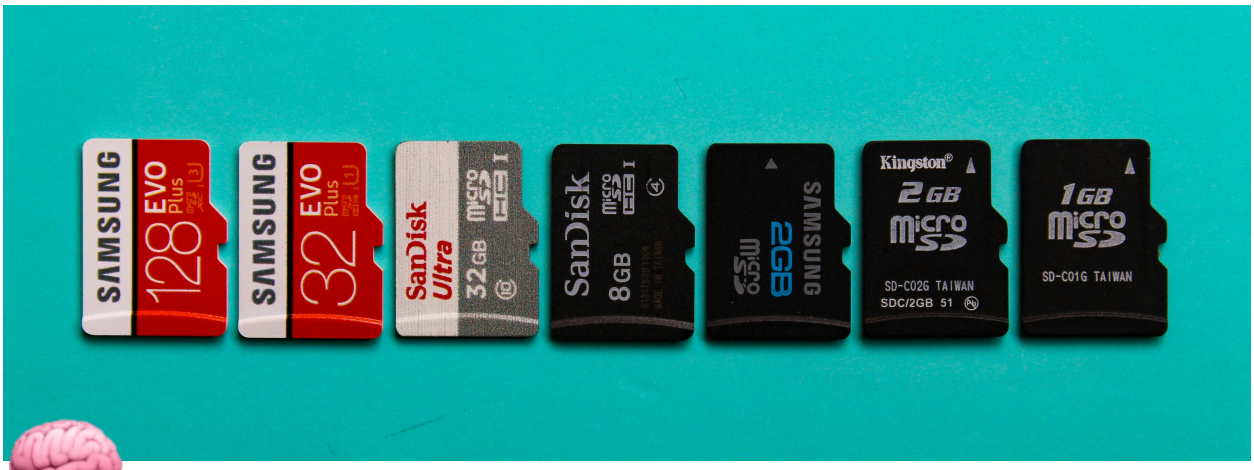


Week 8 - Non Volatile Memory

Memory that persists even without power.

ROM

- Read Only
  - Data can’t be changed.
- Mask-programmed ROM
  - Data is written onto the chip during manufacturing using a physical mask.
- PROM
  - Programmable ROM: user can program
  - Data is burnt onto the PROM by melting the fuses (by using a high voltage)
    - A burnt fuse permanently breaks the circuit.
- EPROM
  - Erasable PROM: Data can be changed more than once.
  - Uses floating gate transistors.
    - A floating gate within the transistor can hold different charge values which can be used to manipulate the threshold voltage and thus the

value of the transistor.

    - an UV light can be flashed through a quartz window to erase the memory. Old data must be erased to write new data.
- Flash Memory
    - Electronically erasable.
    - Can only erase on block or sector at a time.
- EEPROM
    - Electronically Erasable PROM
    - Individual bytes are erasable.

Device complexity and Cost (in increasing order):

PROM → EPROM → Flash → EEPROM

ATmega328p

- 1 Kb EEPROM
- Byte addressable (0-1023) (4 registers)
- Potentially any data can be stored on the EEPROM.
- External NVMs can be used for more memory.

File Systems

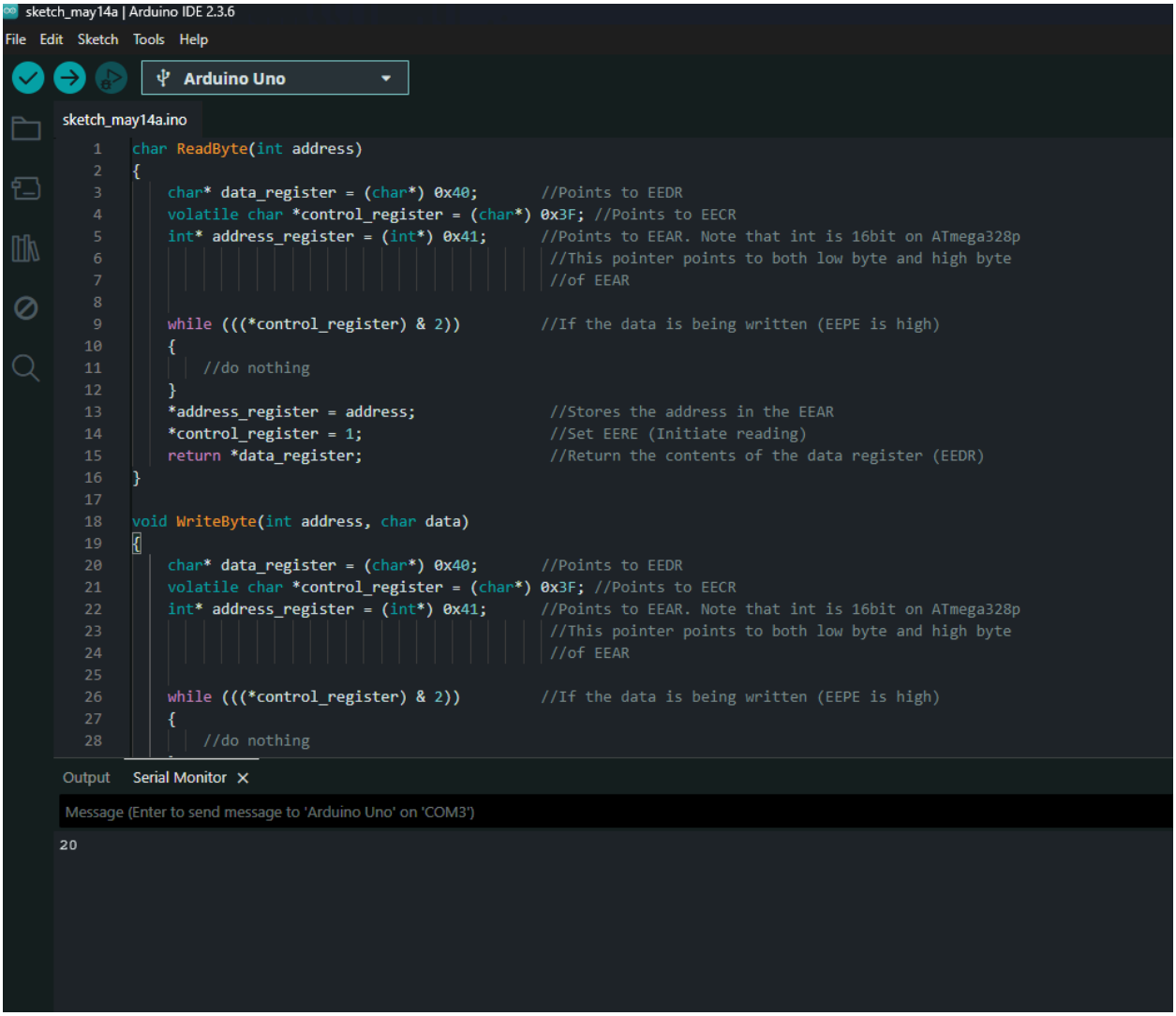
- A standard that defines formats for organizing files and folders
    - Where a file begins and ends in memory
    - how metadata is stored
    - how location information is stored etc.

Raspberry Pi 3

- Primary storage is SD Card (Flash Memory)
- Uses a file system managed by the Linux OS.

Exercise (Arduino)

Writing a Value to EEPROM then Reading it




```
char ReadByte(int address)
{
   char* data_register = (char*) 0x40;      //Points to EEDR
```

```
   volatile char *control_register = (char*) 0x3F; //Points to EECR
   int* address_register = (int*) 0x41;      //Points to EEAR. Note that int is 16bit on
                                 //This pointer points to both low byte and high byte
                                 //of EEAR

   while (((*control_register) & 2))        //If the data is being written (EEPE is high)
   {
      //do nothing
   }
   *address_register = address;             //Stores the address in the EEAR
   *control_register = 1;                  //Set EERE (Initiate reading)
   return *data_register;                   //Return the contents of the data register (EE
}

void WriteByte(int address, char data)
{
   char* data_register = (char*) 0x40;       //Points to EEDR
   volatile char *control_register = (char*) 0x3F; //Points to EECR
   int* address_register = (int*) 0x41;      //Points to EEAR. Note that int is 16bit on
                                 //This pointer points to both low byte and high byte
                                 //of EEAR

   while (((*control_register) & 2))        //If the data is being written (EEPE is high)
   {
      //do nothing
   }
   *address_register = address;             //Stores the address in the EEAR
   *data_register = data;                   //Stores the data in the EEDR
   *control_register = 4;                  //Enable Master Write (Set EEMPE)
   *control_register |= 2;                 //Start writing (Set EEPE)
}

int main()
{
   Serial.begin(9600);
```

```
    WriteByte(115, 20);                  //Write a value of 20 to the memory location 1
    unsigned char a = ReadByte(115);          //Read the value at memory location 11
    Serial.println(a);                   //Prints the value

    return 0; // Assuming a typical C/C++ main return
}
```