# Sensor Simulator with EEPROM FS

In this assignment we have main three components to simulate sensors and EEPROM. See figure 1.

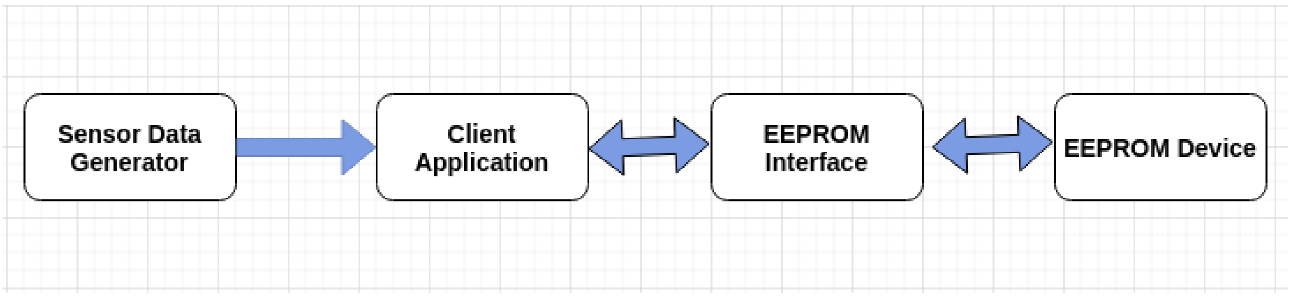


**Figure 1**

**Sensor Data generator** : It will generate random RFID tag ID with timestamp in millis. So, client application has to read it and perform EEPROM operations.Sequence of data generation:

1. millis (data type is long long)
2. RFID Tag ID (data type is String)

It will genearate 100 such samples.

**Client Application** : This one is basically your arduino code that you are suppose to implement on hardware , but due to lack of hardware , you will implement File System on virtual EEPROM and for that you have EEPROM interface. First read millis and convert it into hours,minutes and seconds, then read Tag ID. You don't have any mechanism to get filename from user , so use first 3 characters of tagID as filename.

**EEPROM Interface** : In arduino code , you are using EEPROM objects and its functions but here you will use below 3 functions to Read/Write from EEPROM.

1. int EEPROM_Write(int address ,unsigned char data);
2. unsigned char EEPROM_Read(int address);
3. void EEPROM_Close(void);

Functions 1 and 2 are known to you (they are similar to arduino EEPROM functions), but 3rd one is important. At the end of execution of Client application just call EEPROM_Close() function once, it will generate EEPROM.log file. That you will use for debugging purpose. Memory is designed as of 4KB only, so use address accordingly otherwise it may cause segmentation fault.

**EEPROM Device** : It is auto generated .log file ,which shows data for each and every EEPROM address.(It will only created if client have called EEPROM_Close()).

**Programming Examples :**

**Platfrom required:** All codes tested under Ubuntu 16.04 LTS 64 bit OS
**Provided Binaries and libraries:**
sensor_simulator : it will generate 100 samples of millis and ID
libeeprom.a : it is library for EEPROM interface

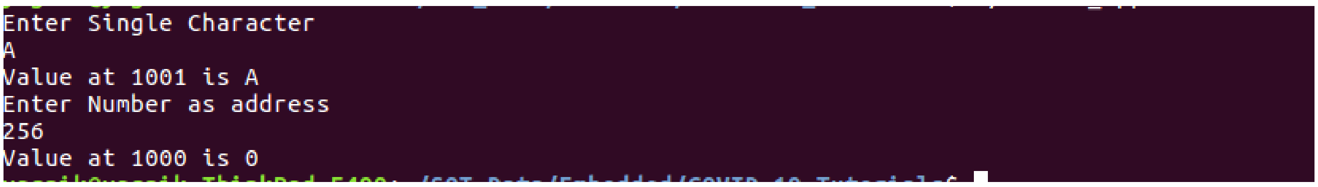
**Initial Setup (After downloading repository from GitHub):**

**make sensor_simulator to be executable by running below command,**
**$ sudo chmod a+x sensor_simulator**

**Example 1:**
**First I will show you how to read/write some random data without Filesystem on EEPROM.**

```
$ gcc -static example1.c -o exe1 libeeprom.a
$ ./exe1
```

O/P

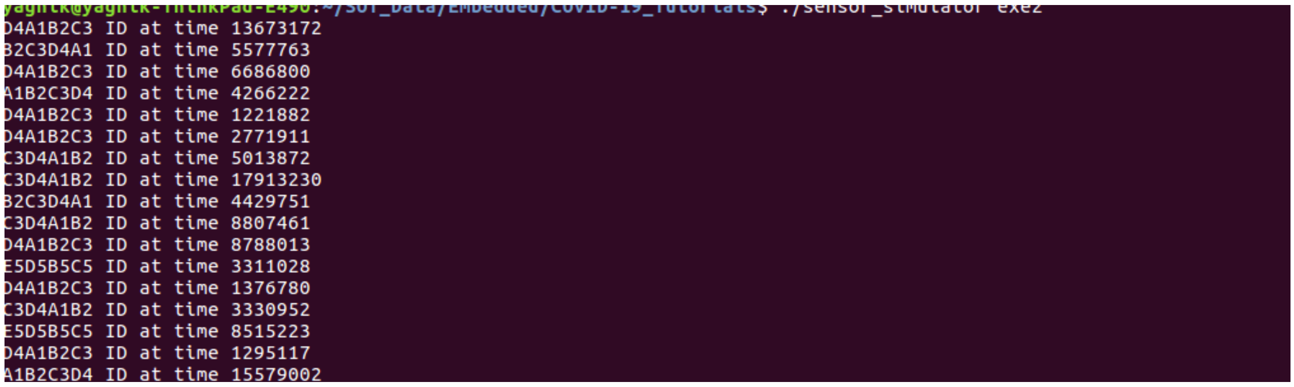


```
Enter Single Character
A
Value at 1001 is A
Enter Number as address
256
Value at 1000 is 0
```

**Example 2 :**
**In this example , you will pass your application as argument to sensor_simulator and read data from simulator.**

```
$ gcc example2.c  -o  exe2
$ ./sensor_simulator  exe2
```

O/P



```
yagnik@yagnik-ThinkPad-E490:~/SOT_Data/Embedded/COVID-19_Tutorials$ ./sensor_simulator exe2
D4A1B2C3 ID at time 13673172
B2C3D4A1 ID at time 5577763
D4A1B2C3 ID at time 6686800
A1B2C3D4 ID at time 4266222
D4A1B2C3 ID at time 1221882
D4A1B2C3 ID at time 2771911
C3D4A1B2 ID at time 5013872
C3D4A1B2 ID at time 17913230
B2C3D4A1 ID at time 4429751
C3D4A1B2 ID at time 8807461
D4A1B2C3 ID at time 8788013
E5D5B5C5 ID at time 3311028
D4A1B2C3 ID at time 1376780
C3D4A1B2 ID at time 3330952
E5D5B5C5 ID at time 8515223
D4A1B2C3 ID at time 1295117
A1B2C3D4 ID at time 15579002
```

From example 1 and 2 , you will come to know about EEPROM interface and simulator , now you just need to implement File system.

FileSystem is same as you are using in RFID data logger practical.

**Hint: Your Solution needs to be compiled by below command(consider your source code file name is sol.c)**

**gcc -static sol.c -o solution libeeprom.a**

**and to run your code ,**

**./sensor_simulator solution**

# EEPROM FileSystem Design

```
typedef struct
{
char file_name[3];//only 3 bytes for file name
char tag_id[8];//UID of RFID tag
uint8_t freeoffset;//For next entry, valid offset from current header address.It varies from 0-255
uint8_t data_size;//size of data_entry element =3B
}File_Header;//13B

//Here data_entry means your timestamp when card is detected.
typedef struct
{
uint8_t HH;
uint8_t MM;
uint8_t SS;
}Data_Entry;//3B

typedef struct
{
uint16_t FreeMemAdd;//Used to get Next Free Address, For file
uint16_t FreeMemSize;//Remaining Memory Size
uint8_t no_files;//Current no. of files/Tags in system
}FS_Header;//5B

typedef struct
{
FS_Header header;
uint16_t tag_header_add[MAX_FILES];
}FS;
```

//**NOTE: Find the below values.**

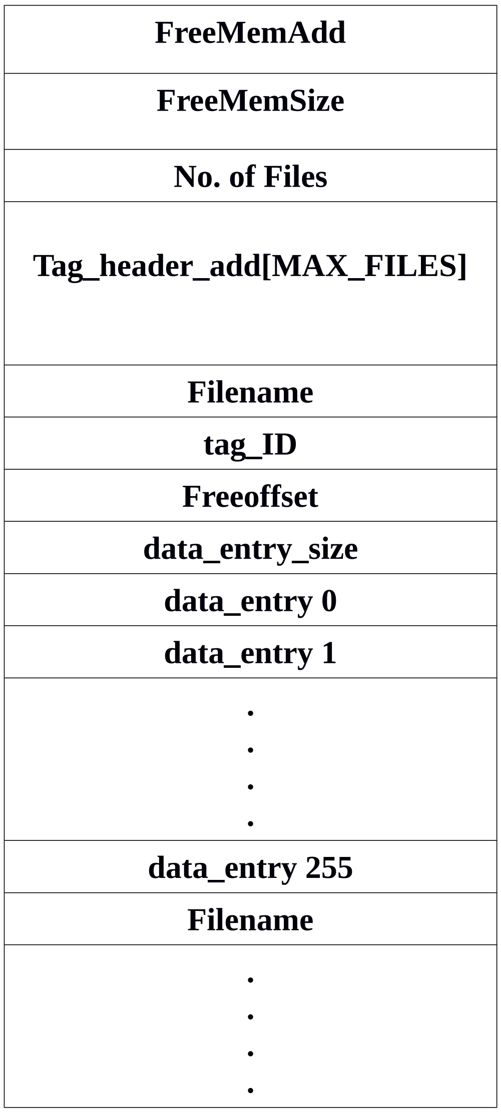
| | **Size** |
|---|---|
| **EEPROM Size** | |
| **MAX_FILES** | |
| **MAX_ENTRIES** | |
| **sizeof(Data_Entry)** | |
| **sizeof(FS)** | |
| **Address bus width** | |
| **Data bus width** | |
| **File header size** | |
| **Max File size** | |

| | | **Default Value** | **+/- Offset** |
|---|---|---|---|
| **FreeMemAdd** | It always points to Next Free File address. | | |
| **FreeMemSize** | It shows remaining size of EEPROM . | | |
| **no_files** | It indicates current no. Of files present in system. | | |
| **Filename** | It indiacares filename for the each Tag. | | |
| **freeoffset** | It always point to Next free Data_Entry structure’s data. | | |

**//Address calculation Formulas:**

**You will alway save address of each file header , which will work as reference address for rest of that file.  So , List out all required address  with reference of file header.**

**Filesystem block diagram:**


FreeMemAdd
FreeMemSize
No. of Files
Tag_header_add[MAX_FILES]
Filename
tag_ID
Freeoffset
data_entry_size
data_entry 0
data_entry 1
.
.
.
.
data_entry 255
Filename
.
.
.
.