

AIM OF THE PROJECT

Logic gates can be utilized to implement complex logic functions in a circuit, but doing all the necessary calculations can be a tedious and time-consuming process. However, memory chips can serve as a simple alternative because they can also be used to represent logic functions.

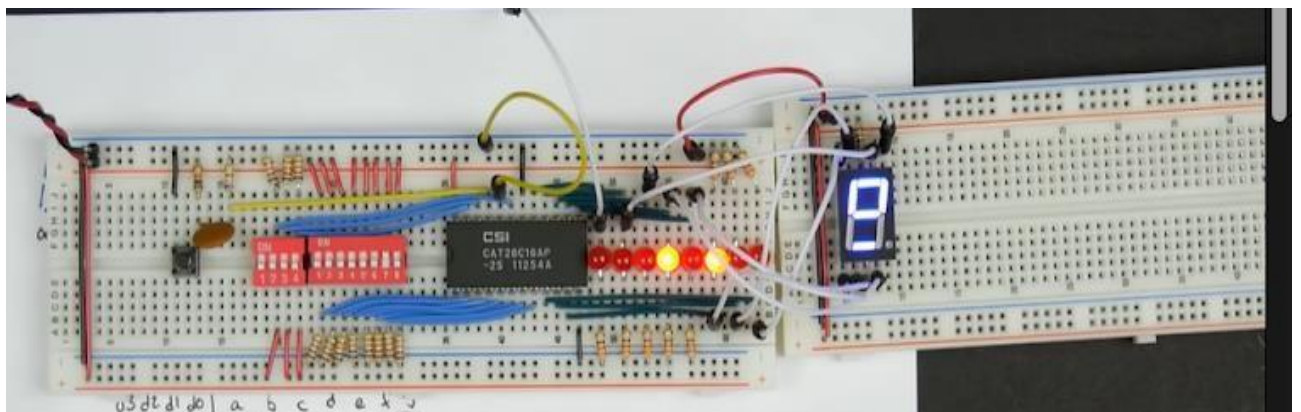
Work well turns out you can replace any combinational logic circuit with a ROM and so rather than having to design this complicated circuit with all of those gates, we could replace all of them with a ROM that's because remember where this came from we started with this basic truth table which just describes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

In this project, we'll wire up an EEPROM (In Arduino) 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 combinational logic circuit such as the 7-segment decoder and many more

Complete parts list :

- - (1x 28C16) EEPROM in Arduino Uno / AC218C6
- - 8x LEDs
- - 8x 330 Ω resistors
- - 1x 8-position DIP switch
- - 1x 4-position DIP switch
- - 12x 10k Ω resistors
- - 1x 100nF capacitor
- - 1x 680 Ω resistor
- - 1x momentary tact switch
- - 1x Common Anode 7-segment display
- - 1x 100 Ω resistor.



INTRODUCTION

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. Based on computer memory working features, memory is divided into two types.

- **Volatile Memory (RAM)**
- **Non-volatile Memory (ROM)**

Before understanding ROM, we will first understand what exactly volatile and non-volatile memory is. Non-volatile memory is a type of computer memory that is used to retain stored information during power is removed. It is less expensive than volatile memory. It has a large storage capacity. ROM (read-only memory), and flash memory are examples of non-volatile memory. Whereas volatile memory is a temporary memory. In this memory, the data is stored till the system is capable, but once the power of the system is turned off the data within the volatile memory is deleted automatically. RAM is an example of volatile memory.

What is RAM?

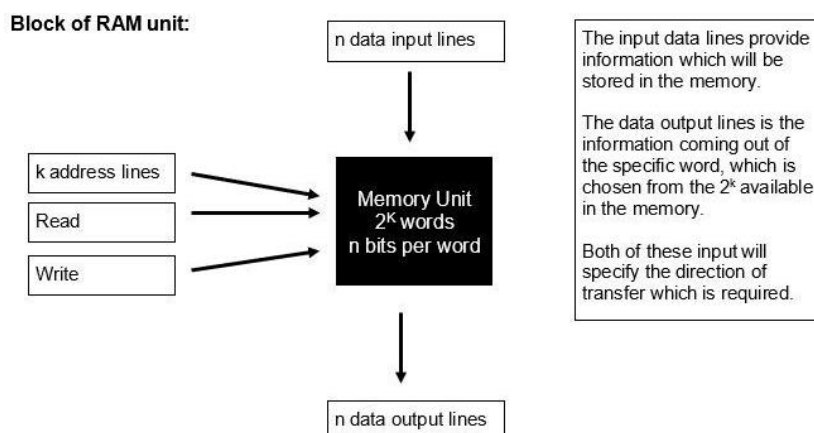
RAM = Random Access Memory

Random Access memory is present on the motherboard and the computer's data is temporarily stored in RAM. As the name says, RAM can help in both Read and write. 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.

In order to better understand RAM, imagine the blackboard of the classroom, the students can both read and write and also erase the data written after the class is over, some new data can be entered now.

Architecture of RAM

Due to RAM architecture, the memory cells can be accessed for information from anywhere on the computer system. This communication between different peripherals and RAM is achieved by data input and output lines, control lines which specify the direction of transfer, and address selection lines.



What is ROM?

ROM = “Read Only Memory”

ROM is a non-volatile read only storage unit within electronic systems, however, with ROM, data that is stored inside the memory units cannot be electrically modified after the device has been manufactured, as it is hard wired.

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:



Information in the form of binary is stored permanently inside ROM by the manufacture, the information is injected in the form of bits. ROM consists of logic gates only, arranged in a way that they store specified bits.

Block structure:

The unit consists of k input lines and n out lines.

- The k input lines take the input address from where we want to access the content of the ROM.
- Since the input lines are either 0 or 1 (binary form). The input lines can be referred to as 2^k total addresses and each of these addresses contains n bit of information, which will be given out as the output of the ROM. This is specified as $2^k \times n$ ROM

Internal structure:

The internal structure consists of two components: the decoder and OR (logic) gates.

- The decoder is a combinational circuit. It is used to decode any encoded form like binary to understandable forms like decimal form. Within the ROM structure, the input into a decoder will be binary and the output will be represented in decimal form.
- All the OR logic gates will have outputs of the decoder as their input.

The ROM does not need a read-control line since at any given time, the output lines automatically provide the n bits of the word selected by the address value. Because the outputs are a function of only the present inputs (the address lines), a ROM is classified as a

combinational circuit. In fact, a ROM is constructed internally with decoders and a set of OR gates. There is no need for providing storage capabilities as in RAM, since the values of the bits in the ROM are permanently fixed.

ROMs find a wide range of applications in the design of digital systems. As such, it can implement any combinational circuit with k inputs and n outputs. When employed in a computer system as a memory unit, the ROM is used for storing fixed programs that are not to be altered and for tables of constants that are not subject to change. ROM is also employed in the design of control units for digital computers. As such, they are used to store coded information that represents the sequence of internal control variables needed for enabling the various operations in the computer. A control unit that utilizes a ROM to store binary control information is called a microprogrammed control unit.

Features of ROM (Read-Only Memory):

- ROM is a non-volatile memory.
- Information stored in ROM is permanent.
- Information and programs stored on it, we can only read.
- Information and programs are stored on ROM in binary format.
- It is used in the start-up process of the computer.

WHY EEPROMS?

To understand why we use EEPROMS, let us discuss what different types of Read-Only Memory (ROMs) are:

1. **MROM (Masked read-only memory)**
2. **PROM (Programmable read-only memory)**
3. **EPROM (Erasable programmable read-only memory)**
4. **EEPROM (Electrically erasable programmable read-only memory)**

1. MROM (Masked read-only memory): We know that ROM is as old as semiconductor technology. MROM was the very first ROM that consisted of a grid of word lines and bit lines joined together with transistor switches. This type of ROM data is physically encoded in the circuit and only be programmed during fabrication. It was not so expensive.

2. PROM (Programmable read-only memory): PROM is a form of digital memory. In this type of ROM, each bit is locked by a fuse or anti-fuse. The data stored in it are permanently stored and can not be changed or erasable. It is used in low-level programs such as firmware or microcode.

3. EPROM (Erasable programmable read-only memory): EPROM also called EROM, is a type of PROM but it can be reprogrammed. The data stored in EPROM can be erased and

reprogrammed again by ultraviolet light. Reprogrammed of it is limited. Before the era of EEPROM and flash memory, EPROM was used in microcontrollers.

4. EEPROM (Electrically erasable programmable read-only memory): As its name refers, it can be programmed and erased electrically. The data and program of this ROM can be erased and programmed about ten thousand times. The duration of erasing and programming of the EEPROM is near about 4ms to 10ms. It is used in microcontrollers and remote keyless systems.




Advantages of ROM:

- It is cheaper than RAM and it is non-volatile memory.
- It is more reliable as compared to RAM.
- Its circuit is simple as compared to RAM.
- It doesn't need refreshing time because it is static.
- It is easy to test.

Disadvantages of ROM:

- It is a read-only memory, so it cannot be modified.
- It is slower as compared to RAM.
- Difference between PROM and EPROM.

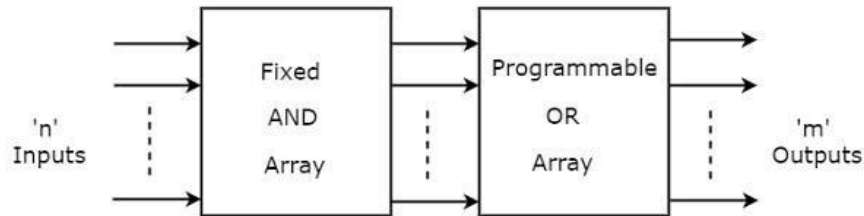
Memory Type	Category	Erase	Write Mechanism	Volatility
Random-access memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile
Read-only memory (ROM)	Read-only memory	Not possible	Masks	Nonvolatile
Programmable ROM (PROM)			Electrically	
Erasable PROM (EPROM)	Read-mostly memory	UV light, chip-level		
Electrically Erasable PROM (EEPROM)		Electrically, byte-level		
Flash memory		Electrically, block-level		

01	PROM		<ul style="list-style-type: none"> • A PROM that can be modified only by users • Stands for PROGRAMMABLE READ ONLY MEMORY • Reprogrammable only once
02	EPROM		<ul style="list-style-type: none"> • A programmable ROM that can be erased and used • Stands for ERASABLE PROGRAMMABLE ROM • Can be reprogrammed using UV light
03	EEPROM		<ul style="list-style-type: none"> • A user modifiable ROM that can be erased and reprogrammed • Stands for ELECTRICALLY ERASABLE ROM • Can be reprogrammed using electrical charge

PROM:

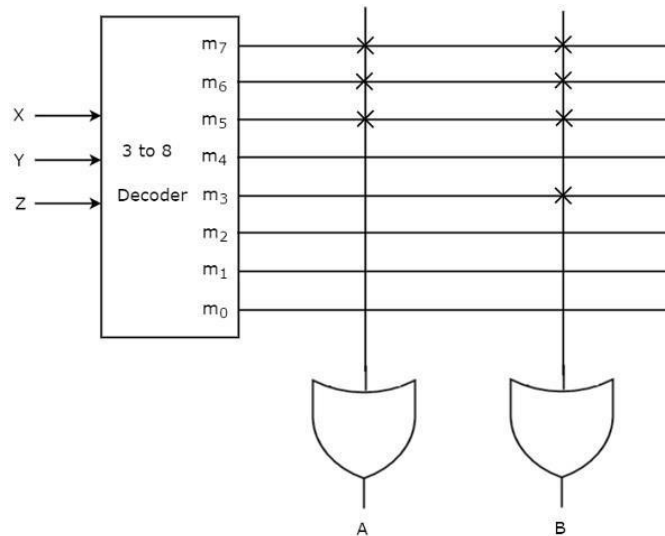
The user has the flexibility to program the binary information electrically once by using a PROM programmer.

PROM is a programmable logic device that has a fixed AND array & Programmable OR array. The block diagram of PROM is shown in the following figure.



Here, the inputs of AND gates are not of programmable type. So, we have to generate 2^n product terms by using 2^n AND gates having n inputs each. We can implement these product terms by using the $n \times 2^n$ decoder. So, this decoder generates ' n ' minterms.

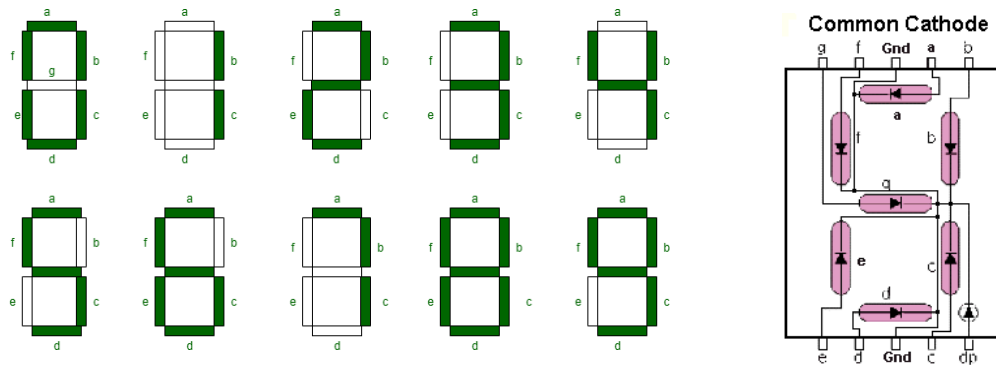
Here, the inputs of OR gates are programmable. That means, we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate. Therefore, the outputs of PROM will be in the form of a sum of minterms.



THE TRADITIONAL APPROACH TO LOGIC GATES

Let's first take a look at how the circuit mentioned above could be designed with logic gates in mind. Typically, you start by creating a truth table that contains all relevant combinations of your input signals, as well as the corresponding value of the output with that combination. For any combinational circuit first, we need is the truth table of the logic expression and then we have to obtain equations through k-maps, for every variable different equation needs to be formulated and therefore the task becomes tedious.

For example, we can look at the design of a 7-segment decoder



Suppose the column for segment a shows the different combinations for which it is to be illuminated. So 'a' is active for the digits 0, 2, 3, 5, 6, 7, 8, and 9.

BCD to common cathode 7 segment truth table:

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

A truth table containing all relevant combinations of your input signals and the corresponding value of the output with that combination.

From the above truth table, the Boolean expressions of each output functions can be written as

$$a = F1(A, B, C, D) = \sum m(0, 2, 3, 5, 7, 8, 9)$$

$$b = F2(A, B, C, D) = \sum m(0, 1, 2, 3, 4, 7, 8, 9)$$

$$c = F3(A, B, C, D) = \sum m(0, 1, 3, 4, 5, 6, 7, 8, 9)$$

$$d = F4(A, B, C, D) = \sum m(0, 2, 3, 5, 6, 8)$$

$$e = F5(A, B, C, D) = \sum m(0, 2, 6, 8)$$

$$f = F6(A, B, C, D) = \sum m(0, 4, 5, 6, 8, 9)$$

$$g = F7(A, B, C, D) = \sum m(2, 3, 4, 5, 6, 8, 9)$$

This step involves constructing Karnaugh's map for each output term and then simplifying them to obtain a logical combination of inputs for each output.

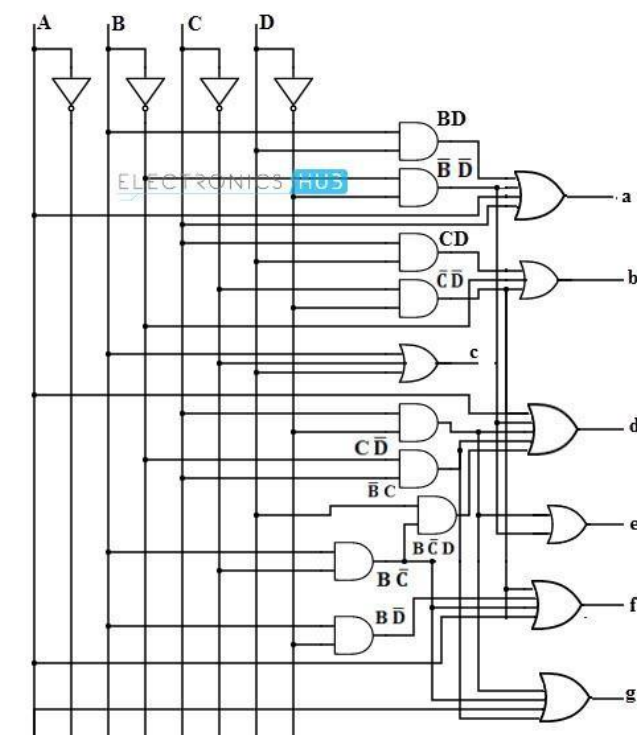
K-Map Simplification:

The below figures shows the k-map simplification for the common cathode seven-segment decoder to design the combinational circuit.

From the above simplification, we get the output values as

$a = A + C + BD + \overline{B}\overline{D}$
 $b = \overline{B} + \overline{C}\overline{D} + CD$
 $c = B + \overline{C} + D$
 $d = \overline{B}\overline{D} + C\overline{D} + B\overline{C}D + \overline{B}C + A$
 $e = \overline{B}\overline{D} + C\overline{D}$
 $f = A + \overline{C}\overline{D} + B\overline{C} + B\overline{D}$
 $g = A + B\overline{C} + \overline{B}C + C\overline{D}$

The final step involves drawing a combinational logic circuit for each output signal. Once the task was accomplished, a combinational logic circuit can be drawn using 4 inputs (A,B,C,D) and a 7-segment display (a,b,c,d,e,f,g) as output.



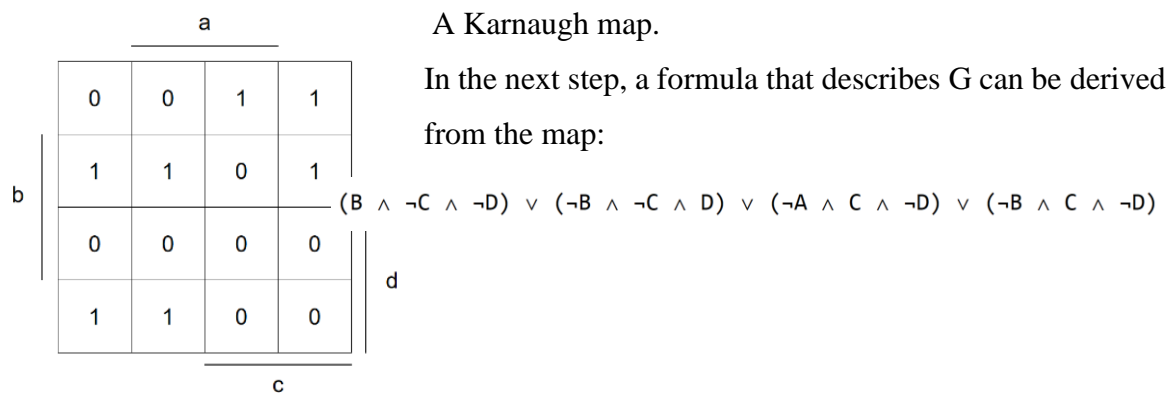
As you can see, each number is represented by a unique combination of the four input signals. A through G are the outputs of the logic circuit that will connect to the corresponding inputs on a seven-segment display.

Calculating the Output Functions

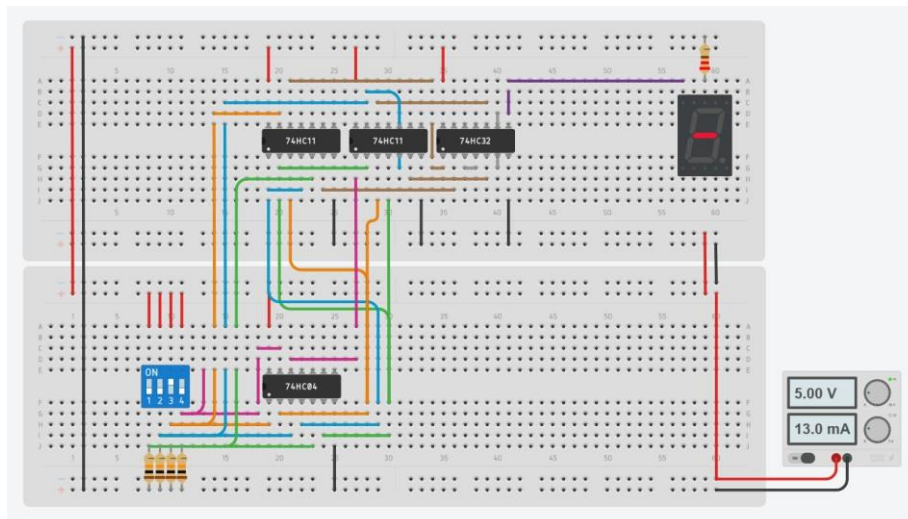
After creating the truth table, you have to determine a logic function for each output that exactly matches the values of the truth table for each combination of the input signals.

If you look at the input “1000” in the table above, the output G has to be high, and for the input “0000”, the result of G is low. I’ll only outline the process for one output function. The other ones can be calculated in the same way.

The easiest way to find a logic function that describes the behavior of G is to create a Karnaugh map and then simplify it.



Once that’s done, you can implement the logic circuit for that single segment:



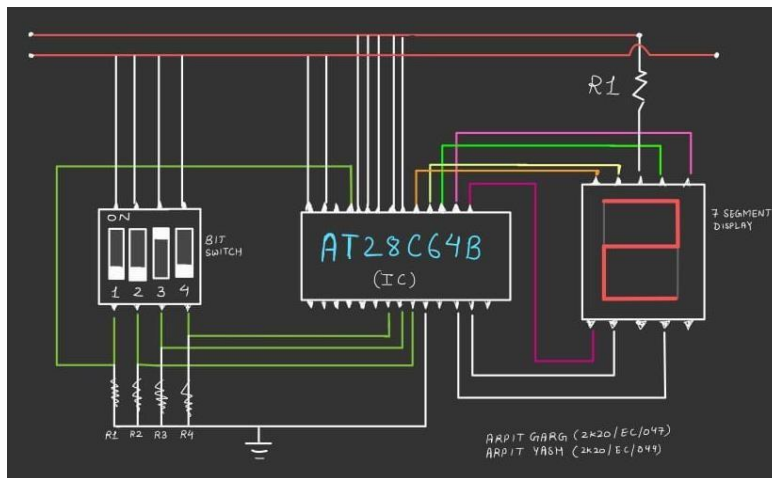
Implementation of the logic circuit for a specific segment.

Note that you have to repeat the entire process for every single segment of the display. The resulting logic circuit will, therefore, become quite complex — especially if you consider how simple the task is that it performs.

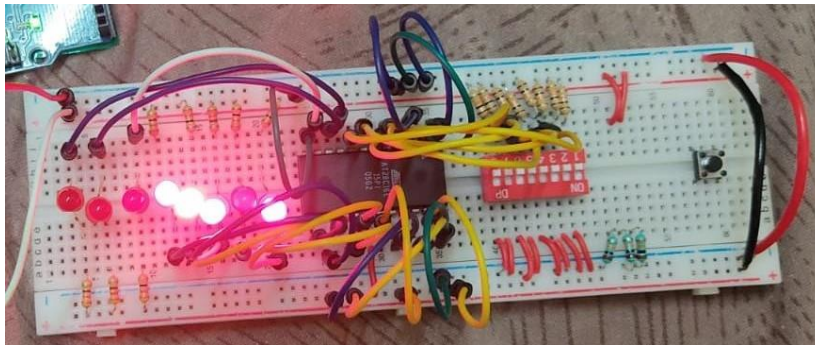
AN ALTERNATIVE TO LOGIC GATES: USING A ROM CHIP

In some cases, it may make more sense to replace the entire logic part of the circuit with a simple memory chip. Note that any memory chip will work. However, only persistent memory types, like ROMs or Flash memory, make sense in the real world because the volatile memory chips won't retain the stored data when they are powered off. That said, any parallel ROM chip with at least four addresses and seven output lines, like the AT28C64B, will work just fine.

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:



Connecting the display to the EEPROM.



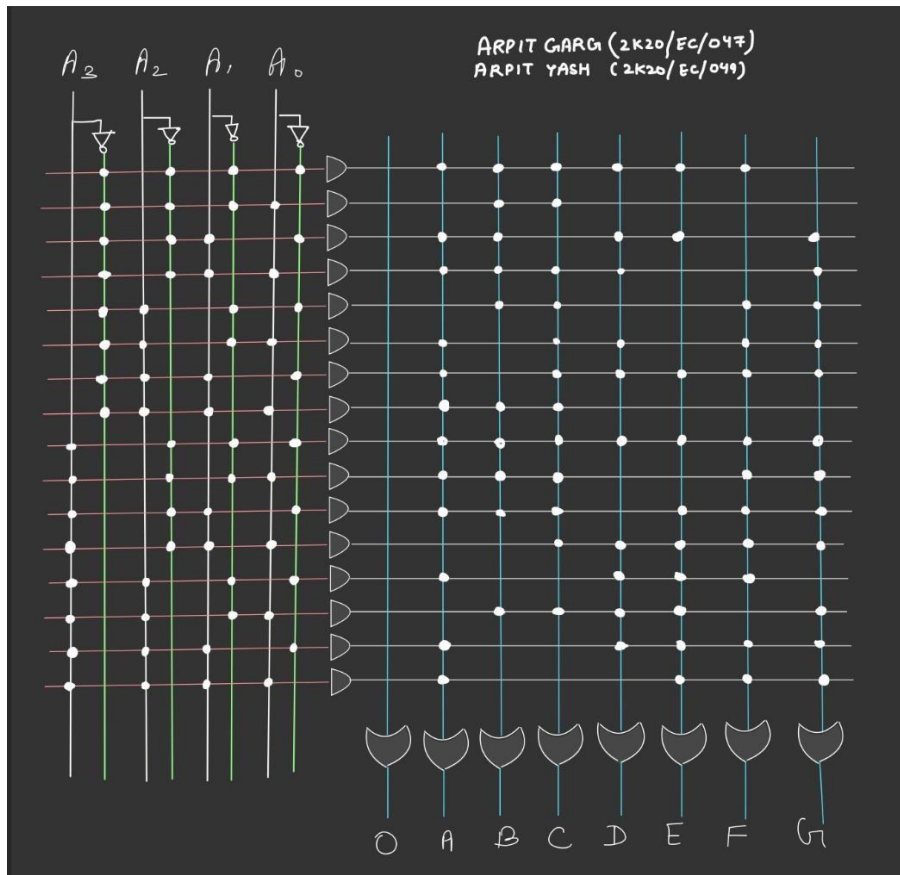
Make sure that you adjust the data bits to the pins you chose to use. I chose to arrange the four outputs in an order that makes it easier to connect the EEPROM to the display.

Once you made all the physical connections, you just have to store the values of A through G in the EEPROM at the correct address:

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

The data values can directly be taken from the truth table that we created earlier.

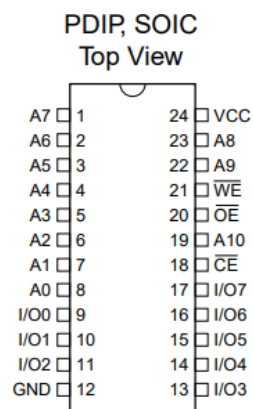


EEPROM AT28C16 IC

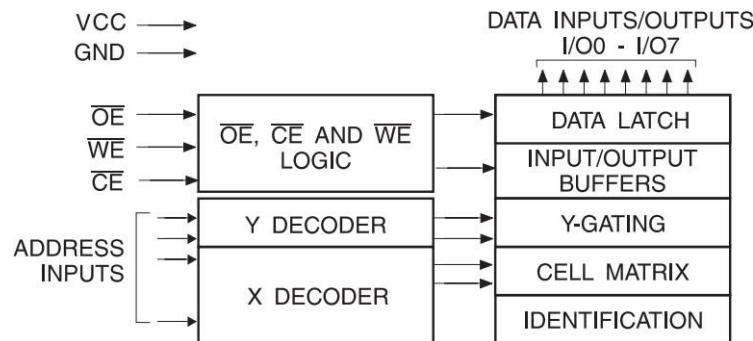
The AT28C16 is a low-power, high-performance Electrically Erasable and Programmable Read Only Memory with easy to use features. The AT28C16 is a 16K memory organized as 2,048 words by 8 bits.

PIN CONFIGURATIONS

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. Once the end of a write cycle has been detected, a new access for a read or a write can begin.



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. This dual line control gives designers increased flexibility in preventing bus contention.

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. The address location is latched on the last falling edge of WE (or CE); the new data is latched on the first rising edge. Internally, the device performs a self clear before write. Once a byte write has been started, it will automatically time itself to completion. Once a programming operation has been initiated and for the duration of t_{WC} , a read operation will effectively be a polling operation.

WRITE PROTECTION: Inadvertent writes to the device are protected against in the following ways: (a) VCC sense—if VCC is below 3.8V (typical) the write function is inhibited; (b) VCC power on delay—once VCC has reached 3.8V the device will automatically time out 5 ms (typical) 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.

EEPROM LIBRARY (ARDUINO)

The microcontroller on the Arduino and Genuino AVR-based board has EEPROM: memory whose values are kept when the board is turned off (like a tiny hard drive). This library enables you to read and write those bytes.

The supported micro-controllers on the various Arduino and Genuino boards have different amounts of EEPROM:

- 1024 bytes on the ATmega328P,
- 512 bytes on the ATmega168 and ATmega8,
- 4 KB (4096 bytes) on the ATmega1280 and ATmega2560.

To use this library: `#include <EEPROM.h>`

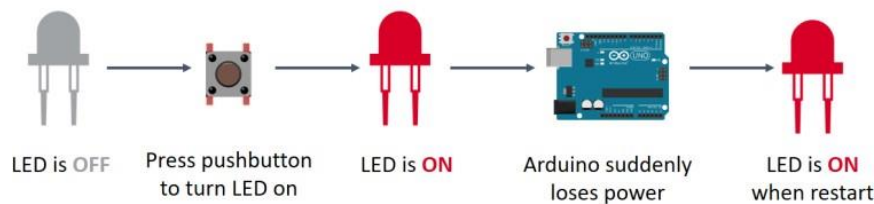
- **read():** Reads a byte from the EEPROM. Locations that have never been written to have the value of 255.

Syntax: `EEPROM.read(address)`

- **write():** write a byte to the EEPROM.

Syntax: `EEPROM.write(address,value)`

Save LED state on EEPROM



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;
  }
}
```

```

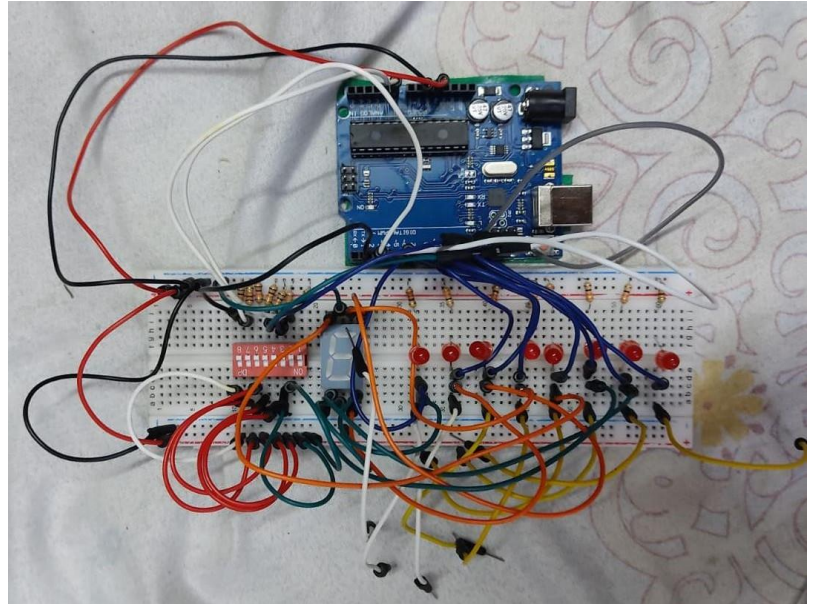
void setup()
{
  Serial.begin(9600);
  Serial.print("Erasing EEPROM \n");
  for (int address = 0; address <=15; address += 1)
  {
    EEPROM.write(address,0xFF);           //Erase EEPROM by setting all data
    bits high
  }
  Serial.print("Erased EEPROM \n");
  for (int address = 0; address <=15; address += 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

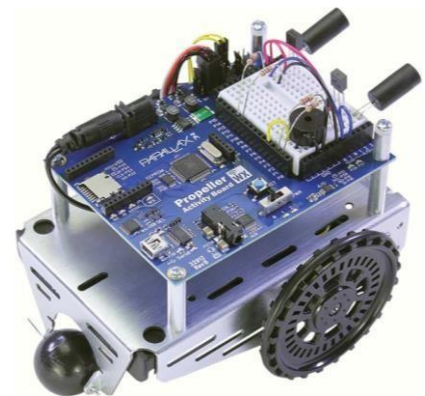
Erasing EEPROM	Program EEPROM
Erased EEPROM	Programmed EEPROM
0 : 0xFF	0 : 0x7E
1 : 0xFF	1 : 0x30
2 : 0xFF	2 : 0x6D
3 : 0xFF	3 : 0x79
4 : 0xFF	4 : 0x33
5 : 0xFF	5 : 0x5B
6 : 0xFF	6 : 0x5F
7 : 0xFF	7 : 0x70
8 : 0xFF	8 : 0x7F
9 : 0xFF	9 : 0x7B
10 : 0xFF	10 : 0x77
11 : 0xFF	11 : 0x1F
12 : 0xFF	12 : 0x4E
13 : 0xFF	13 : 0x3D
14 : 0xFF	14 : 0x4F
15 : 0xFF	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

If memory chips can be used to represent this behavior, then why go through all the hassle to calculate the logic functions and build a complicated circuit? The most important reason that comes to my mind is the price. ROM chips are usually quite pricey (around ten times the cost of simple logic ICs) and, most of the time, you won't use a lot of the storage space.

Furthermore, memory chips are much slower than logic ICs. Reading a value from a typical ROM will take around 150 nanoseconds, while the propagation delay on a typical logic IC is around five nanoseconds.

On the other hand, if you use a ROM chip, you can easily correct mistakes in your design by simply reprogramming the memory chip. Furthermore, a single chip uses less valuable board space on a PCB.

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.

REFERENCES :

1. <https://www.geeksforgeeks.org/bcd-to-7-segment-decoder/>
2. <https://docs.arduino.cc/learn/programming/eeprom-guide>
3. Using an EEPROM to replace combinational logic/ben-eater/youtube.com
4. <https://www.alldatasheet.com/datasheet-pdf/pdf/95099/ATMEL/AT28C16-15PI.html>