실험제목 : 타이머/카운터 I (Timer/Counter I)

실험목적

ATmega328PB에 내장된 타이머/카운터(Timer/Counter)의 구조와 모드별 동작원리 및 제어방법을 이해하고, 이를 이용한 실험을 통해 마이크로 컨트롤러의 시간 개념에 대해 알아본다.

실험 준비물

Atmel Studio 7

Atmega328PB Xplained Mini

4-Digit SSD Module

5×7 Dot Matrix LED Display

실험에 필요한 예비지식

1. Timers/Counters

ATmega328PB에는 5개의 타이머/카운터가 내장되어 있다. 각각의 타이머/카운터에 대한 특징은 아래의 표와 같다.

	Width (bits)	ATmega328P	ATmega328PB
Timer/Counter0	8	○	○
Timer/Counter1	16	○	○
Timer/Counter2	8	○	○
Timer/Counter3	16	X	○
Timer/Counter4	16	X	○

2. Timer/Counter0

8-비트 Timer/Counter0 (TC0)는 모두 4가지의 동작 모드를 가지고 있으며, Timer/Counter Control Registers A(TCCR0A)와 Timer/Counter Control Registers B(TCCR0B)에 할당되어 있는 Waveform Generation mode bits (WGM02, WGM01 및 WGM00)와 Compare Output mode bits (COM01 및 COM00)에 의해 동작 모드가 결정된다. 각 동작 모드에 대한 자세한 설명은 다음과 같다.

1) Normal Mode (WGM[02:00] = 0b000)

① TCNT0는 항상 0부터 255까지 상향 계수를 한 후, 다시 0으로 돌아간다.

② TCNT0가 255에서 0으로 돌아가면 Overflow Flag인 TOV0가 1로 세트된다.

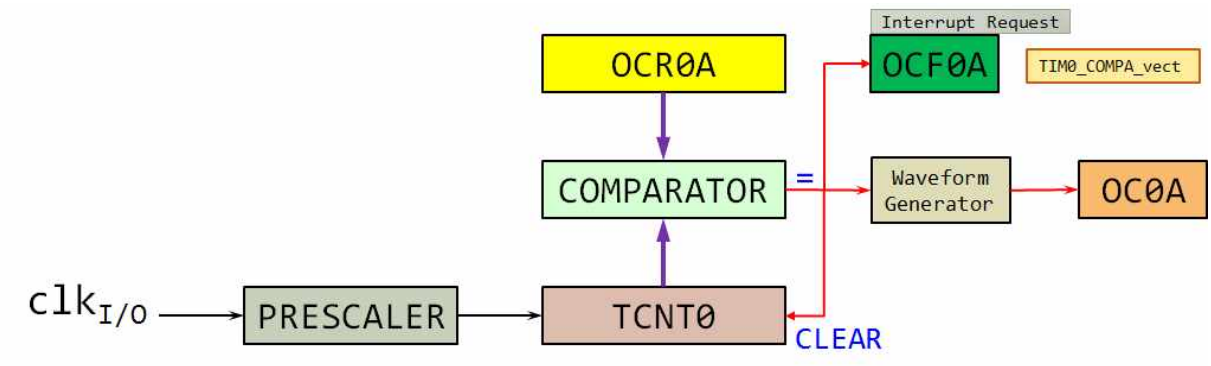
③ 만일 TC0 Overflow Interrupt를 사용한다면, TC0 Overflow Interrupt Service Routine으로 분기할 때 이 Overflow Flag인 TOV0 비트는 자동적으로 0으로 클리어 된다.

2) Clear Timer on Compare Match (CTC) Mode (WGM[02:00] = 0b010)

① TCNT0는 항상 0부터 상향 계수를 시작하며, 그 값이 OCR0A의 값과 일치하면 다시 0으로 돌아간다.

② 일치하는 순간에 OCF0A Flag가 1로 세트되며, 해당 인터럽트가 활성화 되어 있으면 인터럽트가 발생한다.

③ COM0A[1:0]=0b01로 설정하면 TCNT0와 OCR0A의 값이 일치하는 순간에 OC0A 핀으로 출력되는 값을 토글시킬 수 있으며, 이 기능을 사용하여 원하는 주파수를 가진 구형파 신호를 얻을 수 있다.

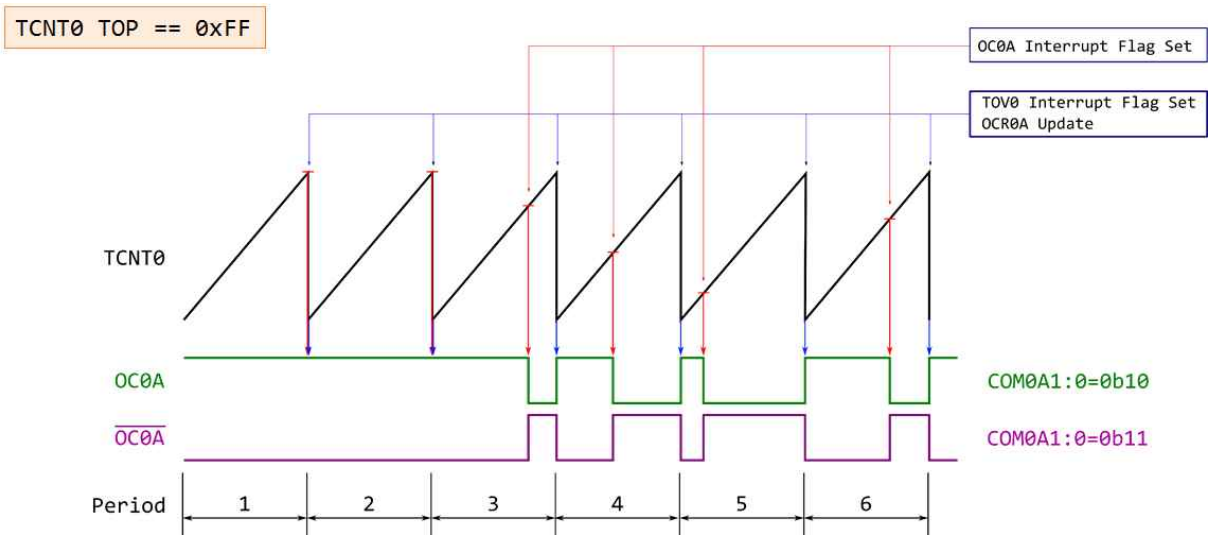


3) Fast PWM Mode (WGM[02:00] = 0b011 or WGM[02:00] = 0b111)

① TCNT0는 0부터 0xFF(WGM[02:00] = 0b011) 혹은 OCR0A(WGM[02:00] = 0b111) 값까지의 계수를 반복한다.

② 비반전(Non-Inverting) 모드 일 때, OC0A/B 출력은 0≤TCNT0≤OCR0A/B인 동안 논리값 '1'이 출력되고, OCR0A/B≤TCNT0≤TOP인 동안 논리값 '0'이 출력된다.

③ 반전(Inverting) 모드 일 때, 에는 비반전 모드일 때, OC0A/B 출력은 0≤TCNT0≤OCR0A/B인 동안 논리값 '0'이 출력되고, OCR0A/B≤TCNT0≤TOP인 동안 논리값 '1'이 출력된다.

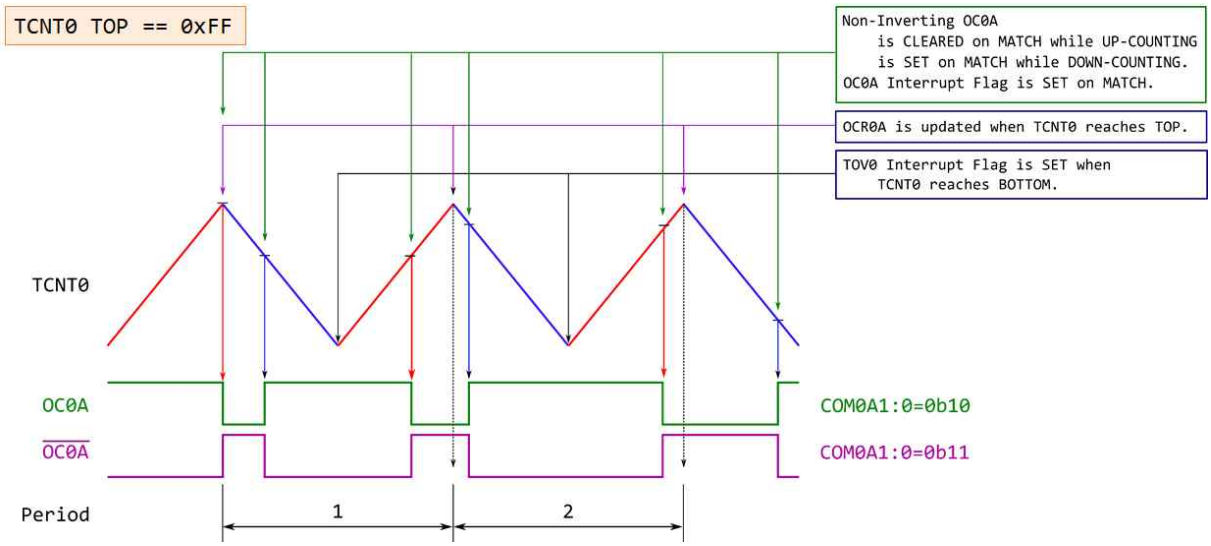


④ TCNT0의 계수값이 TOP에 도달하는 순간에 Timer/Counter Overflow Flag (TOV0)가 1로 세트되고, 해당 인터럽트가 활성화 되어 있으면 인터럽트가 발생하며, 이 Interrupt Service Routine에서 비교값을 갱신할 수 있다.

4) Phase Correct PWM Mode (WGM[02:00] = 0b001 or WGM[02:00] = 0b101)

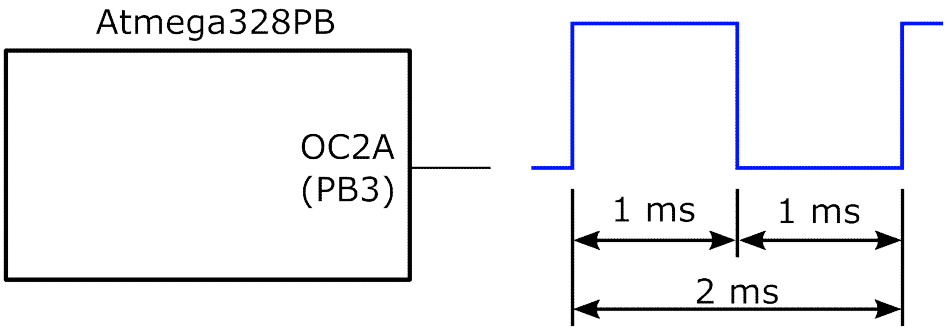
① TCNT0는 0부터 0xFF(WGM[02:00] = 0b001) 혹은 OCR0A(WGM[02:00] = 0b101) 값까지의 계수를 반복한다.

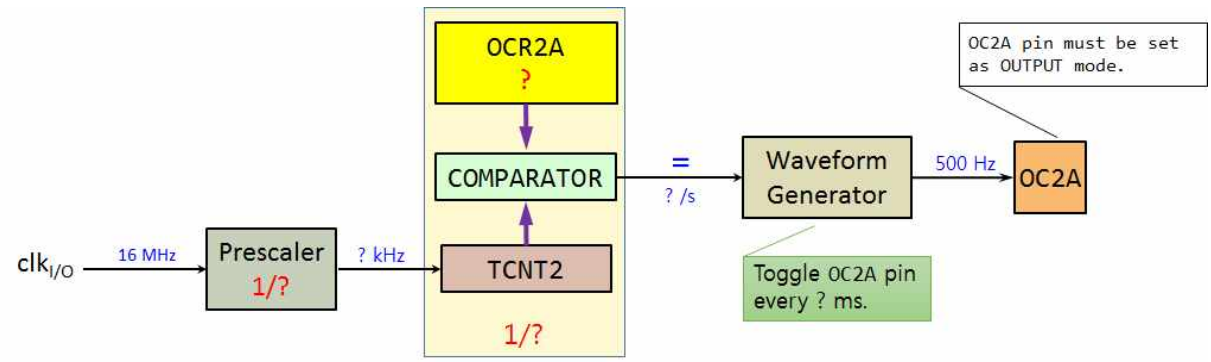
② 반전(Inverting) 모드 및 비반전(Non-Inverting) 모드 일 때의 OC0A/B 출력은 다음 그림과 같이 출력된다.



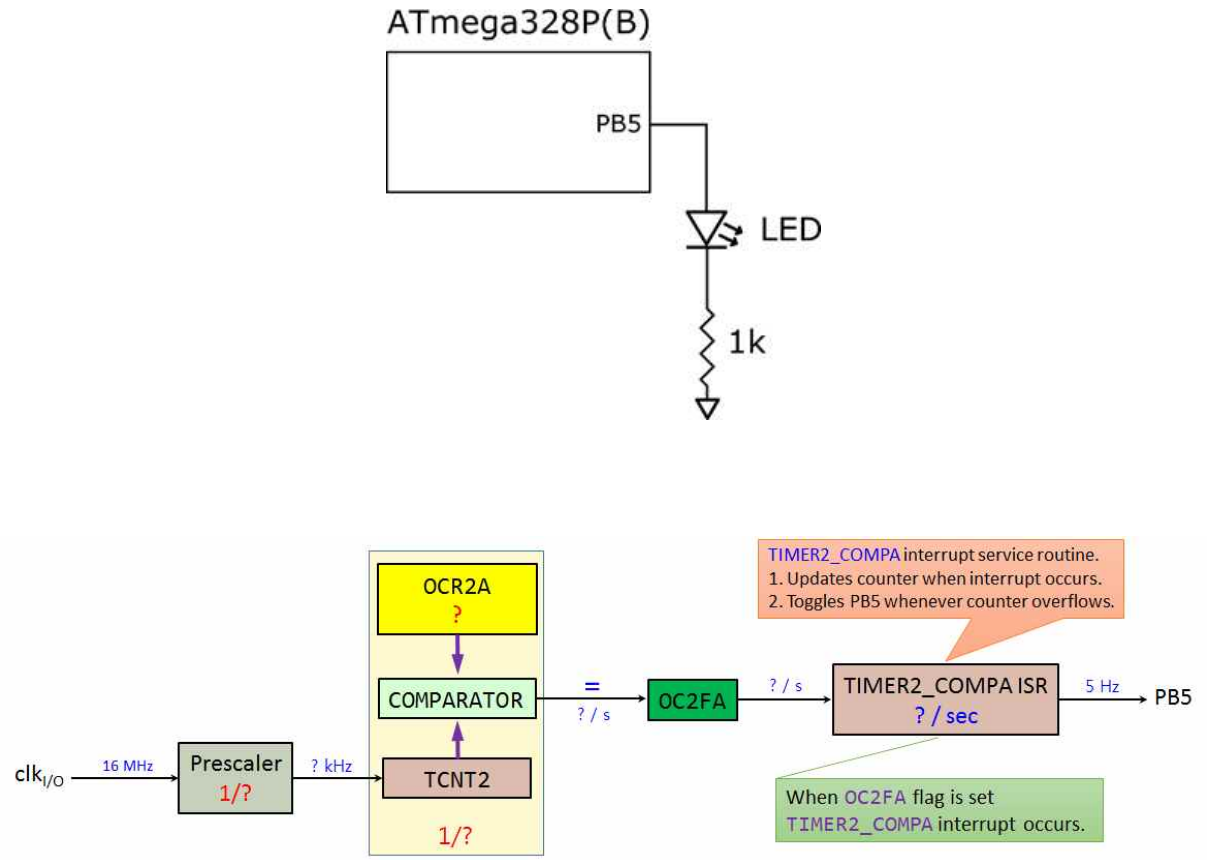
실험 내용

1. ATmega328PB에 내장된 8 비트 Timer/Counter2의 CTC Mode를 이용하여 OC2A(PB3)핀으로 500 Hz의 구형파가 출력되도록 프로그램을 작성하시오. 단, 시스템 클럭 주파수는 16 MHz이다.
   이 방식으로 얻을 수 있는 최고 주파수와 최저 주파수는 얼마인가? 각각에 대해 프로그램을 작성하고 oscilloscope를 통해 확인하시오.





2. ATmega328PB에 내장된 Timer/Counter2의 CTC Mode와 Timer/Counter2 Output Compare Match A Interrupt를 이용하여 PB5에 연결된 LED가 매 100 ms마다 토글 되도록 프로그램을 작성하시오. 단, 시스템 클럭 주파수는 16 MHz이다.



3. 시분할 방식에 의한 4-Digit SSD 표시 제어

1) Four SSD Lab Shield는 아래의 <그림 10>과 같이 4-Digit SSD module (S-5462CSR2/C) 및 ULN2803 transistor array가 하나의 Board에 연결되어 있으며 ATmega328PB X-mini board의 커넥터에 직접 연결하여 사용할 수 있다.

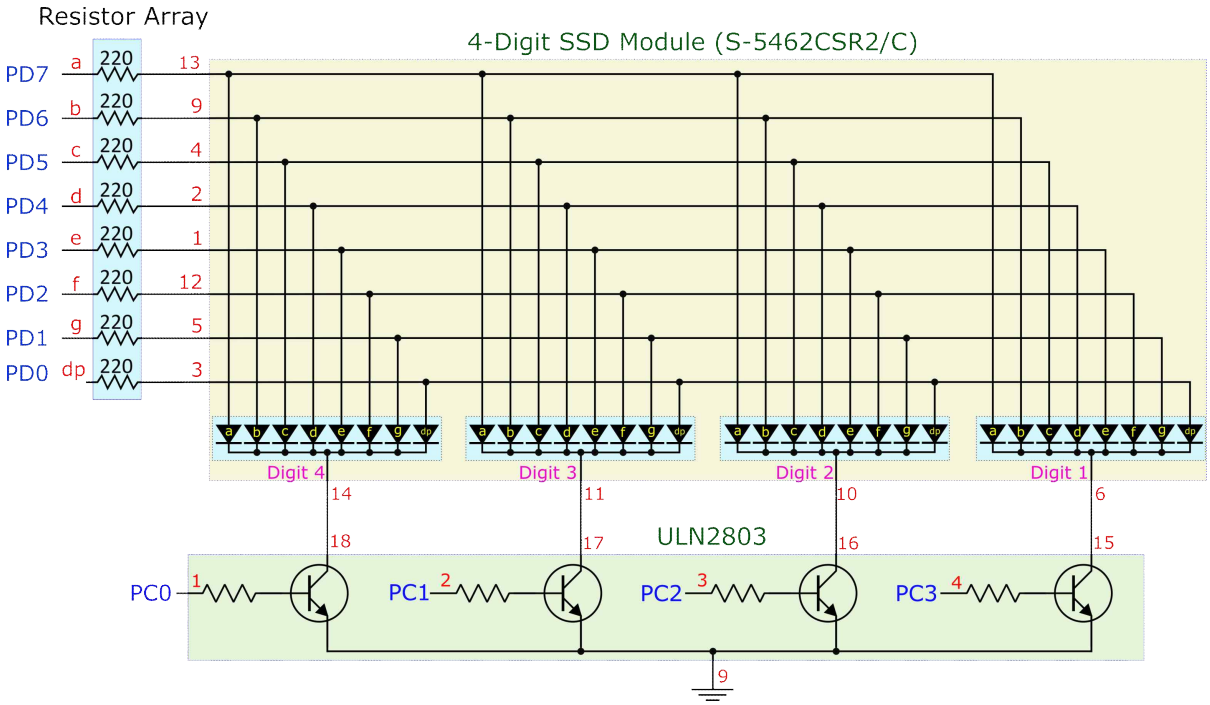


그림 10. 4-Digit SSD Module 연결도

2) Mod-10 Counter 구현 (4-Digit SSD module의 이해)

(1) <그림 10>을 참고하여 `main()` 함수의 시작 부분에서 숫자 '0' 부터 '9' 를 Seven Segment Display(SSD)에 표시할 수 있는 배열 `font[10]`를 정의하시오.

```
uint8_t font[10] = {
                /* abcdefg. */
                 0b11111100,          // 0
                   ...
                 0b11110110};         // 9
```

그림 11. SSD font

(2) `main()` 함수의 `font[10]` 선언부에 이어 `unsigned char` 또는 `uint8_t` 형의 변수 `mod_10_counter`를 선언하고 그 값을 0으로 초기화 하시오.

```
uint8_t ____________________;
```

그림 12. `mod_10_counter` 변수의 선언 및 초기화

(3) 이어서 ATmega328PB의 GPIO에 대한 입출력 모드를 설정하시오. 즉, Port D의 `PD[7:0]`와 Port C의 `PC[3:0]`의 입출력 모드를 설정하시오.

```
// Initialize PORTs
DDRD  _____________;    // Set PD[7:0] to OUTPUT mode
PORTD _____________;    // Clear PD[7:0]

DDRC _____________;     // Set PC[3:0] to OUTPUT mode
PORTC _____________;    // Keep PC[7:4] and Clear PC[3:0] -> Clear all SSDs
```

그림 13. GPIO 초기화

(4) `main()` 함수의 `while` loop 안에서 `_delay_ms()` 함수를 사용하여 매 500 msec 마다 변수 `mod_10_counter`의 값을 1 씩 증가시키시오. 단, 이 변수의 값이 10이 되면 0으로 되돌린다.

```
while (1)
{
   _delay_ms(500);
   ____________________________;  // Increment mod_10_counter by 1
   ____________________________;  // Reset mod_10_counter if it equals to 10
      ...
}
```

그림 14. 매 500 msec마다 `mod_10_counter` 변수의 증가

(5) 증가된 변수 mod_10_counter의 값을 4-Digit SSD module의 Digit 1에만 표시하시오. 즉, Digit 1을 제외한 Digit 2, Digit 3 및 Digit 4는 모두 소등되어 있어야 한다. 또한, Digit 1의 표시가 매 0.5 초마다 1씩 증가하여야 한다.

```
while (1)
{
   _delay_ms(500);
   ____________________________;  // Increment mod_10_counter by 1
   ____________________________;  // Reset mod_10_counter if it equals to 10

   PORTD = ____________________;  // Output font data
   PORTC ____________________;  // Display mod_10_counter on Digit 1 SSD
}
```

그림 15. mod_10_counter의 값을 Digit 1에만 표시

(6) 위 (5)항의 실험을 성공했으면 변수 mod_10_counter의 값을 4-Digit SSD module의 Digit 2에만 표시하시오

```
while (1)
{
   _delay_ms(500);
   ____________________________;  // Increment mod_10_counter by 1
   ____________________________;  // Reset mod_10_counter if it equals to 10

   PORTD = ____________________;  // Output font data
   PORTC ____________________;  // Display mod_10_counter on Digit 2 SSD
}
```

그림 16. mod_10_counter의 값을 Digit 2에만 표시

(7) 위 (6)항의 실험을 성공했으면 변수 mod_10_counter의 값을 4-Digit SSD module의 Digit 3에만 표시하시오.

```
while (1)
{
   _delay_ms(500);
   ____________________________;  // Increment mod_10_counter by 1
   ____________________________;  // Reset mod_10_counter if it equals to 10

   PORTD = ____________________;  // Output font data
   PORTC ____________________;  // Display mod_10_counter on Digit 3 SSD
}
```

그림 17. mod_10_counter의 값을 Digit 3에만 표시

(8) 위 (7)항의 실험을 성공했으면 변수 mod_10_counter의 값을 4-Digit SSD module의 Digit 4에만 표시하시오.

```
while (1)
{
   _delay_ms(500);
   ____________________________;  // Increment mod_10_counter by 1
   ____________________________;  // Reset mod_10_counter if it equals to 10

   PORTD = ____________________;  // Output font data
   PORTC ____________________;  // Display mod_10_counter on Digit 4 SSD
}
```

그림 18. mod_10_counter의 값을 Digit 4에만 표시

이 과정을 통해 4-Digit SSD module을 어떻게 구동해야 하는지를 배웠다.

3) 네 자리 숫자값('1234')을 표시 (**새로운 프로젝트에서 구현**)

(1) `main()` 함수에 위 2)의 (3)에서 구현한 방법과 동일한 방법으로 ATmega328PB의 GPIO에 대한 입출력 모드를 설정하시오.

```
// Initialize PORTs
DDRD  _____________;    // Set PD[7:0] to OUTPUT mode
PORTD _____________;    // Clear PD[7:0]

DDRC _____________;    // Set PC[3:0] to OUTPUT mode
PORTC _____________;    // Keep PC[7:4].  and Clear PC[3:0] -> Clear all SSDs
```

그림 19. GPIO 초기화

(2) ATmega328PB의 Timer/Counter0를 CTC 모드로 동작시켜 매 5 msec마다 Compare Match A 인터럽트가 발생하도록 설정한다.

```
// Initialize Timer0 to generate Compare Match A interrupt every 5 msec
TCCR0A = _____________;        // Set Timer0 to CTC mode
TCCR0B = _____________;        // Set Prescaler
OCR0A  = _____________;        // Set OCR0A

TIMSK0 = _____________;        // Enable Output Compare Match A Interrupt
_____________;                 // Enable global interrupt
```

그림 20. Compare Match A 인터럽트 설정

(3) `main()` 함수의 `while` loop 안에서는 아무 일도 하지 않는다.

```
while (1)
{
        // Do nothing
}
```

그림 21. Compare Match A 인터럽트 설정

(4) Timer0의 Compare Match A 인터럽트 처리 루틴(ISR)을 구현한다. ISR의 시작 부분에 위 2)의 (1)에서 선언한 것과 동일한 font[10]를 정의한다.

```
ISR(TIMER0_COMPA_vect)
{
    static uint8_t font[10] = {
                /* abcdefg. */
                 0b11111100,          // 0
                   ...
                 0b11110110};         // 9
        ...
}
```

그림 22. Compare Match A 인터럽트 처리 루틴 구현 및 font[10] 정의

(5) font[10] 선언부에 이어 표시할 Digit의 위치를 나타내는 static unsigned char 또는 static uint8_t형의 변수 position을 선언하고 그 값을 0으로 초기화 하시오.

```
ISR(TIMER0_COMPA_vect)
{
   static uint8_t font[10] = {
        ...

   static uint8_t ___________________;
}
```

그림 23. potion 변수의 선언 및 초기화

(6) 매 5 msec마다 인터럽트가 발생하면 TIMER0_COMPA ISR이 호출되어 실행된다.

① 처음 인터럽트가 발생했을 때 제일 왼쪽 자리인 Digit 4의 자리에 '1' 을 표시한다. (position 변수의 값이 0)
② 두 번째 인터럽트가 발생하면 왼쪽에서 두 번째 자리인 Digit 3의 자리에는 '2' 를 표시한다. (position 변수의 값이 1)
③ 이어서 세 번째 인터럽트가 발생하면 왼쪽에서 세 번째 자리인 Digit 2의 자리에는 '3' 을 표시한다. (position 변수의 값이 2)
④ 네 번째 인터럽트가 발생하면 제일 오른쪽에 있는 Digit 1의 자리에는 '4' 를 표시한다. (position 변수의 값이 3)

따라서 매 인터럽트가 발생할 때마다 <그림 24>와 같이 ISR 내에서는 position 변수의 값에 따라 해당하는 자리에 해당하는 숫자를 표시해 주면 된다.

```
ISR(TIMER0_COMPA_vect)
{
   static uint8_t font[10] = {
        ...

   if (position == 3)
   {
      PORTD ____________________;       // Display '1' on Digit 4
      PORTC ____________________;
   }
   else if (position == 2)
   {
      PORTD ____________________;       // Display '2' on Digit 3
      PORTC ____________________;
   }
   else if (position == 1)
   {
      PORTD ____________________;       // Display '3' on Digit 2
      PORTC ____________________;
   }
   else
   {
      PORTD ____________________;       // Display '4' on Digit 1
      PORTC ____________________;
   }
      ...
}
```

그림 24. SSD 모듈에 '1234' 라고 표시

(7) position 변수의 값에 따른 숫자의 표시가 끝났으면 position 변수가 다음번 표시할 숫자의 위치를 가리키도록 이 변수의 값을 1 증가시킨 후, 이 값이 4가 되면 다시 0으로 되돌린다.

```
ISR(TIMER0_COMPA_vect)
{
   static uint8_t font[10] = {
        ...
   ____________________;        // Update position variable
   ____________________;
}
```

그림 25. position 변수가 다음번 표시할 숫자의 위치를 가리키도록 갱신

4) Mod-10000 카운터의 구현 (새로운 프로젝트에서 구현)

이 실험에서는 0부터 9,999까지 계수할 수 있는 mod-10000 카운터를 만들고, 이 카운터의 값을 앞에서 사용했던 시분할(time-division) 방식을 이용하여 4-Digit SSD Module(S-5462CSR2/C)에 표시하는 프로그램을 구현한다.

(1) `main()` 함수 시작 전에 `unsigned char` 또는 `uint8_t`형의 전역 변수 `count_buf[4]`를 선언한다.

```
unsigned char count_buf[4];

int main(void)
{
        ...
}
```

그림 26. 전역 변수 `count_buf[4]`를 선언

(2) `main()` 함수 안에 `unsigned int` 또는 `uint16_t`형의 변수 `mod10k_counter`를 선언하고 0으로 초기화한다.

```
int main(void)
{
   uint16_t ____________________;
        ...
}
```

그림 27. 변수 `mod10k_counter`를 선언하고 0으로 초기화

(3) mod10k_counter 변수 선언이 끝나면 ATmega328PB의 GPIO에 대한 입출력 모드를 설정한다.

```
int main(void)
{
        ...

   // Initialize PORTs
   DDRD  _____________;       // Set PD[7:0] to OUTPUT mode
   PORTD _____________;       // Clear PD[7:0]

   DDRC _____________;        // Set PC[3:0] to OUTPUT mode
   PORTC _____________;       // Keep PC[7:4]. Clear PC[3:0] -> Clear all SSDs
}
```

그림 28. GPIO 초기화

(4) 이어서 ATmega328PB의 Timer/Counter0를 CTC 모드로 동작시켜 매 5 msec마다 Compare Match A 인터럽트가 발생하도록 설정한다.

```
int main(void)
{
        ...

   // Initialize Timer0 to generate Compare Match A interrupt every 5 msec
   TCCR0A = _____________;           // Set Timer0 to CTC mode
   TCCR0B = _____________;           // Set Prescaler
   OCR0A  = _____________;           // Set OCR0A

   TIMSK0 = _____________;           // Enable Output Compare Match A Interrupt
   _____________;                    // Enable global interrupt
}
```

그림 29. Compare Match A 인터럽트 설정

(5) main() 함수의 while loop 안에서 변수 mod10k_counter의 값을 각각 천의 자리, 백의 자리, 십의 자리 및 일의 자리로 분리한다. 즉, 변수 mod10k_counter의 값이 1234라면 천의 자리는 1, 백의 자리는 2, 십의 자리는 3 그리고 일의 자리는 4가 된다. 이렇게 분리된 값을 전역 변수인 count_buf[]에 저장한다.

```
int main(void)
{
        ...

   while (1)
   {
      count_buf[3] = _____________;          // 1000의 자리 분리
      count_buf[2] = _____________;          // 100의 자리 분리
      count_buf[1] = _____________;          // 10의 자리 분리
      count_buf[0] = _____________;          // 1의 자리 분리
         ...
   }
}
```

그림 30. mod10k_counter의 값을 분리

(6) main() 함수의 while loop 안에서 변수 mod10k_counter의 값을 분리한 후에는 mod10k_counter의 값을 1 증가시키고, 이 값이 10,000에 도달했으면 0으로 되돌린다. 마지막으로 _delay_ms() 함수를 사용하여 10 msec 지연시킨다.

```
int main(void)
{
        ...

   while (1)
   {
         ...
      _____________;                  // Update mod10k_counter
      _____________;

      _delay_ms(10);
   }
}
```

그림 31. 매 10 msec마다 mod_10k_counter 변수의 증가

(7) Timer0의 Compare Match A 인터럽트 처리 루틴(ISR)을 구현한다. ISR의 시작 부분에 표시할 Digit의 위치를 나타내는 `static unsigned char` 또는 `static uint8_t`형의 변수 `position`을 선언하고 그 값을 0으로 초기화한다.

```
ISR(TIMER0_COMPA_vect)
{
   static uint8_t ____________________;  // position 변수의 선언 및 초기화
        ...
}
```

그림 32. Compare Match A 인터럽트 처리 루틴 구현과 potion 변수의 선언 및 초기화

(8) `position` 변수의 선언에 이어 위 2)의 (1)에서 선언한 것과 동일한 `font[10]`를 정의한다.

```
ISR(TIMER0_COMPA_vect)
{
        ...
   static uint8_t font[10] = {
      /* gfedcba */
      0b0111111,              // 0
        ...
      0b1101111};             // 9
        ...
}
```

그림 33. `font[10]` 정의

(9) 매 5 msec 마다 인터럽트가 발생하면 `TIMER0_COMPA` ISR이 호출되어 실행된다.

① 처음 인터럽트가 발생했을 때 제일 왼쪽 자리인 Digit 4의 자리에는 `count_buf[3]`에 저장되어 있는 값(1000의 자리)에 해당하는 `font` 데이터를 찾아 출력한다. (`position` 변수의 값이 0)
② 두 번째 인터럽트가 발생하면 왼쪽에서 두 번째 자리인 Digit 3의 자리에는 `count_buf[2]`에 저장되어 있는 값(100의 자리)에 해당하는 `font` 데이터를 찾아 출력한다. (`position` 변수의 값이 1)
③ 이어서 세 번째 인터럽트가 발생하면 왼쪽에서 세 번째 자리인 Digit 2의 자리에는 `count_buf[1]`에 저장되어 있는 값(10의 자리)에 해당하는 `font` 데이터를 찾아 출력한다. (`position` 변수의 값이 2)
④ 네 번째 인터럽트가 발생하면 제일 오른쪽에 있는 Digit 1의 자리에는 `count_buf[0]`에 저장되어 있는 값(1의 자리)에 해당하는 `font` 데이터를 찾아 출력한다. (`position` 변수의 값이 3)

따라서 매 인터럽트가 발생할 때마다 <그림 34>와 같이 ISR 내에서는 position 변수의 값에 따라 표시될 값을 count_buf[]에서 찾아 변수 number에 저장한 후, 이 number 값에 해당하는 font 데이터를 가져와 position 변수가 가리키는 자리에 표시해 주면 된다.

```
ISR(TIMER0_COMPA_vect)
{
      ...

   // position 변수에 해당하는 count_buf[]의 원소를 가져온다.
   uint8_t number = ____________________;

   // 이 number 변수에 해당하는 font 데이터를 가져와 출력한다.
   PORTD = ____________________;
   PORTC = ____________________;
      ...
}
```

그림 34. SSD 모듈에 '1234'라고 표시

(10) position 변수의 값에 따른 숫자의 표시가 끝났으면 position 변수가 다음번 표시할 숫자의 위치를 가리키도록 이 변수의 값을 1 증가시킨 후, 이 값이 4가 되면 다시 0으로 되돌린다.

```
ISR(TIMER0_COMPA_vect)
{
       ...
   ____________________;        // Update position variable
   ____________________;
}
```

그림 35. position 변수가 다음번 표시할 숫자의 위치를 가리키도록 갱신

모든 프로그램이 정상적으로 동작한다면 4-Digit SSD Module에는 10 msec마다 1씩 증가하는 네 자리 숫자가 표시되며, 이 값이 9999에 이른 후에는 0으로 되돌아가는 것을 관찰할 수 있다.

4. 시분할 방식에 의한 Dot Matrix LED Display 표시 제어

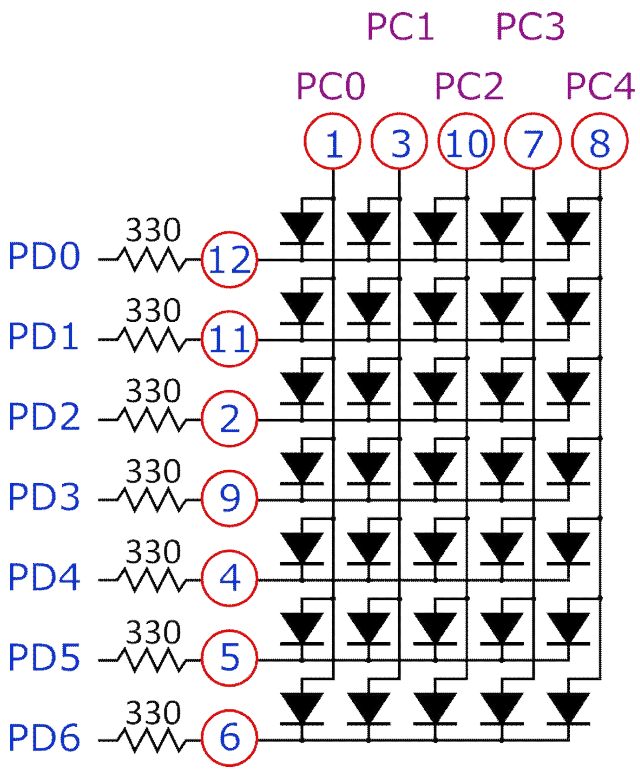
아래의 그림과 같이 회로를 구성하시오. ATmega328PB XMini의 PB7에 연결된 스위치를 누를 때마다 16진 카운터의 값을 1씩 증가시킨 후, 그 값이 아래의 그림에 나타낸 바와 같이 PD[6:0]와 PC[4:0]에 연결된 5×7 Dot Matrix LED Display에 표시되도록 프로그램을 작성하시오.

(1) 단, 소프트웨어에 의한 스위치 디바운싱(debouncing) 기법을 사용할 것.

(2) ATmega328PB에 내장된 Timer/Counter0의 CTC Mode와 Timer/Counter0 Output Compare Match A Interrupt를 이용하여 시분할(Time-division) 방식으로 5×7 Dot Matrix LED Display에 내용을 표시할 것.

(3) Refresh Rate를 200Hz로 할 것.

(4) 핀 배치도: 제품 옆면의 품명이 마킹되어 있는 쪽의 맨 외쪽부터 1번 핀이며, 반시계 방향으로 핀 번호가 증가함. (http://www.eleparts.co.kr/EPX37XMT 참조)



참고: 5×7 Fonts

```
uint8_t font[16][5] = {
        { 0x3E, 0x51, 0x49, 0x45, 0x3E },           // 0
        { 0x00, 0x42, 0x7F, 0x40, 0x00 },           // 1
        { 0x42, 0x61, 0x51, 0x49, 0x46 },           // 2
        { 0x21, 0x41, 0x45, 0x4B, 0x31 },           // 3
        { 0x18, 0x14, 0x12, 0x7F, 0x10 },           // 4
        { 0x27, 0x45, 0x45, 0x45, 0x39 },           // 5
        { 0x3C, 0x4A, 0x49, 0x49, 0x30 },           // 6
        { 0x01, 0x71, 0x09, 0x05, 0x03 },           // 7
        { 0x36, 0x49, 0x49, 0x49, 0x36 },           // 8
        { 0x06, 0x49, 0x49, 0x29, 0x1E },           // 9
        { 0x7E, 0x11, 0x11, 0x11, 0x7E },           // A
        { 0x7F, 0x49, 0x49, 0x49, 0x36 },           // B
        { 0x3E, 0x41, 0x41, 0x41, 0x22 },           // C
        { 0x7F, 0x41, 0x41, 0x22, 0x1C },           // D
        { 0x7F, 0x49, 0x49, 0x49, 0x41 },           // E
        { 0x7F, 0x09, 0x09, 0x01, 0x01 } };         // F
```

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

uint8_t mod_16_counter;

int main(void)
{
    // Initialize PORTs

    // Initialize Timer/Counter0

    // Enable Interrupts

    while (1)
    {
        // Update mod_16_counter whenever SW is pressed
    }
}

ISR(TIMER0_COMPA_vect)     // Timer/Counter0 Output Compare Match A ISR
{
    static uint8_t font[16][5] = {
        { 0x3E, 0x51, 0x49, 0x45, 0x3E },      // 0
                ...
        { 0x06, 0x49, 0x49, 0x29, 0x1E } };    // 9

    static uint8_t column_counter = 0;

    // Get a byte from the font[][] array and output it to PORTD

    // Output logic '1' to the next column

    // Update column_counter

}
```

보고서에 포함 시킬 내용

1. 위의 실험 가운데 **[3. 시분할 방식에 의한 4-Digit SSD 표시 제어]**에 관한 source program에 본인이 이해한 내용을 바탕으로 주석을 추가하여 제출할 것.

2. 모두 10자리의 SSD를 위 **[3. 시분할 방식에 의한 4-Digit SSD 표시 제어]**의 방법으로 구동할 때 필요한 GPIO 핀의 수는 얼마인가? 계산 근거를 함께 제출하시오.

3. ATmega328PB의 Timer/Counter를 CTC mode로 동작시킬 때 응용할 수 있는 분야를 나열하시오.