타이머

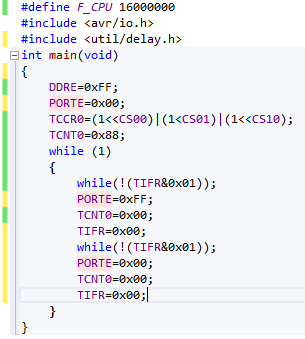
이장부터 사용되는 레지스터의 수가 많이 늘어난다. 또한 이 레지스터를 사용하기 위해서는 각 비트마다의 기능을 알아둬야 되는데 외우는것도 가능하긴 하겟지만 외우기 보단 가급적이면 데이터 시트를 읽고 어떤 기능인지를 파악할수 있는 능력을 기르는 것을 권장한다.

앞서 \_delay\_ms()라는 함수를 사용하고 넘어갔는데 이 함수는 마이크로 컨트롤러를 정확한 시간만큼 대기시키는 함수다. 이 기능은 마이크로 컨트롤러 내에 내장된 타이머 기능을 사용한다.

이 과정을 자세히 설명하면 다음과 같다. 마이크로 컨트롤러는 동작을 하기위해 외부에서 수정등을 통해 클럭을 제공받는다. 이 클럭은 주로 수정등을 통해 발진되는데 정밀한 발진주파수를 가지고 있기 때문에 이 클럭을 통해서 실제 시간을 측정할수 있다.

여기서 설명하는 마이크로 컨트롤러는 8비트 AVR인 ATmega128A을 기준으로 설명하는데 이 128A는 16Mhz의 수정을 통해 클럭을 공급받는다. 분주를 거치지 않으면 이 클럭의 주기는 약 1회에 0.0000000625초 이다. 이 주기만큼 세주면 원하는 시간을 계산을 할수있다. 만약 이 주기가 너무 빠르다면 분주기를 이용해서 원하는 수준으로 맞춰 줄수가 있다. 클럭의 횟수는 별도의 레지스터에 저장되며 이 레지스터가 꽉 차게 되면 이 레지스터가 꽉 찼다는 신호를 보내준다. 이를 통해 원하는 시간을 재때 확인할수 있다.

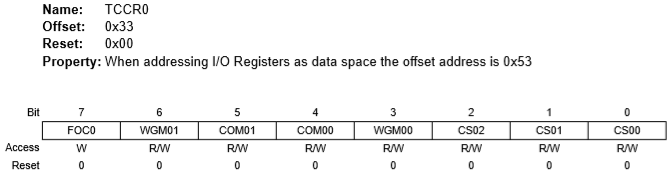
실제로 작성한 코드를 보면 이런식이다.



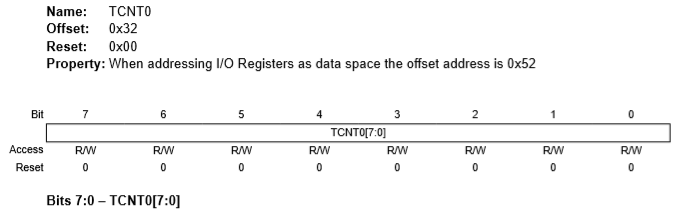
이 코드는 8비트 타이머를 이용해서 일정한 주기의 클럭을 발생 시키는 예제이다. 이 예제코드는 16Mhz클럭을 1024분주해서 약 60Hz의 클럭을 만들어낸다.

이 예제코드 서 다루고 있는 레지스터는 일단 3개정도로 제어용 레지스터인 TCCR0와 타이머 저장 레지스터인 TCNT0, 그리고 타이머 오버플로우 플래그 레지스터인 TIFR이 쓰였다.

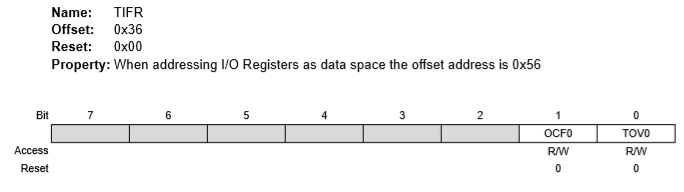
일단 이 예제에 쓰인 레지스터를 하나 하나씩 짚어보고 넘어가자.



TCCR은 Timer/Counter Control Register약자로 타이머/카운터를 제어한다는 의미를 가지고 있다. 각 비트 마다 이상한 단어가 들어가 있는데 이들은 각각의 비트가 의미하는 기능들을 의미한다.

* FOC0=특정 모드에서만 쓰는 제어 레지스터로 이장에서는 사용하기 않는다.
* WGM00,01=타이머의 기능을 설정해주는 레지스터로 레지스터에 입력된 값에 따라 다른기능으로써 동작하게 해준다.
* COM00,01=출력 비교기 관련 제어비트다.
* CS00,01,02=클럭 분주기. 클럭의 분주율,외부클럭 사용등 을 세팅 할수있다.

이렇게 타이머를 세팅하면 타이머 클럭을 기록하는 TCNT 레지스터가 1 주기마다 1 씩 올라간다. 8비트 타이머에서 TCNT레지스터는 255까지 기록되며 이 레지스터에서 오버 플로우가 발생되었을 경우 타이머 오버 플로우 인터럽트가 발생한다. TCNT레지스터는 직접 기록이 가능한 레지스터이기 때문에 타이머가 올라가기 전에 미리 TCNT레지스터에 미리 기록을 해둬서 원하는 수만큼의 클럭을 센뒤에 오버플로우 카운터가 올라가게 만들어서 원하는 주기만큼의 시간을 재는 것이 가능하다. 위의 코드에서는 이 오버플로우 인터럽트가 올라올라갈 때 while문을 이용해 대기하는데 여기에서 TIFR레지스터가 나온다. TIFR은 타이머 인터럽트 플래그 레지스터로 타이머에서 인터럽트 기능이 발동될 때 마다 해당되는 기능의 플래그를 올려준다. 8비트 타이머/카운터에서는 2개의 인터럽트 플래그가 존재하는데 이는 아래의 표에서 볼수있다.



* OCF= 아웃풋 캡처 플래그. 아웃풋 캡처 인터럽트가 발생할 경우 1이되는 플래그다
* TOV=타이머 오버플로우 플래그로 TCNT가 꽉 차서 오버플로우가 발생할 때 1이 되는 플래그다.

위의 예시에서는 타이머0의 인터럽트 부분만 다루고 있기 때문에 2비트만 보이나 실제로 TIFR레지스터는 타이머0 뿐만 아니라 다른 8비트 타이머와 16비트 타이머 또한 공유하고 있기 때문에 나머지 비트에는 해당 타이머에 할당된 플래그 비트가 있다. TIFR말고도 이렇게 레지스터를 공유하는 기능이 많기 때문에 데이터시트를 읽어서 정확히 어떤 비트에 어떤 기능이 있는지를 확인해 둬야 제대로 사용이 가능하다.

위의 코드에선 이 타이머 오버플로우 인터럽트가 발생 할 때 까지 while문을 이용해 대기하고 만약 TOV0플래그가 올라간 것이 확인되면 while 문을 탈출해서 while문 아래에 있는 내용들을 실행하는 구조를 가지고 있다. 이때 인터럽트 플래그가 올라오는 시간은 약 0.016초로 주파수로 따지면 약 61Hz가 된다. 인터럽트 플래그가 올라온 뒤에는 수동으로 직접 내려준다

지금까지 타이머의 기초적인 구성을 살펴보았다. \_delay\_ms()함수에 비하면 타이머 기능을 직접 이용하는 것이 무슨 이득이 있는가 생각할수 있지만 딜레이 함수가 기본으로 제공되지 않는 마이크로컨트롤러라면 타이머 기능을 이용해 직접 시간을 짜줘야 되는 만큼 꼭 알아둬야 되는 기능이며 아직 다루지 않은 타이머의 기능들 까지 포함하면 반드시 알아둬야 되는 기능이다.

만약 8비트 타이머의 타이머 레지스터가 너무 적다면 16비트 타이머의 레지스터를 쓰는것도 한가지 방법중 하나이다. 16비트 타이머는 8비트 타이머의 2배인 16비트 타이머/카운터 레지스터를 제공하기 때문에 더 긴 주기와 더 많은 시간을 측정할수 있다. 또한 일반적으로 16비트 타이머에 탑재되는 기능이 더 많기 때문에 16비트 타이머에 대해서 따로 공부해 놓는 것 또한 좋은 자세다.

주의할 사항이 몃가지가 있는데 모든 마이크로 컨트롤러가 같은 클럭을 사용하는 것은 아니기 때문에 자신이 사용하는 마이크로 컨트롤러의 주파수에 대한 이해가 필요하며 또 전압에 따라서 수정의 발진 주파수가 달라지는 경우의 수도 존재하는데 이경우 타이머의 측정결과가 영향을 받기 때문에 주의해야된다.