USING

TO REPLACE COMBINATIONAL LOGIC

PRESENTED BY-
ARPIT GARG 2K20/EC/047

INTRODUCTION

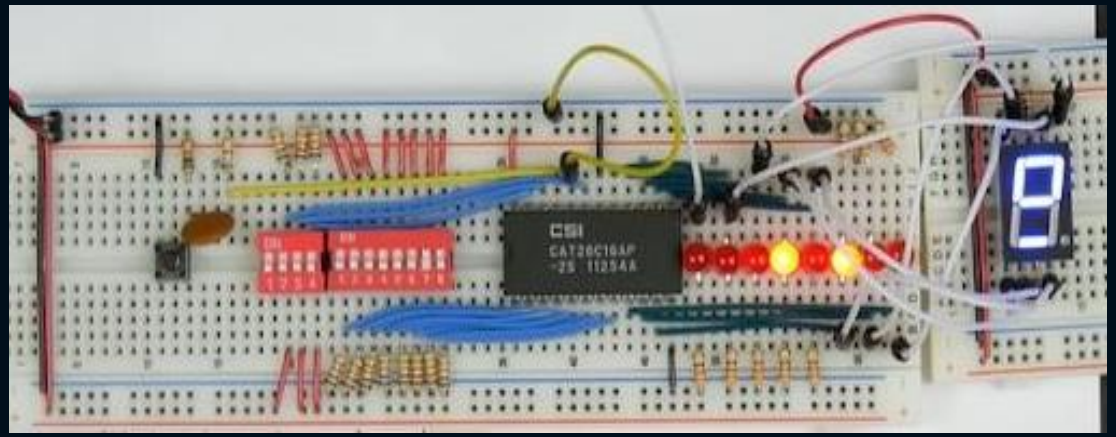
Work well turns out you can **replace** any **combinational logic** circuit with a ROM and so rather than having to design this complicated circuit with all of these gates and everything going on here we could **replace** all of this with a ROM that's because remember where this came from we started with this basic truth table which just describes how the data bits coming in relation to whether the segments are turned ON or OFF well we can actually program this truth table into a ROM and the data bits.

Become the address and then the data output of the ROM becomes these outputs over here. So at address 0 we could program in 000 0001 and address one in the ROM we can program in 100 1111 one and so forth and then we could put the ROM in the circuit instead of this and instead of these switches feeding the input of this fairly complicated **logic** circuit, these switches could feed the address of our ROM and then the outputs instead of coming out of this collection of OR gates and so

In this project, we'll wire up an EEPROM (28C16) so we can read its contents. We'll also take a look at the data sheet to learn how to program it, and try programming some values. Finally, we'll see how the EEPROM can be used to replace any a combinational logic circuits such as the 7-segment decoder and many more

Complete parts list (everything in this video):

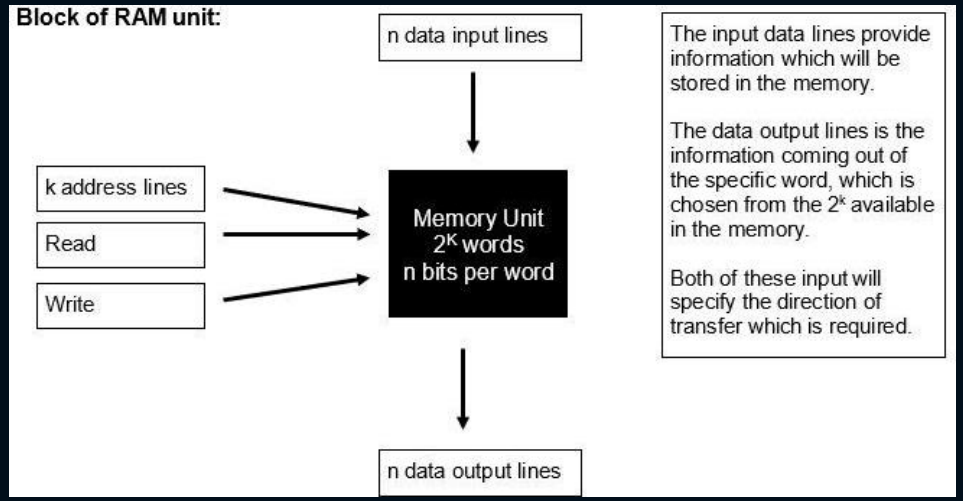
- -1x 28C16 EEPROM
- -8x LEDs
- -8x 330Ω resistors
- -1x 8-position DIP switch
- -1x 4-position DIP switch
- -12x 10kΩ resistors
- - 1x 100nF capacitor
- - 1x 680Ω resistor
- - 1x momentary tact switch
- - 1x Common Anode 7-segment display
- -1x 100Ω resistor.



In a computer system, memory is a very essential part of the computer system and is used to store information for instant use or permanently.

Volatile Memory (RAM):

Random Access memory is present on the motherboard and the computer's data is temporarily stored in RAM. RAM can help in both Read and writing. RAM is a volatile memory, which means, it is present as long as the Computer is in ON state, as soon as the computer turns OFF, the memory is erased.



Architecture of RAM

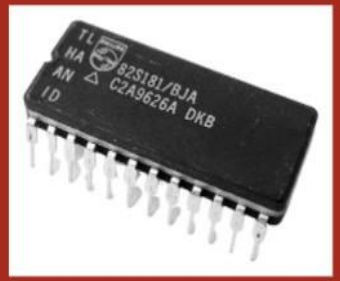
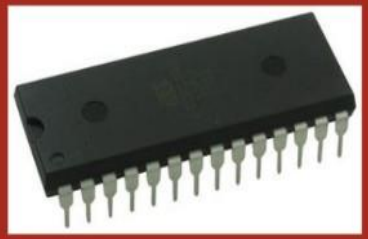
Non-volatile Memory (ROM):

ROM is a non-volatile read only storage unit within electronic systems.ROM is a type of memory, which is useful at storing information, which doesn't change during its lifespan in the system, this is referred to as firmware.

Read-only memory is memory, which is hard-wired, this can include a diode matrix, for instance, a system that cannot be electronically changed.

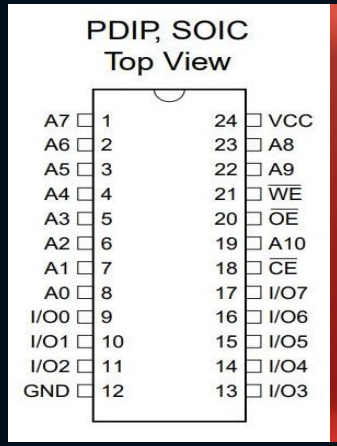
Architecture of ROM

Different Types Of Rom

01	PROM		• A PROM that can be modified only by users • Stands for PROGRAMMABLE READ ONLY MEMORY • Reprogrammable only once
02	EPROM		• A programmable ROM that can be erased and used • Stands for ERASABLE PROGRAMMABLE ROM • Can be reprogrammed using UV light
03	EEPROM		• A user modifiable ROM that can be erased and reprogrammed • Stands for ELECTRICALLY ERASABLE ROM • Can be reprogrammed using electrical charge

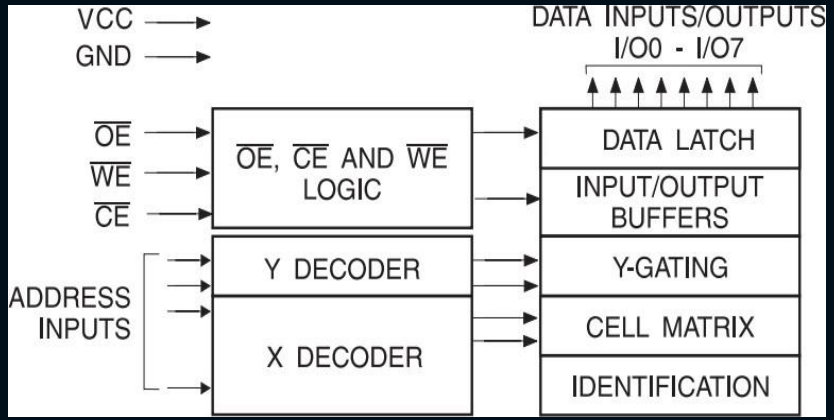
EEPROM AT28C16 IC

The AT28C16 is a low-power, high-performance Electrically Erasable, and Programmable Read-Only Memory with easy-to-use features. The AT28C16 is a 16K memory organized as 2,048 words by 8 bits.



Pin Name	Function
A0 - A10	Addresses
$\overline{CE}$	Chip Enable
$\overline{OE}$	Output Enable
$\overline{WE}$	Write Enable
I/O0 - I/O7	Data Inputs/Outputs
NC	No Connect
DC	Don't Connect

The AT28C16 is accessed like a static RAM for the read or write cycles without the need of external components. During a byte write, the address and data are latched internally, freeing the microprocessor address and data bus for other operations. Following the initiation of a write cycle, the device will go to a busy state and automatically clear and write the latched data using an internal control timer. The end of a write cycle can be determined by DATA POLLING of I/O7.



Device Operation

- **READ:** The AT28C16 is accessed like a Static RAM. When CE and OE are low and WE is high, the data stored at the memory location determined by the address pins is asserted on the outputs. The outputs are put in a high impedance state whenever CE or OE is high.

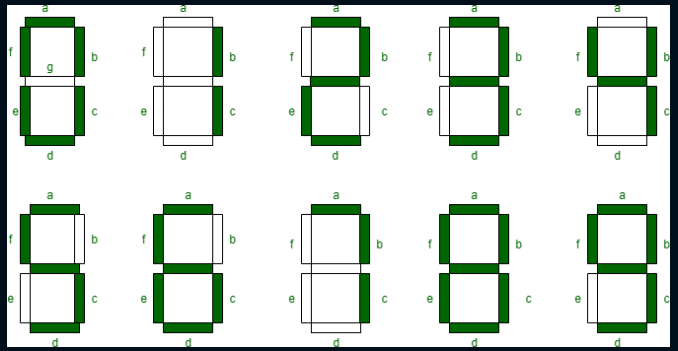
- **BYTE WRITE:** Writing data into the AT28C16 is similar to writing into a Static RAM. A low pulse on the WE or CE input with OE high and CE or WE low (respectively) initiates a byte write.

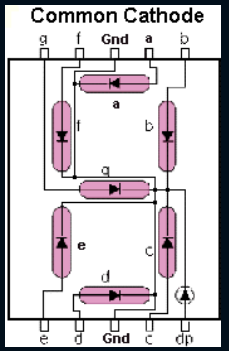
- **WRITE PROTECTION:** Inadvertent writes to the device are protected against in the following ways: (a) VCC sense—if VCC is below 3.8V the write function is inhibited; (b) VCC power on delay—once VCC has reached 3.8V the device will automatically time out 5 ms before allowing a byte write; and (c) write inhibit—holding any one of OE low, CE high or WE high inhibits byte write cycles.

- **CHIP CLEAR:** The contents of the entire memory of the AT28C16 may be set to the high state by the CHIP CLEAR operation. By setting CE low and OE to 12 volts, the chip is cleared when a 10 msec low pulse is applied to WE.

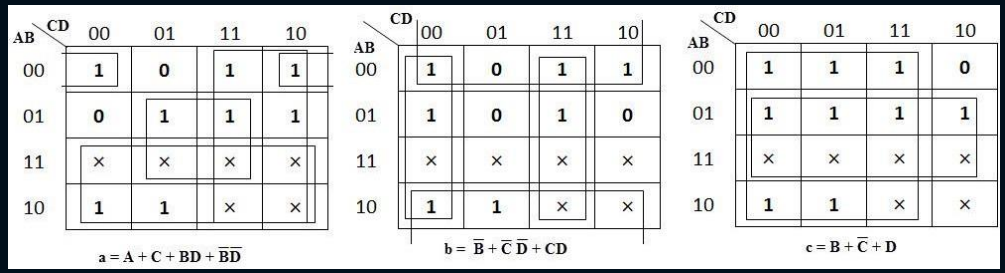
THE TRADITIONAL APPROACH TO LOGIC GATES

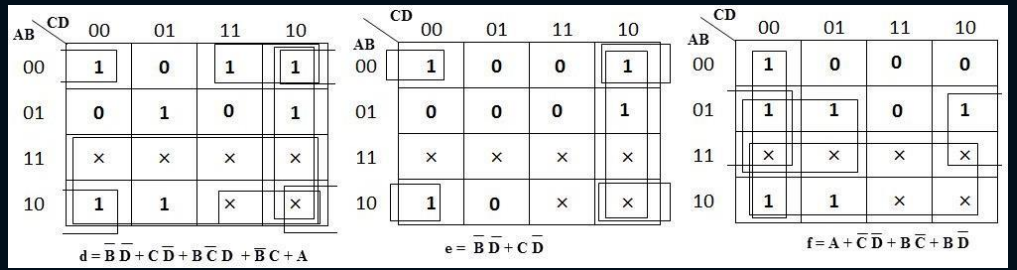
For any combinational circuit first, we need is the truth table of the logic expression and thenwe have to obtain equations through k-maps, for every variable different equation needs to be formulated and therefore the task becomes tedious





A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1





AB \ CD	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	×	×	×	×
10	1	1	×	×

$g = \bar{B}C + C\bar{D} + B\bar{C} + B\bar{C} + A$

From the above simplification, we get the output values as

$a = A + C + BD + \bar{B}\,\bar{D}$

$b = \bar{B} + \bar{C}\,\bar{D} + CD$

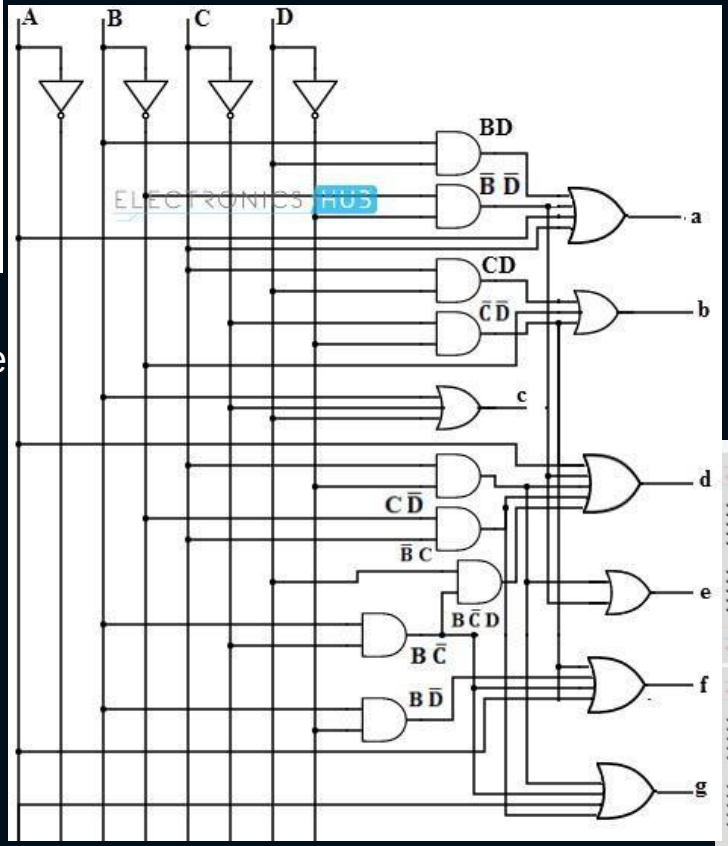
$c = B + \bar{C} + D$

$d = \bar{B}\,\bar{D} + C\,\bar{D} + B\,\bar{C}\,D + \bar{B}\,C + A$

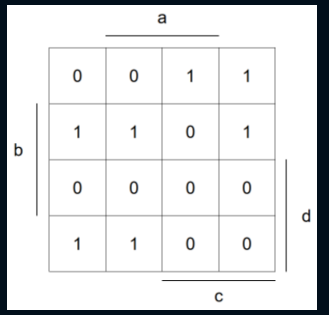
$e = \bar{B}\,\bar{D} + C\,\bar{D}$

$f = A + \bar{C}\,\bar{D} + B\,\bar{C} + B\,\bar{D}$

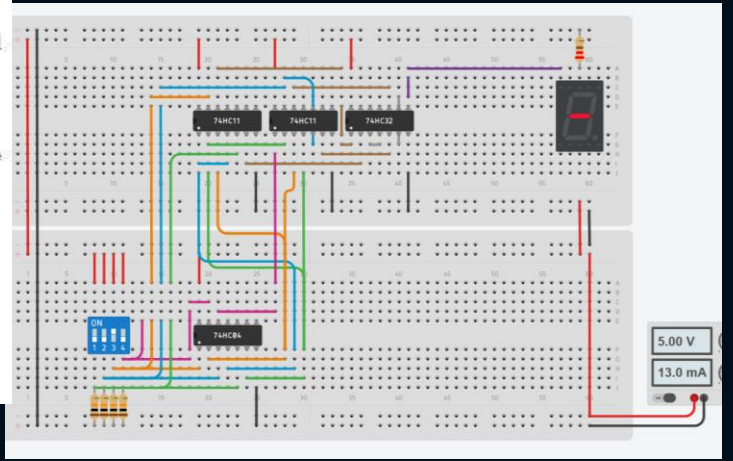
$g = A + B\,\bar{C} + \bar{B}\,C + C\,\bar{D}$



A through **G are the outputs of the logic circuit** that will connect to the corresponding inputs on a 7 - segment display.

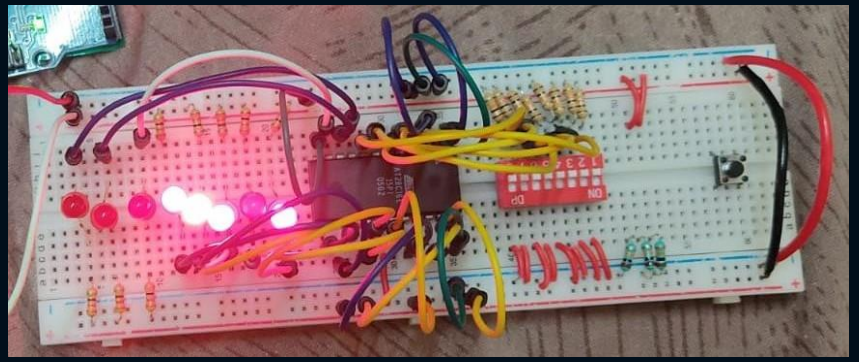


$(B \wedge \neg C \wedge \neg D) \vee (\neg B \wedge \neg C \wedge D) \vee (\neg A \wedge C \wedge \neg D) \vee (\neg B \wedge C \wedge \neg D)$

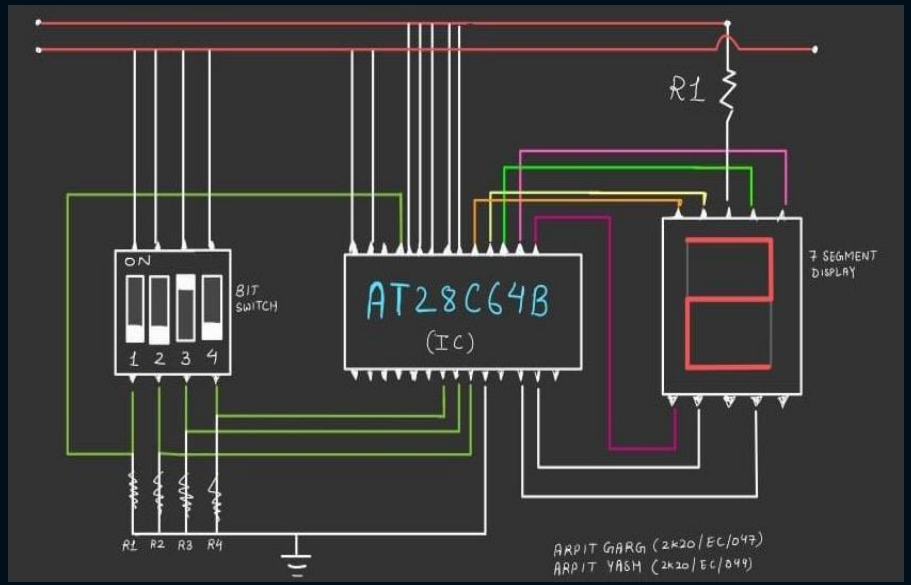


Logic circuit of G segment

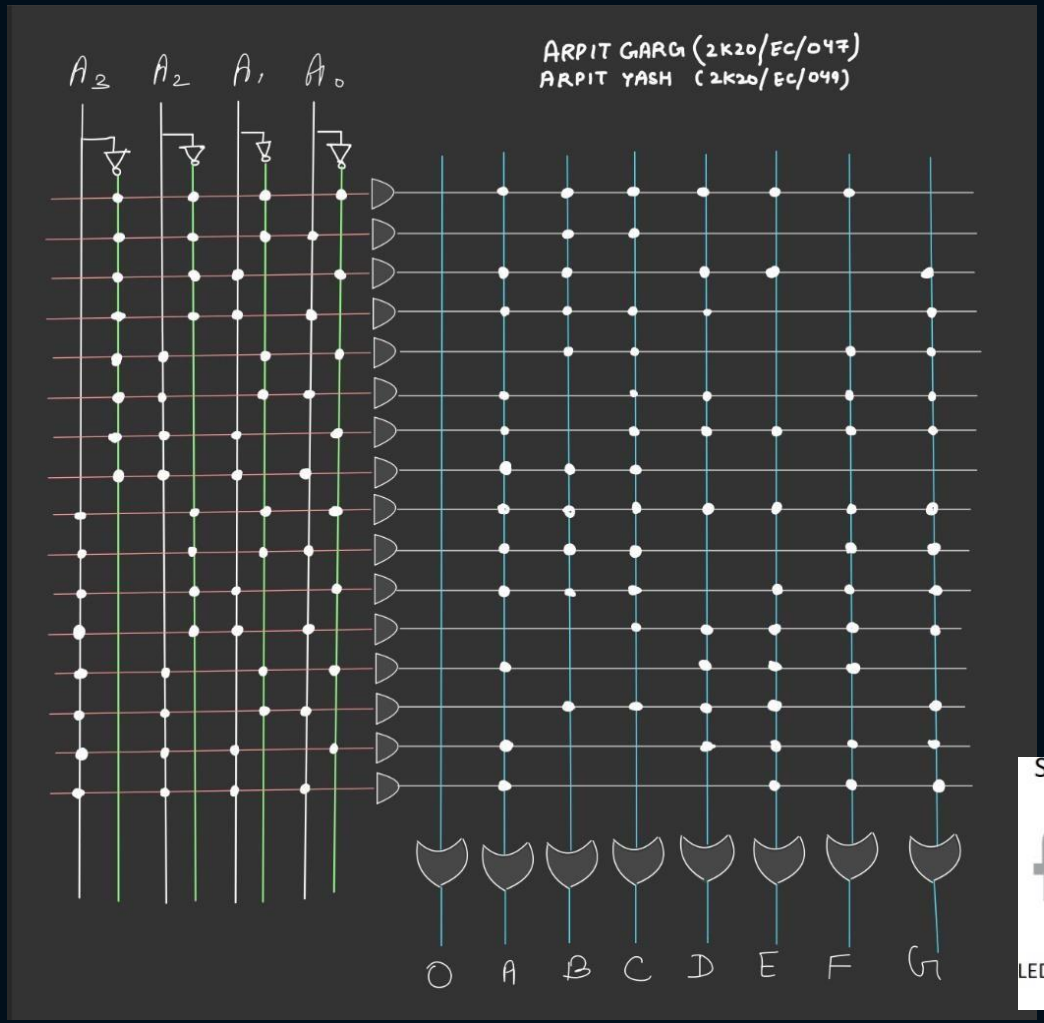
AN ALTERNATIVE TO LOGIC GATES: USING A ROM CHIP



The four inputs (A to D) get connected to any four address lines of the EEPROM. The seven segments of the display (A to G) connect to any seven data lines of the EEPROM:



Address (A12-A0)	Data (I/O7-I/O0)
0000 0000 0000	01111110
0000 0000 0001	00110000
0000 0000 0010	01101101
0000 0000 0011	01111001
0000 0000 0100	0110011
0000 0000 0101	01011011
0000 0000 0110	01011111
0000 0000 0111	01110000
0000 1000 0000	01111111
0000 1000 0001	01111011



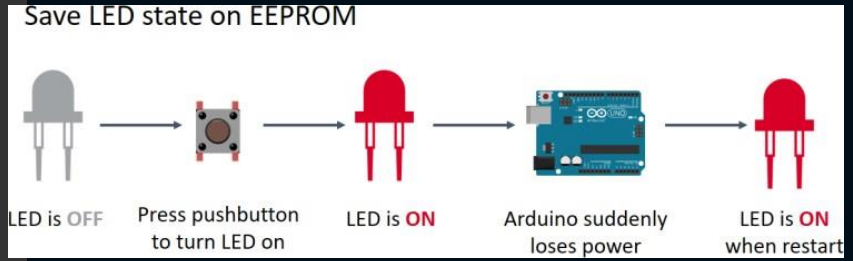
EEPROM LIBRARY (ARDUINO)

The microcontroller on the Arduino and Genuino AVR-based board has EEPROM

- 1024 bytes on the ATmega328P,

#include <EEPROM.h>

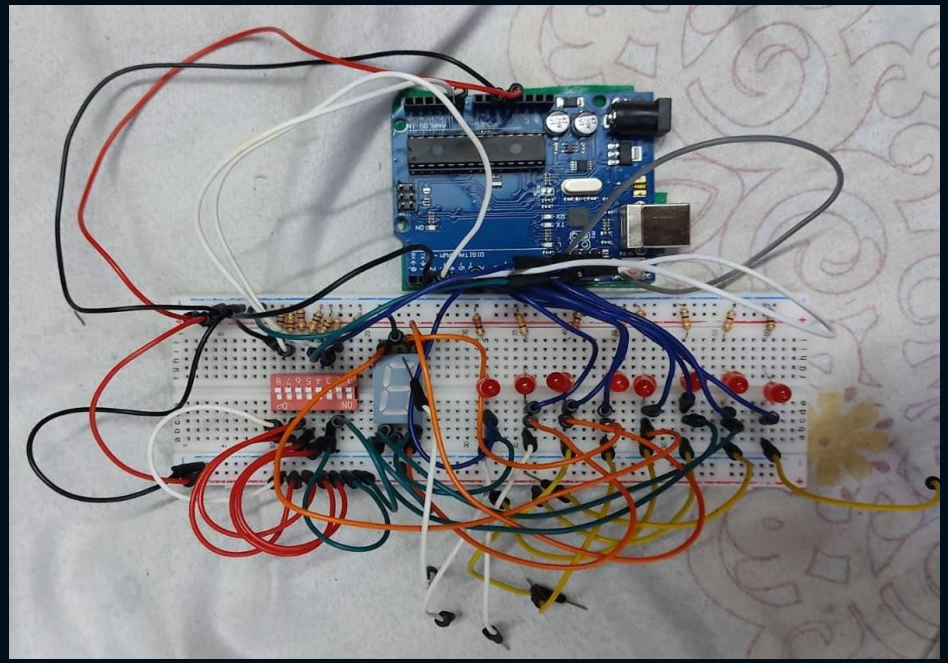
- **read():** Reads a byte from the EEPROM. Locations that have never been written to have the value of 255.
  Syntax: EEPROM.read(address)
- **write():** write a byte to the EEPROM.
  Syntax: EEPROM.write(address,value)



ARDUINO EEPROM CODE

```
#include<EEPROM.h>
#define EEPROM_D0 5
#define EEPROM_D7 12
// 4-bit hex decoder for common cathode 7-segment display
byte data[] = { 0x7e, 0x30, 0x6d, 0x79, 0x33, 0x5b, 0x5f, 0x70, 0x7f,
0x7b, 0x77, 0x1f, 0x4e,0x3d, 0x4f, 0x47 };
int value;
void writeEEPROM(int address, byte data) {
  for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1)
  {  pinMode(pin, OUTPUT);
  }
  for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1)
  { digitalWrite(pin, data & 1);
    data = data >> 1;
  }
}
void setup()
{
  Serial.begin(9600);
  Serial.print("Erasing EEPROM \n");
for (int address = 0; address <=15; address += 1)
  {
    EEPROM.write(address,0xFF);
    //Erase EEPROM by setting all databits high
  }
  Serial.print("Erased EEPROM \n");
  for (int address = 0; address <=15; address += 1)
  //Print data present at all the address lines after erasing EEPROM
  {
  value = EEPROM.read(address);
  Serial.print(address);
  Serial.print(" :0x");
  Serial.println(value,HEX);
  writeEEPROM(address, value); Serial.print("\n");
    delay(50);
  }
  Serial.print("Program EEPROM \n");
  for (int address = 0; address <=15; address += 1)//Write data at required address lines
  { EEPROM.write(address, data[address]);
  }
  Serial.print("Programmed EEPROM \n");
  for (int address = 0; address <=15; address += 1)
  //Print data present at all the address lines after writing EEPROM
{
value = EEPROM.read(address);
Serial.print(address);
Serial.print(" :0x");
Serial.println(value,HEX);
writeEEPROM(address, value);
Serial.print("\n");
delay(5000);
  }
}
void loop()
{
}
```

OUTPUT



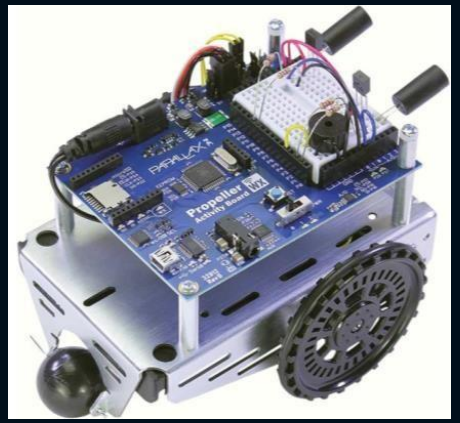
```
Erasing EEPROM
Erased EEPROM
0  :0xFF

1  :0xFF

2  :0xFF

3  :0xFF

4  :0xFF

5  :0xFF

6  :0xFF

7  :0xFF

8  :0xFF

9  :0xFF

10  :0xFF

11  :0xFF

12  :0xFF

13  :0xFF

14  :0xFF

15  :0xFF
```

```
Program EEPROM
Programmed EEPROM
0  :0x7E

1  :0x30

2  :0x6D

3  :0x79

4  :0x33

5  :0x5B

6  :0x5F

7  :0x70

8  :0x7F

9  :0x7B

10  :0x77

11  :0x1F

12  :0x4E

13  :0x3D

14  :0x4F

15  :0x47
```

SCOPE OF USE

Electrically Erasable Programmable Read-Only Memory (EEPROM) is a steady, non-volatile memory storage scheme that is used for storing minimal data quantities in computer and electronic systems and devices, such as circuit boards. This data may be stored, even with no enduring power source, as device patterns or calibration tables.

As I was working on a project which sends the real-time data to the ground control station the connection gets broken sometimes so instead of data being lost we can store it in EEPROM and send it back as soon as the connection is set up thus avoiding the use of external memory storage in our robot.



CONCLUSION

Many complex logic circuits can be replaced by a single ROM chip. However, ROM chips are typically much slower and more expensive than logic ICs. Therefore, it only really makes sense to use ROM chips in circuits with many inputs and in circuits where speed isn’t important.