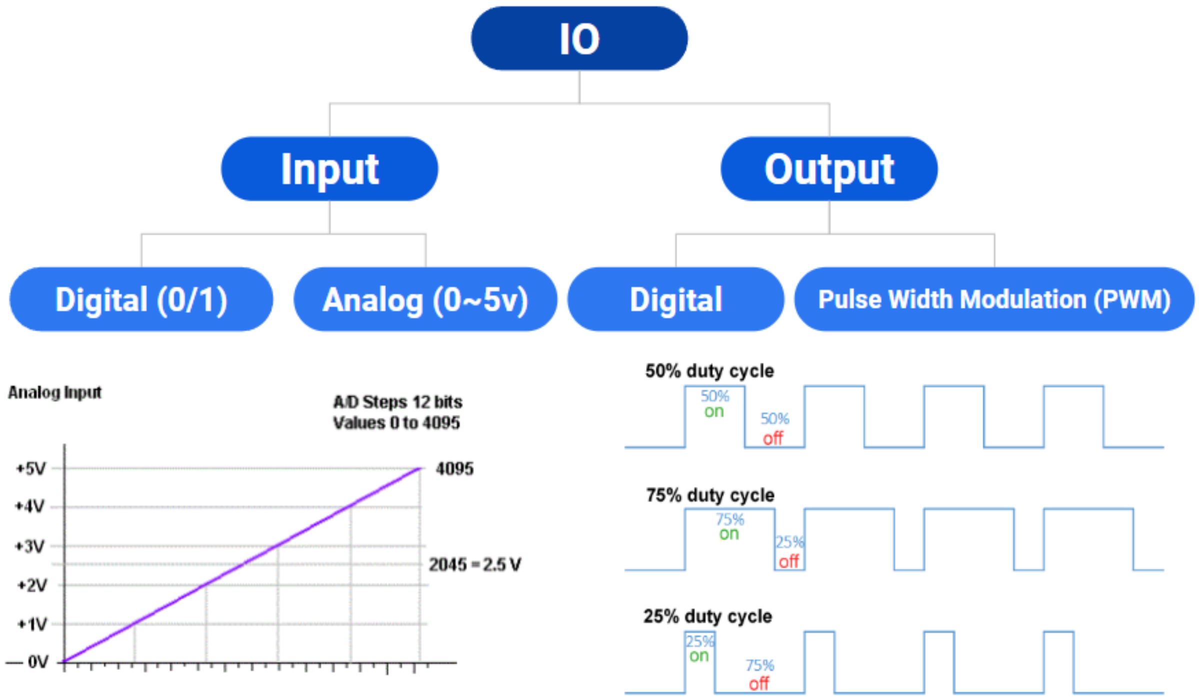
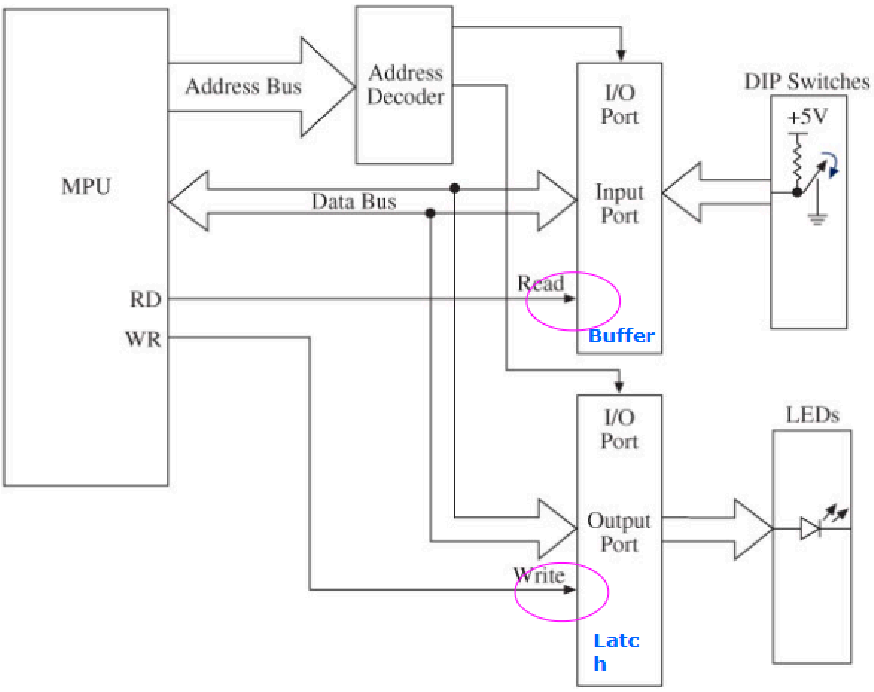
GPIO

Input Output



Block Diagram of I/O Interfacing



Buffer keep the level of 0 or 1 (to resolve fan in problem)
Latch hold the value (0/1) until it is cleared

Explanation: MPU selects the port it needs to use using the address bus by putting the address of that particular port in the address bus. Address decoder selects which port needs to be activated. Next control bus is used. If the microprocessor uses an input port, it will enable the read pin using the control bus. The input device connected to the input port then sends data to the port and port to the Data bus.

Order of use for Input device: Address Bus—->Control Bus—->Data Bus

When the microprocessor needs to write(uses output port/working with an output device like LED's). Similarly it will first use the address bus to select the port it wants to use. The address decoder selects the port. The microprocessor then sends the data using the Data bus. The microprocessor then enables the write pin using Control Bus and then the data is sent from port to the output device.

Order of use for Output device: Address Bus—->Data Bus—->Control Bus

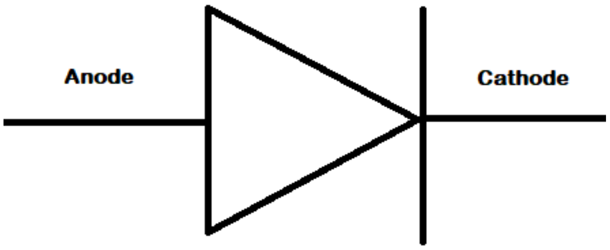
Some Output Devices:

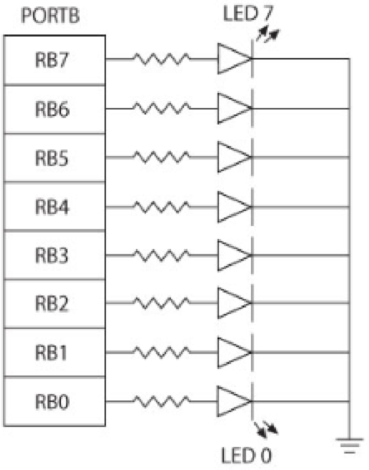
- Light Emitting Diode (LED)
- Seven segment display (Alphanumeric)
- Matrix Display
- Liquid Crystal Display (LCD)
- Buzzer
- DC Motor
- Servo Motor
- Stepper Motor

Interfacing LED:

Two ways to connect LEDs to I/0 ports,

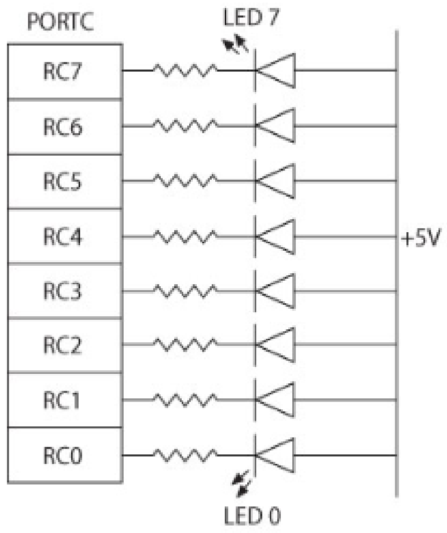
- Common Cathode: The end that is connected to GND.(Current Sourcing)
- Common Anode: The end that is connected to VCC.(Current Sinking)





The cathode end of all the 8 LEDs are commonly connected externally to the GND. The Anode ends of each LEDs are connected to each pin of port B.

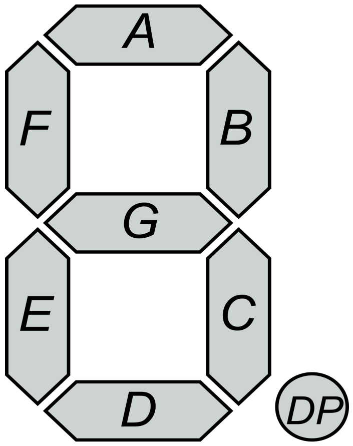
Common Cathode



Common Anode

In the case of common anode the idea is the same but opposite. The anode end of all the 8 LEDs are commonly connected externally with the VCC. The VCC can be supplied by a battery or any other type of external source. The cathode ends of each of the LEDs are connected to each pin of port C. Now if the microprocessor sends 0 through any of the pins of port C then that particular LED will light up.

Interfacing Seven-Segment Display:

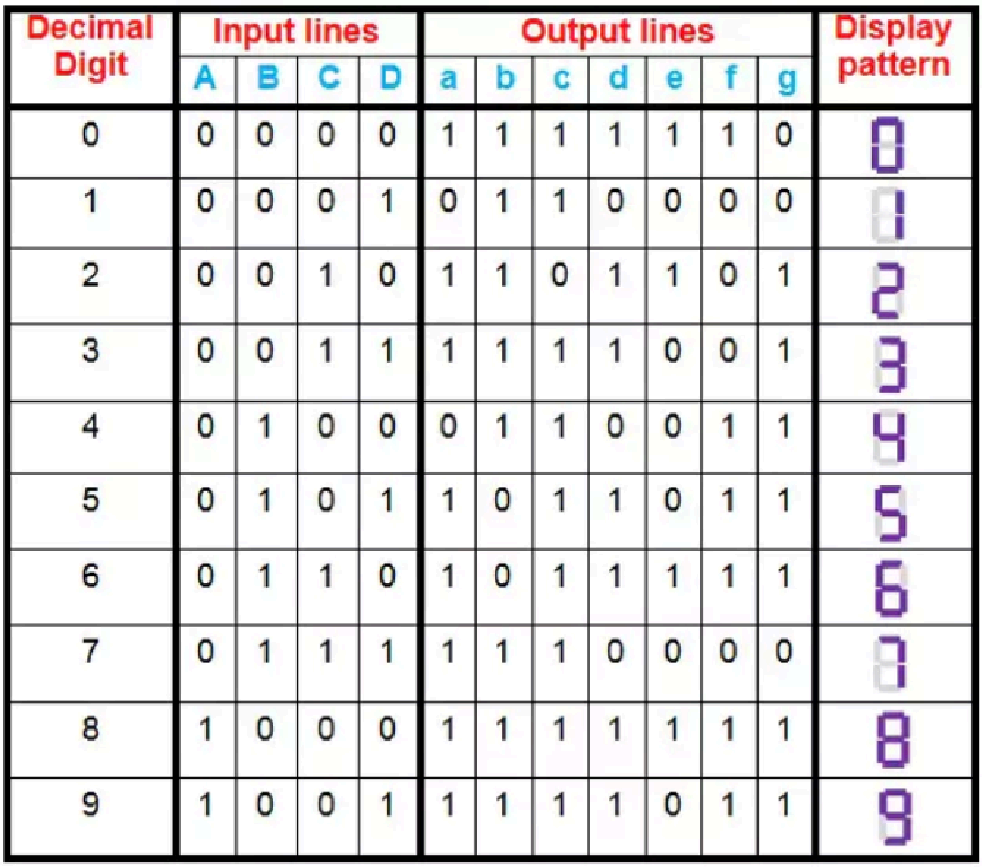


Can show numbers (0 to 9) and a few alphabets. In most cases numbers are displayed. It actually contains 7 LED’s and they are interfaced as either common cathode or common anode.

*Remember: If we want to display “1” then we will need to light up only B and C segment

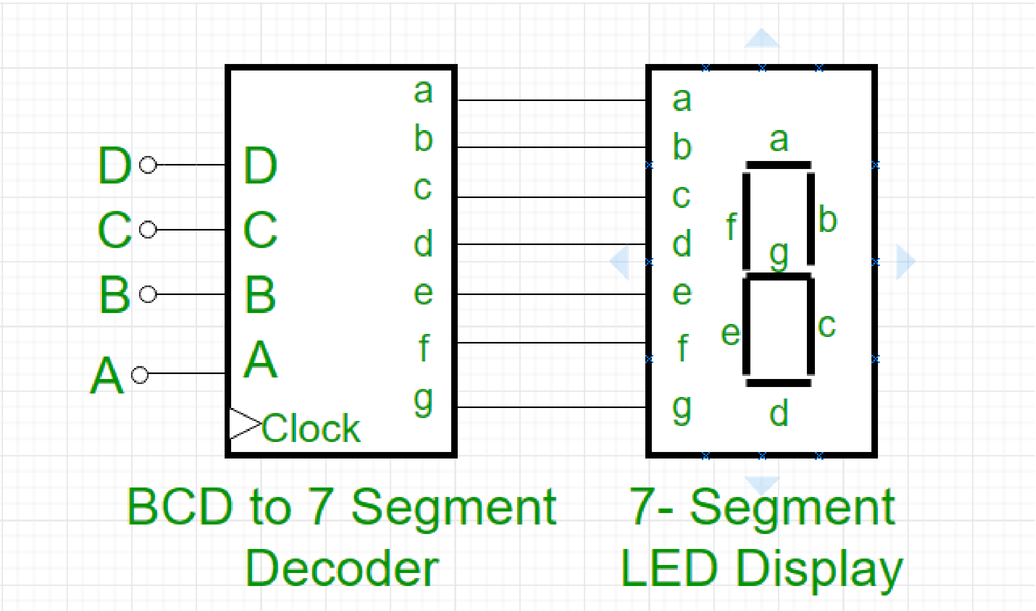
A	B	C	D	E	F	G	Display
1	1	1	1	1	1	0	0
0	1	1	0	0	0	0	1
1	1	0	1	1	0	1	2
1	1	1	1	0	0	1	3
0	1	1	0	0	1	1	4
1	0	1	1	0	1	1	5
1	0	1	1	1	1	1	6
1	1	1	0	0	0	0	7
1	1	1	1	1	1	1	8
1	1	1	1	0	1	1	9

Now, to generate such combinations in order to display numbers, the CPU uses a BCD to seven-segment driver in order to display it.

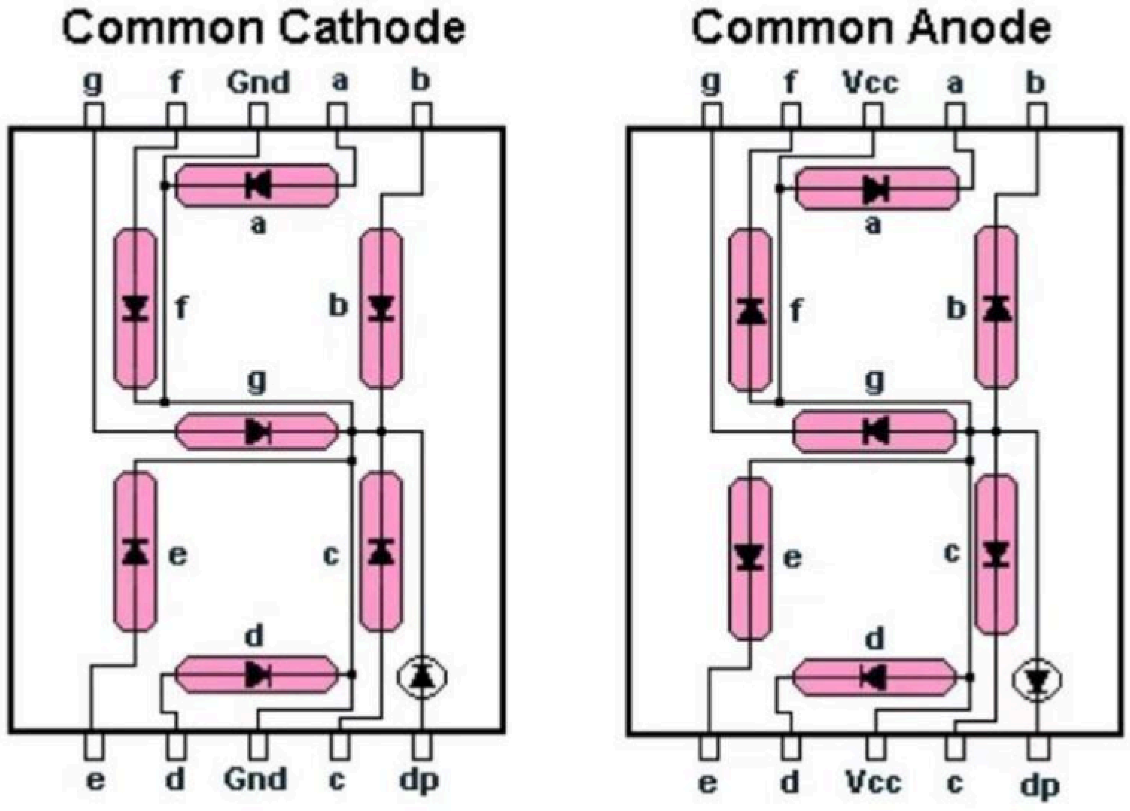


Decimal Digit	Input lines				Output lines							Display pattern
	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	0
1	0	0	0	1	0	1	1	0	0	0	0	1
2	0	0	1	0	1	1	0	1	1	0	1	2
3	0	0	1	1	1	1	1	1	0	0	1	3
4	0	1	0	0	0	1	1	0	0	1	1	4
5	0	1	0	1	1	0	1	1	0	1	1	5
6	0	1	1	0	1	0	1	1	1	1	1	6
7	0	1	1	1	1	1	1	0	0	0	0	7
8	1	0	0	0	1	1	1	1	1	1	1	8
9	1	0	0	1	1	1	1	1	0	1	1	9

The driver takes an input of 4 digits or 4 bits to present 9, if we take 7 bits we can only represent upto 8. The driver is built and designed in a way that whenever it gets a combination of for example “0110” (binary representation of 6, it will automatically give an output combination of “1011111” which will basically display the 6.



Common cathode and common anode configuration:

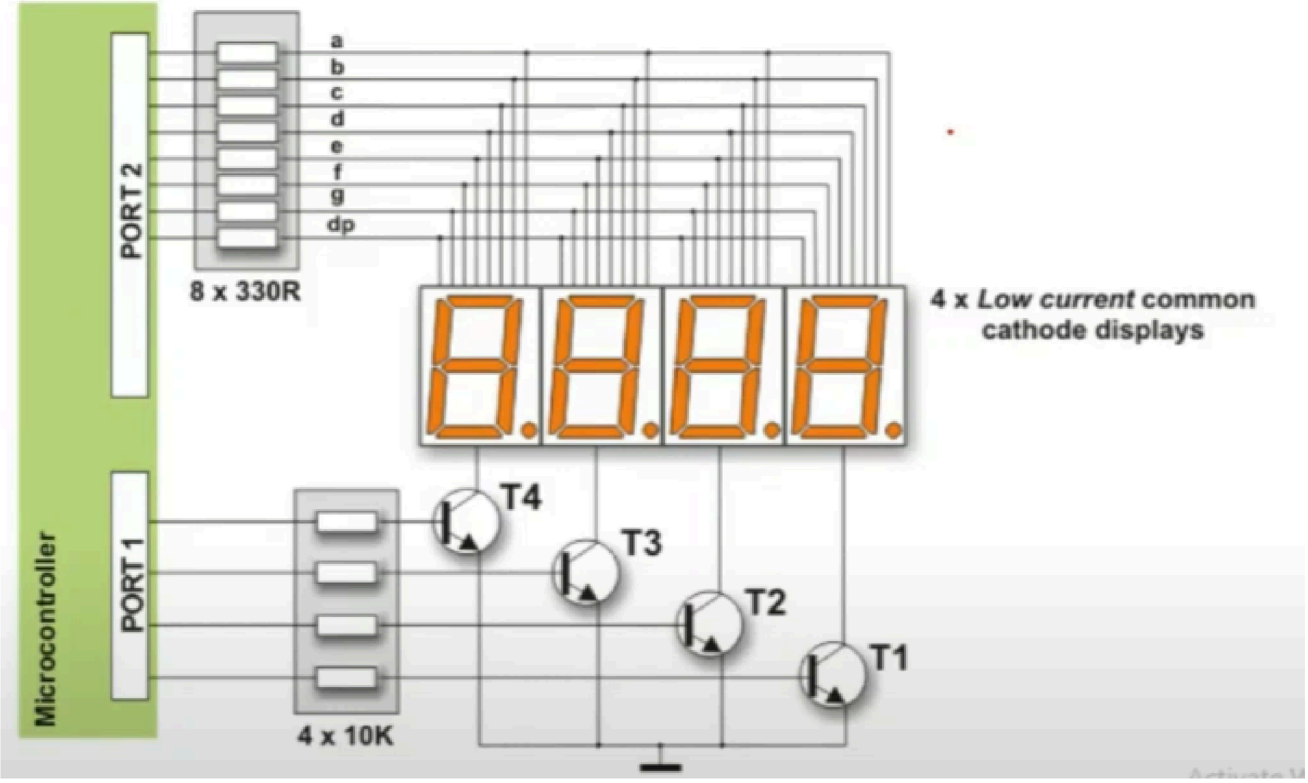


SSD Configuration

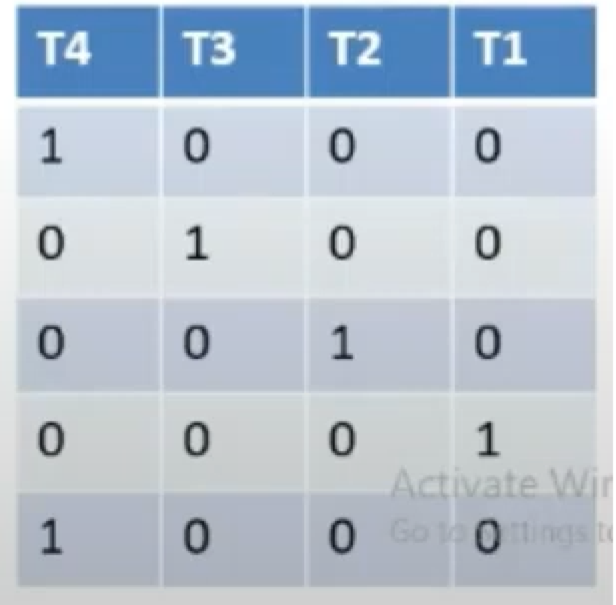
The tables we have seen above are for common cathode configuration. Now if we use common anode the combinations will just get reversed for example: to display 5 the combinations are “1011011” for common cathode, if we use common anode configuration it will be the reverse “0100100” (1 turns to 0 and 0 turns to 1).

Now, if we look at the above diagram, we can see that for common cathode the grounds are common and the VCC are individually given. If we pass “1” through any of the a,b,c,d,e,f,g pins then that corresponding LED will light up. Whereas in the common anode configuration the VCC are common but GND is individually connected so if we send “0” through any of the a,b,c,d,e,f,g pins then that corresponding LED will light up

Interfacing to Multiple 7-Segments

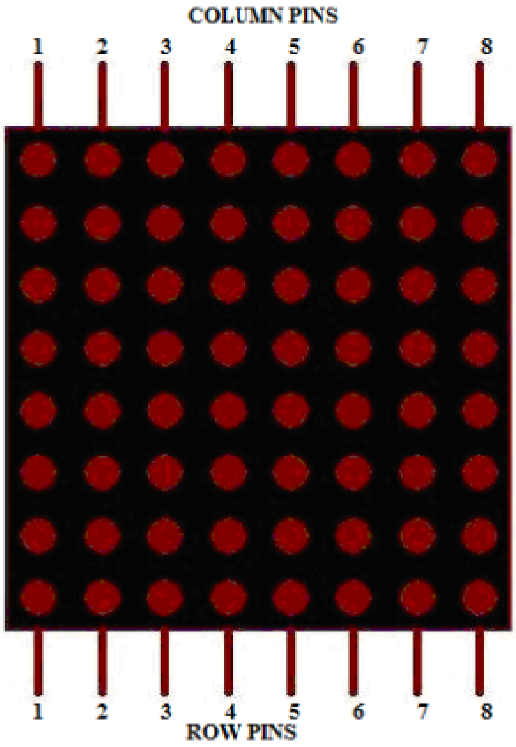


There is a BCD to seven-segment decoder in port 2 that is commonly connected to the a,b,c,d,e,f,g of the seven segment displays and the dp is externally connected to the CPU. Now we might think that these displays will show the same digits whenever the CPU sends a combination as the pins are commonly connected but no. To independently show different digits in different displays we can see the connection of port 1. There are 4 transistors connected to port 1 who have a common GND connection. Basically these transistors are working as switches. When we send “1” through the base of the transistor the internal circuit gets completed (emitter and collector gets connecter) so as the emitter is connected to the GND so the collector is also grounded and it directly passes the ground of the seven-segment displays. If we send “0” through the base of the transistor then the circuit is incomplete and the GND is not passed to the displays. So, as a result we can use this mechanism to individually keep the displays on one at a time. We can use the combination below for the transistors in order to operate the displays individually. At a time “1” is passed through only one transistor and the remaining transistors are passed “0”. These rotations are so fast that we cannot see it with our bare eyes and we see a constant light. But in reality at a time only one display is on.

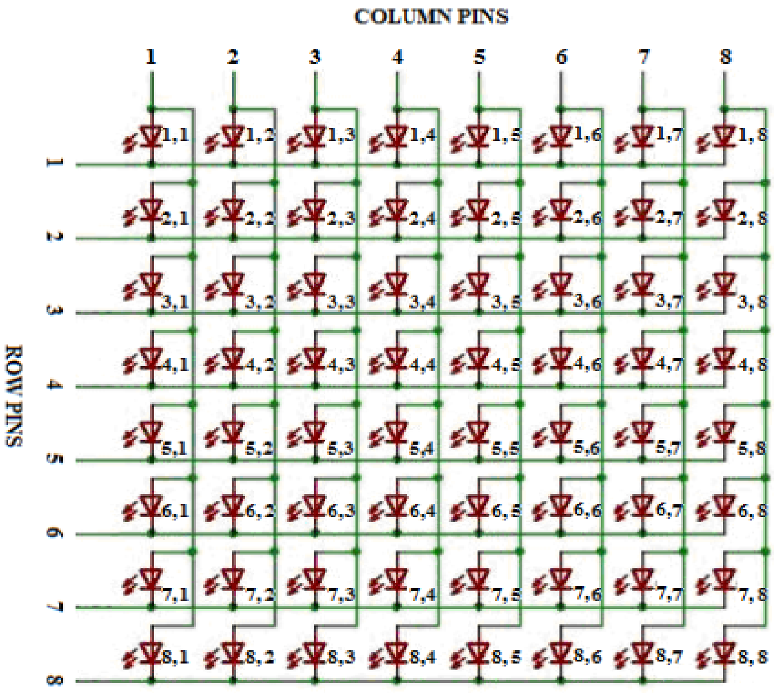


T4	T3	T2	T1
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

LED Matrix Display:

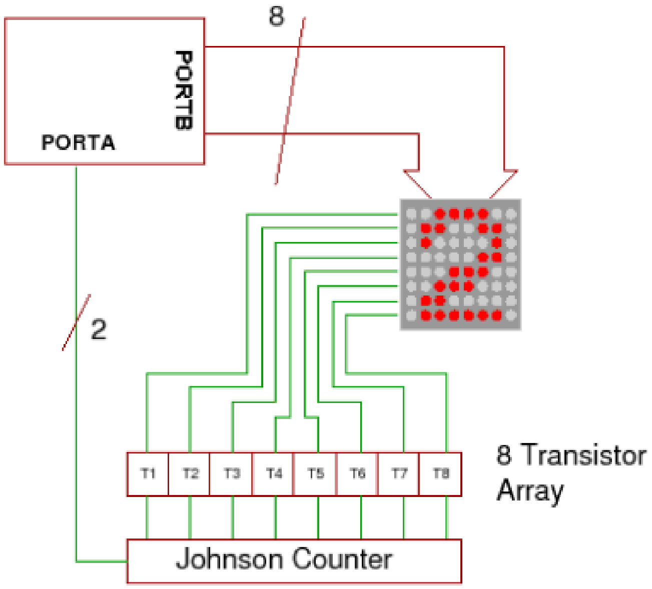


A standard LED matrix display contains 64 LEDs



If we look into the structure of these Led's, then we can see that each row is commonly connected to GND and each column is commonly connected to VCC. Now what if we want to light up a specific LED out of all these 64 LED's?. For example we want to light up the LED from row 5 and column 4. Now if we send "1" through column 6 then all the LED's in column 6 will be connected to VCC. If we send "0" through row 5 then all the LED's in row 5 will receive GND. Now if we look closely we can see that the LED in row 5 and column 4 is the only LED that received both VCC and GND so it will light up. What happens to the other LED's in row 5 and column 4? The LEDs in row 5 will only receive GND but no VCC so it wont light up and in case of column 4, all the LED is connected to VCC but not GND so they wont light up either as we know an LED needs to be connected to both GND and VCC in order to light up. We use this mechanism to light up individual LEDs and display different alphabets.

As we can see, it requires a total of 16 pins to use a LED matrix display (8 rows, 8 columns) which is not feasible. To avoid this redundancy a new circuit is introduced which is Johnson counter as shown below:



The purpose of this Johnson counter is to reduce the pin numbers from the microcontroller port. If we look at the diagram on the left, we can see that the 8 pins of port B handle the columns of the led matrix display. The row is handled by the johnson counter. Johnson counter is connected to port A using 2 pins to configure itself and johnson counter configures the 8 rows of the led matrix display. Suppose if we do not use johnson counter then we would have need to connect all the 8 pins of port A to the row of the led matrix display which sums up to 16 pins in total but with with the johnson counter connected we only need 10 pins in total (8 pins of port B for column and 2 pins of port A to configure johnson counter).

How does Johnson counter work?

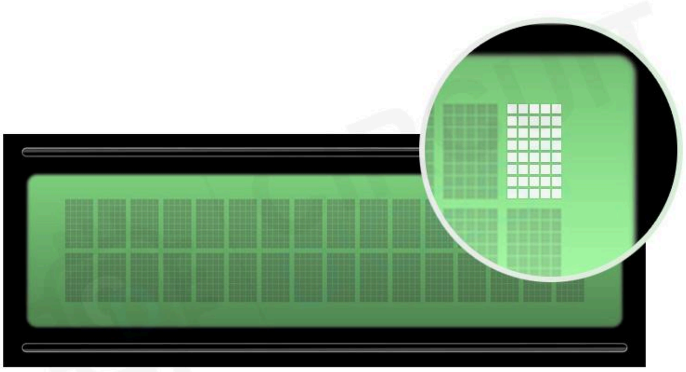
Out of the 2 pins connected to the johnson counter of port A, 1 pin is used to enable the johnson counter. The other pin is used to send clock pulse to the johnson counter. From the diagram we can see that the johnson counter contains an array of 8 transistors. All these 8 transistors are connected to all the 8 rows of the led matrix display. The task of this johnson counter is to provide GND through the rows of the led matrix display. Suppose we want to light up the middle 6 LEDs of the last row so the johnson counter will send a combination of “11111110”, as we can see that the last bit is “0” that means only the last row of the led matrix display is receiving GND and now if the microprocessor sends VCC through the 2nd, 3rd, 4th, 5th and 6th column then all the middle 6 LEDs of the last row will light up. With each clock pulse the transistor array of johnson counter left shifts by one bit and this rotation goes on. At a time only one row is active. This rotation is very fast, frequent and smooth that it cannot be seen through our bare eyes.

*Remember: Initially when the johnson counter is enabled by microprocessor, the initial combination is “11111110” for common cathode connection. For common anode connection, the initial combination right after enable is “00000001” (VCC is commonly connected to anode end of the LEDs so “1” is sent at the last row in order to send VCC in the last row.

LCD Display

LCD
1
16
VSS VDD V0 RS RW E D0 D1 D2 D3 D4 D5 D6 D7 A K
Hello World!
LCD Tutorial
Last Minute ENGINEERS.com

LCD displays are better than the previous displays we have seen above. We can display many alphabets or symbols using this display but only those that have ASCII values. LCD displays work depending on 8 bit ASCII code. At a time it can show 32 characters in total. If we look closely at the display we can see 2 lines with 16 segments in each of the lines like shown in the picture below.



Contents of LCD displays:

Three control signals:

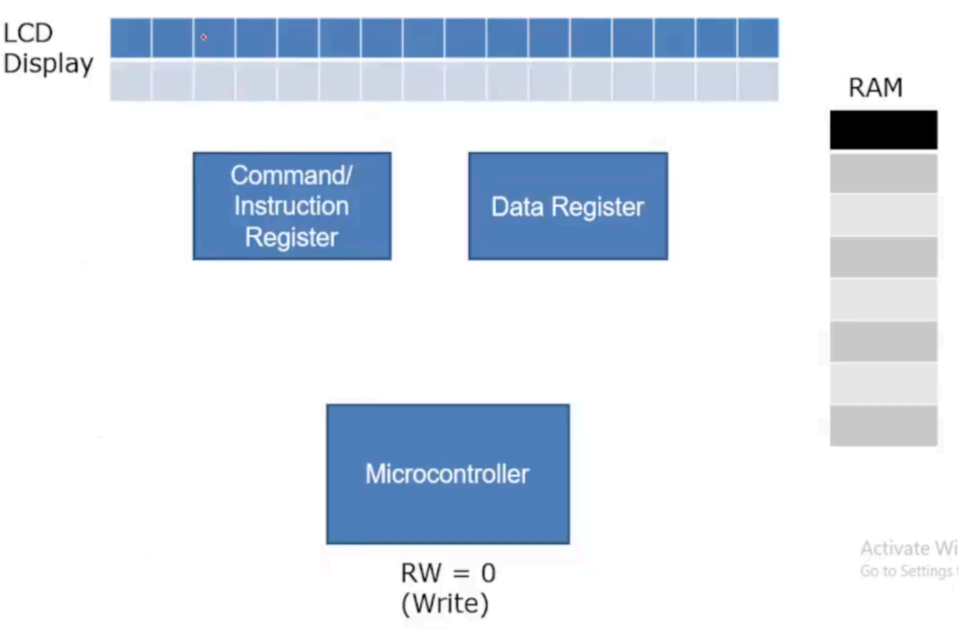
- RS (Register Select)
- RW (Read/Write)
- E (Enable)

These three pins create a connection between the microcontroller and the LCD display. These are the control pins that are responsible for communication between the microprocessor and the LCD display. There are some other pins for sending data from the microprocessor to the LCD display about which alphabet of the ASCII value we want to display. There are 8 pins allocated for this data communications each of 8 bits as we know each ASCII codes are of 8 bits. These 8 pins are from (D0 to D7) connected from microcontroller to LCD.

There are three 8 bit internal registers in the LCD display:

- Instruction Register (IR) to write instructions to set up LCD
- Data Register (DR) to write data (ASCII characters)
- Data RAM stores data in 8-bit character code in ASCII.

Communication Between RAM and LCD Display:



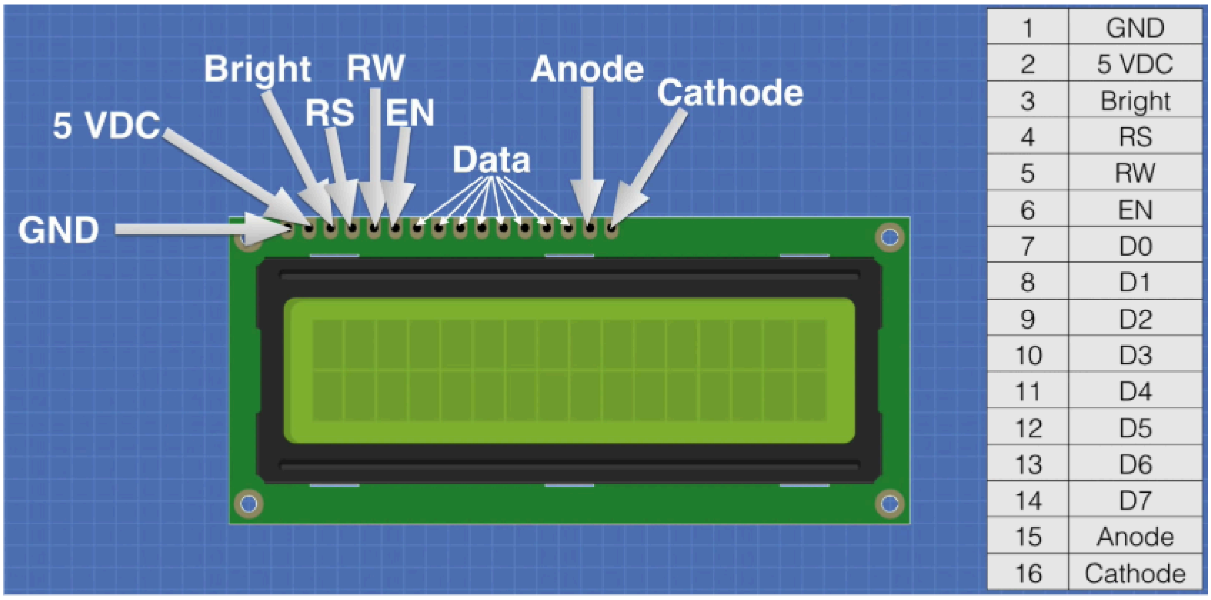
LCD has its own RAM that is integrated into the LCD display. As we know there are in total 32 segments in the LCD display with 2 lines each containing 16 segments. Each of these segments are indexed from 0 to 15. On the other hand the RAM is also indexed from 0 to 31, in total 32 RAM slots (as LCD has 32 segments). When the microcontroller sends a ASCII value of 8 bits, it is stored in the RAM. Suppose the first slot of the RAM has the ASCII value of the alphabet “A” stored in it. There is a circuit connected between the LCD display and the RAM that converts the ASCII value to its corresponding symbols/alphabets. After this conversion, the alphabet “A” is displayed on the display in the same index segment that it was stored in the RAM. In simple words if the ASCII value was stored in index 1 then the alphabet will be displayed in the index 1 of the display.

Communication between microcontroller and LCD Display:

The microcontroller is connected to the LCD pin using three pins as we have seen before (RS, RW, E). There are more 8 pins (D0 to D7) that the microcontroller uses to send data. At first if the microcontroller wants to display data on the display it enables the LCD using the enable (E) pin which is an active low pin (mean microcontroller sends “0” to enable the LCD). The next step is configuring the LCD. We know that the microcontroller is connected to the RS (Register Select) pin of the LCD. Microcontroller sends a “0” through the RS pin to the LCD. The purpose of this “0” is that now the microcontroller will now send data of 8 bit using which the LCD will be configured. Now, this 8 bit data contains information like from which index the alphabets will start from, indentation, line break, etc basically in which slot of the RAM, these ASCII values should be stored. This 8 bit data is sent through (D0 to D7) pins to the Command register. The LCD will configure itself according to the provided configuration. Suppose for example if the configuration says that the sentence will start from the 4th index of the display then the ASCII values are also stored from the fourth slot of the RAM. After the configuration steps are complete, the microcontroller starts sending data. Before that it turns the RS pin “1” which means that the data that will be received now should be displayed on the display. After the RS pin is set to “1” the microcontroller starts sending ASCII codes of 8 bit through the pins (D0 to D7). These ASCII codes are stored in the Data Register first then from the data register these ASCII values are sent to RAM (according to the configurations done before). And finally from the RAM sends the ASCII values to the display accordingly which passes through another circuit that converts ASCII codes to its corresponding alphabet/symbol as we have seen before and then the LCD displays it.

Purpose of RW pin:

Whenever the microcontroller sends any data that can be either command register data or data register data. RW will always be 0. It will never be “1” because we know read = 1 and write = 0. The LCD is an output device and the microcontroller will never read anything from the LCD, it will always write data to the LCD. So, that is why RW will always be 0 as the microcontroller will never read anything from the LCD.



1	GND
2	5 VDC
3	Bright
4	RS
5	RW
6	EN
7	D0
8	D1
9	D2
10	D3
11	D4
12	D5
13	D6
14	D7
15	Anode
16	Cathode

According to the above diagram, there are in total 16 pins in a LCD Display:

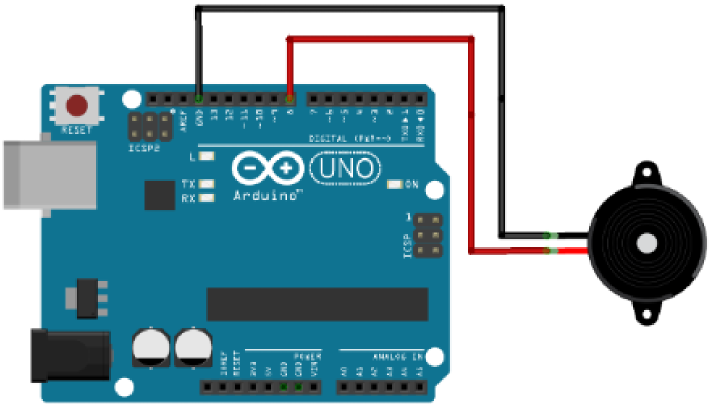
Pin 1 GND
Pin 2 VCC
Pin 3 Bright → controls brightness
Pin 4 RS → Register Select used for configuration when 0 and 1 for sending data.
Pin 5 RW → For read/write which is always “0”
Pin 6 E → Used to enable the LCD
Pin 7 D0 → (D0 to D7) is used for configuration when RS is 0 and sends ASCII values when RS is 1
Pin 8 D1
Pin 9 D2
Pin 10 D3
Pin 11 D4
Pin 12 D5
Pin 13 D6
Pin 14 D7
Pin 15 Anode
Pin 16 Cathode

Pin 15 (Anode) and pin 16 (Cathode) are used to turn on the backlight of the display which is basically an LED.

LCD can be interfaced either in 8-bit or 4-bit mode:
Till now we have seen the 8-bit configuration of LCD display, as every data sent to the LCD is of 8 bit as the ASCII values are of 8 bit each. There is another mode which is 4-bit mode, used in some special cases.

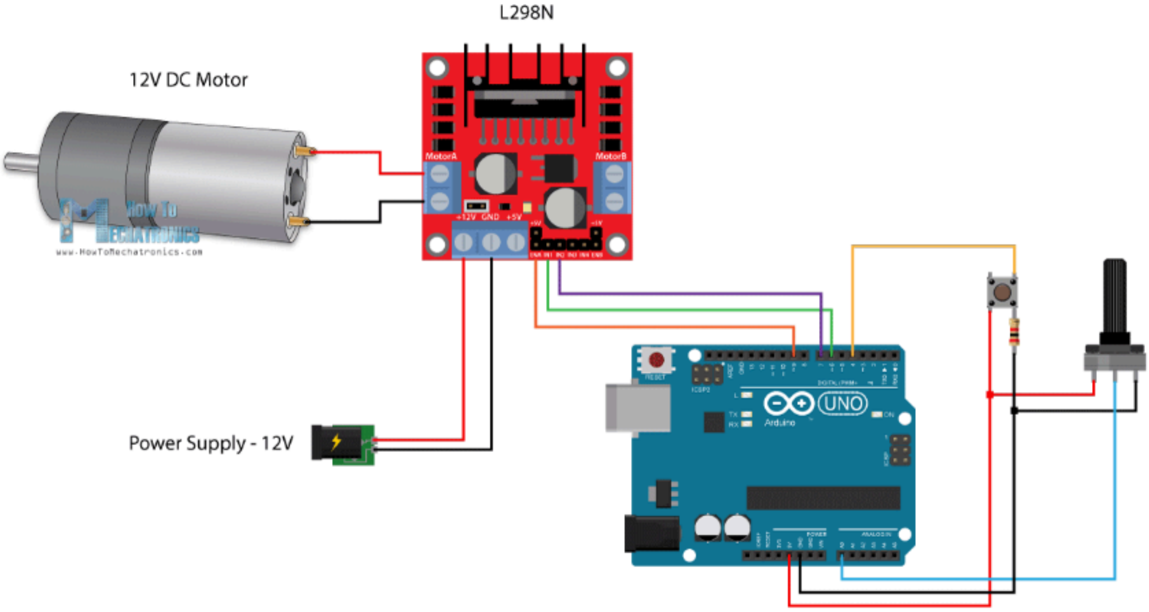
8-bit mode → Data transfers in 1 clock pulse through D0 to D7.
4-bit mode → Also sends 8 bit data but uses two clock pulses to send 8 bit data through D4 to D7. In the first clock pulse it sends the LSB 4 bit and in the second clock pulse it sends the MSB 4 bit.

Buzzer:



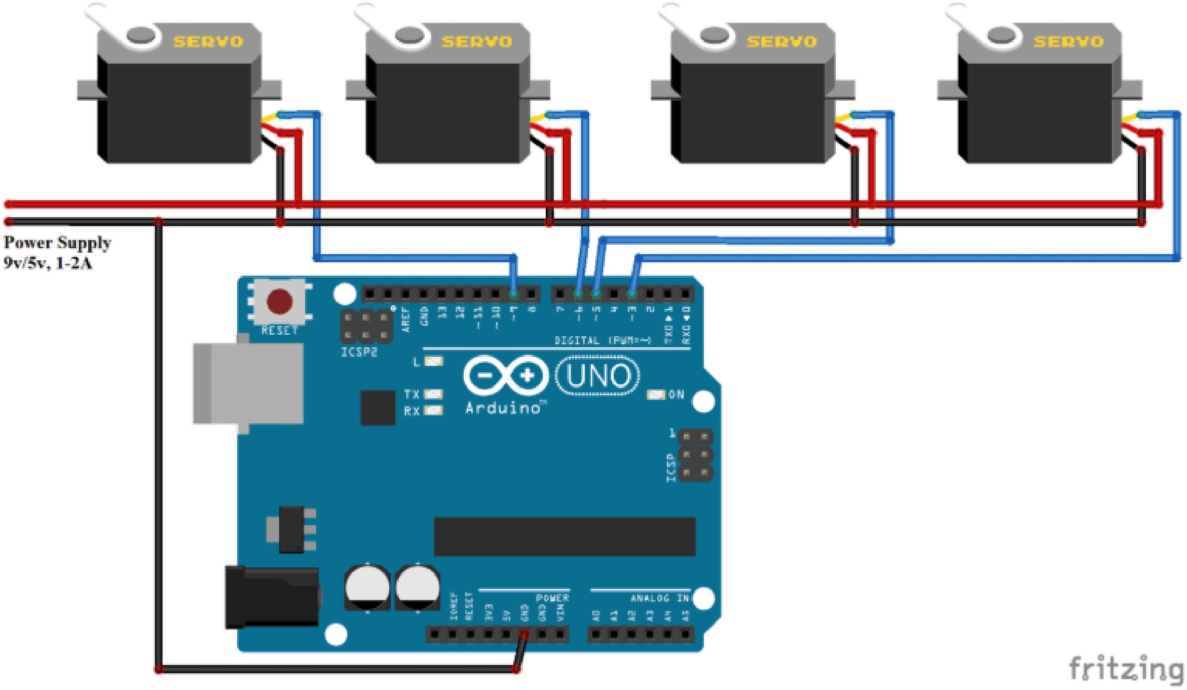
The VCC (red wire) of the buzzer is connected to an output pin of the arduino and the GND (black wire) is connected to the ground pin of the arduino. Suppose if we connect a temperature sensor with the arduino and we write arduino codes about a making a condition that if the room temperature rises above a certain level then a high signal is sent through the pin that provides VCC to the buzzer making the buzzer ring. So the input to the buzzer depends on the output of the temperature sensor here. We can modify buzzers using arduino codes for example setting delays like how many seconds it will wait before the next beep.

DC Motor:



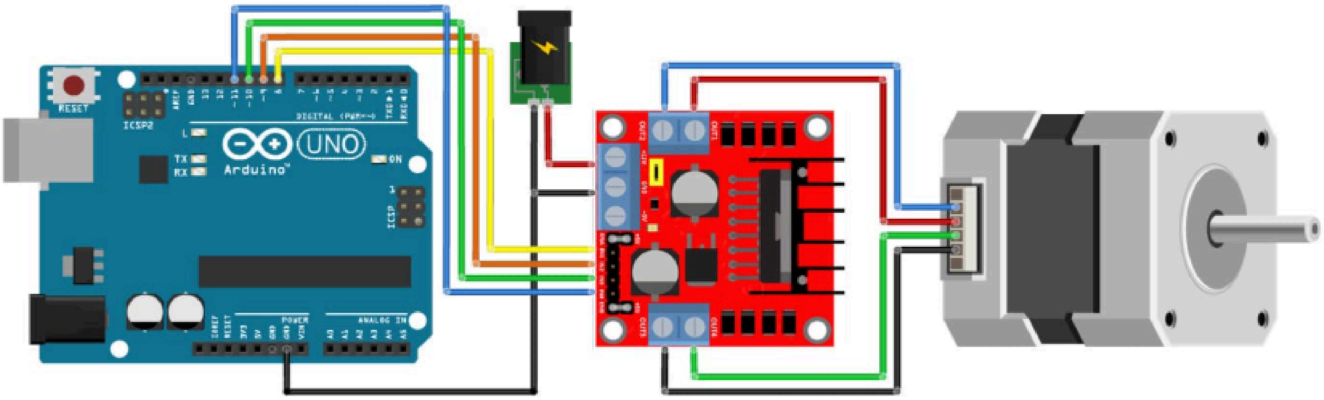
Arduino is not directly connected to the DC motor. There is a red circuit we can see in the diagram which is basically the driver circuit. Arduino cannot supply high power, it can at max provide 5v. This is why the driver circuit is used to provide power to the motor. It can handle two motors at a time. There is an external power supply connected to the driver circuit that it uses to power up the motor. We can see there are three wires that are connected to the driver circuit from the arduino. The driver circuit itself does not control anything, it just supplies power to the motor but all the control is done by the arduino, for example: when the motor will rotate, which direction to rotate, pauses between rotations, etc. Driver circuits just help to execute these controls. These controls are sent through these three wires connected to the driver circuit from the arduino.

Servo Motor:



Servo motor does not require a driver circuit because it does not require much power to rotate the motor. Servo motors basically deal with angles and these angles change according to the clock pulse of the microcontroller. If the microcontroller wants to change the angle of the servo motor, it will send a clock pulse to the servo motor and the servo motor will change the angle by a few degrees with each clock pulse.

Stepper Motor:

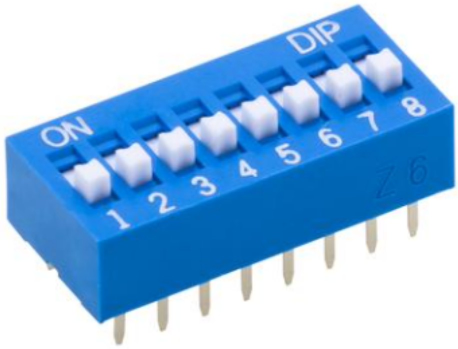


Stepper motor uses a driver circuit. There is a coil inside the stepper motor. A magnetic field is needed to be generated in this coil. To generate this magnetic field, a sufficient amount of current needs to be passed which cannot be provided by the microcontroller itself. So, therefore a driver circuit is required.

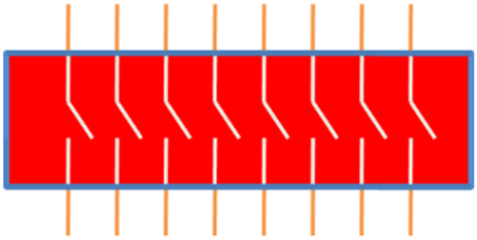
Till now all the devices we learned about are all output devices, now we will learn about low-level input devices.

- Dip Switch
- Push-Button Switch
- Matrix Keyboard
- Analog Input (Sensors)

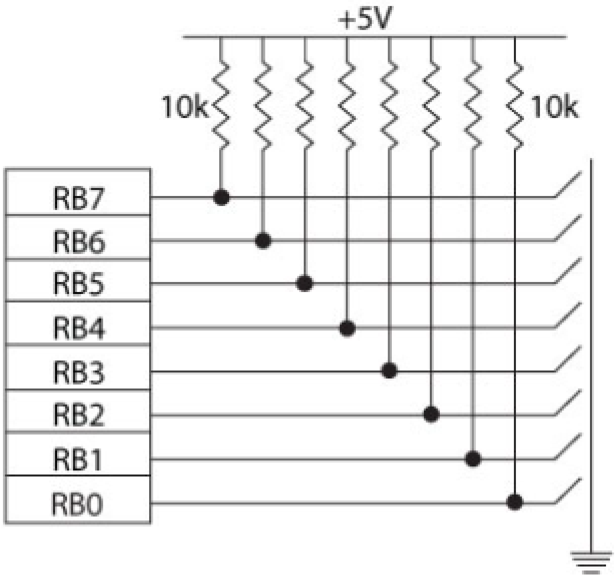
Dip Switch:



A block of 8 switches. One side of the switch is tied high (to a power supply through a resistor called a pull-up resistor), and the other side is grounded. The logic level changes when the position is switched. When we set this switch into a breadboard, we connect the "on" pin to the VCC of the breadboard. For any type of switches we use, we provide an external VCC for it.

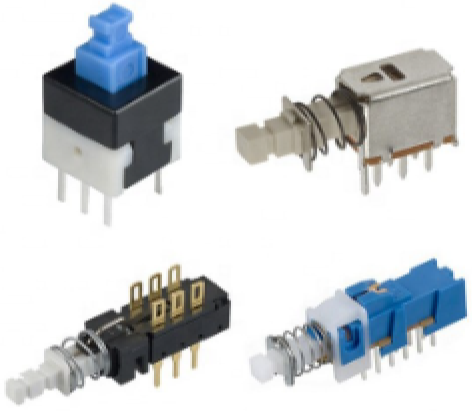


DIP Switch Schematic

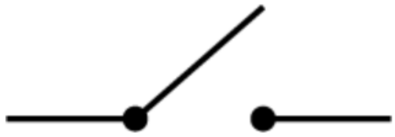


The diagram on the right is the internal circuit of a dip switch. We can see that there are 8 switches in the diagram. For every switch, one end is connected to an external VCC of 5V and this same end is connected to let's say an 82C55 IC port B. So the 82C55 is constantly receiving an input of "1" as the 5v VCC is going towards the pins of 82C55 IC port B pins. There is also an unconnected ground. If we press a switch, it establishes a connection with the ground. As a result, the 5V VCC instead of going towards the pins of 82C55 IC, it goes towards the ground. As a result the pins will get an input of "0" instead of "1". And when we receive a "0" as an input from a switch, it means the switch is pushed as switch is an active low output device.

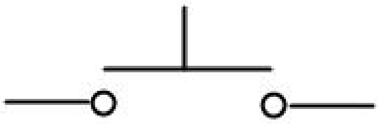
Push-button key:

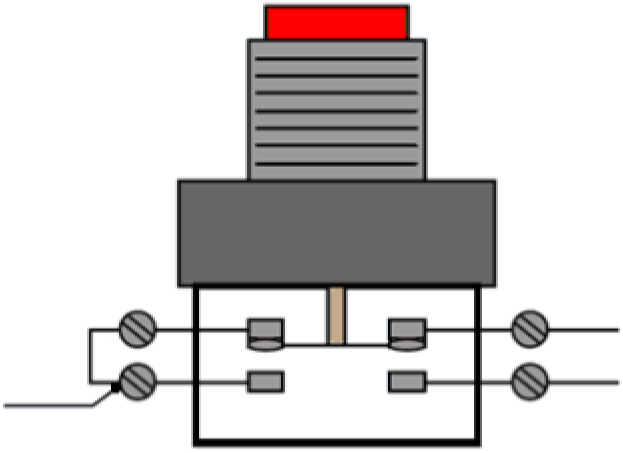


The connection is the same as in the DIP switch. Both are active low output devices. The only difference is in its mechanism. Normally switches are represented as below:



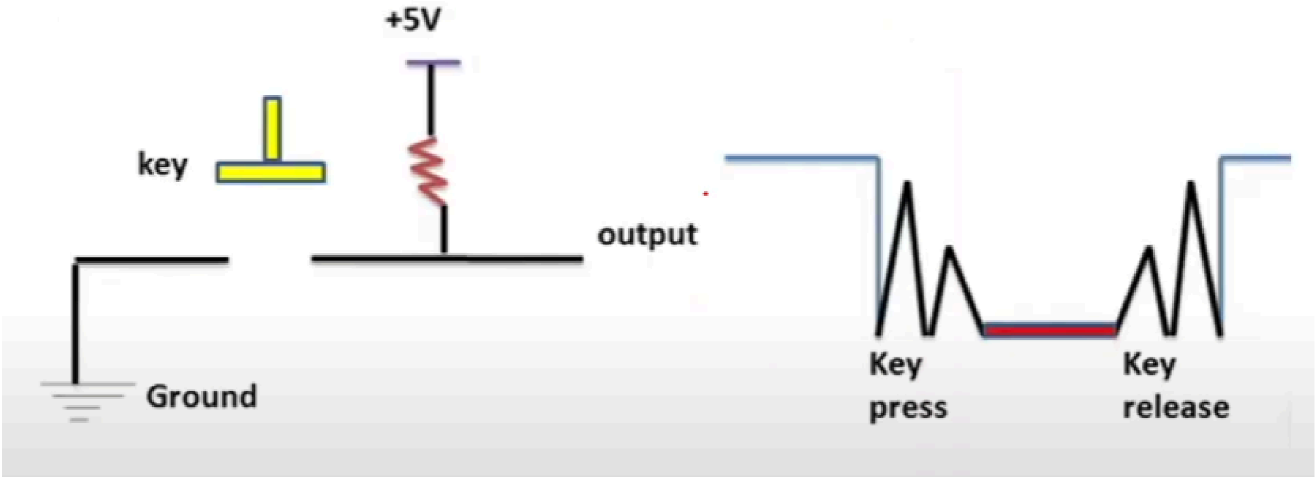
But push button has a different mechanism so they are represented as below:





There is two wires with a gap in the middle. So when we push the button down, it establishes a connection between the wires below.

Key Debouncing:



When we press a switch and the switch is descending, there is a moment when the switch is not fully touching the wire below but still have a little touch, at that very moment we receive a distorted signal as shown in the diagram above right. We suddenly get a 1 signal then a 0 signal and again 1 signal and finally when the switch is fully touching the wires below, we get a “0” output. So this distortion signal phenomenon is known as key debouncing. So due to this distortion (constantly changing between 1 and 0) there is a high chance of the circuit getting shorted, or a wrong output.

Same goes for when we unpress/release the switch. Again there is a moment when the switch is not fully touching the wire below but still have a little touch and this phenomenon happens again. We can understand this better if we look at the diagram above right, there are two phases of key debouncing. One when we press the switch and again when releasing the key and when the switch is fully in contact with the wires below we receive a “0” output as shown with a red line.

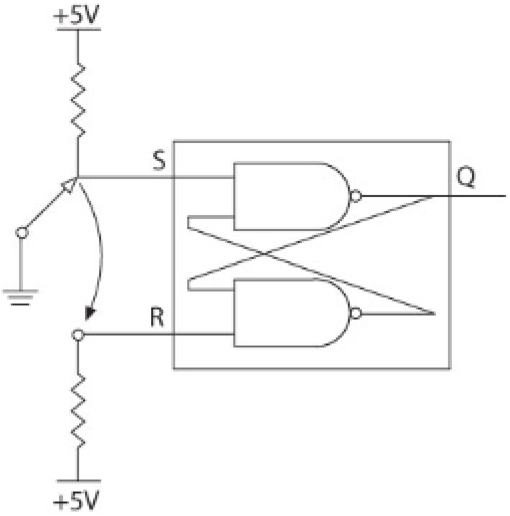
Now if we talk about our regular keyboards that we use in our pc. Key debouncing can cause many problems. For example if i press the letter “A” on the keyboard and due to key debouncing multiple A’s might appear on the screen.

There are two ways to solve such problems of reading one contact as multiple inputs which result in multiple outputs.

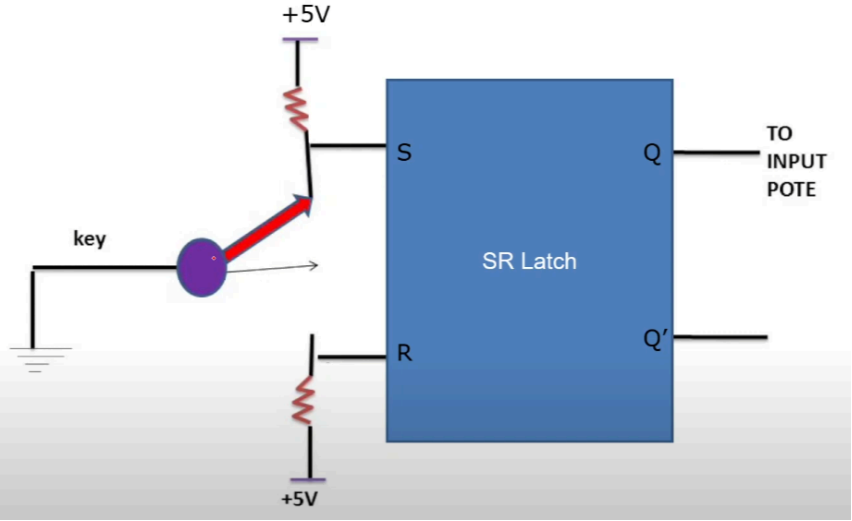
- Software key debouncing
- Hardware key debounce Techniques

Software debouncing: When a switch is pressed we create a delay of 20ms where we wait for 20ms before taking the input. Distortions happen in this 20ms so we make a delay of 20ms to avoid this distortion. Similarly the output is also taken after 20ms. So this technique helps us avoid the distortion of input/output and successfully solving key debouncing issues.

Hardware key debounce Techniques:



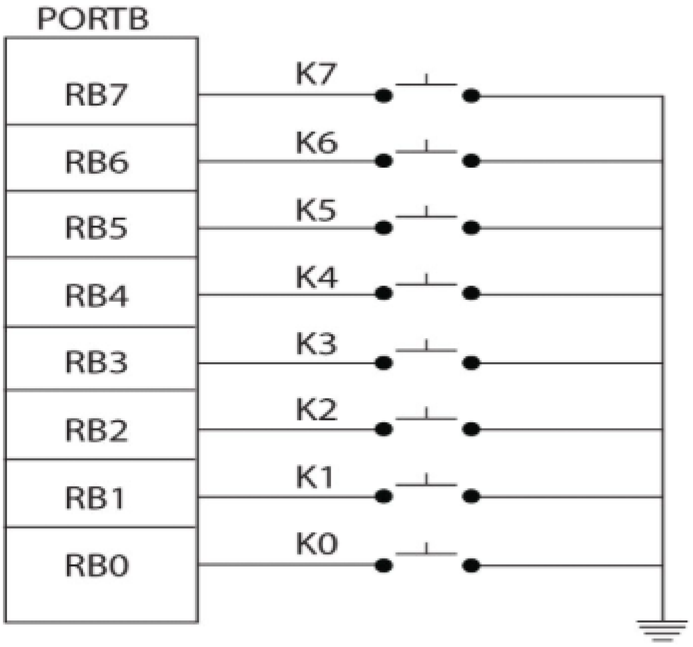
We use S-R latch to solve key debouncing. If we look into the internal circuit , we can see two NAND gates. Depending on the input given into S, we get an output through Q. This output Q is given as input to R. The Output we get from R is again given as input to S which is required for creating the output.



When the key is connected to S, the key is in the pressed state and when the key is in unpressed state, it is connected to R. If we look closely into the circuit we can see S and R are both connected to an external VCC of 5V. We will get the output through Q.

When the switch is pressed (connected to S), the 5V VCC travels towards the ground and no voltage passes through S. As a result we get a “0” input through S and R is connected to a 5V VCC so it is sending a “1” as input through R. If we trace the internal circuit then we can see that we will get a “0” through Q and a “1” through Q’. Now, when we release the key, it moves towards R. As a result the 5V VCC connected to S now travels through S instead of going towards ground. But still we will receive a “0” through Q. This is because, initially before release we got a o from Q which is given as input to R and another input for R was already “1” (connected to VCC). So, if we NAND 0 and 1 we get 1 as output which is sent as input for S. After we release the switch we are receiving 1 through one input of S and 1 through the other input of S (output of R). Therefore if we NAND 1 and 1 we get 0 and we get this o through Q. Unless the key is fully released and gets connected to R the output remains 0. The output (Q) turns 1 when the key is in contact with R. As a result we get no distorted signal here.

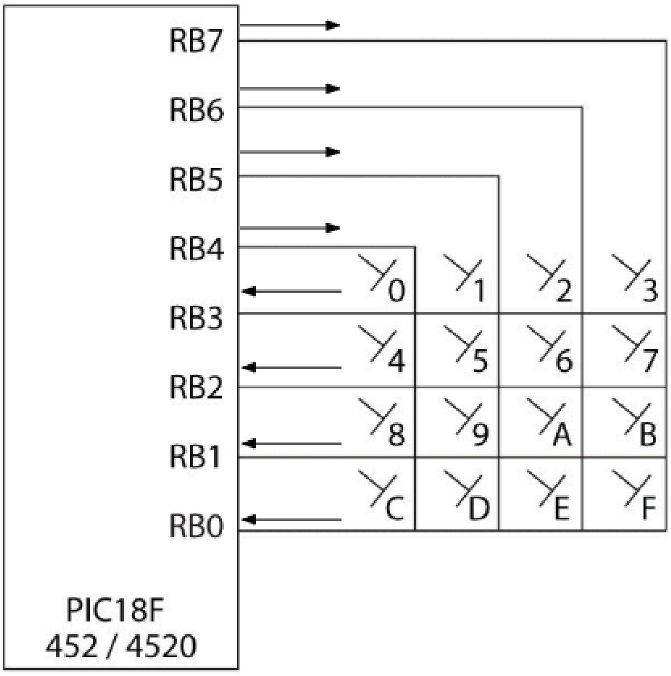
Illustration: Interfacing Push-Button Keys (1 of 6)



Problem statement

- A bank of push-button keys are connected as inputs to PORTB.
- The pull-up resistors are internal to PORTB.
- Write a program to recognize a key pressed, debounce the key, and identify its location in the key bank with numbers from 0 to 7.

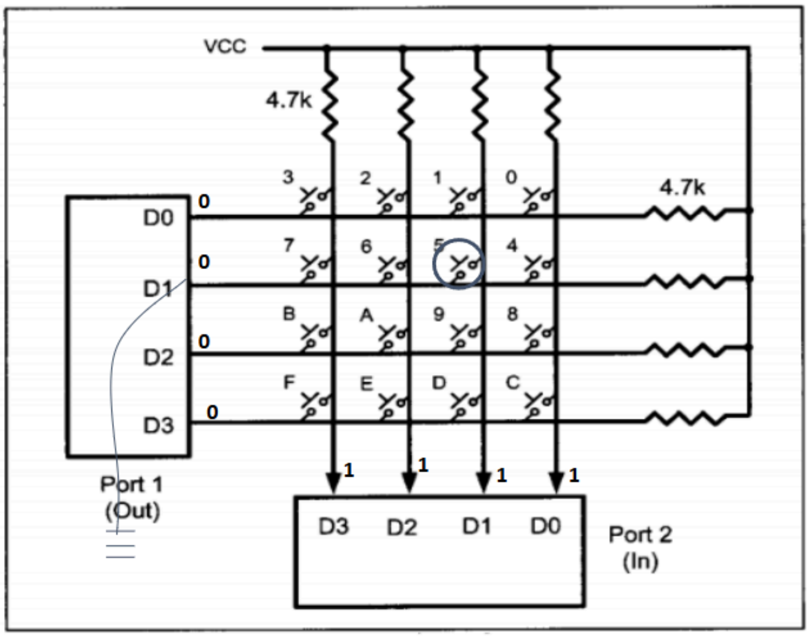
Interfacing a Matrix Keyboard:



Matrix keyboard is also an input device. The switches are giving output but these outputs are sent as input to the microprocessor. A matrix keyboard has 16 keys (0 to 9) and (A to F). It is a low level device.

We use two ports of 82C55 to interface a matrix keyboard. To configure 16 switches it requires 4 rows and 4 columns. One port works as an output port (row) and another port is used as input port (column).

Microprocessor will send some values through the output port/row and based on those values it will receive some input through the input port/ column. Based on the input it received it will distinguish what key has been pressed.



From the diagram on the left we can see that all the rows and columns wires are connected to an external VCC. Between every intersection between the rows and columns there is a switch connected. When we press those switches, it connects the row wire and column wire of that specific switch we press. And establishes a connection.

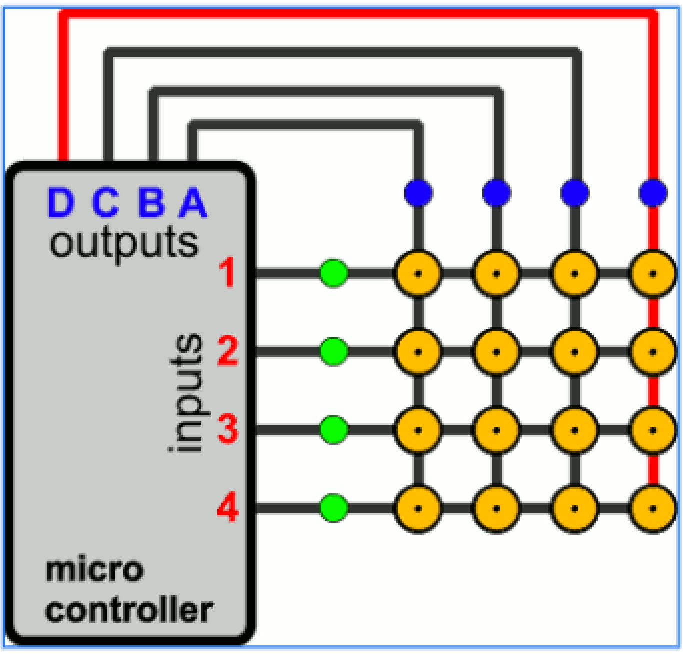
There are three steps how the microprocessor identifies which switch is pressed:

- Column identification
- Row identification
- Key identification

Initially the input pins receive 1 (column). The output pins receive 0 (row). When a key is pressed the row and column wire gets connected. As a result the column pin turns zero for that specific column the switch belongs from and the rest of the column pins stays 1. The column identification is done here.

Next, for row identification the microprocessor by putting 0 in one row and putting 1 on the rest and checks if the input changes from 1111 to something else. In each iteration the zero is shifted to the next row for example: iteration 1: 0111, iteration 2: 1011, iteration 3: 1101, iteration 4: 1110. At every iteration it puts 0 on one of the rows and checks if the input changes for that combination. When it finds out that the input changes if we put 0 in a certain row then it identifies that the switch might belong from that particular row and row identification is done.

During column identification the input value is kept in a register. Also the row that is identified, all the switches that belong to that row are brought and compared. In the comparison the switch corresponding to the 0 is the switch that is pressed and that value is sent to the keypressed variable.



- Ground one column at a time and check all the rows in that column.
- Once a key is identified, it is encoded based on its position in the column.