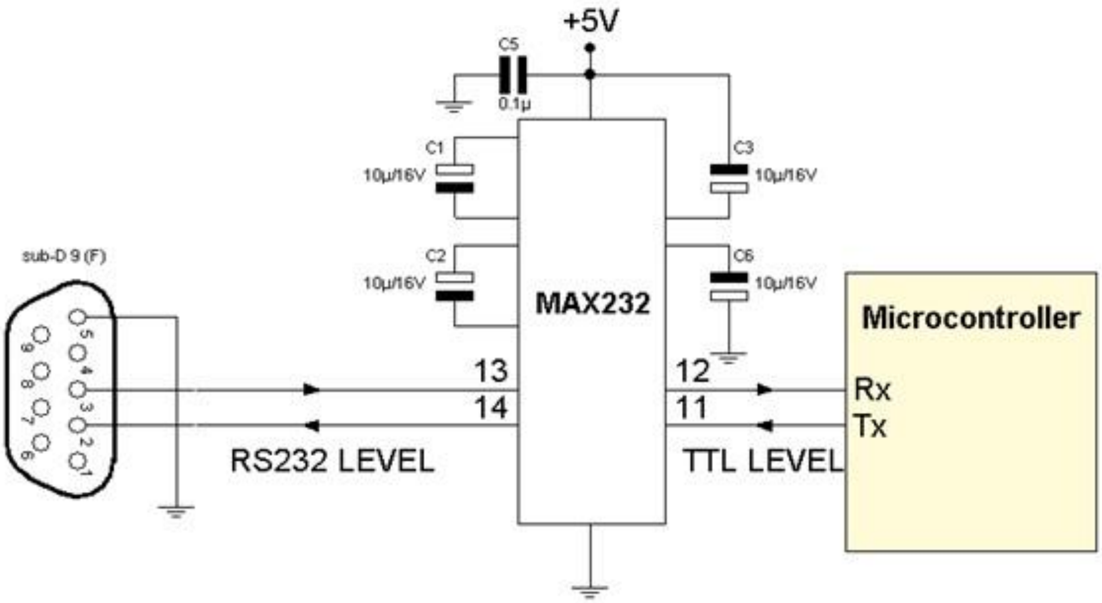


Figure 14: Connection Diagram of RS–232 with Max232 chip

Softwares Used

Following softwares were used throughout the project —

- *Microsoft Visual C# 2010 Express*

  We used Microsoft Visual C# 2010 express to compile the C# program by which the user will control the stepper motor. This software has a user friendly outlook and is heavily used for developing Graphical User Interface applications along with Windows Forms applications.

- *AVR Studio 4*

  AVR studio 4 is used to write a firmware which will be burned into the ATMega32 microcontroller. It provides the integrated development environment (IDE) for developing and debugging AVR microcontroller based applications. Here we can write, build and debug applications written in C/C++ or Assembly code.

- *Extreme Burner*

  Extreme burner burns the *.hex* file generated by AVR studio 4 into the ATmega32 microcontroller.

- *Prolific PL–2303 Vista Driver*

  This software is a virtual serial port driver which identifies the connector RS–232 and assigns a specific port name to it every time it is inserted through the USB Port into the PC. This software also helps to interface the serial port in *Proteus* simulation circuit.

COMPONENTS USED

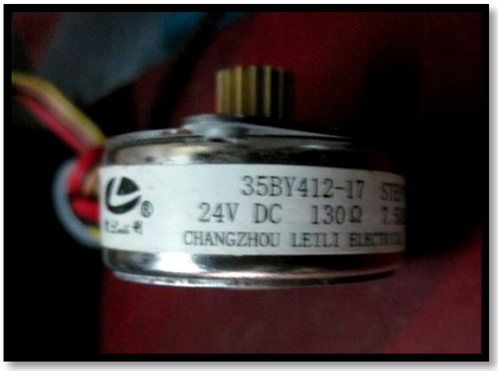


Figure 15: A Typical 24 V Stepper Motor

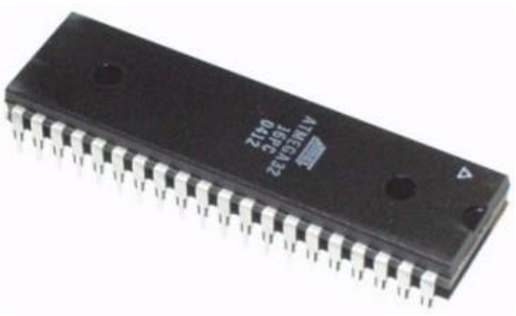


Figure 16: ATMega32 Microcontroller

Figure 17: RS–232 Connector Cable

Figure 18: Max232, Voltage Level Converter IC

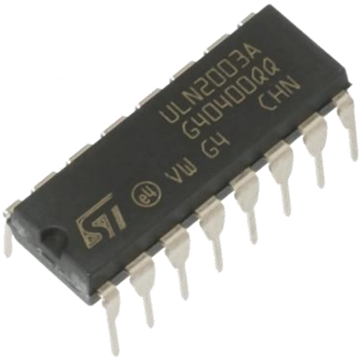


Figure 19: ULN2003, Motor Driver IC

Here,
The RS–232 cable first takes serial data from PC USB Port and passes it to the microcontroller via Max232.

The Max232 IC then converts the serial data voltage level to suitable level for microcontroller.

The ATMega32 microcontroller then starts driving the motor through ULN2003 IC according to the main instructions stored via loading a code.

The ULN2003 IC supplies the necessary amount of current for driving the motor.

The Stepper Motor being used here is a 24 V, 130 Ω Unipolar motor, driven in both full step and half step mode in both clockwise and counter-clockwise directions.

Main Code

This is the source code for the Motor Control Interfacing Window –

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.Net;

namespace Blitzkrieg__stepper_motor_controller_
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            button2.Enabled = false;
            radioButton1.Enabled = false;
            radioButton2.Enabled = false;
            radioButton3.Enabled = false;
            radioButton4.Enabled = false;
            button1.Enabled = false;
            checkBox1.Enabled = false;
        }
        string pname;
        string msg = " ";
        string url;
        string fromnet = " ";
        Thread t;
        bool f = false;
        string prevmsg;

        private void button1_Click(object sender, EventArgs e)
        {
            /*radioButton1.Enabled = false;
            radioButton2.Enabled = false;
            radioButton3.Enabled = false;
            radioButton4.Enabled = false;
            button1.Enabled = false;
            button2.Enabled = true; */

            if (radioButton1.Checked && radioButton3.Checked)
                msg = "A";
            else if (radioButton1.Checked && radioButton4.Checked)
                msg = "B";
            else if (radioButton2.Checked && radioButton3.Checked)
                msg = "C";
            else if (radioButton2.Checked && radioButton4.Checked)
                msg = "D";
            if (!serialPort1.IsOpen)
            {
                serialPort1.PortName = pname;
                serialPort1.Open();
            }
            serialPort1.Write(msg);
        }

        private void button2_Click(object sender, EventArgs e)
        {
            /*radioButton1.Enabled = true;
            radioButton2.Enabled = true;
            radioButton3.Enabled = true;
            radioButton4.Enabled = true;
            button1.Enabled = true;
            button2.Enabled = false;*/

            msg = "O";
            if (!serialPort1.IsOpen)
            {
                serialPort1.PortName = pname;
                serialPort1.Open();
            }
            serialPort1.Write(msg);
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {
        }
        private void button3_Click(object sender, EventArgs e)
        {
            f = true;
            pname = textBox1.Text;
            if (pname != "")
            {
                radioButton1.Enabled = true;
                radioButton2.Enabled = true;
                radioButton3.Enabled = true;
                radioButton4.Enabled = true;
                button1.Enabled = true;
                button2.Enabled = true;
                textBox1.Enabled = false;
                button3.Enabled = false;
                textBox2.Enabled = false;
                checkBox1.Enabled = true;

                url = textBox2.Text;
                if (url == "")
                    url = "http://www.mytextfile.com/public?s=HBKdcYIf16GqrXAtQA91w42rhejrw38aLaBzm3cg";
                t = new Thread(downit);
                t.Start();
                this.Refresh();
            }
        }

        void downit()
        {
            while (true)
            {
                Thread.Sleep(1000);

                if (checkBox1.Checked)
                {
                    try
                    {
                        WebClient wc = new WebClient();
                        string data = wc.DownloadString(url);
                        string data2 = data.Substring(82, 1);
                        fromnet = data2;
                    }

                    catch (Exception ex)
                    {
                        MessageBox.Show("Incorrect URL or Network error", "Error!");
                    }
                    if ((fromnet == "A") || (fromnet == "B") || (fromnet == "C") || (fromnet == "D") || (fromnet == "O"))
                    {
                        if (fromnet != prevmsg)
                        {
                            prevmsg = fromnet;
                            if (!serialPort1.IsOpen)
                            {
                                serialPort1.PortName = pname;
                                serialPort1.Open();
                            }
                            serialPort1.Write(fromnet);
                        }
                    }
                }
            }
        }

        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (f)
                t.Abort();
        }

    }
}
```

USER INTERFACE

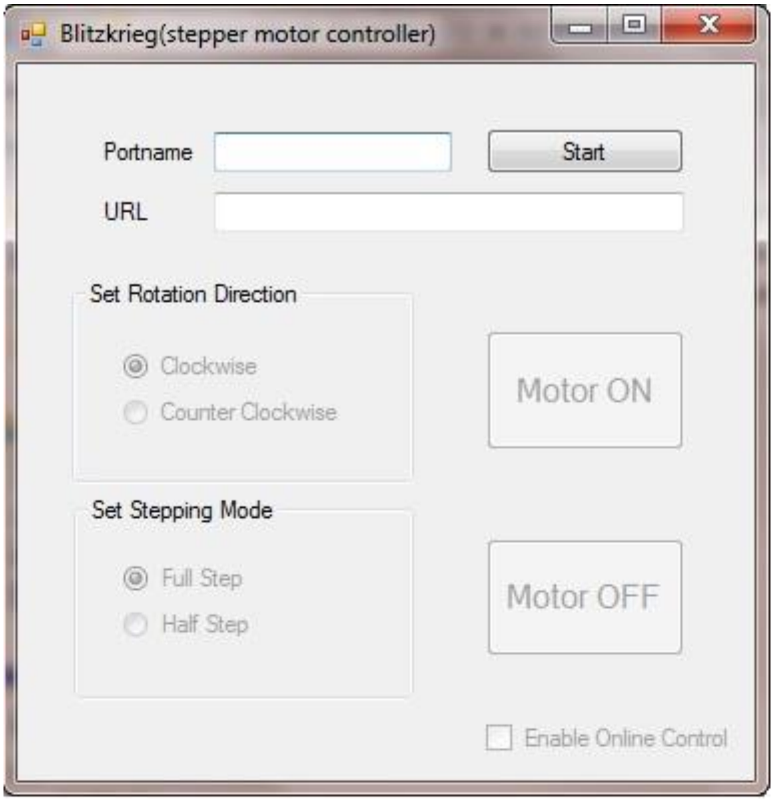


Figure 20: Motor Control Interfacing Window

Here, for online control, at first, a web link is inserted into the 'URL' input and the 'Enable Online Control' checkbox is selected.

Then —

- For Clockwise Full Step motion ⇒ Signal 'A' is sent
- For Clockwise Half Step motion ⇒ Signal 'B' is sent
- For Counter-Clockwise Full Step motion ⇒ Signal 'C' is sent
- For Counter-Clockwise Half Step motion ⇒ Signal 'D' is sent
- For turning the motor OFF ⇒ Signal 'O' is sent

Circuit Diagram

The final circuit diagram (schematic) —

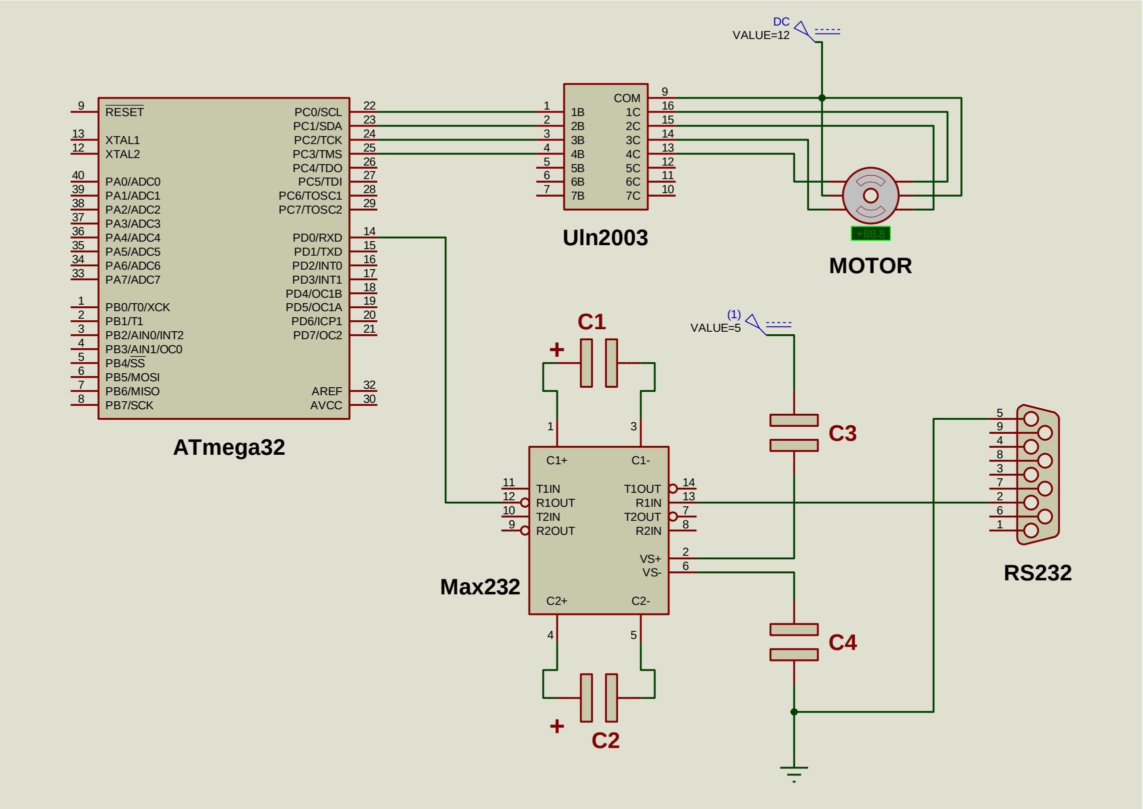


Figure 21: Final Circuit Diagram (Schematic)

Operation

Operation – Step by Step

The operation of the circuit is given below —

- ✓ Computer sends the *serial data* via COM Port, which is the RS–232 cable.
- ✓ The serial data through the RS–232 cable is then passed to the Max232. A traditional PC sends data with a voltage of □0.3 to □12 V for HIGH signal and 0.3 to 12 V for LOW signal. Max232 converts the voltage level from 0 to 5 V, which ATMega32 can handle without burning.
- ✓ The data byte is then passed to Serial Data Receive Port of ATMega32, which is PD0/RXD. In the ATMega32, Port C is used as the output port. A *specific code* is loaded into the for the purpose of the project.

- ✓ After the data comes out from ATMega32, it is passed to ULN2003. This data carries in a *very low current*, which is not capable of running a motor. ULN2003 converts this low current to a suitable value for running the motor.
- ✓ The motor then runs according the command it gets which may be *'A', 'B', 'C', 'D'* or *'O'*.

❒ Microcontroller Code

The code loaded into the microcontroller for the purpose of motor control is given here —

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

unsigned char got = 'O';
ISR(USART_RXC_vect)
{
        got = UDR;
}

void my_delay(void)
{
        int i,j;
        for(i = 0; i < 30000; i++)
                  for(j = 0; j < 1000; j++)
                  ;
        ;
}

int main(void)
{
        DDRC = 0xFF;
        DDRD = 0x00;
        UCSRB = (1 << RXEN) | (1 << RXCIE);
        UCSRC = (1 << UCSZ1) | (1 << UCSZ0) | (1 << URSEL);
        UBRRL = 0x33;
        sei();
        while(1)
        {
                  if (got == 'A')
                  {
                            PORTC = 0x03;
                            my_delay();
                            PORTC = 0x09;
                            my_delay();
                            PORTC = 0x0C;
                            my_delay();
                            PORTC = 0x06;
                            my_delay();
                  }
                  else if (got == 'B')
                  {
                            PORTC = 0x03;
                            my_delay();
                            PORTC = 0x01;
                            my_delay();
                            PORTC = 0x09;
                            my_delay();
                            PORTC = 0x08;
                            my_delay();
                            PORTC = 0x0C;
                            my_delay();
                            PORTC = 0x04;
                            my_delay();
                            PORTC = 0x06;
                            my_delay();
                            PORTC = 0x02;
                            my_delay();
                  }
                  else if (got == 'C')
                  {
                            PORTC = 0x06;
                            my_delay();
                            PORTC = 0x0C;
                            my_delay();
                            PORTC = 0x09;
                            my_delay();
                            PORTC = 0x03;
                            my_delay();
                  }
                  else if(got == 'D')
                  {
                            PORTC = 0x02;
                            my_delay();
```

```
                        PORTC = 0x06;
                        my_delay();
                        PORTC = 0x04;
                        my_delay();
                        PORTC = 0x0C;
                        my_delay();
                        PORTC = 0x08;
                        my_delay();
                        PORTC = 0x09;
                        my_delay();
                        PORTC = 0x01;
                        my_delay();
                        PORTC = 0x03;
                        my_delay();
                }
                else if(got == 'O')
                {
                        PORTC = 0x00;
                        my_delay();
                        my_delay();
                        my_delay();
                        my_delay();
                }
        }
        return 0;
}
```

ASSEMBLED CIRCUIT

Our finally assembled circuit –

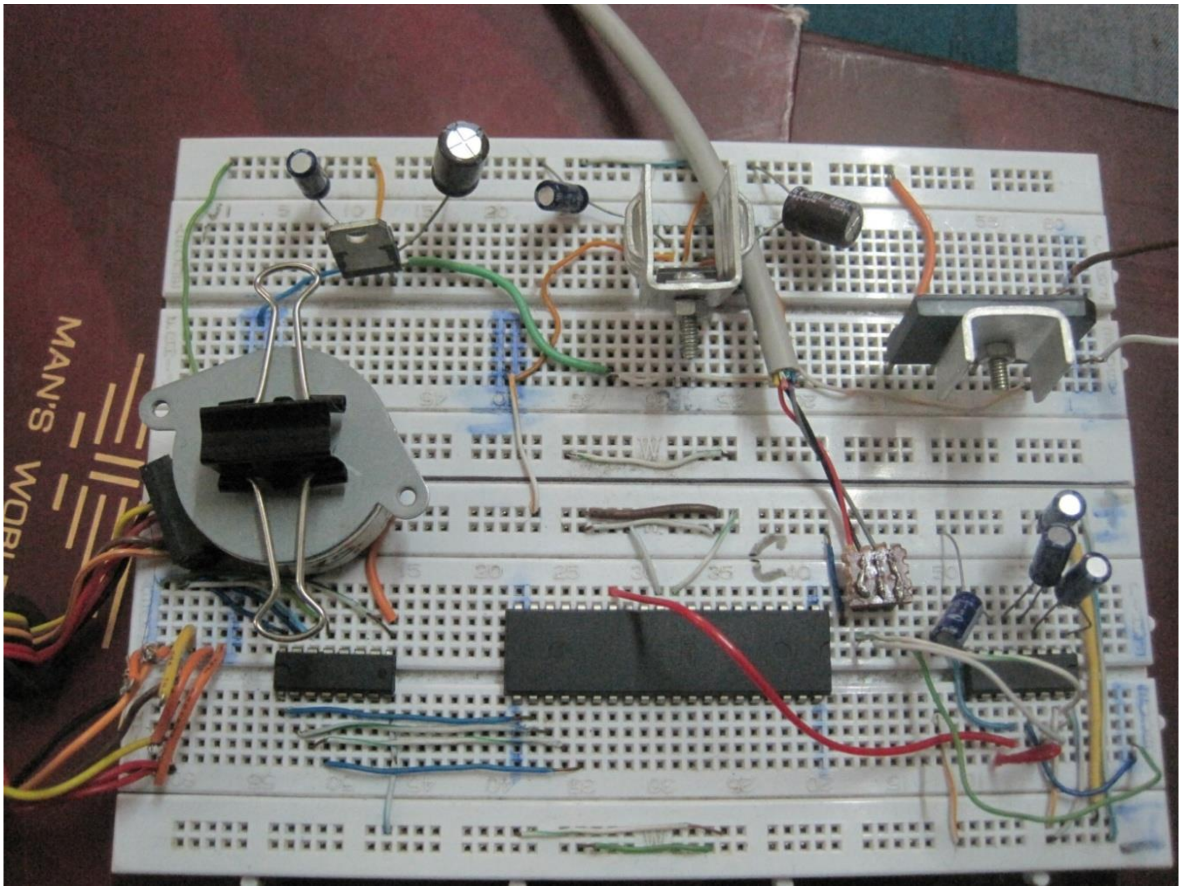


Figure 21: Finally Assembled Circuit

Cost Analysis

The cost analysis of our project is presented here —

	Component	Quantity	Price (Tk.)
1	Transformer	1	120
2	Bridge Rectifier	1	60
3	7805 IC + Heat Sink	1 + 1	7 + 35 = 42
4	RS–232 Connector	1	250
5	RS–232 Male - Female Cable	1	200
6	Max232 IC	1	65
7	ATMega32	1	450
8	Stepper Motor	1	230
9	ULN2003 IC	1	23
10	Capacitor	8	25
11	Breadboard	2	150 × 2 = 300
12	Wire	—	30
Total Cost			**Tk. 1795**

Discussion

Our project has certain practical implications. We can use this at industrial environment where motor must be controlled from remote place. The internet control gives a new dimension to the project and increases its effectiveness.

This project can further be developed according to the need of the users, like *rotation at a certain angle*, *automated motor control* depending on time of the day etc. Moreover, this project can be helpful for *Robotics Projects*.