AVR - HW3

임베디드스쿨1기
Lv1과정
2020. 09. 24
박성환

1. Review-AVR Ext Interrupt

```c
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define sbi(PORTX, DitX)    (PORTX |= (1 << DitX))

SIGNAL(INT1_vect)
{
    PORTB = 0x20;    // PB5(13번핀) 출력 '1'로 설정
    _delay_ms(200);  // 200ms 동안 LED 'on'
}
int main(void)
{
    sbi(SREG, 7);        // SREG I 비트 1로 셋팅
    sbi(EIMSK, INT1);    // EMISK INT1 1로 셋팅
    EICRA = 0x08;        // INT1 falling edge 셋팅

    DDRB = 0x20;         // PB5(13번핀) 출력 핀으로 사용
    DDRD = 0x00;         // 입력으로 사용 (INT1 = 3번핀 INT0 = 2번핀)
    PORTD = 0xff;        // 풀업저항

    /* Replace with your application code */
    while (1)
    {
        PORTB = 0x00;    // PB5(13번핀) 출력 '0'로 설정
    }
}
```

1. Review- ELF, HEX, BIN, JTAG 간단요약

- 보통 컴파일 -> Axf(=elf) 파일 생성
- Axf(=elf)파일에는 여러 디버깅info들, 변수/함수 주소 + bin 등으로 구성되어 있음(디버깅시 사용됨)
- Hex, Bin은 Optional로 구성된 경우가 많고 실제 Flash memory에는 bin파일이 올라감
- Hex는 Bin파일에 주소가 추가 되어있다고 간단히 생각하기(bin파일 입력 시 주소정보 필요)
- ST-Link 와 TI-Link는 Breaking Point 사용하며 디버깅 가능하기 때문에 JTAG기능을 한다고 볼 수 있음

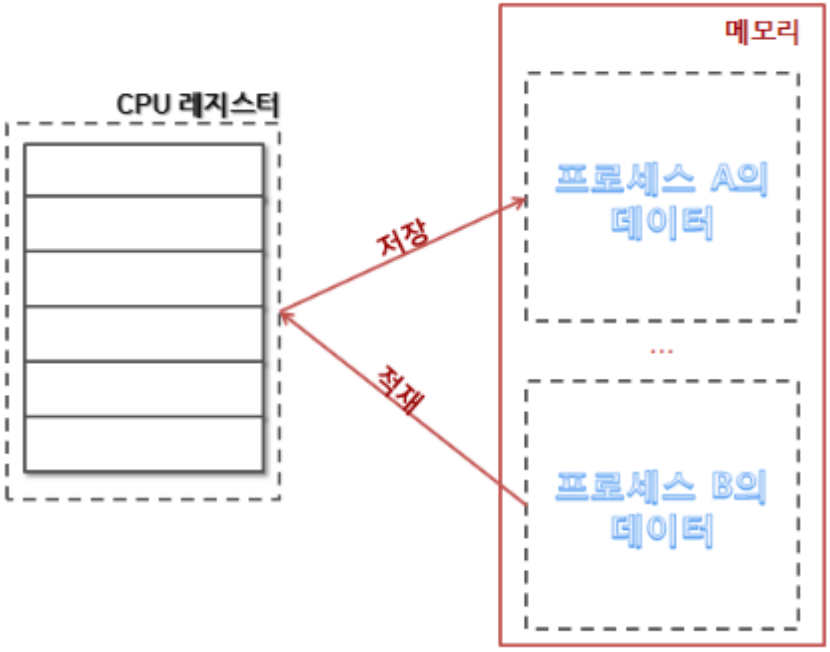
2. Context Switching

1. Context Switching이란

- 현재 진행하고 있는 Task(Process, Thread)의 상태나 레지스터 값을 저장하고 다음 진행할 Task의 상태나 레지스터값을 적용하는 과정

2. PCB(Process Control Block)

- Context Switching시 PCB에는 프로세스 상태 또는 레지스터의 값등과 같은 관련 데이터들이 저장됨
- 프로세스 생성 시 만들어지며, 프로세스마다 고유한 PCB를 가짐(Task(Process/Thread)에 대한 정보로 대부분의 정보는 Register에 저장되며 PCB로 관리됨

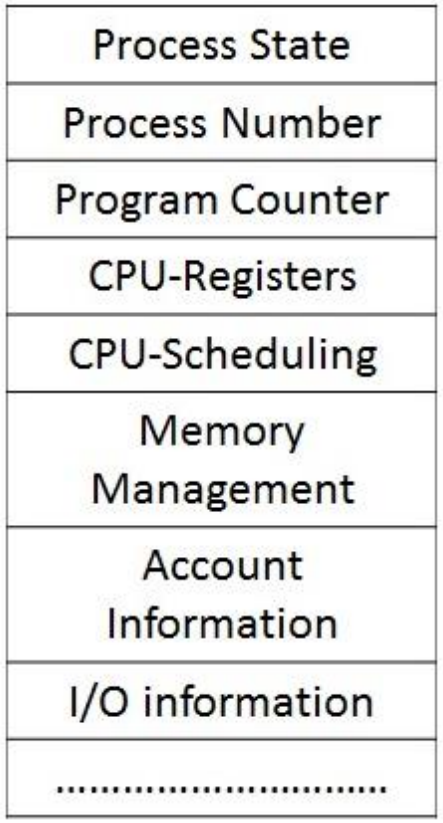


2. Context Switching

3. PCB 저장정보
1) 프로세스 식별자(Process ID)
2) 프로세스 상태(Process State) :
생성(create), 준비(ready), 실행 (running), 대기(waiting), 완료(terminated)
3) 프로그램 계수기(Program Counter) : 이 프로세스가 다음에 실행할 명령어의 주소
4) 사용 중인 레지스터 정보
5) CPU 스케줄링 정보 : 우선 순위, 최종 실행시각, CPU 점유시간 등
6) 메모리 관리 정보(Memory limits) : 사용 가능한 메모리 공간 정보
7) 입출력 상태 정보 : 프로세스에 할당된 입출력장치 목록, 사용 파일 목록 등
8) 포인터 : 부모 프로세스에 대한 포인터, 자식 프로세스에 대한 포인터,
프로세스가 위치한 메모리 주소에 대한 포인터, 할당된 자원에 대한 포인터 정보

4. Context Switching Cost
Cache 초기화
Memory Mapping 초기화
Kernel은 항상 실행되어야 함(메모리 접근을 위해)
PCB를 저장하고 가져오는 오버헤드 시간

5. Context Switching 발생시점
Hardware를 통한 I/O 요청이나,
OS/Driver 레벨의 Timer기반 Scheduling에 의해 발생



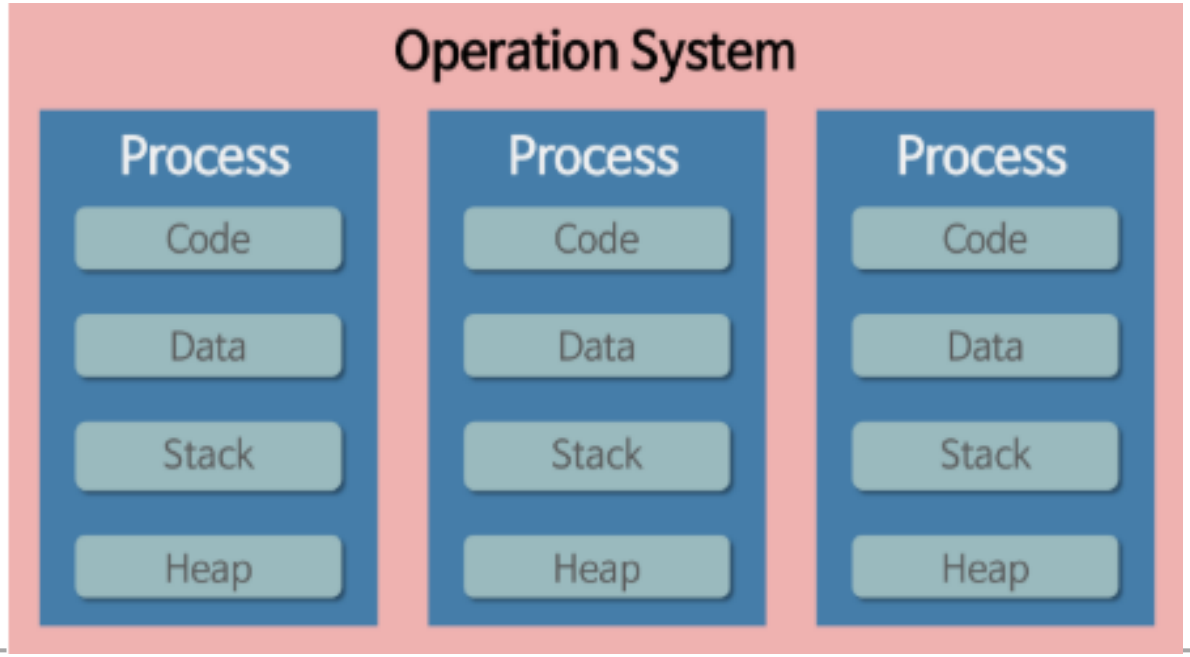
PCB 구조

참고. Process

1. Process 정의
컴퓨터에서 연속적으로 실행되고 있는 컴퓨터 프로그램
메모리에 올라와 실행되고 있는 프로그램의 인스턴스
운영체제로부터 시스템 자원을 할당받는 작업의 단위

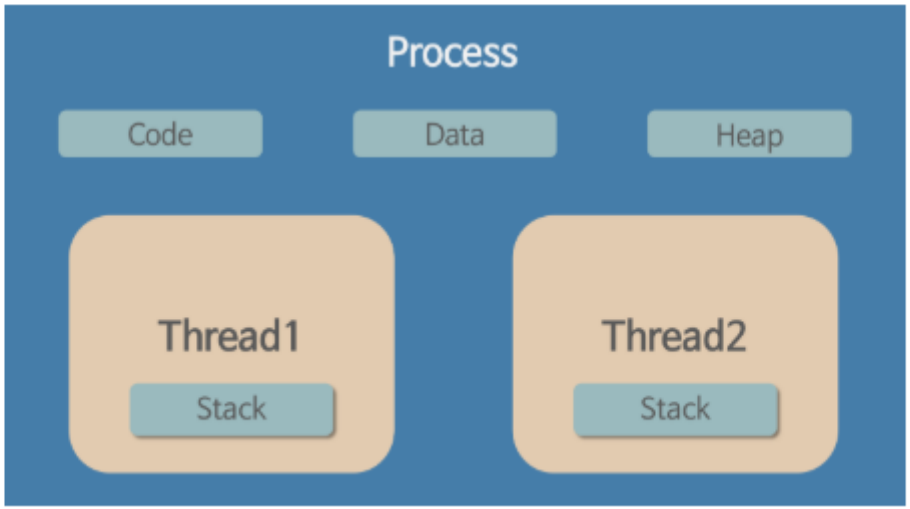
2. 할당받는 시스템 자원 예
CPU시간
운영되기 위해 필요한 주소공간
Code, Data, Stack, Heap의 구조로 되어 있는 독립된 메모리 영역

3. 특징
기본적으로 프로세스당 최소 1개의 스레드를 가지고 있음
각 프로세스는 별도의 주소 공간에서 실행되며, 한 프로세스는 다른 프로세스의 변수나 자료구조에 접근할 수 없음
한 프로세스가 다른 프로세스의 자원에 접근하려면 프로세스 간의 통신(IPC)을 사용해야 함



참고. Thread

스레드(Thread) 란

- 사전적 의미
  - “프로세스 내에서 실행되는 여러 흐름의 단위”
  - **프로세스의 특정한 수행 경로**
  - 프로세스가 할당받은 자원을 이용하는 실행의 단위
- 특징
  - 

  - 스레드는 프로세스 내에서 각각 Stack만 따로 할당받고 Code, Data, Heap 영역은 공유한다.
  - 스레드는 한 프로세스 내에서 동작되는 여러 실행의 흐름으로, 프로세스 내의 주소 공간이나 자원들(힙 공간 등)을 같은 프로세스 내에 스레드끼리 공유하면서 실행된다.
  - 같은 프로세스 안에 있는 여러 스레드들은 같은 힙 공간을 공유한다. 반면에 프로세스는 다른 프로세스의 메모리에 직접 접근할 수 없다.
  - 각각의 스레드는 별도의 레지스터와 스택을 갖고 있지만, 힙 메모리는 서로 읽고 쓸 수 있다.
  - 한 스레드가 프로세스 자원을 변경하면, 다른 이웃 스레드(sibling thread)도 그 변경 결과를 즉시 볼 수 있다.

참고. 멀티 프로세스 vs 멀티 스레드

멀티 프로세스

- 멀티 프로세싱이란
  - 하나의 응용프로그램을 여러 개의 프로세스로 구성하여 각 프로세스가 하나의 작업(태스크)을 처리하도록 하는 것이다.
- 장점
  - 여러 개의 자식 프로세스 중 하나에 문제가 발생하면 그 자식 프로세스만 죽는 것 이상으로 다른 영향이 확산되지 않는다.
- 단점
  - Context Switching에서의 오버헤드
    - **Context Switching** 과정에서 캐쉬 메모리 초기화 등 무거운 작업이 진행되고 많은 시간이 소모되는 등의 오버헤드가 발생하게 된다.
    - 프로세스는 각각의 독립된 메모리 영역을 할당받았기 때문에 프로세스 사이에서 공유하는 메모리가 없어, Context Switching가 발생하면 캐쉬에 있는 모든 데이터를 모두 리셋하고 다시 캐쉬 정보를 불러와야 한다.
  - 프로세스 사이의 어렵고 복잡한 통신 기법(IPC)
    - 프로세스는 각각의 독립된 메모리 영역을 할당받았기 때문에 하나의 프로그램에 속하는 프로세스들 사이의 변수를 공유할 수 없다.
- 참고 Context Switching란?
  - CPU에서 여러 프로세스를 돌아가면서 작업을 처리하는 데 이 과정을 Context Switching라 한다.
  - 구체적으로, 동작 중인 프로세스가 대기를 하면서 해당 프로세스의 상태(Context)를 보관하고, 대기하고 있던 다음 순서의 프로세스가 동작하면서 이전에 보관했던 프로세스의 상태를 복구하는 작업을 말한다.

참고. 멀티 프로세스 vs 멀티 스레드

멀티 스레드

- 멀티 스레딩이란
  - 하나의 응용프로그램을 여러 개의 스레드로 구성하고 각 스레드로 하여금 하나의 작업을 처리하도록 하는 것이다.
  - 윈도우, 리눅스 등 많은 운영체제들이 멀티 프로세싱을 지원하고 있지만 멀티 스레딩을 기본으로 하고 있다.
  - 웹 서버는 대표적인 멀티 스레드 응용 프로그램이다.
- 장점
  - 시스템 자원 소모 감소 (자원의 효율성 증대)
    - 프로세스를 생성하여 자원을 할당하는 시스템 콜이 줄어들어 자원을 효율적으로 관리할 수 있다.
  - 시스템 처리량 증가 (처리 비용 감소)
    - 스레드 간 데이터를 주고 받는 것이 간단해지고 시스템 자원 소모가 줄어들게 된다.
    - 스레드 사이의 작업량이 작아 Context Switching이 빠르다.
  - 간단한 통신 방법으로 인한 프로그램 응답 시간 단축
    - 스레드는 프로세스 내의 Stack 영역을 제외한 모든 메모리를 공유하기 때문에 통신의 부담이 적다.
- 단점
  - 주의 깊은 설계가 필요하다.
  - 디버깅이 까다롭다.
  - 단일 프로세스 시스템의 경우 효과를 기대하기 어렵다.
  - 다른 프로세스에서 스레드를 제어할 수 없다. (즉, 프로세스 밖에서 스레드 각각을 제어할 수 없다.)
  - 멀티 스레드의 경우 자원 공유의 문제가 발생한다. (동기화 문제)
  - 하나의 스레드에 문제가 발생하면 전체 프로세스가 영향을 받는다.

참고. 멀티 프로세스 vs 멀티 스레드

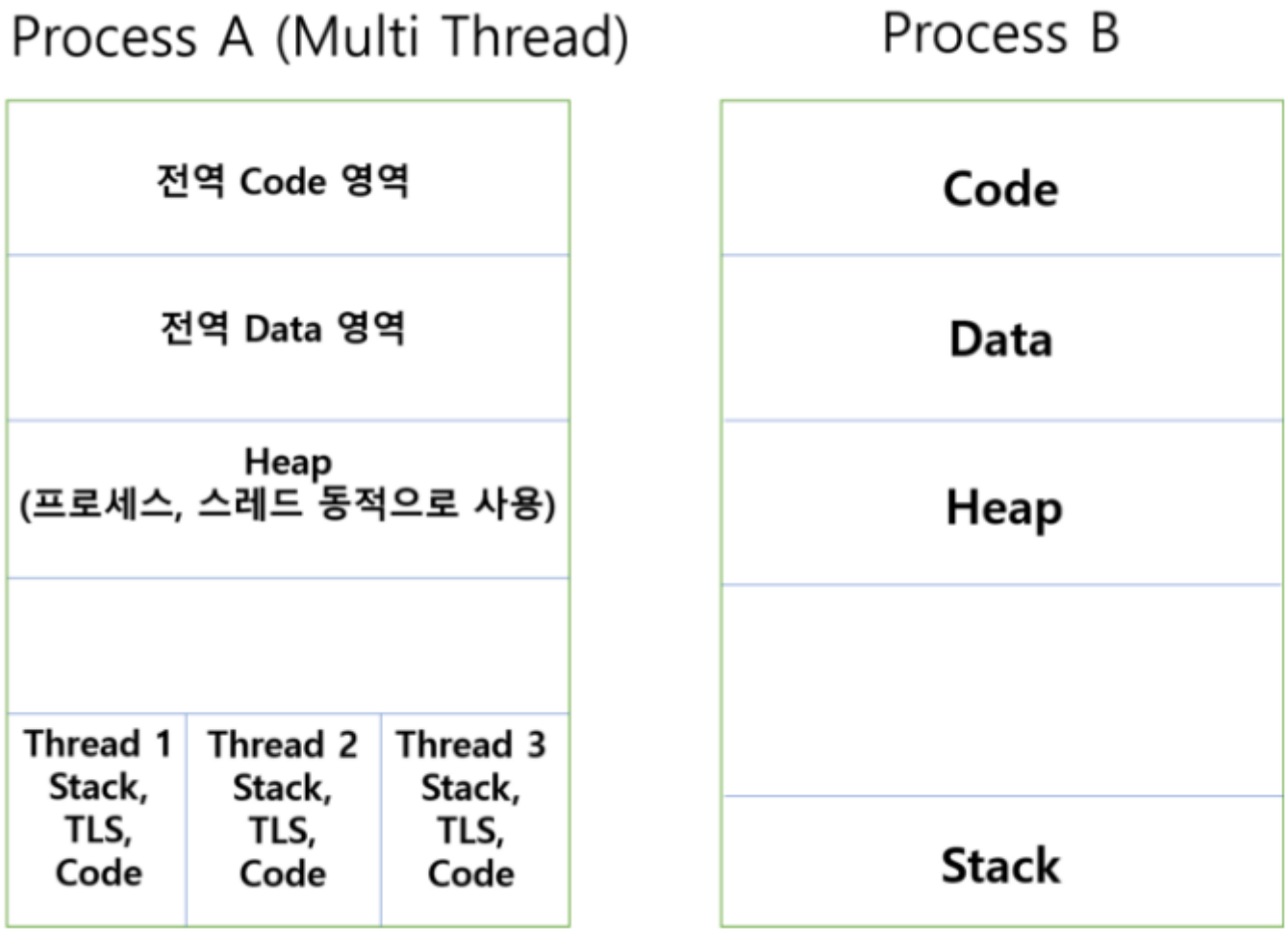
멀티 프로세스 대신 멀티 스레드를 사용하는 이유?

- 멀티 프로세스 대신 멀티 스레드를 사용하는 것의 의미?
  - 쉽게 설명하면, 프로그램을 여러 개 키는 것보다 하나의 프로그램 안에서 여러 작업을 해결하는 것이다.

- 여러 프로세스(멀티 프로세스)로 할 수 있는 작업들을 하나의 프로세스에서 여러 스레드로 나눠가면서 하는 이유?
  - i. 자원의 효율성 증대
    - 멀티 프로세스로 실행되는 작업을 멀티 스레드로 실행할 경우, **프로세스를 생성하여 자원을 할당하는 시스템 콜이 줄어들어** 자원을 효율적으로 관리할 수 있다.
    - -> 프로세스 간의 Context Switching시 단순히 CPU 레지스터 교체 뿐만 아니라 RAM과 CPU 사이의 캐쉬 메모리에 대한 데이터까지 초기화되므로 오버헤드가 크기 때문
    - 스레드는 프로세스 내의 메모리를 공유하기 때문에 독립적인 프로세스와 달리 스레드 간 데이터를 주고 받는 것이 간단해지고 시스템 자원 소모가 줄어들게 된다.
  - ii. 처리 비용 감소 및 응답 시간 단축
    - 또한 프로세스 간의 통신(IPC)보다 스레드 간의 통신의 비용이 적으므로 작업들 간의 통신의 부담이 줄어든다.
    - -> 스레드는 Stack 영역을 제외한 모든 메모리를 공유하기 때문
    - 프로세스 간의 전환 속도보다 스레드 간의 전환 속도가 빠르다.
    - -> Context Switching시 스레드는 Stack 영역만 처리하기 때문

- 주의할 점!
  - **동기화 문제**
  - 스레드 간의 자원 공유는 전역 변수(데이터 세그먼트)를 이용하므로 함께 상용할 때 충돌이 발생할 수 있다.

참고. 멀티 프로세스 vs 멀티 스레드



스케줄링 단위가 Thread이면 context Switching시 TCB만 바꾸면 됨

RTOS에서 말하는 Task는 어떻게 보면 Thread와 같은 개념으로 보면 되는가?

참고. 멀티 프로세스 vs 멀티 스레드

차이	프로세스	스레드
자원 할당 여부	실행 시마다 새로운 자원을 할당	자신을 실행한 프로세스의 자원을 공유
자원 공유 여부	일반적으로 자원을 공유하지 않는다. 같은 프로그램의 프로세스일 경우 코드를 공유하기는 한다.	같은 프로세스 내 스레드들은 스택을 제외한 나머지 세 영역을 공유한다.
독립성 여부	일반적으로 독립적	일반적으로 프로세스의 하위 집합
주소 소유 여부	별개의 주소 공간을 갖는다	주소 공간을 공유한다.
통신 여부	오직 시스템이 제공하는 IPC 방법으로만 통신	공유 변수 수정 등 자유롭게 다른 스레드와 소통
Context Switch	일반적으로 프로세스보다 스레드의 Context Switching이 더 빠를 수 있다.	하지만 상황에 따라 그렇지 않을 수도 있다.

3. Timer/Counter

1. 타이머/카운터란?

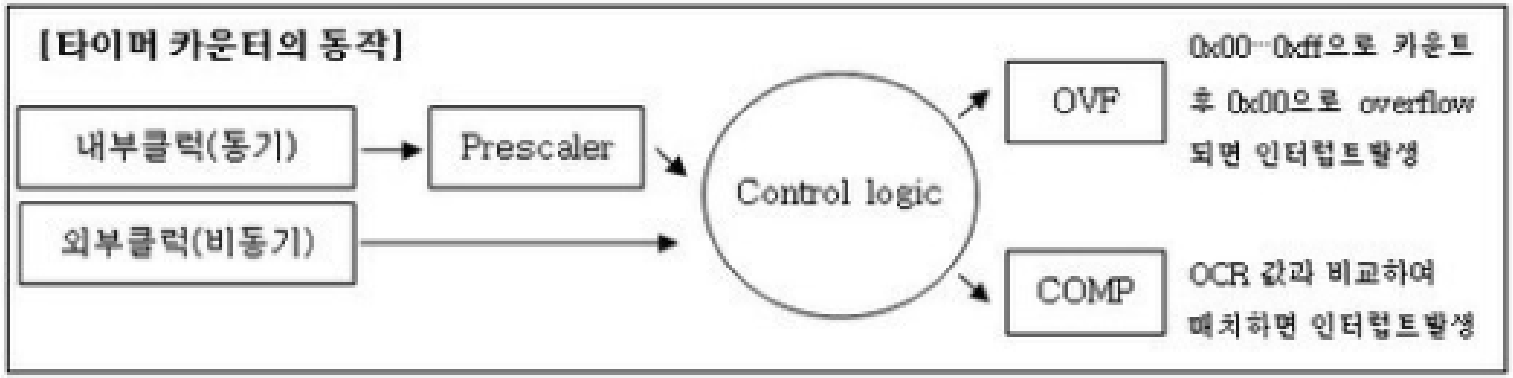
타이머: 내부 클럭을 이용하여 일정시간 간격의 펄스를 만들어 내거나 일정시간 경과 후에 인터럽트를 발생시키는 기능을 말함

카운터: 외부핀(TOSC1, TOSC2, T1, T2, T3)을 통해서 들어오는 펄스를 계수하여 Event Counter로서 동작되는 것을 말함

입력소스가 다르긴 한데 사용목적과 결과가 같은 이유로 사용되기 때문에 혼용되어 불린다.

타이머 – 내부클럭(빠름/분주가능:범위 내에서 클럭선택가능) – 동기모드

카운터 – 외부클럭(느림/분주불가능:외부클럭 그대로 사용) – 비동기모드



감사합니다.