SMART LAB PROJECT

I developed a secure access control system designed for a laboratory environment to restrict entry to authorized students only. Each student is assigned a unique ID and password for authentication. The system features an Admin Panel with capabilities to:

- Add new members during runtime.
- Delete existing members.
- Change any password.
- Count the number of members currently inside the lab.
- Lock the lab to prevent entry.
- Display the total number of stored members.
- Show the last 12 member's ID have entered.

Password Storage Innovation:

To optimize memory usage, I implemented a compact algorithm to store 8-character passwords (including letters, numbers, or symbols) using just 8 bits per password in external EEPROM.

Hardware Features:

- A push button inside the lab allows members to exit easily.
- Two IR sensors used one installed in front of the door and the other behind to ensure accurate detection of entry and exit direction.
- An additional push button resets the system if someone tries to enter while another person is leaving.

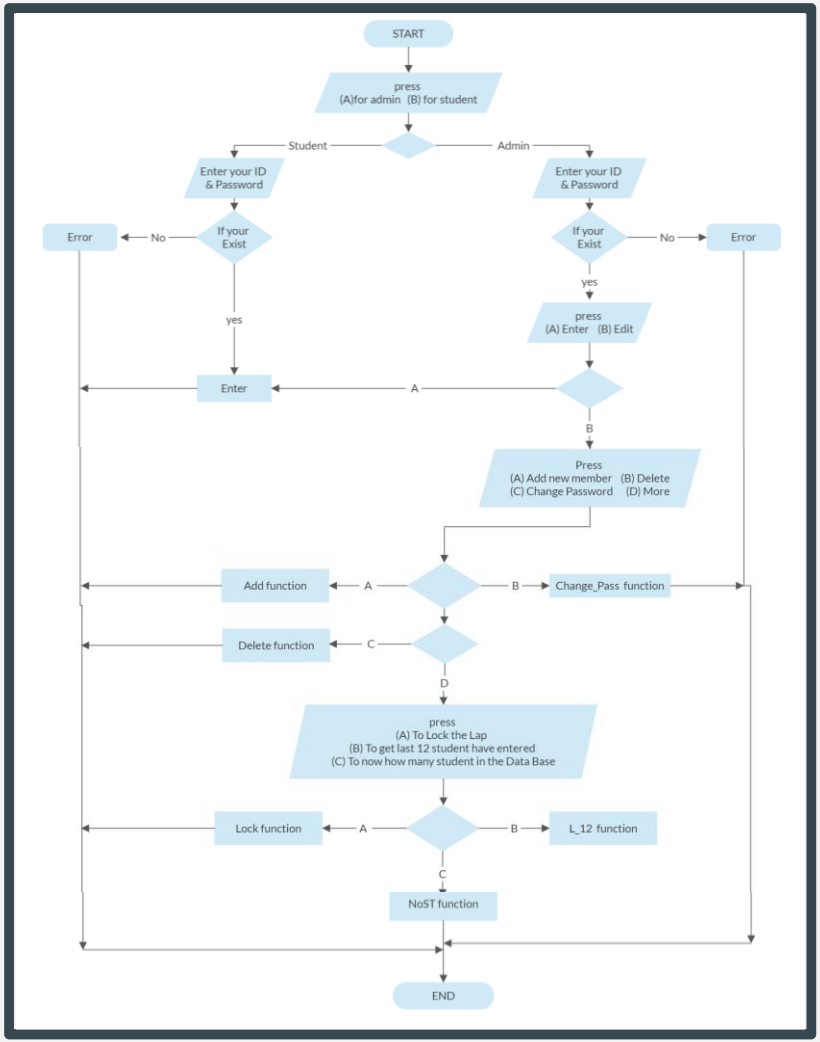
Video

submitted by\ Mahammad Heshmat Abd El-Raheem

FLOW CHART

We will discuss only the main functions which we made:

- EEPROM Mapping
- GET_ID FUNCTION
- GET_PASSWORD FUNCTION
- IR SENSOR



EEPROM Memory Management

The system uses a 2 KB (2048 bytes) external EEPROM that communicates with the ATmega32 microcontroller over I2C. To keep the design efficient and organized, I reserved only the first 256 addresses (0x0000–0x00FF) as follows:

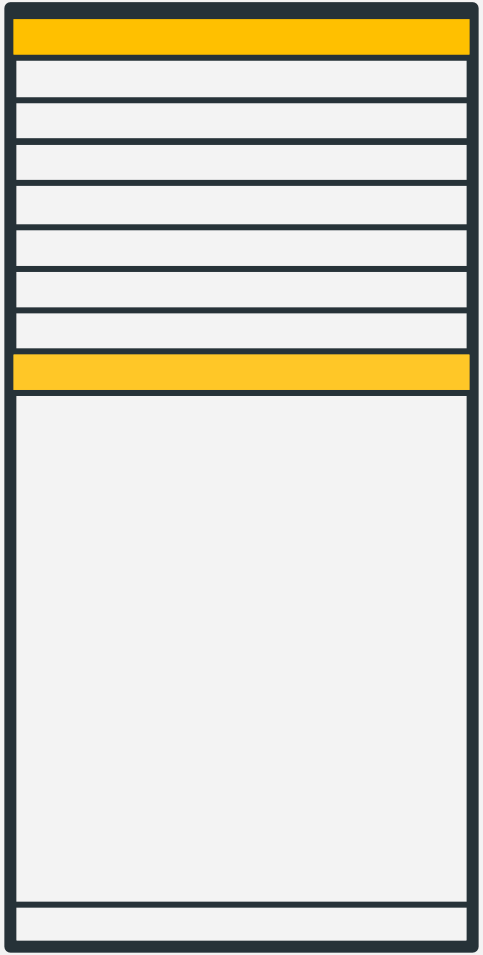
Addresses 1–254:
Each address corresponds directly to a student ID. The 8-bit data stored in that location represents the student's compressed 8-bit password.

Address 0:
Used as a counter to track the number of currently registered students. This enables dynamic adding and removing of members at runtime without losing track of capacity.

Address 255:
Reserved for storing the admin's ID and password securely.

This structure allowed me to:

- Quickly map IDs to EEPROM addresses for fast lookups.
- Minimize memory footprint with only 1 byte per password, using a custom encoding algorithm.
- Keep the admin credentials isolated from user data.



GET_ID FUNCTION

Student ID Generation & Obfuscation:
To make the IDs look less predictable and harder to reverse-engineer, I designed a simple encoding scheme:

Internal Storage:
Each student record is stored sequentially starting from EEPROM address 1 (0x0001) up to address 254, so internally the IDs are just 1–254.

External Display (to users)**:**
When the system issues an ID to a student, it does not display this raw address number. Instead, it uses a transformation to generate a 4-digit obfuscated ID:

Public ID = (Internal ID + 38) × 34

This ensures all displayed IDs:

- Are 4-digit numbers
- Appear unrelated to their real position in memory
- Are harder to guess or increment manually

For Retrieving Data:
When a user enters their ID, the system reverses this calculation to map it back to the correct EEPROM address.

For example:

If the internal ID = 1:
(1 + 38) × 34 = 1326

If the internal ID = 255:
(255 + 38) × 34 = 9962

GET_PASS FUNCTION

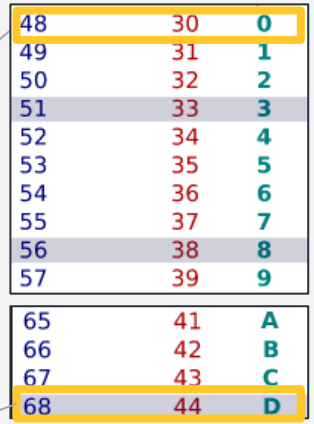
Password Compression Algorithm

To reduce memory usage while still distinguishing different passwords, I designed a custom algorithm that encodes an 8-character password into a single 8-bit value stored in EEPROM, Here's how it works:

1- Input Handling:

The user enters an 8-character password.
Allowed characters:
- Numbers (0–9) with ASCII codes 48–57
- Letters A–D with ASCII codes 65–68

The characters are received and stored in an array called MIN[].
Example input: "79A46B14"
MIN = [55, 57, 65, 52, 54, 66, 49, 52]



48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
65	41	A
66	42	B
67	43	C
68	44	D

2- Splitting the Password:

The array is split into two groups of 4 characters each
- Group 1: MIN[0] to MIN[3] and adding 20 to them The offset ensures all results are positive.
- Group 2: MIN[4] to MIN[7]

The offset = 20 because according to the worst case if (0 - D) which 0= 48 and D=68 the result will be -20.

GET_PASS FUNCTION

Example input: "79A46B14"
MIN = [55, 57, 65, 52, 54, 66, 49, 52]

3- Difference Calculation:

t=0: (20+55)-54 = 21
t=1: (20+57)-66 = 11
t=2: (20+65)-49 = 36
t=3: (20+52)-52 = 20

```
for (u8 t=0 ;t<4 ;t++)
{
    FIN[t]= (20+(MIN[t]))-(MIN[t+4]);
}
```

FIN[0]	=	20 +	MIN[0]	-	MIN[4]
FIN[1]	=	20 +	MIN[1]	-	MIN[5]
FIN[2]	=	20 +	MIN[2]	-	MIN[6]
FIN[3]	=	20 +	MIN[3]	-	MIN[7]

4- Summing the results:

- Resulting FIN array: [21,11,36,20]
- Password = FIN[0] + FIN[1] + FIN[2] + FIN[3]
- Password = 21 + 11 + 36 + 20 = 88

This single byte (88) is stored in the EEPROM at the address corresponding to the student ID.

GET_PASS FUNCTION

For the worst case:

Example input: “DDDD0000"

MIN = [68, 68, 68, 68, 48, 48, 48, 48]

t=0: (20+68) - 48 = 40

t=1: (20+68) - 48 = 40

t=2: (20+68) - 48 = 40

t=3: (20+68) - 48 = 40

```
for (u8 t=0 ;t<4 ;t++)
{
    FIN[t]= (20+(MIN[t]))-(MIN[t+4]);
}
```

- Password = 40 + 40 + 40 + 40 = **160**. This single byte (160) is the biggest number will store in the EEPROM which can contain up to 255.

Why this approach?

- It saves space (only 1 byte per password).
- It still produces different values for different inputs.
- It makes brute-force guessing more difficult because the transformation is non-obvious.

IR SENSOR

Why Two IR Sensors?

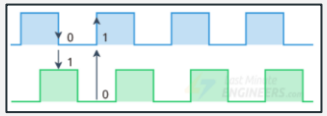
I integrated two infrared (IR) sensors at the lab entrance to accurately detect the direction of movement—whether someone is entering or exiting.

Using just a single sensor would only detect that something passed, but not the direction, making it impossible to:

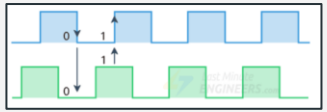
- Count how many people are inside reliably.
- Prevent errors when one person leaves while another enters.

How it works?

Entering: Sensor 1 triggers before Sensor 2



Exiting: Sensor 2 triggers before Sensor 1




```
void FUNCTION_IR_IN (u8*COUNTER)
{
    u8 I=1 , II=1 ;
    while (I)
    {

        if ( DIO_u8GetPinValue (DIO_U8_PIN16) == 0 )
        {
            while ( II )
            {
                if ( DIO_u8GetPinValue (DIO_U8_PIN17) == 0 )
                {
                    (*COUNTER)++;
                    _delay_ms(1500);
                    FUNCTION_CLOSE_DOOR();
                    I=0;
                    II=0;
                    _delay_ms(1000);
                }
            }
        }

        if ( DIO_u8GetPinValue (DIO_U8_PIN17) == 0 )
        {
            while (DIO_u8GetPinValue (DIO_U8_PIN8) == 0)
            {
                FUNCTION_BUZZER(ALARM);
            }

        }

    }
}
```

THANK YOU