Chapter 10:
Memory and Data Management

If you use or reference these slides or the associated textbook, please cite the original authors' work as follows:

Toulson, R. & Wilmshurst, T. (2016). Fast and Effective Embedded Systems Design - Applying the ARM mbed (2nd edition), Newnes, Oxford, ISBN: 978-0-08-100880-5.

Memory function types

A microprocessor needs memory for two reasons: to hold its program, and to hold the data that it is working with; we often call these *program memory* and *data memory*. To meet these needs there are a number of different semiconductor memory technologies available, which can be embedded on the microcontroller chip.

Memory technology is divided broadly into two types: volatile and non-volatile.

- **Non-volatile memory** retains its data when power is removed, but tends to be more complex to write to in the first place. For historical reasons it is still often called ROM (Read Only Memory). Non-volatile memory is generally required for program memory, so that the program data is there and ready when the processor is powered up.

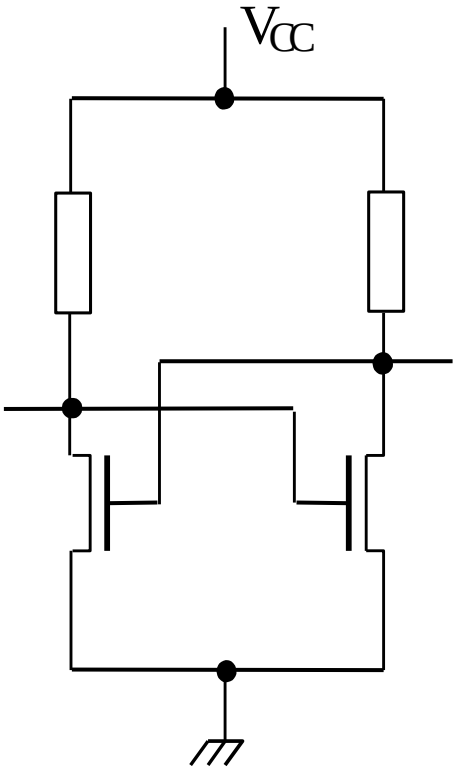
- **Volatile memory** loses all data when power is removed, but is easy to write to. Volatile memory is traditionally used for data memory; it's essential to be able to write to memory easily, and there is little expectation for data to be retained when the product is switched off. For historical reasons it is often called RAM (Random Access Memory).

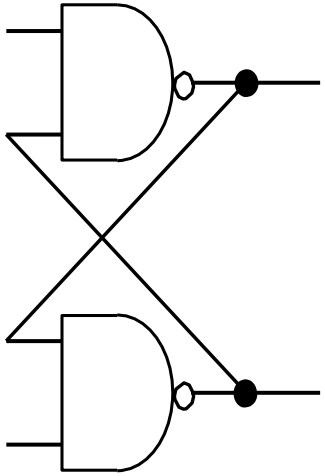
Essential electronic memory types

- In any electronic memory we want to be able to store all the 1s and 0s which make up our data.
- A simple one-bit memory is a coin. It is stable in two positions, with either “heads” facing up, or “tails”.
- We can try to balance the coin on its edge, but it would pretty soon fall over.
- We recognise that the coin is stable in two states, we call this *bistable*.
- It could be said that “heads” represents logic 1, and “tails” logic 0. With 8 coins, an 8-bit number can be represented and stored.
- If we had 10 million coins, we could store the data that makes up one photograph of good resolution, but that would take up a lot of space indeed!
- There are a number of electronic alternatives to the coin, which take up much less space.
- One is to use an electronic bistable (or “flip-flop”) circuit.

Essential electronic memory types

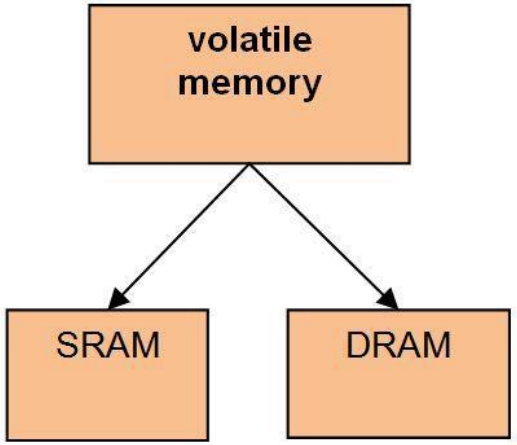
The two logic circuits shown are stable in only two states, and each can be used to store one bit of data.

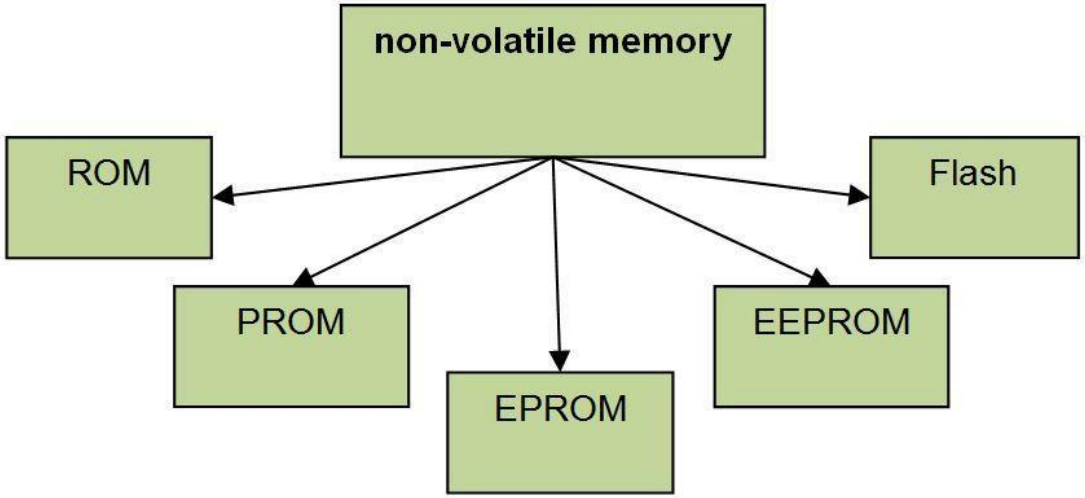




Essential electronic memory types

There are a number of different types of volatile and non-volatile memory:





RAM	Random Access Memory
SRAM	Static RAM
DRAM	Dynamic RAM

ROM	Read Only Memory
PROM	Programmable ROM Electrically
EPROM	Programmable ROM Electrically
EEPROM	Erasable PROM
FLASH	A cheap and rapid form of EEPROM

Introducing pointers

Before delving into the embedded world of memory and file access, we first need to cover a little C/C++ background on *pointers*.

Pointers are used to indicate where a particular element or block of data is stored in memory.

When a pointer is defined it can be set to a particular memory address and C/C++ syntax allows us to access the data at that address.

Pointers are required for a number of reasons; one is because the C/C++ standard does not allow arrays of data to be passed to and from functions, so in this case we must use pointers instead.

For example, we may wish to pass an array of 10 data values to a function in order to perform a simple mean average calculation, but in C/C++ this is not possible and causes a syntax error if attempted. Instead, to achieve this, it is necessary to pass a single pointer value as an input argument to the function.

In this instance the pointer essentially describes the memory address of the first element of the data array and is usually accompanied by a single argument that defines the size of the data array in question.

Defining pointers

Pointers are defined similar to variables but by additionally using the ***** operator. The following declaration defines a pointer called **ptr** which points to data of type **int**:

```
int *ptr; // define a pointer which points to data of type int
```

The specific address of a data variable, can also be assigned to a pointer by using the & operator, for example

```
int datavariable = 7;       // define a variable called datavariable with value 7
int *ptr;                   // define a pointer which points to data of type int
ptr = &datavariable;        // assign the pointer to the address of datavariable
```

We can also use the * operator in a program to get the data from a pointer address, for example:

```
int x = *ptr; // get the contents of location ptr and assign to x
```

We can also use pointers with arrays, because an array is really just a number of data values stored at consecutive memory locations; for example:

```
int array[] = {3, 4, 6, 2, 8, 9, 1, 4, 6}; // define an array of arbitrary values
int *ptr;                                   // define a pointer
ptr = &array[0];                            // assign pointer to the address of the first
                                            // element of the array
```

Using pointers with arrays and functions

```
/*Program Example 10.5: Pointers example for an array average fuction
*/
#include "mbed.h"
Serial pc(USBTX, USBRX);                          // setup serial comms
char data[]={5,7,5,8,9,1,7,8,2,5,1,4,6,2,1};      // define some input data
char* dataptr;                                    //define a pointer for the input data
float average;                                    //floating point average variable
float CalculateAverage(char* ptr, char size);     //function prototype

int main() {
    dataptr=&data[0];                          //point to address of the first array element
    average = CalculateAverage(dataptr, sizeof(data));// call function
    pc.printf("\n\rdata= ");
    for (char i=0; i<sizeof(data); i++) {       //loopfor each data value
        pc.printf("%d ",data[i]);                //display all data value
    }
    pc.printf("\n\raverage= %.3f",average);     //display average value
}

//CalculateAverage function definition and code
float CalculateAverage(char* ptr, char size) {
    int sum=0;              //variable for calculating the sum of the data
    float mean;             //variable for floating point mean value
    for (char i=0; i<size; i++) {
        sum=sum + * (ptr+i);//add all data elements together
    }
    mean=(float)sum/size; //divide by size and cast to floating point
    return mean;
}
```

Useful C/C++ library functions for data control

Function	Format	Summary Action
fopen	`FILE *fopen(const char *filename, const char *mode);`	opens the file of name **filename**
fclose	`int fclose(FILE *stream);`	closes a file
fgetc	`int fgetc(FILE *stream);`	gets a character from a stream
fgets	`char *fgets(char *str, int n, FILE *stream);`	gets a string of **n** chars from a stream
fputc	`int fputc(int character, FILE *stream);`	writes **character** to a stream
fputs	`int fputs(const char *str, FILE *stream);`	writes a string to a stream
fprintf	`int fprintf(FILE *stream, const char *format, ...);`	writes formatted data to a stream
fseek	`int fseek(FILE *stream, long int offset, int origin);`	moves file pointer to specified location

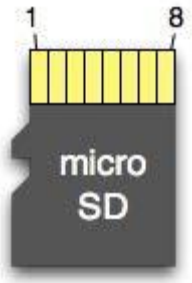
str	An array containing the null-terminated sequence of characters to be written.
stream	Pointer to a FILE object that identifies the stream where the data is to be written (see Section B10 for more information on streams in C/C++.
…	Indicates that additional formatted arguments may be specified in a list

Using external memory with the mbed

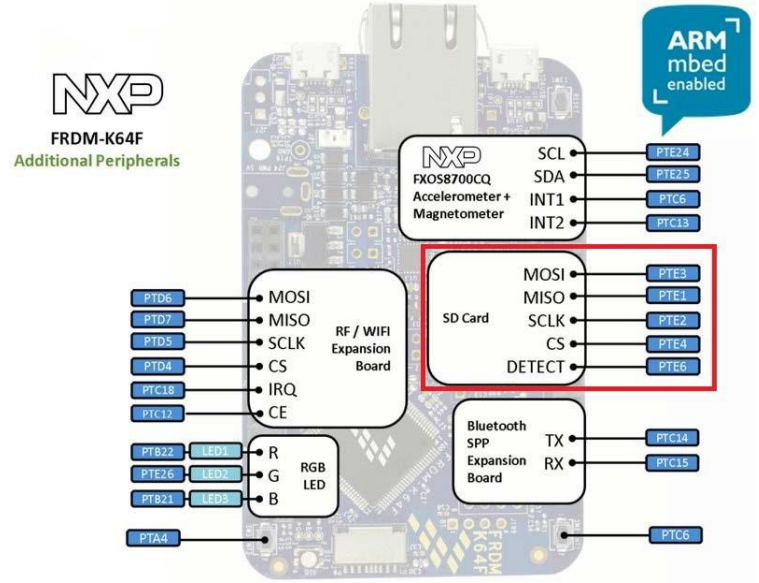
A flash SD (Secure Digital) card can be used with the mbed via the SPI protocol

Using a micro SD card with a card holder cradle (as shown) it is possible to access the SD card as an external memory

The SD card can be connected to the mbed as shown in the following wiring table:



Pin	MicroSD	SPI	mbed pin
1	DAT2	NC	
2	CD/DAT3	CS	PTE4
3	CMD	DI	PTE3(MOSI)
4	VDD	VDD	
5	CLK	SCLK	PTE2
6	GND	GND	
7	DAT0	DO	PTE1(MISO)
8	DAT1	NC	



The following mbed libraries are also required:

SDFileSystem library by Simon Ford.
https://developer.mbed.org/users/simon/code/SDFileSystem

FatFileSystem by mbed.
https://developer.mbed.org/users/mbed_unsupported/code/FatFileSystem

Writing data to an SD Card

```
/* Program Example 10.4: writing data to an SD card
*/

#include "mbed.h"
#include "SDFileSystem.h"

SDFileSystem sd(PTE3, PTE1, PTE2, PTE4, "sd"); // MOSI, MISO, SCLK, CS
Serial pc(USBTX, USBRX);
DigitalOut GLED(LED2);

int main() {
    FILE* File; File = fopen("/sd/sdfile.txt", "w"); // open file
    if(File == NULL) {    // check for file pointer
        pc.printf("Could not open file for writing\r\n");// error if no pointer
    }
    else{
        pc.printf("SD card file opened successfully\r\n");
    }
    fprintf(File, "Here's some sample text on the SD card"); // write data
    fclose(File);// close file
    while(1){
        GLED = !GLED;
        wait(0.5);
    }
}
```

Chapter quiz questions

1. What does the term *bistable* mean?
2. How many bistables would you expect to find in the mbed's SRAM?
3. What are the fundamental differences between SRAM and DRAM type memory?
4. What are the fundamental differences between EEPROM and Flash type memory?
5. Describe the purpose of pointers and explain how they used to access the different elements of a data array.
6. What C/C++ command would open a text file for adding additional text to the end of the current file.
7. What C/C++ command should be used to open a text file called "data.txt" and read the 12th character.
8. Give a practical example where data logging is required and explain the practical requirements with regards to timing, memory type and size.
9. Give one reason why pointers are used for direct manipulation of memory data.
10. Write the C/C++ code that defines an empty 5 element array called **dataarray** and a pointer called **datapointer** which is assigned to the first memory address of the data array.

Chapter review

- Microprocessors use memory for holding the program code (program memory) and the working data (data memory) in an embedded system.
- A coin or a logic flip-flop/bistable can be thought of as a single 1-bit memory device which retains its state until the state is actively changed.
- Volatile memory loses its data once power is removed, whereas non-volatile can retain memory with no power. A number of different technologies are used to realise these memory types including SRAM and DRAM (volatile) and EEPROM and Flash (non-volatile).

- Pointers point to memory address locations to allow direct access to the data stored at the pointed location.
- Pointers are generally required owing to the fact that C/C++ does not allow arrays of data to be passed into functions
- The **stdio.h** library contains functions that allow us to create, open and close files, as well as read data from and write data to files.
- Files can be created on the mbed for storing and retrieving data and formatted text.
- An external SD memory card or USB Flash drive can be interfaced with the mbed to allow larger memory.

Read further:
Try out the data logging mini-project described in Section 10.7