Memory in Embedded Systems

Dongbing Gu

School of Computer Science and Electronic Engineering
University of Essex
UK

Spring 2018

Outline

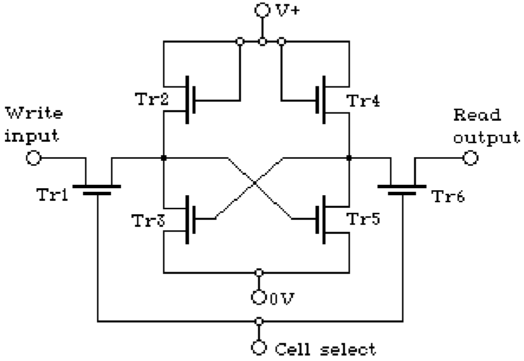
Section 1

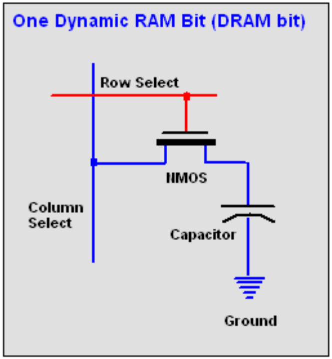
Memory Technologies

Volatile Memory

- SRAM (static RAM)
  - Memory cell uses flip-flop to store bit
  - Requires 6 transistors
  - Holds data as long as power supplied
- SRAM is faster than DRAM (dynamic RAM), but it is also larger (each bit takes up more silicon area).
- DRAM is tended to do the same thing as SRAM with a reduced silicon area.
- One bit is stored in a tiny capacitor.
- Due to the small capability and leakage currents, the memory loses its charge over a short period of time.
- DRAM holds data for only a short time, so each memory location must be periodically refreshed every few milliseconds.

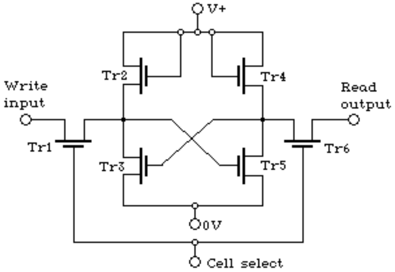
Basic MOS dynamic and static RAM memory cells

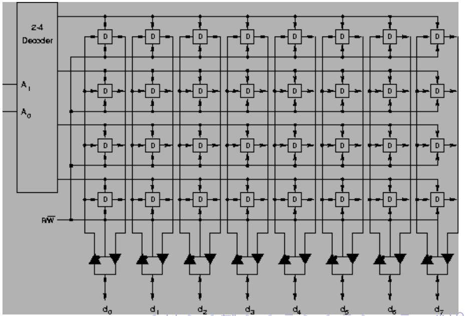




Memory Access

- Stores large number of bits
  - $m \times n$: m words of n bits each
  - $k = log_2 m$ address input signals
  - $m = 2^k$ words
- Memory access
  - r/w: selects read or write
  - enable: read or write only when asserted
  - multiport: multiple accesses to different locations simultaneously





Registers

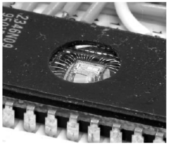
- The most tightly integrated memory in a processor is the registers.
- The size of a word is a key property of a processor architecture.
- The number of registers in a processor is usually small. This is not so much the cost of the register hardware, but rather the cost of bits in an instruction word.
- Cortex-M3 core has 16 user-visible registers.
- Three of these registers have dedicated functions, program counter (PC), link register (LR), stack pointer (SP).
- Processor Status Register (PSR) which is implicitly accessed by many instructions.

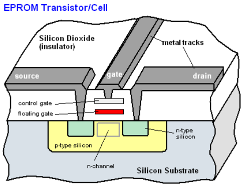
Non-Volatile Memory

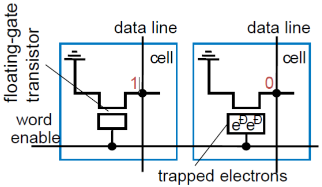
- The most basic non-volatile memory is ROM (read-only memory).
- The contents of ROM is fixed at the chip factory. This can be useful for mass produced products that only need to have a program and constant data stored, and these data never change.

Erasable Programmable ROM (EPROM)

- Uses floating-gate transistor in each cell.
- Programmer uses higher-than-normal voltage so electrons tunnel into the gate.
- Electrons become trapped in the gate.
- Only done for cells that should store 0, Other cells will be 1.
- To erase, a specified device is required: shine ultraviolet light onto chip to give trapped electrons energy to escape.
- Requires chip package to have window.
- EPROM had become widely used, but was replaced by flash memory.



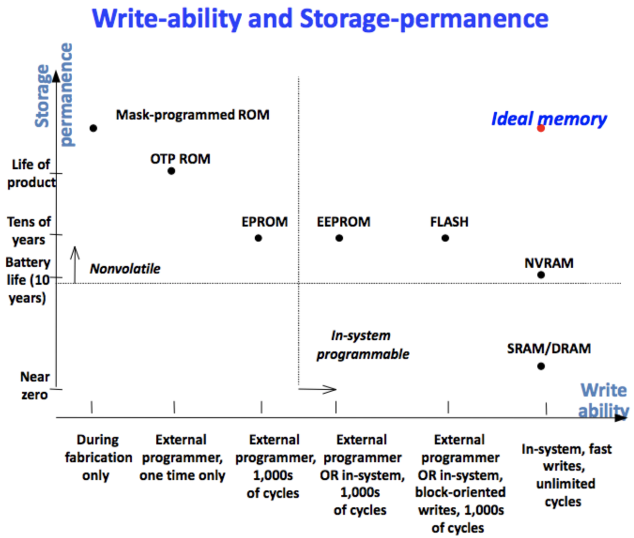




EEPROM and Flash

- EEPROM (electrically-erasable programmable ROM): the write time is typically much longer than the read time, and the number of writes is limited during the lifetime of the device. Erasing one word at a time electronically.
- Flash memory erases whole blocks at any one time.
- Huge advantage: they are very high density.
- Flash memories have reasonably fast read times, but not as fast as SRAM and DRAM, so frequently accessed data will typically have to be moved from the flash to RAM.
- The write times are much longer than the read times, and the total number of writes are limited.
- It is primarily used in main memory, memory cards, USB flash drives, solid-state drives, and similar products.

Storage permanence


Write-ability and Storage-permanence
Storage permanence
Mask-programmed ROM
Ideal memory
OTP ROM
Life of product
Tens of years
Battery life (10 years)
Near zero
EPROM
EEPROM
FLASH
NVRAM
Nonvolatile
In-system programmable
SRAM/DRAM
Write ability
During fabrication only
External programmer, one time only
External programmer, 1,000s of cycles
External programmer OR in-system, 1,000s of cycles
External programmer OR in-system, block-oriented writes, 1,000s of cycles
In-system, fast writes, unlimited cycles

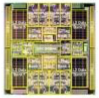
Cortex M3 On chip Memory

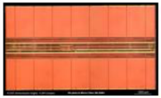
- On-chip Flash memory system: 512 kB of on-chip flash memory can be used for both code and data storage
- On-chip Static RAM (SRAM): 64 kB of on-chip static RAM memory. 32 kB of SRAM, accessible by the CPU and DMA (direct memory access) controller are on a higher speed bus.

Section 2

Memory Management

Memory Latency



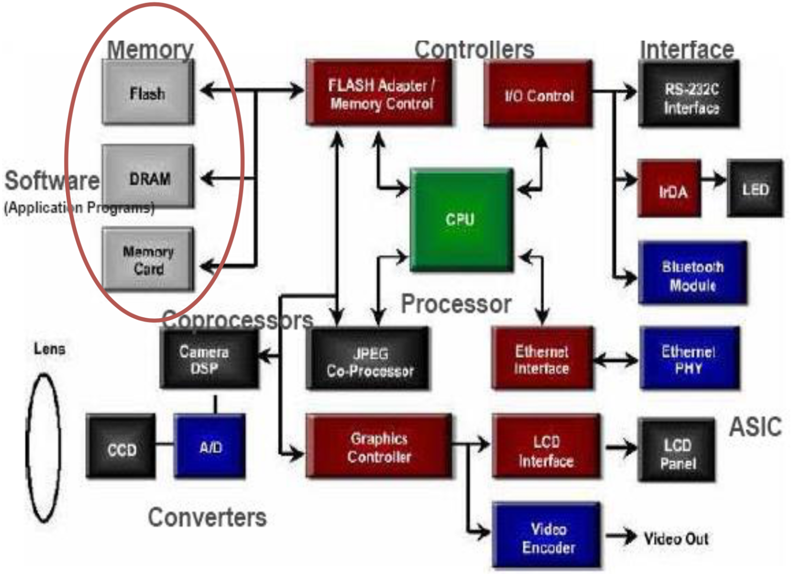


	On-chip memory (SRAM)	Off-chip memory (DRAM)	Solid State Disk (Flash Memory)	Secondary Storage (HDD)
Latency: (Cycles)	1~30	100~300	25000~2000000	>5000000

Embedded Memory Components

- Most microprocessors combine different memory technologies to increase the overall memory capacity while optimising cost, latency, and energy consumption.
- Typically, a relatively small amount of on-chip SRAM will be used with a larger amount of off-chip DRAM.
- The application programmer may not be aware that memory is fragmented across these technologies.
- The operating system and/or the hardware provides address translation, which converts logical addresses in the address space to physical locations in one of the available memory technologies.

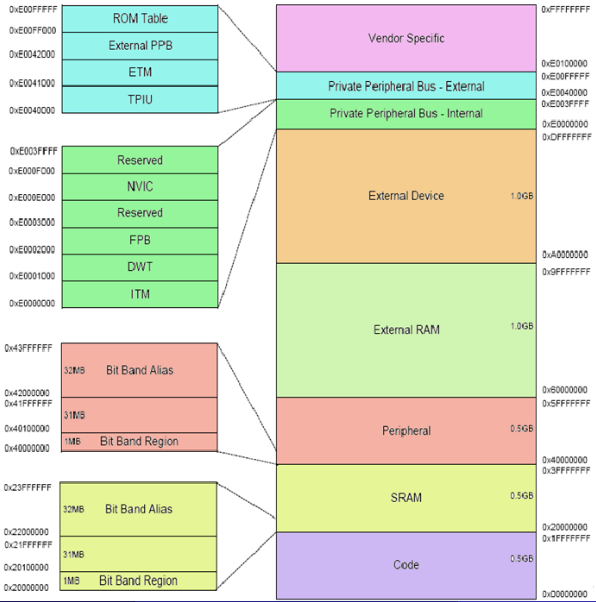
Embedded Memory Components



Cortex-M3 Memory Address Space

- Cortex-M3 has a single “physical” address space of 2^{32} bytes (4 GB).
- The SRAM and peripheral areas are accessed through the system bus.
- The “Code” region is accessed through the ICode (instructions) and DCode (constant data) buses.
- Predefined memory map

Cortex-M3 Memory Map



Cortex-M3 Memory Map

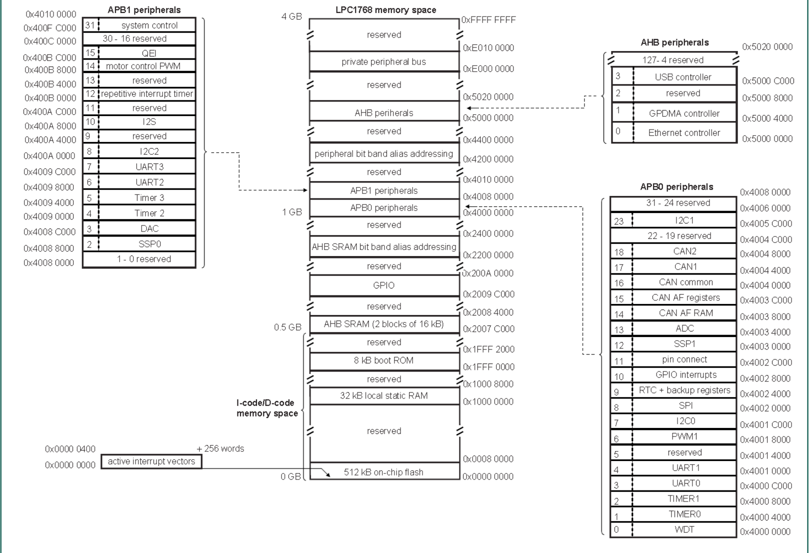
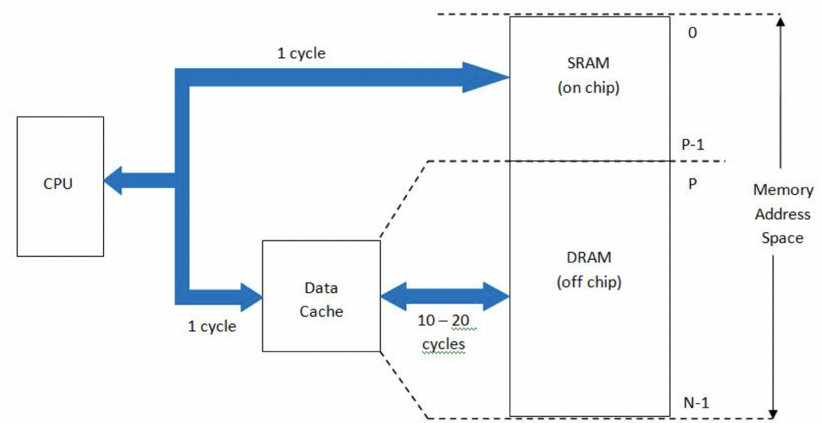


Fig 3. LPC17xx system memory map

Memory Caches

- Some memories are accessed before others.
- The close memory duplicates data in the distant memory with the hardware automatically handling the copying to and from, then it is called a cache.
- Cache memory is RAM that a microprocessor can access more quickly than it can access regular RAM.
- For embedded applications with tight real-time constraints, cache timing behaviour can vary substantially.

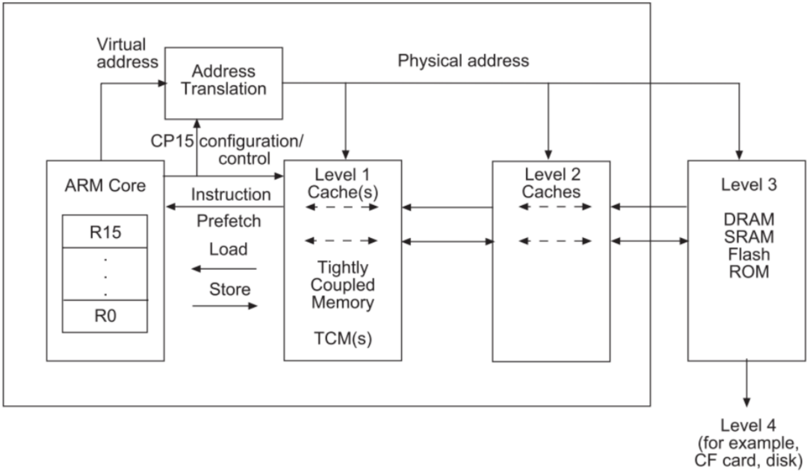
Memory Caches



ARM Memory Caches

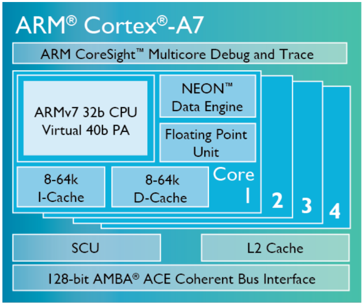
- Level 1 (L1) cache is extremely fast but relatively small, and is usually embedded in the processor chip (CPU).
- Level 2 (L2) cache is often more capacious than L1; it may be located on the CPU or on a separate chip.
- Level 3 (L3) cache is typically specialized memory that works to improve the performance of L1 and L2. It can be significantly slower than L1 or L2, but is usually double the speed of RAM. In the case of multicore processors, each core may have its own dedicated L1 and L2 cache, but share a common L3 cache.

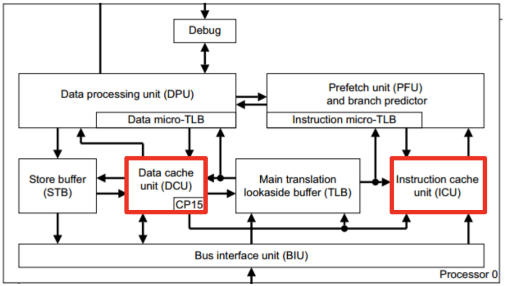
ARM Memory Caches



Cortex-A7 Memory

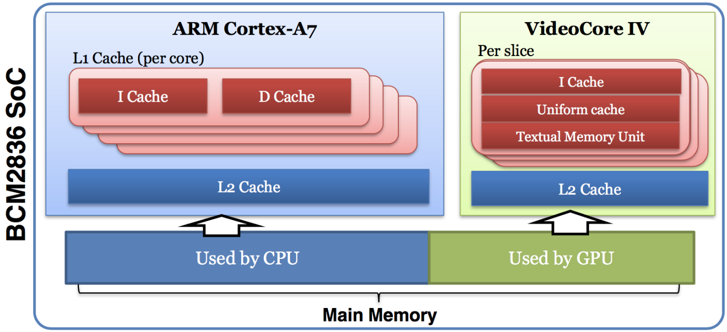
- L1 Cache
  - Instruction Cache (I-Cache): 32-bytes cache line
  - Data Cache (D-Cache): 64-bytes cache line





Raspberry Pi2 - Memory Architecture

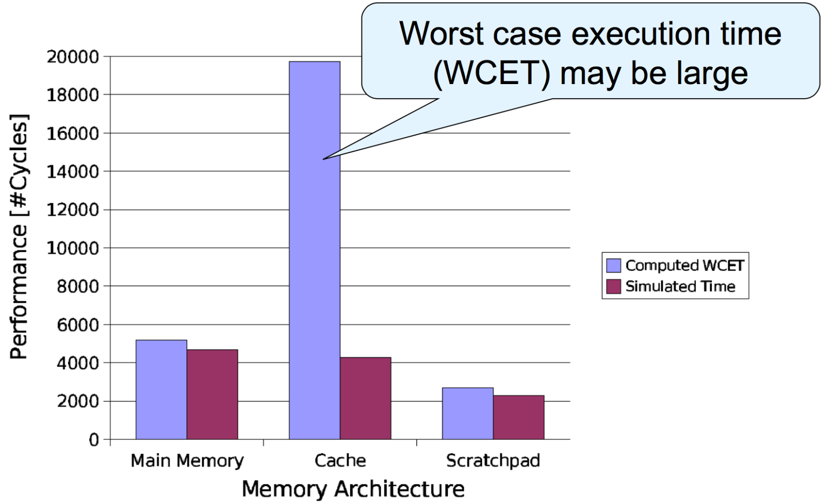
- Broadcom BCM2836 SoC
  - CPU: Quad-core Cortex-A7: L1 and L2 cache
  - GPU: VideoCore IV Processor: exclusive memory system
  - Main Memory: 1GB RAM : Shared by CPU and GPU



Cache operation overview

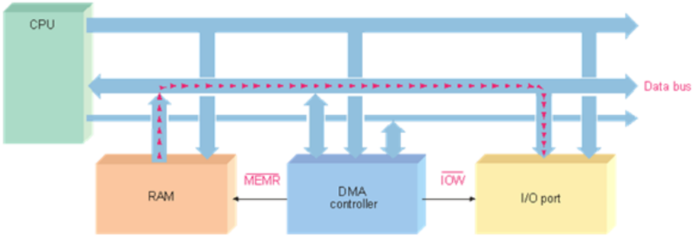
- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

Cache Performance



Direct Memory Access (DMA)

- A direct memory access (DMA) is an operation in which data is transported from one resource to another resource in a computer system without the involvement of the CPU.
- The task of a DMA controller is to execute the copy operation of data from one resource location to another.
  - I/O-device to memory
  - memory to I/O-device
  - memory to memory
  - I/O-device to I/O-device



DMA channels

- DMA consists of several channels which may be individually configured with a transfer mode, source and destination memory address.
- Each channel can also be set to trigger on a DMA request from a specific peripheral.
- LPC1768 has 8 DMA channels (32 bits). The operations can be performed in either burst or single-cycle mode.
- In burst mode, the DMA controller keeps control of the bus until all the data buffered by the requesting device has been transferred to memory.
- In single-cycle mode, the DMA controller gives up the bus after each transfer. This requires that the bus request/acknowledge sequence be performed for every transfer. This overhead can result in a drop in overall system throughput.

Cortex M3 DMA Channel Configuration

- Channel Descriptors: Each DMA channel has two associated channel descriptor structures. These include the source and destination address for the channel as well as information on number of elements to transfer, data size, transfer type etc.
- DMA Register: Common configurations for the DMA, as well as DMA interrupts and trigger sources.
- Registers in trigger peripherals: The DMA request signals from the peripherals are generated in the peripherals, hence it is important to configure the peripherals correctly to generate the desired DMA requests.

Memory Protection Unit (MPU)

- Cortex-M3 has an Memory Protection Unit (MPU).
- Allows access rules to be set up for privileged access and user program access.
- MPU can be used in various ways:
  - Set up by an operating system, allowing data used by privileged code (e.g., the operating system kernel) to be protected from untrusted user programs
  - Can be used to make memory regions read-only, to prevent accidental erasing of data, or to isolate memory regions between different tasks in a multitasking system

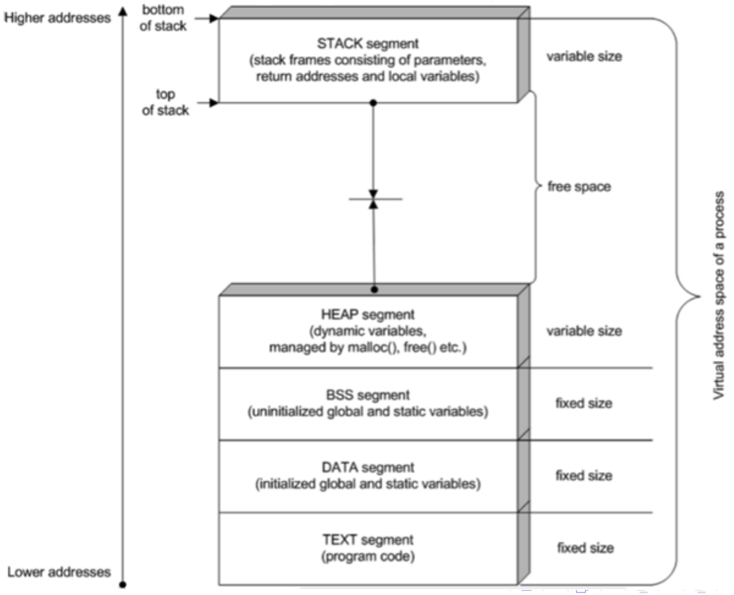
Section 3

Program Memory Model

Local, Global and Static memory

- A local variable is one that occurs within a specific scope. They exist only in the function where they are created.
- A global variable is a variable that is defined outside all functions and available to all functions.
- In local variables, static is used to store the variable in the statically allocated memory instead of the automatically allocated memory.
- Statically allocated memory (global or static) is typically reserved in data segment of the program at compile time.

Program Memory Map



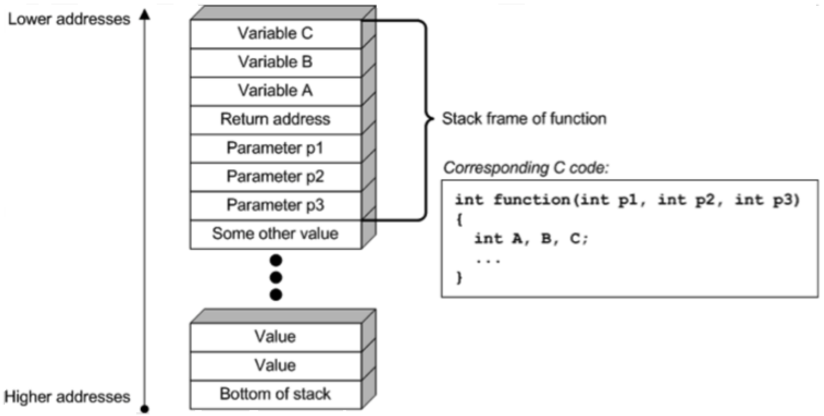
Higher addresses
bottom of stack
STACK segment (stack frames consisting of parameters, return addresses and local variables)
variable size
top of stack
free space
HEAP segment (dynamic variables, managed by malloc(), free() etc.)
variable size
BSS segment (uninitialized global and static variables)
fixed size
DATA segment (initialized global and static variables)
fixed size
TEXT segment (program code)
fixed size
Lower addresses
Virtual address space of a process

Stack

- A stack is a region of memory that is dynamically allocated to the program in a last-in, first-out (LIFO) pattern.
- A stack pointer (typically a register) contains the memory address of the top of the stack.
- When an item is pushed onto the stack, the stack pointer is decreased and the item is stored at the new location referenced by the stack pointer.
- When an item is popped off the stack, the item referenced by the stack pointer is (typically) copied into a register and the stack pointer is increased.

Stack

- It stores types of variables that have a fixed lifetime - local variables
- The stack is relatively small. It is generally not a good idea to do anything that eats up lots of stack space.



```
int function(int p1, int p2, int p3)
{
   int A, B, C;
   ...
}
```

Heap

- The heap segment keeps track of memory used for dynamic memory allocation.
- The heap starts from lower memory, growing up into higher memory.
- In C, when you use the new operator to allocate memory, this memory is allocated in the heap segment.
- Allocated memory stays allocated until it is specifically deallocated or the application ends (at which point the OS should clean it up).
- Because the heap is a big pool of memory, large arrays, structures, or classes can be allocated here.

```
int *ptr = new int; // ptr is assigned 4 bytes in the heap
int *array = new int[10]; // array is assigned 40 bytes in ↩
    the heap
```

Heap

- A program can at any time request (*malloc* or *new* in C) that the operating system allocate additional memory.
- Heap can keep the track of which portions of memory are in use by which application.
- When the program no longer needs access to memory that has been so allocated, it frees the memory (by calling *free* in C).

Memory Leak

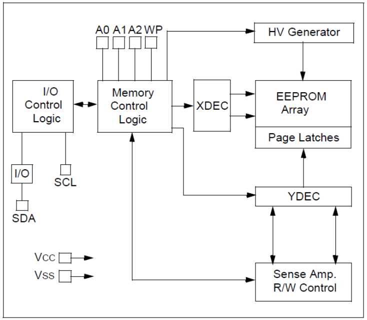
- A garbage collector is a task that runs either periodically or when memory gets tight.
- It automatically frees any portions of memory that are no longer referenced.
- With or without garbage collection, it is possible for a program to inadvertently accumulate memory that is never freed.
- This is known as a memory leak.
- The program will eventually fail when physical memory is exhausted.

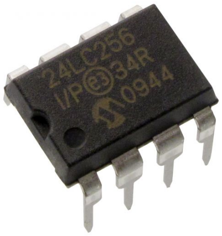
Section 4

External Memories

I2C EEPROM 24LC256

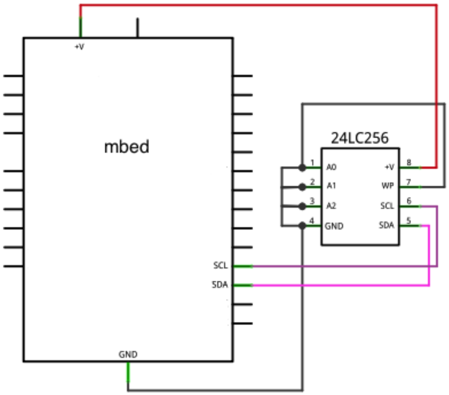
- The Microchip Technology Inc. 24LC256: 32K x 8 (256 Kbit).
- It has a page write capability of up to 64 bytes of data.
- It is capable of both random and sequential reads up to the 256K boundary.
- Up to eight devices on the same bus, for up to 2 Mbit address space.
- 5 ms max. write cycle time

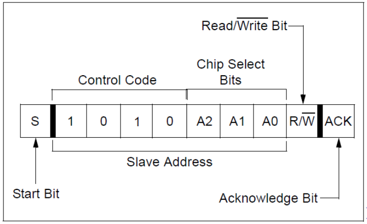




Processor and I2C EEPROM

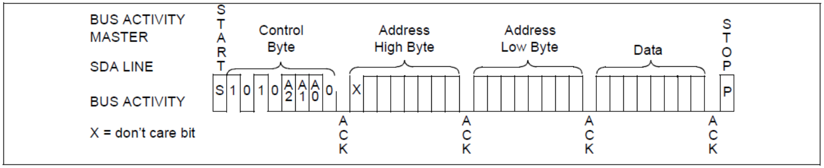
- Three pins to configure the device address (A0, A1 and A2).
- Usually, these pins are hard-wired to high and/or low levels.
- To control these pins with a microcontroller, these pins are driven to 0 or 1 levels before access to the memory.
- WP pin is used to write-protect.





I2C EEPROM Write Operations

- This memory has two write modes: byte write and page write.
- In byte write mode:
  - set the address position where you want to write,
  - the next byte will be the data that you want to store.
  - finally, send the Stop condition.



I2C EEPROM Write Operations

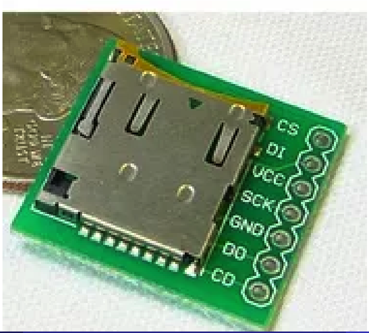
- The 32KB address is from 0x0000 to 0x7FFF.
- 15 bits to address.
- The sequence to access consists in:
  - Generate the Start condition in the I2C bus (SDA low, then SCL low).
  - Send the device address (0x00 or ox01 in above example)
  - Send the high byte of the address to access
  - Send the low byte of the address to access.
  - the next byte will be the data that you want to store.
  - finally, send the Stop condition.

I2C EEPROM write operation

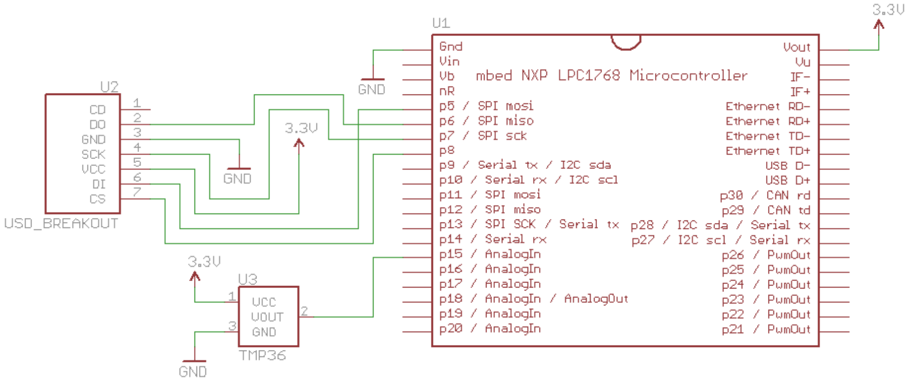
```
#define EEPROM1_WR  0xA0        // 24LC256 Address to write
#define EEPROM1_RD  0xA1        // 24LC256 Address to read
char buf[2];
buf[0]=0x01;
buf[1]=0x00;
buf[2]=0xAA;
I2C i2c(p27, p28);
i2c.write(EEPROM1_WR, buf,3);
```

SD cards on the mbed

- SD Cards are widely used by devices for storage; phones, mp3 players, pc's etc.
- SD cards support SPI protocols.
- SD Cards are block devices. That means you read/write data in multiples of the block size (usually 512-bytes).
- A file system SDFileSystem is an abstraction on top of the SPI protocol.



SD cards on the mbed



Data files on the mbed

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

  SDFileSystem sd(p5, p6, p7, p8, "sd");

  int main() {
    mkdir("/sd/mydir", 0777);

    FILE *fp = fopen("/sd/sdtest.txt", "w");
    if(fp == NULL) {
        error("Could not open file for write\n");
    }
    fprintf(fp, "Hello fun SD Card World!");
    fclose(fp);
}
```