|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **예 비 보 고 서** | | | | |
| 학 과 | 학 년 | 학 번 | 조 | 성 명 |
| 전자공학과 | 3 | 12191505 | - | 윤수연 |
| 실험 제목 | Timer/Counter | | | |
| 1. **자료조사**    1. **Timer/Counter란**   Timer/Counter는 clock을 count하는 기능을 말한다. 이름은 나뉘어진 것 같지만 사실 같은 기능을 말한다. 만약 clock이 외부에서 들어오는 외부 clock이라면 clock에 할당되는 시간이 비교적 불규칙적일 수밖에 없고, 그런 경우 clock을 count한다. 이런 기능을 카운터 모드라고 한다. 반면 clock source가 내부의 clock신호라면 당연히 한 클럭 당 일정 시간이 소요되는 일정한 주기의 신호다. 이를 이용해 몇 번의 주기가 실제로 얼마가 걸리는지 계산할 수 있고 이러한 경우를 타이머모드라고 한다. 이때 정확한 시간을 측정하기 위해서는 주파수나 duty ratio정보가 있어야 한다. 아래 그림은 clock을 count하는 과정을 디지털 논리회로로 나타낸 예시 그림이다.  24. 카운터 (Counter)   * 1. **Timer/Counter의 동작 및 종류**   Timer/Counter 기능을 사용하기 위해서는 T2 pin, OC2pin 두 개의 핀을 사용해야한다. 아주 간단하게 입력 핀과 출력 핀 두 개를 사용한다. T2가 입력 펄스를 받는 역할을 하고 OC2가 출력을 내보낸다. 그런데 Timer/Counter는 clock을 count하는 기능인데 출력할 일이 무엇이 있을까? 바로 clock을 count하며 data를 가공하여 내보낸다. 예를 들면 count한 값이나 주파수나 duty ratio를 가공한 펄스파를 내보내는 등의 기능을 통해 출력이 이루어진다.  아래 표는 ATmega128의 Timer/Counter를 정리한 표다. Atmega128에는 4개의 Timer/Counter가 있는데, 아래 표에서 볼 수 있듯이 구조부터 8비트와 16비트로 완전히 다르고 관련 레지스터 또한 전혀 다르다. 우리는 이번 주 실험에서 가장 간단한 Timer/Counter인 Timer/Counter2에 대해 알아볼 것이다.  1회차 타이머 카운터0 : 네이버 블로그   * 1. **Timer/Counter2**   Timer/Counter2는 표에서도 확인할 수 있듯이 8비트의 가장 간단한 Timer/Counter다. 이 때 8비트는 입력 펄스의 최대 count수가 8bit, 즉 2^8라는 뜻이다. 따라서 이 Timer/Counter2의 경우 일반 모드에서 256번 count된 이후에는 overflow가 발생한다. 이를 이용해 인터럽트가 발생하는 주기나 인터럽트 발생 횟수 등을 계산할 수 있다. 또한 perscaler 분주비를 통해 Timer/Counter2로 입력되는 클럭도 계산할 수 있는데 여기서 prescaler 분주비는 시스템의 clock 주파수를 묶는 단위로, 이후 소개할 관련 레지스터로 설정할 수 있다.  Timer/Counter2의 경우 동작모드가 4가지로 나뉜다. 가장 먼저 우리가 이번 주차 실험에서 알아볼 Normal 모드는 가장 단순한 모드로 Timer/Counter2의 값이 0x00에서 0xFF가 되고 overflow가 일어나 다시 0x00이 되었을 때 overflow interrupt가 발생하며 ISR이 수행된다. 이 때 Normal 모드의 특징은 OC2핀으로 파형 출력이 일어나지 않는다.  Timer/Counter동작을 위해서는 앞서 말한 핀으로 내부와 외부 클럭 모두를 받을 수 있고 출력 파형 또한 생성할 수 있다. 내부 클럭의 경우 prescaler로 입력받는다. 또한 Timer/Counter2에 접근할 수 있는 관련 레지스터는 총 6개인데, 이 중 3개에 대해 자세히 알아보자.   * 1. **USART 제어 레지스터**  1. TCCR2   AVR(Atmega, Mega board) 배우기: Atmega128 Timer  : Timer/Counter 2 제어 레지스터. 초기 setting을 주로 담당한다.   * 비트7 : FOC2, 강제 출력 비교. PWM 모드가 아닌 경우, 출력을 담당한다. 이번 주차에서는 사용하지 않으므로 자세한 설명은 생략한다. * 비트6, 3 : 파형 발생 모드, Waveform Generation Mode를 설정한다. 일반 모드, CTC모드, PWM모드, 고속PWM모드가 있는데 네 개의 동작을 두 개의 비트로 표현한다. 일반 모드의 경우 00을 입력해준다. * 비트5, 4: 비교 일치 출력 모드, Compare Match Output Mode로, OC2 핀의 동작을 설정하나 일반모드에서는 사용하지 않는다. * 비트2~0 : CS22~CS20(클럭 선택, clock select로써, 타이머/카운터2에서 사용할 클럭의 분주비를 선택하는 데에 쓰인다. CS bit라고도 불린다. 구체적인 동작은 아래 표를 참고 바란다.   AVR - Timer/Counter2   1. TCNT2     : 타이머/카운터2의 8비트 카운터 값을 저장하는 레지스터로 언제나 읽기 및 쓰기 동작이 가능하다. 실질적으로 overflow가 일어나는지 관찰되는 레지스터라 그런 기능이 주로 쓰인다.   * Timer/Counter2 출력 비교 레지스터인 OCR2과 함께 진짜 overflow가 아니라 OCR2값과 TCNT2의 값이 같아졌을 때 인터럽트가 발생하도록 설정하기도 한다.  1. TIMSK     : 이름에서도 느껴지듯이 mask 레지스터다. Timer/Counter2의 인터럽트의 발생 여부를 개별적으로 허용하는 기능을 담당한다. 마스크 레지스터가 아무리 허가해도 상태 레지스터 SREG의 인터럽트 허가 비트가 1이 되어야 동작한다.   * 비트7 : OCIE2, 타이머/카운터2 출력 비교 인터럽트 허가 비트. 일반모드에서는 사용하지 않는다. * 비트6 : TOIE2, 타이머/카운터2 오버플로우 인터럽트 허가 비트.  1. 그 외 레지스터들     : 타이머/카운터 인터럽트 플래그 레지스터. Timer/Counter 0과 Timer/Counter2의 인터럽트가 발생하면 인터럽트의 상태를 저장한다.   * 비트7 : OCF2, 출력 비교 플래그 2. 타이머/카운터2에서 overflow가 발생하면 1로 set 되면서 overflow 인터럽트를 요청한다. * 비트6: TOV2, 타이머/카운터2 overflow 플래그. 타이머/카운터2의 TCNT2 레지스터와 출력 비교 레지스터 OCR2의 값을 비교하여 값이 같으면 다음 주기에서 이 비트가 1로 set되면서 인터럽트를 요청한다.  1. **실험**    1. **Timer/Counter구현**       1. **코드**   Timer/Counter는 LCD와 달리 특정 소자를 말하는 것이 아니라 기능을 일컫는 말이다. 그리고 ATmega128에는 Timer/Counter기능을 지원한다. 따라서 나는 헤더파일을 include한 것처럼 자연스럽게 지원하는 기능일 것이라고 생각했다. Timer/Counter기능이 ATmega128에 존재하는 것은 맞지만 코드에서도 그 기능을 위해 어느 정도 구현해주어야 하는 부분이 있다. 애초에 접근할 관련 레지스터가 따로 있고 레지스터에 구체적인 동작 트리거가 없었는 걸 공부했으니 헷갈릴 이유는 없지만 헷갈렸다. 아래는 매주 제공되는 실습 1단계에 대한 코드다.   |  | | --- | | #define F\_CPU 16000000  #include <avr/io.h>  #include <avr/interrupt.h>  // Normal Mode  unsigned char led = 0x00 ;  unsigned int count = 0;  ISR(TIMER2\_OVF\_vect){  count ++;  if (count == 60) {  if(led==0b10000000) {  led=0b00000001;  }  else {  led = led << 1;  }  count = 0;  }  }  int main(){  DDRA = 0xff ;  TCCR2 = 0b00000101 ; //프리스케일러 분주비 1024  TIMSK = 0b01000000 ; //오버플로우 인터럽트 허용  TCNT2 = 0 ; //타이머카운터0 초기화  SREG = 0x80 ; //전체 인터럽트 허가  led=0b00000001;  while(1){  PORTA=led;  }  } |   코드를 확인해보면 헤더파일을 include한 직후 전역 변수를 정의했다. 코드전체에 걸쳐 led소자의 출력을 control하고 일정하게 값을 유지할 led변수와 clock을 세는 count 변수가 정의되었다. 둘 다 자연수 값을 가지기 때문에 unsigned형으로 선언하여 메모리 효율을 높였다.  count변수는 Timer/Counter기능을 사용함에 있어 과연 빠질 수가 있을까? Timer/Counter기능을 생각해봤을 때 없어서는 안되는 변수다. 하지만 사용자가 하나하나 선언해주어야 한다는 점에서 새삼 기계어에 가까운 코드라고 느꼈다. MPU가 임베디드 등으로 사용될 때 무거운 언어를 사용할 수 없기 때문에 어쩔 수 없는 부분이라고 생각한다.   * 1. **실험 예상**      1. **실습 1단계**   타이머/카운터2 일반모드를 이용하여 시간을 측정하고, 1초마다 LED가 하나씩 번갈아 켜지는 프로그램 작성  LED가 번갈아 켜지는 프로그램은 이제껏 가장 많이 본 유형인만큼 간단하게 구현할 수 있을 것 같다. 물론 Timer/Counter로 구현하는 것은 처음이지만 레지스터의 값을 잘 설정하고 나면 interrupt도 구현해본 적이 있으니 금방 할 수 있을 것 같다.   * + 1. **실습 2단계**   LCD와 연동하여 스탑워치 제작  • 버튼1: 시작, 버튼2: 정지, 버튼3: 초기화  • LCD는 2-라인 모두 사용 (아래 그림 참조), 시/분/초 단위로 출력  실습 1단계에서 LCD소자를 추가하고 누적되는 count에 맞추어 스탑워치를 만드는 단계다. Overflow가 일어날 때가 실제로 몇 초일지는 모르겠지만 스탑워치가 count되는 시간을 현실의 1초와 최대한 비슷하게 만들어보고 싶다. 내부 clock소스를 사용하기 때문에 비교적 일정한 간격으로 만들 수 있을 것이다. 다만 시간 단위는 실험 도중 관찰하기 힘들 것 같다.   * + 1. **실습 3단계**   LCD와 연동하여 스탑워치/타이머 제작  • 초기화면에서 스탑워치/타이머 선택  • 시작/정지/초기화 구현, 시/분/초 단위 표시  실습 2단계에 이어 LCD소자를 활용하여 기능하는 단계다. 다른 점은 두 가지 기능을 둘 다 수행한다는 점인데 초기 화면에서 스탑워치와 타이머 둘 중에 한 기능을 선택한다는 점에서 지난 실험 중 LCD로 e book을 만드는 실험을 성공한 사람이 유리할 것 같다. | | | | |
|  | | | | |