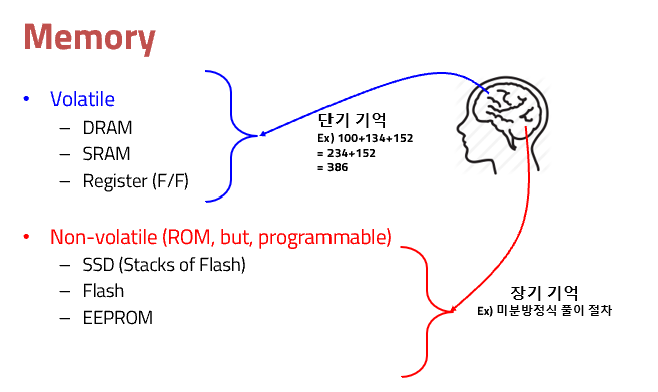
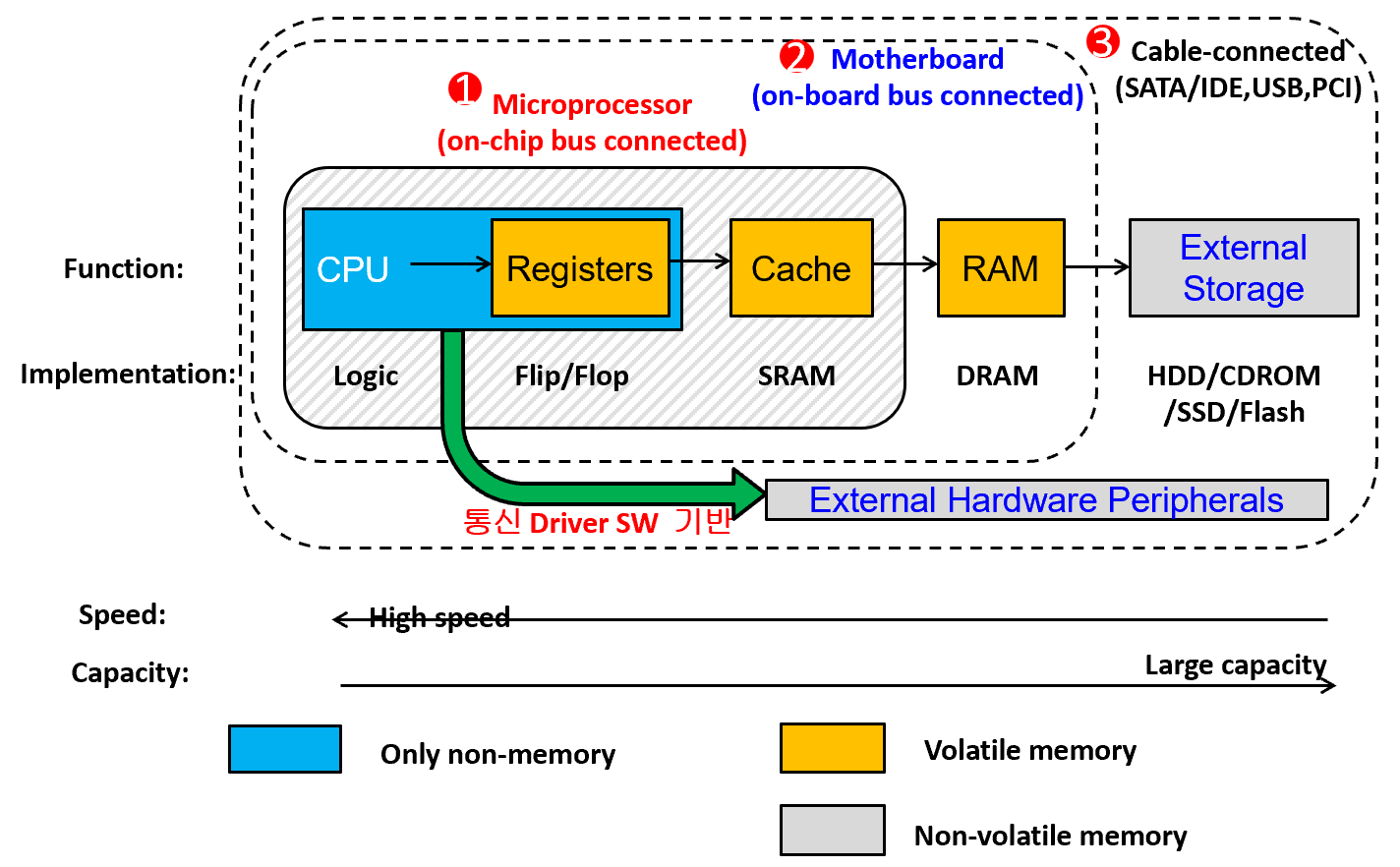
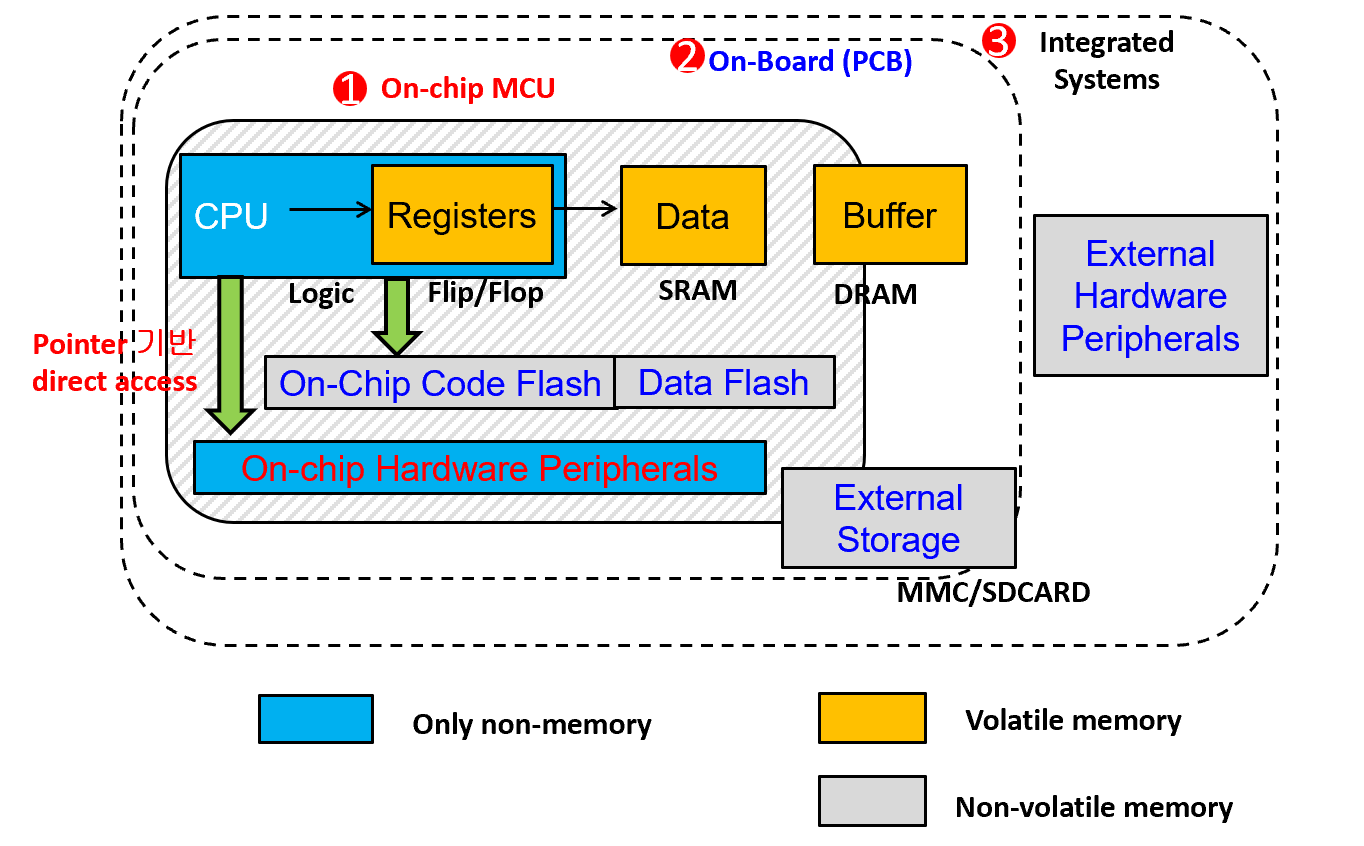
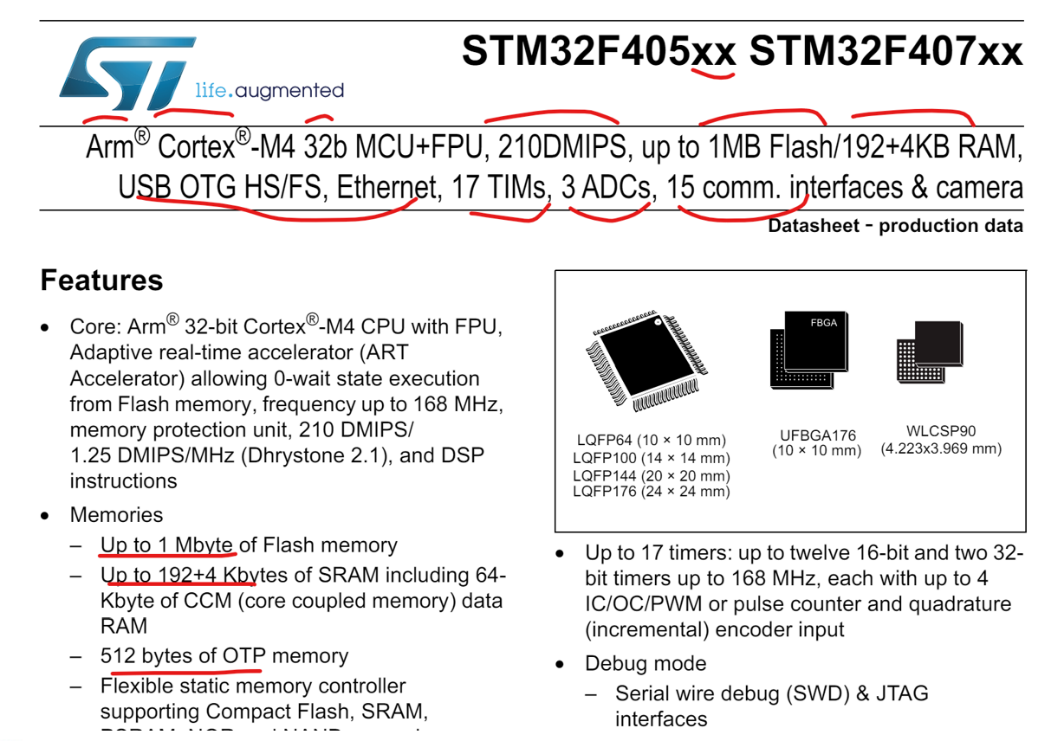
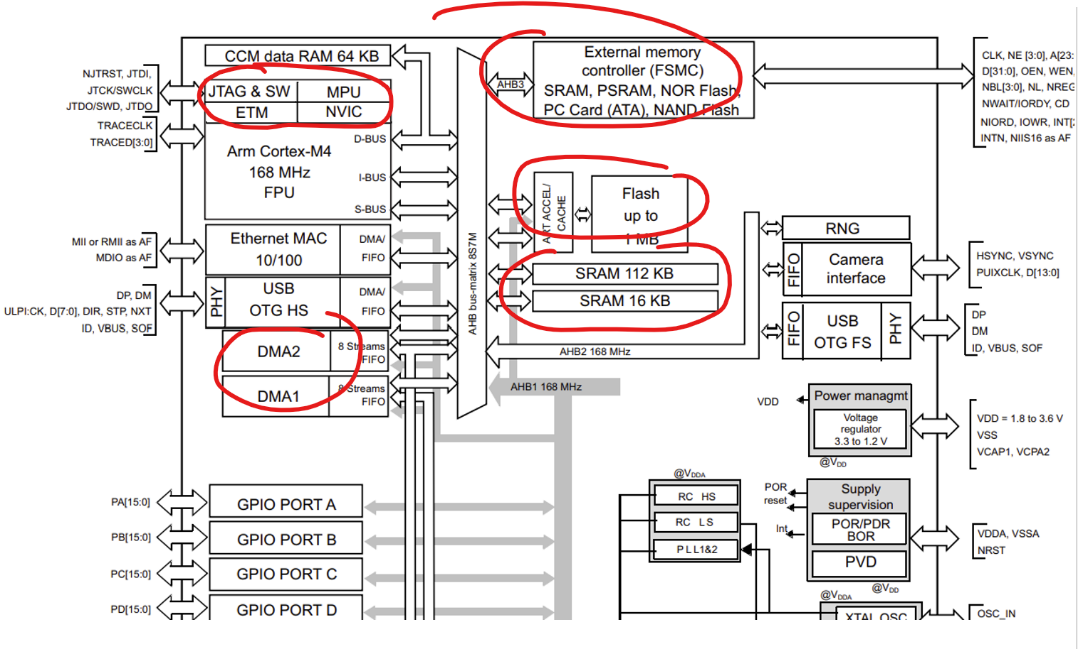
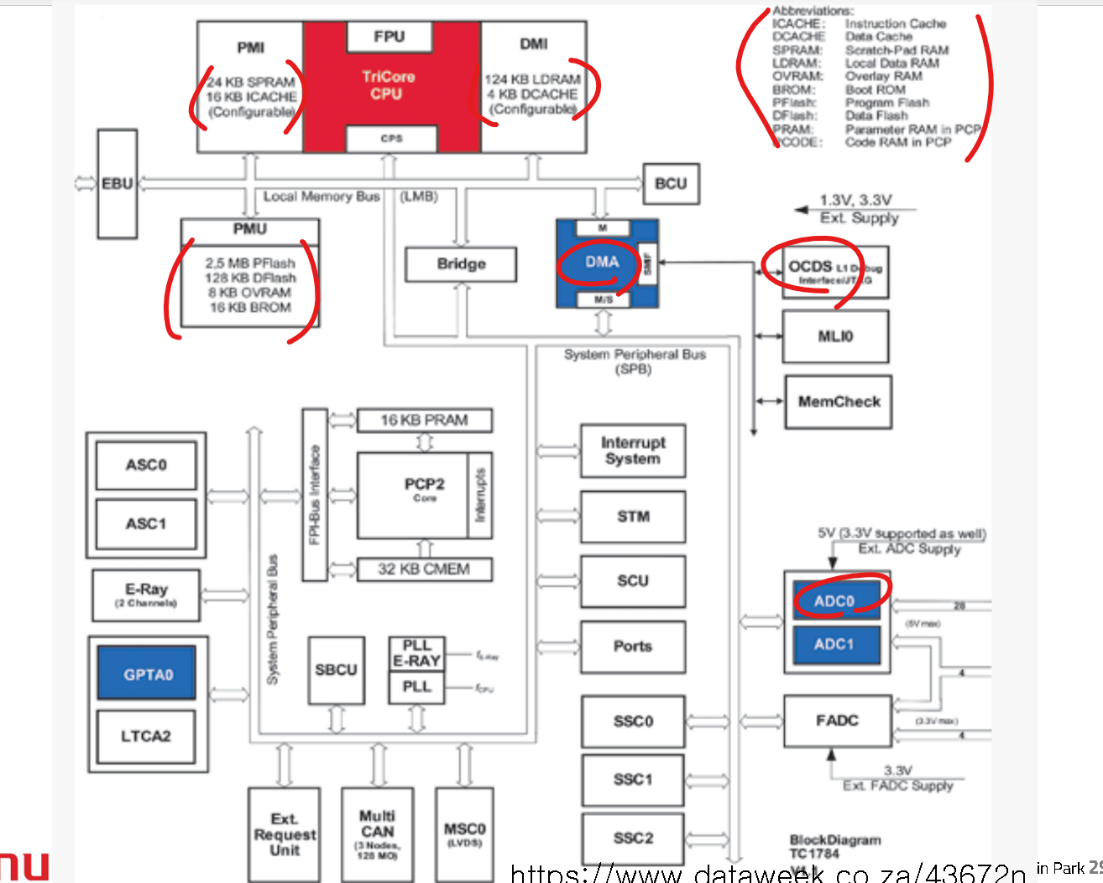
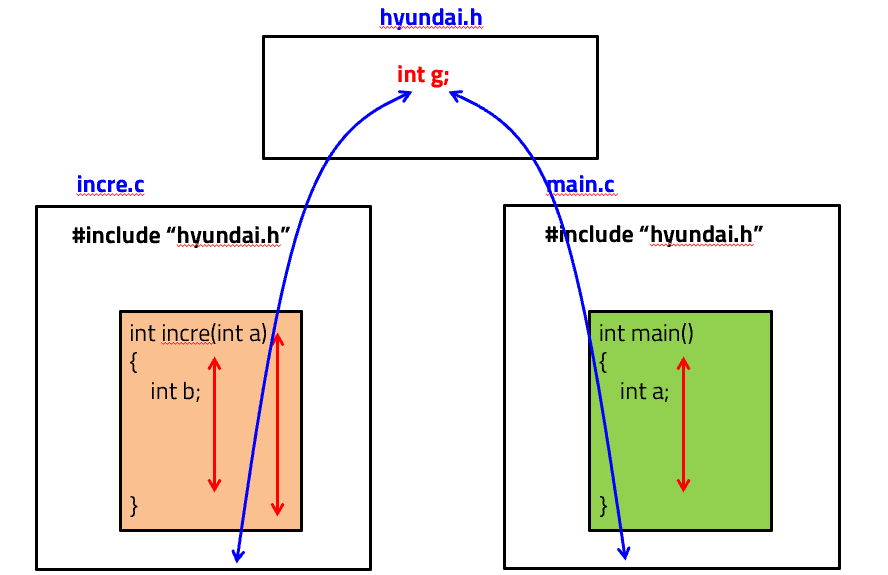
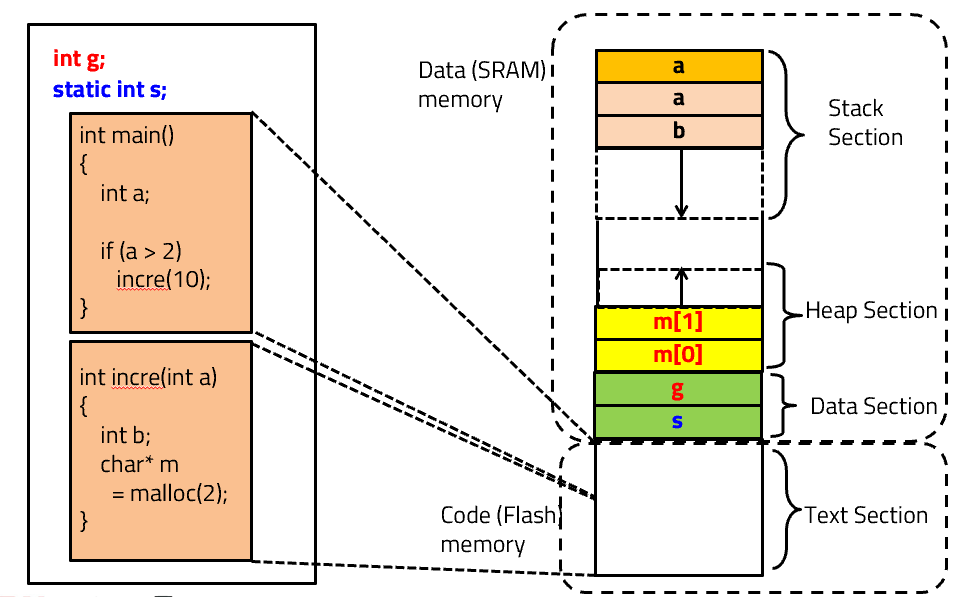
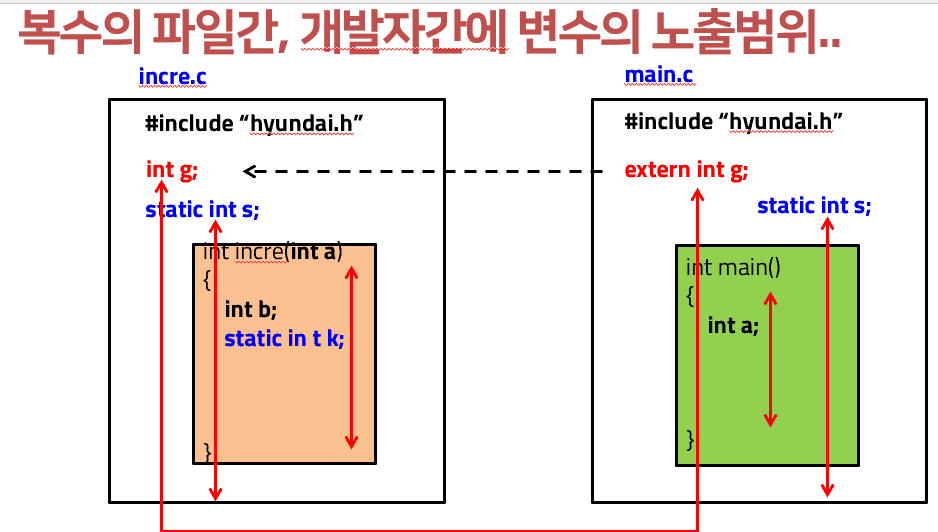
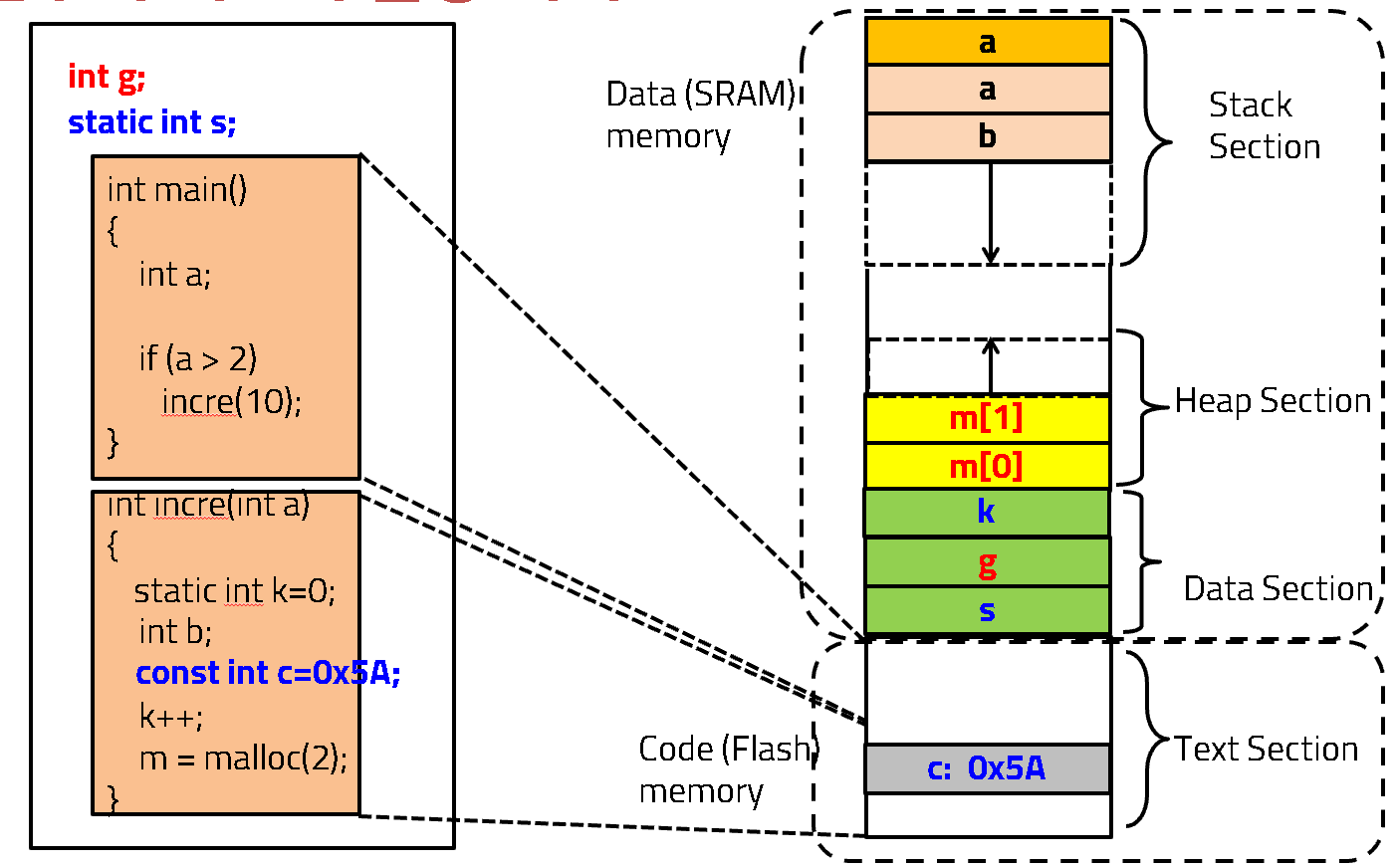
* **1. 임베디드 (메모리) 관점의 내장 HW접근제어**
  + **(1) 임베디드 시스템에 사용되는 메모리 구분.**
    - 임베디드 SW를 구동할려면 반드시 메모리가 필요하다는 사실에 기반하여 메모리를 구분해보자!
      * 
      * flash메모리는 EEPROM 메모리에서, byte단위 write.기능을 삭제
        + write,할때는 all erase후에 한꺼번에 write하면되므로
        + **EEPROM 기능중에 byte/bit.단위 write기능을 제거하여 회로 크기 줄이고 많은 flash memory cell을 삽입함으로써, 용량을 키운 방식임.**
      * flash.메모리 접근 속도는 매우 느리다는 것을 기억하라.
        + **스피드를 높이려면, flash에 적재된 컴파일된 바이너리 코드를 SRAM (data memory)나 scratch pad memory등으로 가져와서 사용자 수준 캐쉬 효과를 낼 수 있도록 구현해야 함.**
      * flash메모리에는 현재 cpu가 읽어서 실행해야 할 코드가 적재
        + 따라서 다른 코드를 업데이트 하고 싶으면
        + (1) 현재 시스템동작을 멈추고, all erase, all write with new firmware
        + (2) 혹은 러닝중에 firmware 업데이트 하고 싶으면

두번째 flash bank에 erase/write하고

다음부터는 두번째 flash bank에 내장된 SW.를 읽고 실행하도록 모드 변경하면됨.

* + **(2) Memoryless vs Memory-Embedded System 기반 HW 접근 제어방식 차이**
    - **A. Memoryless-based Microprocessors**
      * 
      * 대규모 메모리가 필요하고 DRAM을 보드레벨 장착, 외장 SSD 플래쉬 장치를 사용하여 매우 많고 크기가 큰 앱들을 **원할때 설치하고 원할때 실행/종료할 수 있음**
      * 장치들이 **보드 밖에 있으므로 pointer기반 direct access안되고 별도의 드라이버 소프트웨어를 구동**해서 (통신 드라이버) 간접적으로 장치에 접근해야 함. (확장성은 좋으나 느림)
    - **B. Memory-embedded Microcontrollers**
      * 
      * 제어시스템을 SW로 제어하려면, 온칩 메모리는 필수임.
      * **칩내부에 있기 때문에 SW를 메모리에 적재하는 시간이 불필요함.**
      * **directly memory access를 통해 실행할 프로그램을 읽어와서 실행하므로. 성능이 빠름 (엄밀히 말하면 일정함.)**
* 2. 내장 HW 스펙에 따른 내장 SW 동작 특성 연관성
  + **(1) SW실행 특성에 영향을 주는 core 스펙**
    - 
    - 1. ISA (바이너리 호환) 여부
      * ARM cortex, Intel x86, PowerPC, RISC-V, TriCore, 8051, Freescale
    - 2. 컴파일러 관련 (개발환경)의 편리성, 풍부한 라이브러리
      * HAL 드라이버
      * 에뮬레이터/디버거 지원여부
    - 3. DSP 지원여부
    - 4. Floating연산 지원여부
    - **5. 성능 기준 (외부 장치 I/O속도 대비 신호/제어 연산 한계 파악)**
      * modeling simulation통해 underflow/overflow파악
  + **(2) 온칩 HW 연결 구조에 따른 SW 성능변화.**
    - 
    - 
    - 1. 온칩 메모리
      * Data memory (local variable 공간, scratchpad 공간, 캐쉬)
      * Code memory (code flash, data flash)
      * 크기, 및 속도
    - 2. 온칩 디버거 (OCD) 탑재여부
      * 디버거 해상도, 속도, 버퍼링 지원여부.
      * 외장 디버거 지원 제품 종류 파악.
    - 3. MMU 탑재 여부
      * 가상메모리 지원여부 --> OS 설치 가능ㅇ
    - 4. DMA 모드 지원여부
      * 외부 I/O, 통신 채널로부터 내부 장치 redirect 모드 및 채널 갯수
    - 5. 메모리 확장 여부
      * 외부 SPI ROM (Nor Flash, Nand Flash)
      * 외부 롬으로부터 부팅 가능여부.
    - 6. ADC/DAC 해상도 및 속도 여부
      * **speed, 해상도, 채널수에 따라 SW실행 특성/성능이 달라진다.**
    - 7, 통신 지원여부
      * UART, SPI, CAN, LIN, I2C, SENT. DSI
      * **HW탑재 여부에따라 구동 드라이버 코드 방식이 달라진다.**
    - 8. 하드웨어 인터럽트
      * 인터럽트 갯수, 콜백 호출 방식
* **3. 온칩 임베디드 메모리를 이용한 프로그래밍**
  + 지역변수, 전역변수의 live한 범위는 아래와 같다.
    - 
  + **각 변수의 메모리 배치위치는 아래와 같다.**
    - 
  + **(1) 지역변수**
    - **함수 내부에서만 보이는 변수. 함수 밖에서는 접근 X**
    - **함수가 호출될때 스택 영역에 변수가 할당됨.**
    - 함수를 nested call (or recursive call) 형태로 깊이 호출하면 스택 변수가 계속 할당되어, **메모리 사용량이 늘어남. 동적변수/전역변수 공간을 침범할 가능성이 있음.**
    - **함수내의 지역변수의 초기값은, 함수가 호출될때 함수 내부에 포함된 명령어 실행결과로 초기값이 잡힌다**
  + **(2) 전역변수**
    - 전역변수의 장점과 단점
      * **변수의 접근 범위는 프로그램 전체 (파일 전체)**
      * **프로그램 종료전까지는 값을 유지함 (overwrite하지 않으면)**
      * **반면 지역변수는 함수 리턴하면 변수도 없어짐.**
    - 전역변수의 live한 범위에 따라 다음 3가지로 나뉜다
      * 
      * **그냥 전역변수**
        + 파일내의 모든 함수에서 접근가능하고
        + 심지어 다른 파일에서도 모두 접근가능한 변수
        + 그냥 함수 바깥에 변수 하나 선언하면 전역변수임
        + **전역변수를 헤더에 선언하지 말고 c 파일안에 선언후 컴파일, 외부에는 드러나지 않게 만듬, 이 전역변수를 사용하고 싶으면 명시적으로 extern을 사용하여 접근, 누구나 접근할때의 위험성을 어느정도 막을 수 있음 (변수 명을 모르면 전역변수 접근이 쉽지 않게)**
      * **정적 전역변수**
        + **전역변수 앞에 static붙이는것**
        + 변수가 선언된 파일안에 모든 함수에서 접근가능하지만 다른 파일에서는 접근 못함
        + **함수 바깥 전역 변수에 static 한정자를 사용하면 global 변수의 장점(프로그램 종료전까지는 값을 유지)을 유지 면서, 접근 범위를 제한할수 있음 (선언된 현재 파일 내부로만 접근 범위를 제한.)**
      * **정적 지역변수**
        + **지역변수 앞에 static붙임.**
        + 변수선언 위치가 함수 내부이지만 전역변수이며, 단지 접근 범위가 선언된 함수 내부에서만으로 제한될 뿐임.
        + 함수를 호출하고 리턴이 되어도 그 값이 유지됨 (전역변수의 특성 그대로)
        + **함수내의 변수에 static 한정자를 사용하면 global 변수의 장점(프로그램 종료전까지는 값을 유지)을 유지 면서, 접근 범위를 제한할수 있음 (함수 내부로)**
        + **접근 범위가 함수 내부로만으로 제한될뿐, 근본적으로 전역변수임**
        + **따라서 함수 종료후에도 값이 남아있다 다음에 다시 호출했을때 이전에 남겨진 값을 바탕으로 다음 처리가 가능해짐 (인터럽트 서비스 루틴에서 사용함)**
    - 함수가 호출될때 지역변수 값이 초기화되는 반면, 전역변수는 프로그램 실행과 동시에 이미 초기값이 채워져 있다.(매우 중요함)
      * 프로그램이 **실행과 동시에 전역변수가 메모리에 할당되고 초기값이 설정된다**. **전역변수는 읽고 쓸수 있어야 하므로  SRAM에 배치**되어야 할것이다. **초기값은 flash메모리에 있었을 것이다. 이 값이 부팅할때 읽어서 SRAM으로 옮겨져야 한다.**
        + **속도 저하, 메모리 추가 사용의 단점존재**
        + **-> 이것 해결하기 위해 상수 변수 사용 (값을 새로 수정필요없을경우)**
      * main() 함수 실행하기 전에 startup.s 부트 코드에서 초기 값이 설정되어야 함 (컴파일러가 전역변수 초기값 설정하는 코드 생성해줌.)
      * **정적 지역변수의 초기값이 매 함수 호출마다 초기화화 되지 않는 이유임.**
  + **(3) 동적변수**
    - 지역, 전역 변수는 크기가 일정함 (배열이라 하더라도, 배열의 크기는 상수여야 함)
    - 따라서 런타임에 메모리가 모자라지 않도록 넉넉하게 잡아야 한다. 이때 비효율적인 메모리 낭비의 가능성이 커진다.\
    - **동적변수는 런타임의 사용자 요청에 따라 사용자가 지정한 크기만큼만 메모리에 변수를 할당하고 더이상필요없을 경우 반환도 할 수 있다.**
    - malloc()함수를 직접 구현해야 한다 .(임베디드 용으로)
  + **(4) 상수변수**
    - 
    - 변수를 SRAM에 배치하는게 아니라, flash메모리 영역에 배치 (flash굽는과정에서 c변수가 배치된 영역에 0x5A가 저장됨)
    - const를 쓰지 않으면, 최초 부팅할때 변수를 SRAM영역에 배치해야 하고, 초기값 0x5A를 쓰는 명령어를 실행해야 한다 (메모리 사용량 증가 및 코드 실행시칸 증가)
    - **따라서 const 키워드를 붙이면 굳이 변수를 RAM에 할당하고 초기값 설정하는 코드를 실행시킬 필요없으므로 메모리도 아끼고 초기화 시간도 아낄수 있음.**
  + **(5) 포인터변수**
    - **A. 변수의 주소를 얻기.**
      * 모든 변수 및 하드웨어 장치는 메모리의 특정 영역에 배치되어 있다.
      * 특정 영역이라 함은 주소를 말한다.
      * 변수의 메모리 상에 배치된 주소를 얻고 싶으면 변수 이름 앞에 & 기호를 붙인다.

|  |  |
| --- | --- |
|  | **#include <stdio.h>**  **main() {**  **...**  **unsigned int k1 = 0x5A1234FF;**      // 변수는 메모리의 특정 주소에 배치됨      // 그 주소는 변수이름앞에 &기호를 붙이면 된다.  **printf("k1's address on memory : 0x%p\n", &k1);** |

* **B. 주소 값도 값이므로 변수에 담을 수 있다. 변수에 담아보자!**
  + **주소값을 저장하고 있는 변수를 포인터 변수**라고 한다.
  + 모든 변수에는 **변수안에 담길 변수의 값의 종류를 표현해야 한다 (타입이라고 한다)**

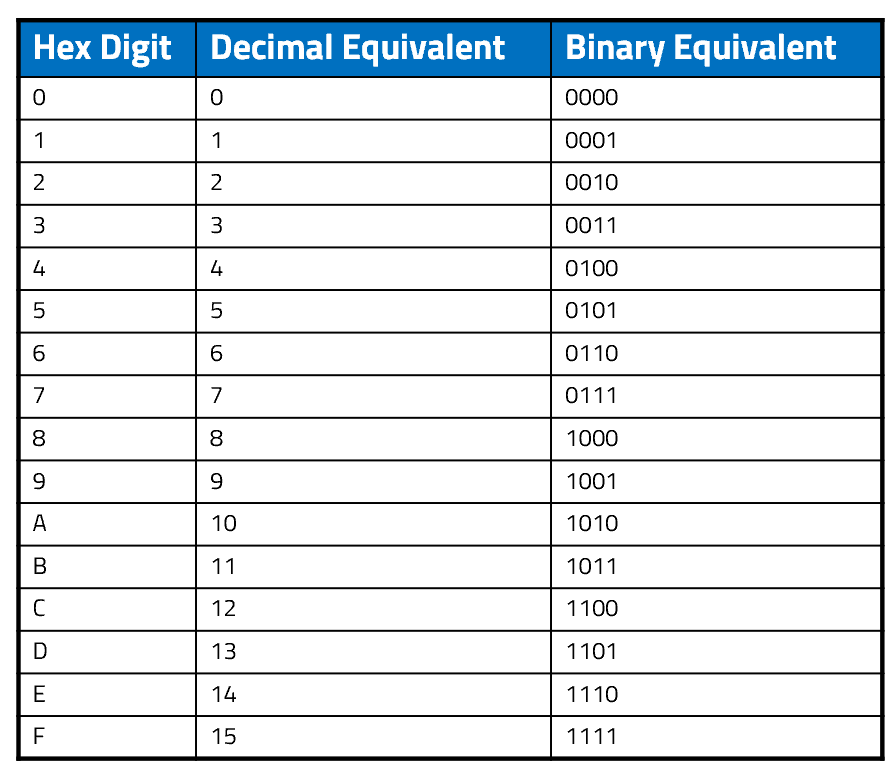
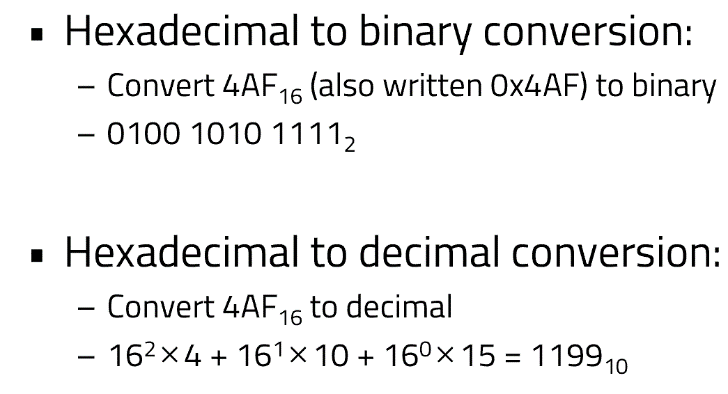
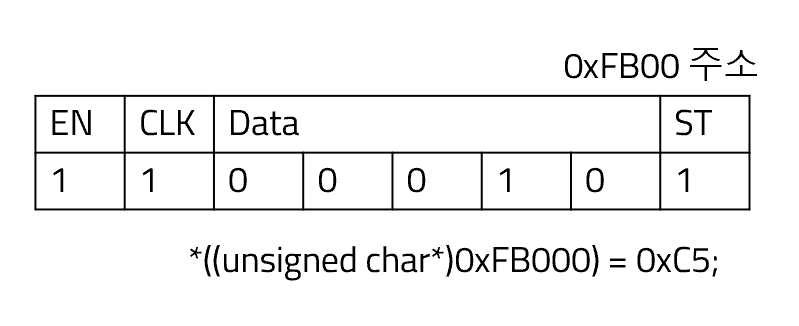
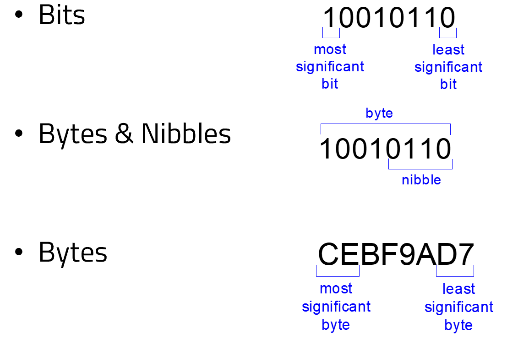
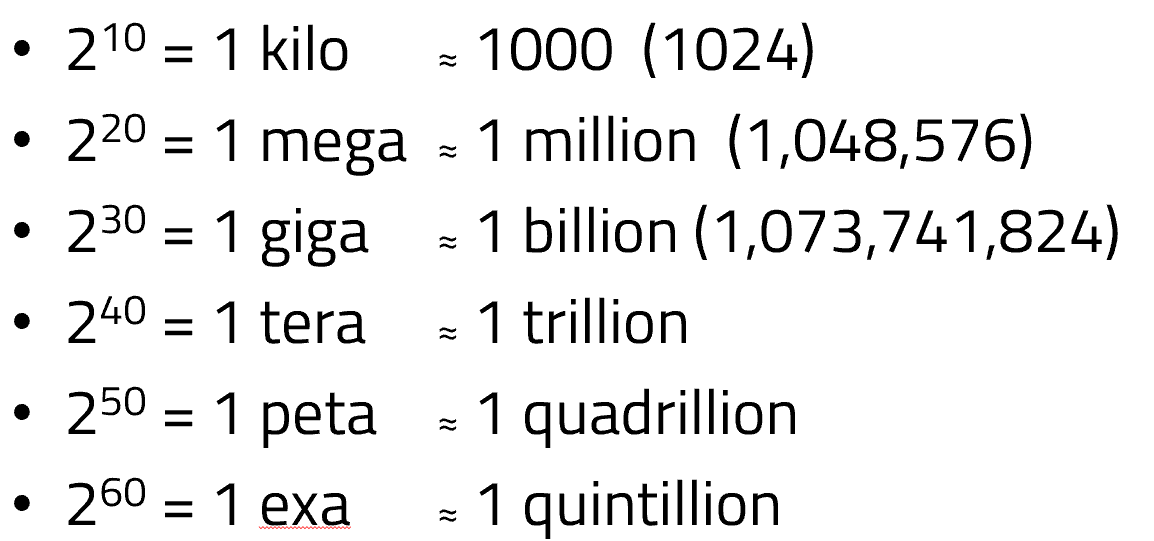
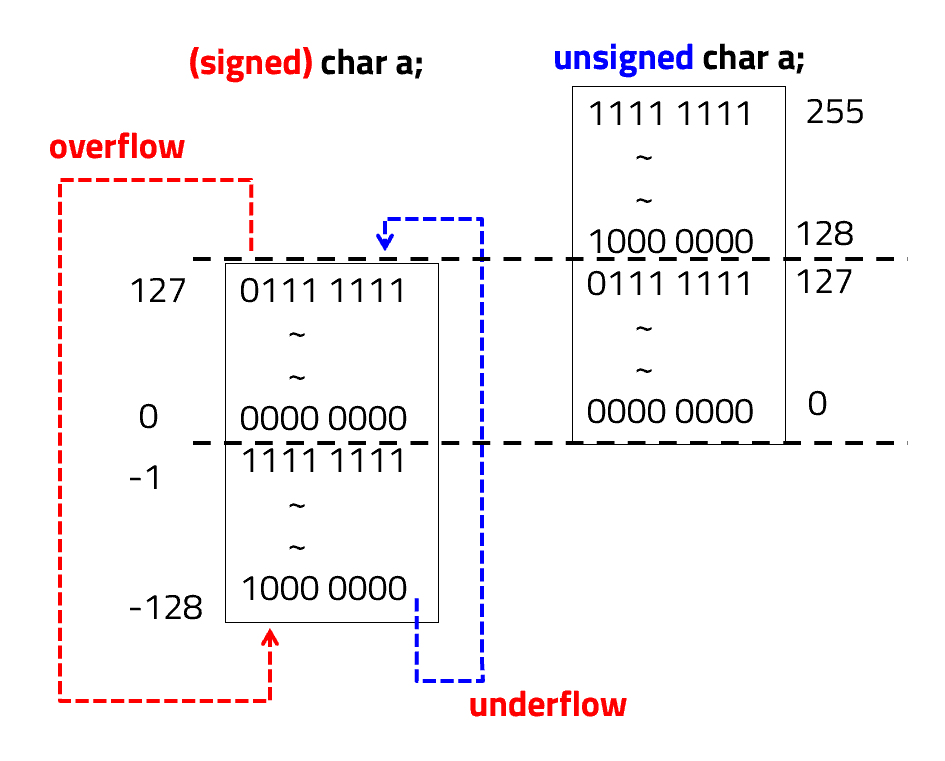
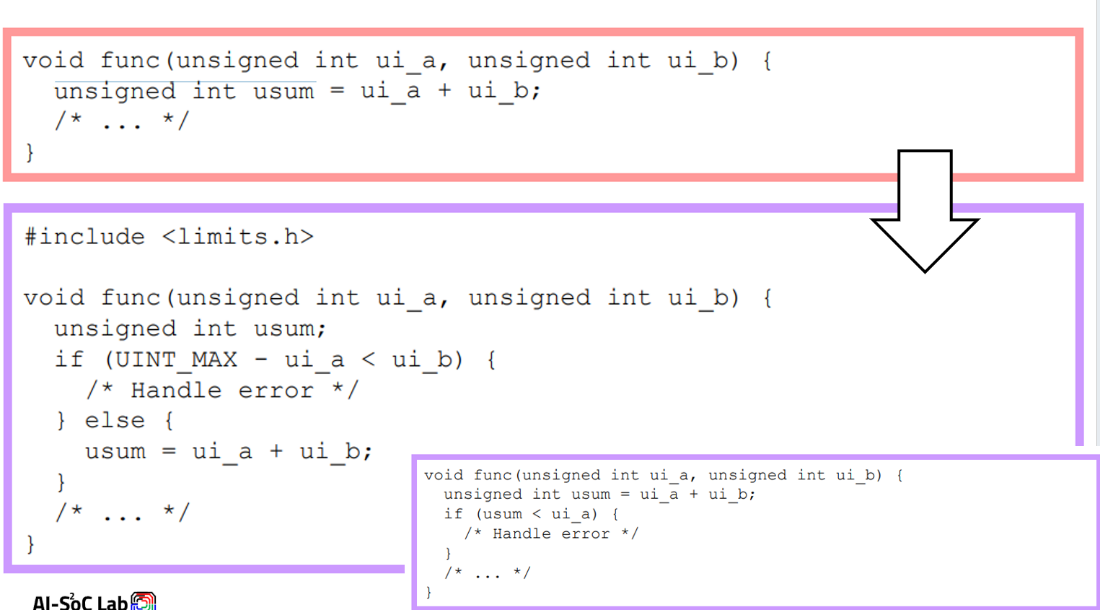
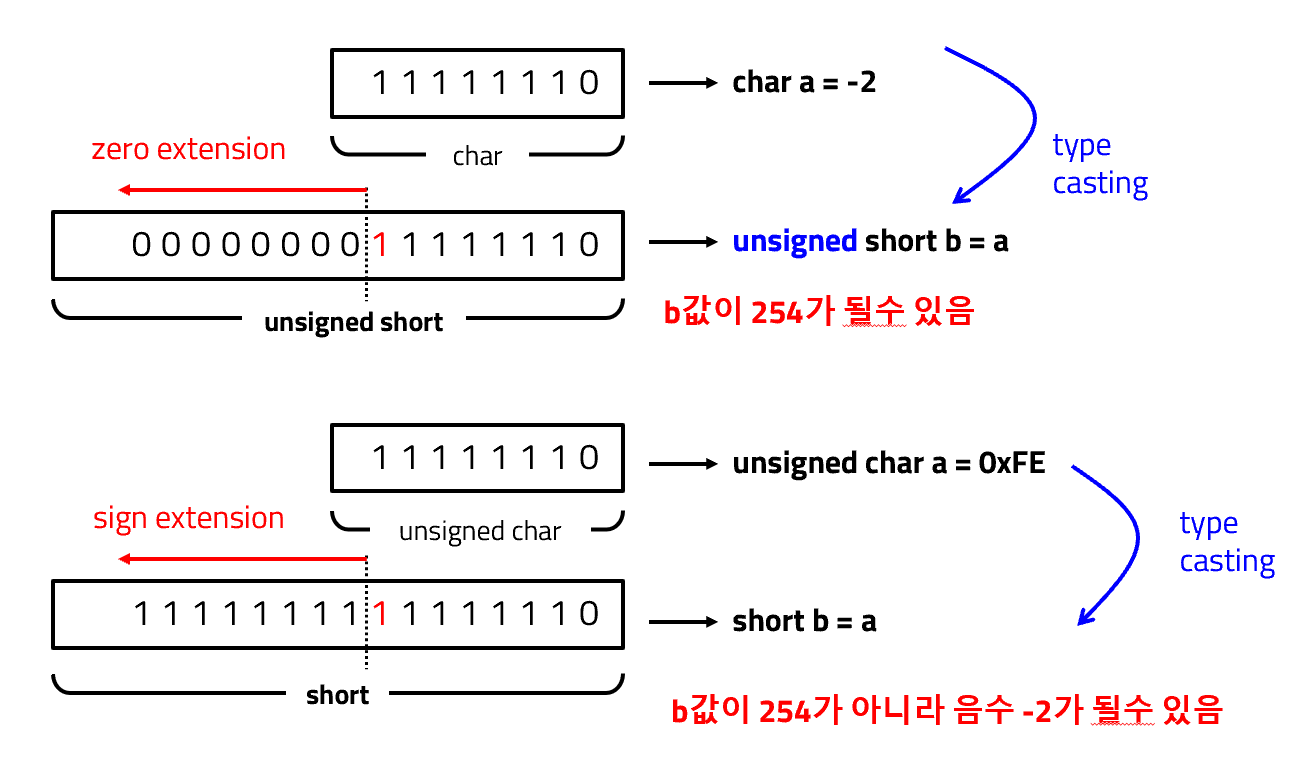
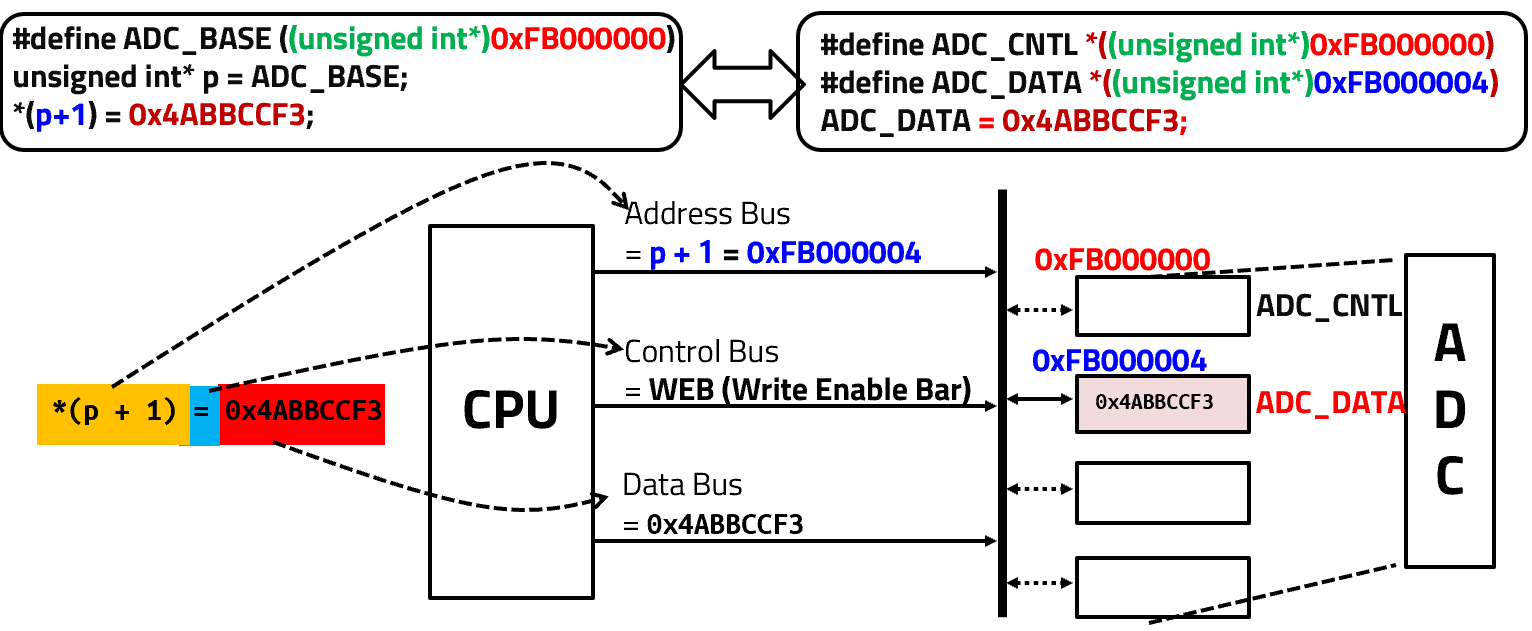
|  |  |
| --- | --- |
|  | // k1의 주소를 변수에 담아보자.      // 이제 p라는 변수안에는 주소가 담겨있다.      // 그려면 p라는 변수에 들어갈 값을 구체적으로 타입으로 정의해야함      // **변수p의 타입을 int\* 로 표시**하면 의미는 다음과 같다.      // **변수p에 담길 값은 주소값이고, 가리키는 대상이 4바이트 정수**라는 의미  **unsigned int\*** p = **(unsigned int\*)&k1**; |

* **C. 이제 주소값을 이용해서 대상에 접근해보자.**
  + **포인터 변수 이름앞에 \*를 붙이면된다.**

|  |  |
| --- | --- |
|  | // 자 이제 이 주소값을 이용해서 대상에 접근해보자.      \*p = 0x12345678;      // 변수 p안에 담긴 주소를 이용하여 간접적으로 k1 값이 변경됨을 확인.      printf("k1: 0x%08X\n", k1);      // 변수에 접근해서 값을 쓰는 행위는 다음의 3가지로 표현할수 있다.      //                    **k1**       = 0xFFBBCCDD      //                  **\*(&k1)**      = 0xFFBBCCDD;      // \*((unsigned int\*)**0x700195FC**) = 0xFFBBCCDD;  **\*((unsigned int\*)0x700195FC) = 0xFFBBCCDD;** |

* 이렇게 #define을 사용하면 좀더 직관적으로 보인다.

|  |  |
| --- | --- |
|  | // 이런 원리로 이미 할당된 주소영역에 접근하여 값을 쓰는 방법.  **#define ADC\_DATA \*((unsigned int\*)0x700195FC)**      ADC\_DATA = 0x55BBCCFF; |

* **4. 변수의 타입과 값**
  + **(1) 2진수, 10진수, 16진수**
    - **A. 변환 테이블.**
      * **아래 테이블을 영혼에 각인하라!**
      * ****
    - **B. 16진수, 2진수 변환을 자유자재로 해야 하는이유 (HW 제어는 비트 단위로 해야 하는데 C로 코딩할때는 16진수로 표현해야 함)**
      * 
      * **만약. 아래처럼 특정 HW 레지스터를 셋팅하고 싶으면**
      * 
  + **(2) bit, byte, nibble**
    - 
    - **n비트로 표현할때 값의 범위..**
    - 
    - n비트로 표현할수 있는 서로 다른 값의 종류 (혹은 범위)
    - **16비트이면, 64x1024개의 값 (0~64x1024-1 까지)을 표현할 수 있다.**
    - **핸들을 360도로 회전하면 센서 출력 전압이 0~5V로 변하고 이전압을 MCU내부의 ADC에 의해 16비트로 샘플링하면, 이는 360도/64x1024의 해상도로 감지하고 그때 최소 전압변화는 5V/(64x1024) 임.**
  + **(3) overflow, underflow**
    - **MCU는 유한한 정보를 저장하고 처리할 수 있다. 일정 이상 입력 데이타 범위가 커지면 (혹은 곱하기 등에 의해 계산결과가 너무 커져도 마찬가지) 결국 overflow현상을 막을수는 없다**
      * **1. 데이타 입력 범위를 적절히 조절해야 한다.**
      * **2. 계산결과 얻어진 값이 overflow, underflow되지 않았는지 검사해야 한다.**
      * **3. 굳이 음수 표현이 필요없다면 unsigned로 양수 데이타 타입을 써라 (2배 더 overflow/underflow 발생하지 않는 여유가 생긴다.)**
      * 
    - **Safe coding방법, 데이타 무결성 채크를 늘 하라**
      * ex) 더하기 결과값을 믿지 말고 overflow 확인
      * 
  + (4) 음수, 양수 변수 혼합해서 사용하지 마라
    - **비트 확장/축소시 의도치 않는 변환이 생길수도 있다.**
    - 
* **5. 온칩 임베디드 SW가 내장 HW와 연동하여 동작하는 원리.**
  + **(1) 내장 HW에 접근하기 위해 온칩 버스 구동..**
    - 메모리에 매핑된 하드웨어 (변수가 배치된 SRAM을 포함한 모든 하드웨어 장치)에 접근하는 행위는 다음의 절차를 거친다.
      * 1. 주소신호 구동
      * 2. 컨트롤 신호 구동 (R/W)
      * 3. 데이타 전송
    - **이 한줄이 임베디드 내장 HW에 접근하는 방법의 모든것을 담고 있다.**
      * cpu는 아래 c코드를 디코딩해서, 위에 버스 3개를 구동하는 회로임.
* 
* **(2) 포인터(주소)를 사용하면 프로그램이 빨라지는 이유..**
  + **변수의 값은 N바이트이지만, 주소는 항상 고정크기 4바이트임.**
    - 함수로 변수 값 전체를 넘기면 메모리도 추가할당되어야 하고, 값을 복사하는 시간도 필요함
  + **하지만 변수의 주소만 인자로 넘기면.. 4바이트만 복사되면 끝난다.**
    - 대신 주소를 넘겨받은 함수에서, 이 주소를 이용하여 변수에 직접 접근한다.
* 