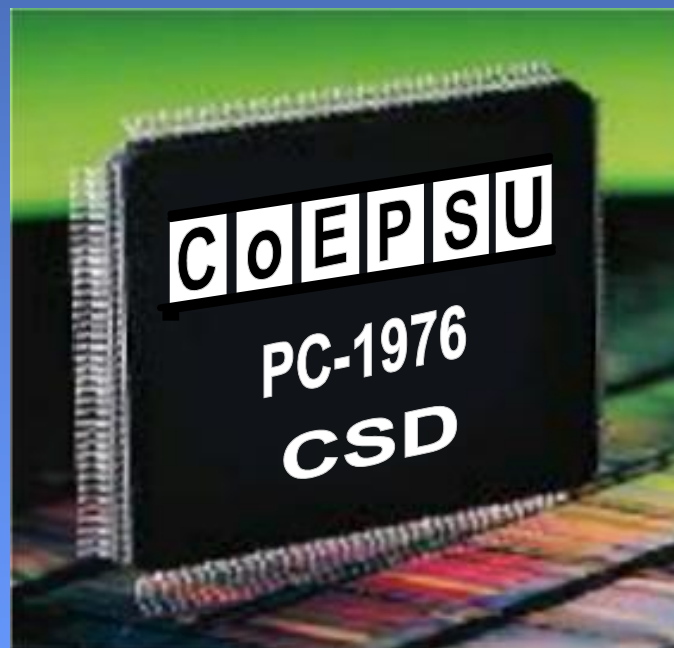


**Department of Computer Engineering**  
**Faculty of Engineering**  
**Prince of Songkla University**

**240-319**

# **Embedded System Developer Module**



March 23, 2023

*Associate Prof. Dr. Panyayot Chaikan*  
*[panyayot@coe.psu.ac.th](mailto:panyayot@coe.psu.ac.th)*



# Chapter 4

## Timer/Counter





# เนื้อหา

การจับเวลาโดยใช้ซอฟต์แวร์

การจับเวลาโดยใช้วงจรรนับ/จับเวลา

วงจรรนับ/จับเวลาหมายเลข 0 และหมายเลข 2 (ขนาด 8 บิต)

การเขียนโปรแกรมบริการการขัดจังหวะของวงจรรนับ/จับเวลา

วงจรรนับ/จับเวลาหมายเลข 1 (ขนาด 16 บิต)





# Chapter 4-part1

วงจรนับ/จับเวลา (Counter/Timer) ขนาด 8 บิต



# วิธีการจับเวลา



## ◆ ใช้ซอฟต์แวร์

- ◆ ไม่มีประสิทธิภาพ
- ◆ ควบคุมเวลาได้ยากหากเขียนด้วยภาษาซี
- ◆ เขียนโปรแกรมได้ค่อนข้างยากแม้จะใช้แอสเซมบลี

## ◆ ใช้วงจรนับ/จับเวลา

- ◆ ซีพียูไม่ต้องเสียเวลาดวนลูปหน่วงเวลาโดยเปล่าประโยชน์
- ◆ ประสิทธิภาพสูง
- ◆ สามารถพัฒนาได้ง่ายทั้งในภาษาแอสเซมบลีและซี

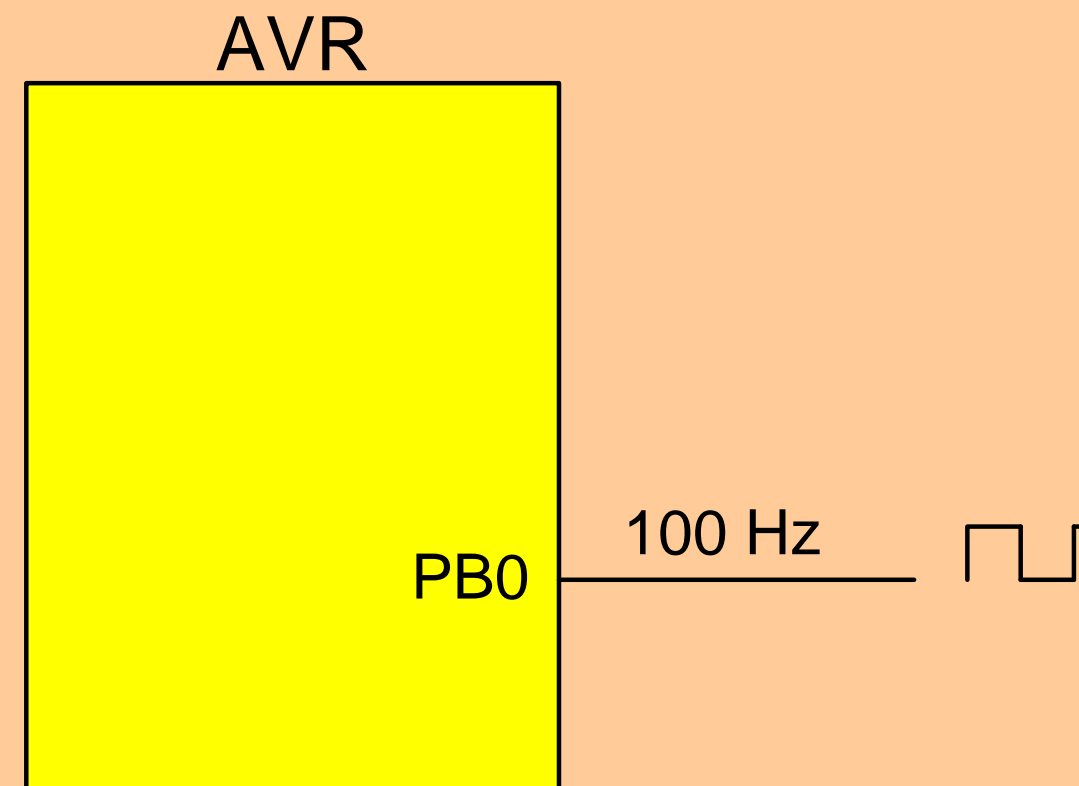


# ตัวอย่าง 10.1 การเขียนโปรแกรมวนซ้ำเพื่อจับเวลา

◆ จงเขียนโปรแกรมเพื่อวนซ้ำ

ให้ซีพียูส่งพัลส์ความถี่ 100 Hz

ออกสู่พอร์ต B บิต 0





# การเขียนโปรแกรมวนลูปเพื่อส่งพัลส์

- ◆ ความถี่ 100 Hz -> 1 คาบ เท่ากับ 10 mS
- ◆ จะต้องเขียนโปรแกรมวนซ้ำให้ชิพียูทำการ Toggle พอร์ต B บิต 0 ทุก ๆ 5 mS
- ◆ สมมุติให้ชิพียู AVR ที่ใช้มีความถี่ 8 MHz



# โปรแกรมวนซ้ำเพื่อส่งพัลส์

START:

```
ldi    var_B,0x00           ;1 clk
BIGLOOP: inc    VAR_B         ;39 clks

loop1:  ldi    VAR_A, 0x00     ;1 clk
        inc    VAR_A         ;255 clks
        cpi    VAR_A, 0xFF    ;255 clks
        brlo   LOOP1         ;(2*254)+1 clks

        Cpi    VAR_B,39       ;39 clks
        brlo   BIGLOOP        ;(38*2)+1 clks

        ;---    COM PORTB
        IN      R16, PORTB
        COM     R16
        OUT     PORTB,R16
        rjmp    START
```

} =1020 clks \* 39  
= 39,780 clks

คำสั่ง brlo ใช้

-2 clk หากมีการบรานซ์

-1 clk หากไม่มีการบรานซ์

- ◆ จำนวน clock ทั้งสิ้น 39,941 ลูก
- ◆ แต่ละลูกมีคาบเท่ากับ 125 nS (เมื่อซีพียูรันที่ 8 MHz)
- ◆ การหน่วงเวลาเกิดขึ้นเท่ากับ  $39,941 * 125 \text{ nS} = 4.99265 \text{ mS}$





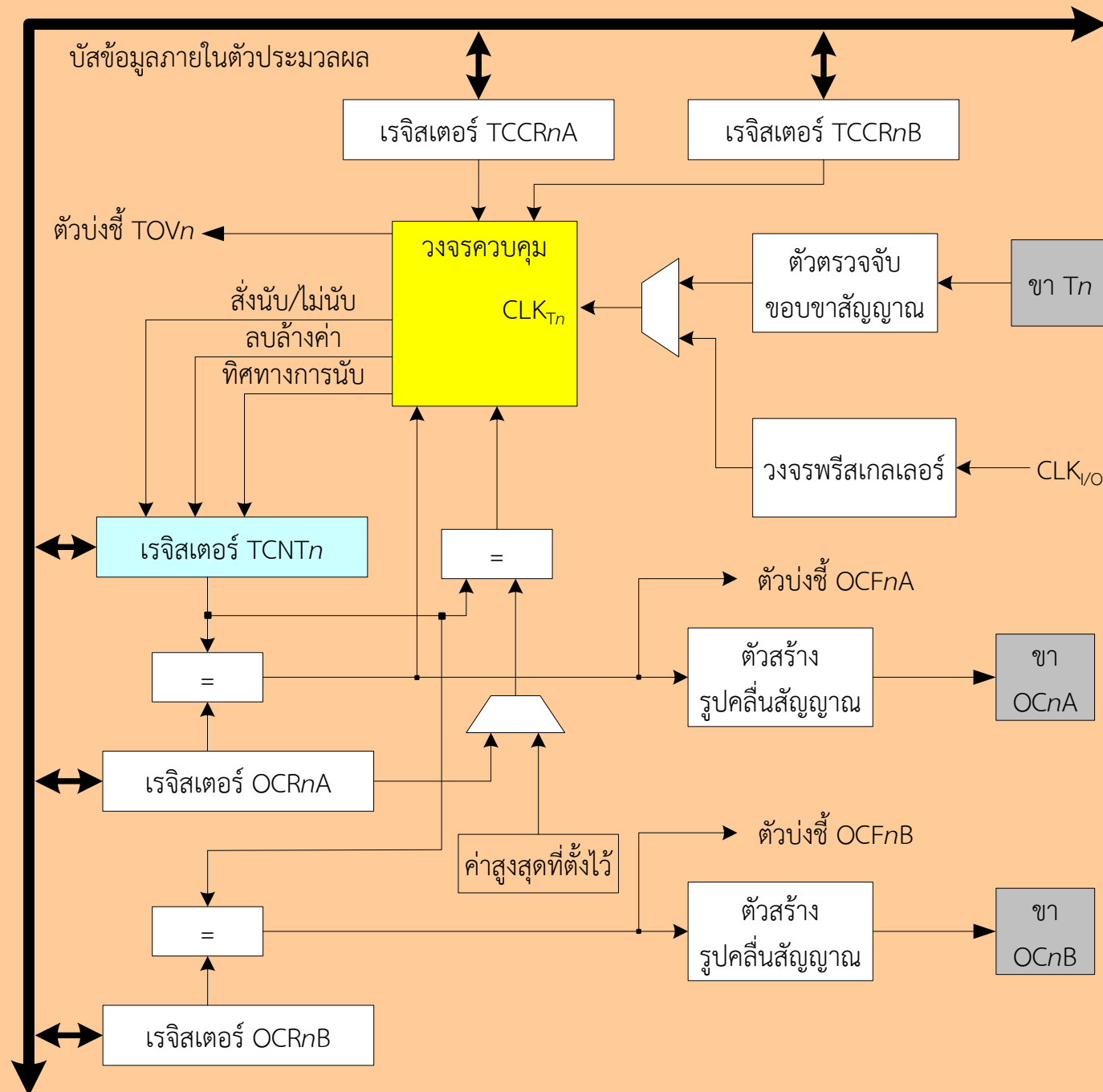
# วิธีที่ 2 - การสร้างพัลส์ด้วยวิธี Timer interrupt

```
int main (void)
{
    sei();
    TCNT0 = 155;
    //-----enable timer 0
    ...
    ...
    while(1)
    {
        function_forever();
    }
}
```

```
ISR(vector of timer 0)
{
    PORTB = ~portB;
    TCNT0 = 155;
    ...
    ...
}
```

ในตัวอย่างให้ทำการขัดจังหวะซีพียู  
ทุกๆ 151 clk

# วงจรรนับ/จับเวลาหมายเลข 0 (Timer/Counter0)





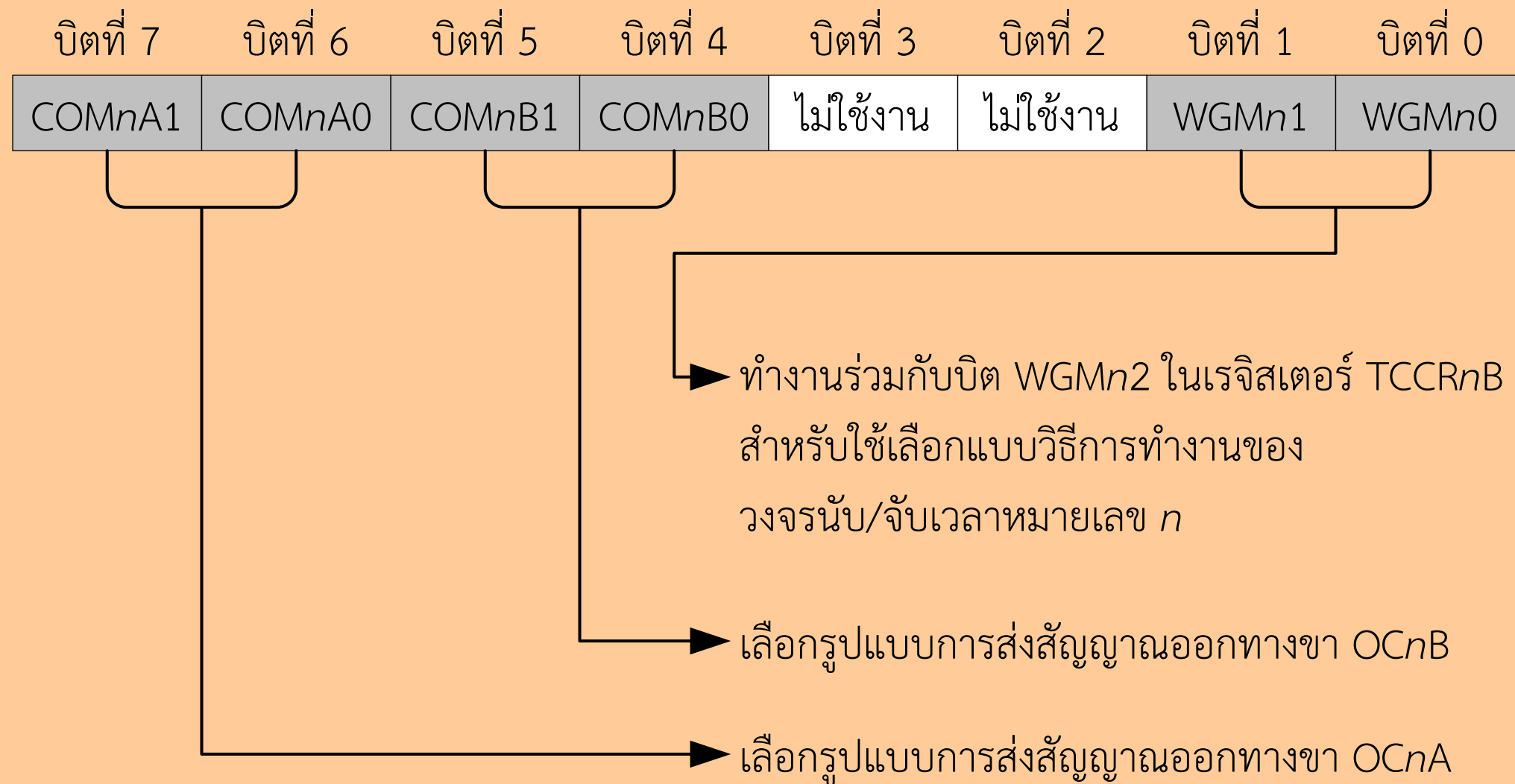
# เรจิสเตอร์ที่เกี่ยวข้องกับ Timer/Counter0

- ◆ TCNT0 Timer/Counter0
- ◆ OCR0A Output Compare Register 0 A
- ◆ OCR0B Output Compare Register 0 B
- ◆ TIFR0 Timer Interrupt Flag Register 0
- ◆ TIMSK0 Timer Interrupt Mask Register 0



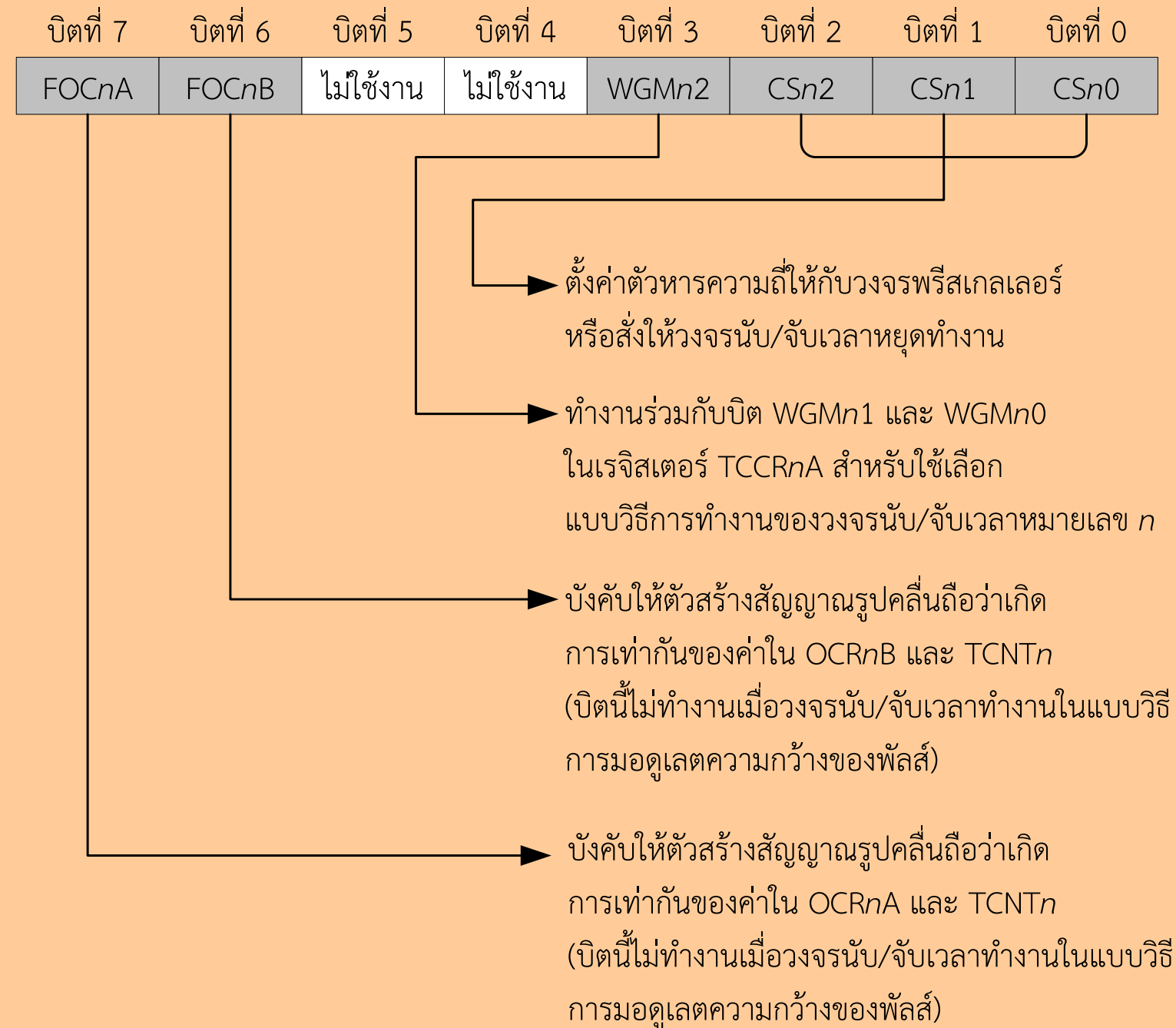


# TCCR<sub>n</sub>A – Timer/Counter Control Register A





# TCCR $n$ B – Timer/Counter Control Register B



# แบบวิธีการทำงานวงจรรนับ/จับเวลาหมายเลข 0, 2



| บิตควบคุมการทำงาน |       |       | ชื่อแบบวิธีการทำงาน                             | ค่าสูงสุดที่นับได้ |
|-------------------|-------|-------|---|--------------------|
| WGMn2             | WGMn1 | WGMn0 |   |                    |
| 0                 | 0     | 0     | แบบวิธีปกติ                                     | 0xFF               |
| 0                 | 0     | 1     | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเฟสถูกต้อง | 0xFF               |
| 0                 | 1     | 0     | แบบวิธี CTC                                     | กำหนดใน OCRnA      |
| 0                 | 1     | 1     | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเร็ว       | 0xFF               |
| 1                 | 0     | 0     | ไม่มีการใช้งาน                                  | -                  |
| 1                 | 0     | 1     | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเฟสถูกต้อง | กำหนดใน OCRnA      |
| 1                 | 1     | 0     | ไม่มีการใช้งาน                                  | -                  |
| 1                 | 1     | 1     | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเร็ว       | กำหนดใน OCRnA      |







# การเลือกสัญญาณนาฬิกาของวงจรนับ/จับเวลา

| CSn2 | CSn1 | CSn0 | สัญญาณสำหรับป้อนวงจรนับ/จับเวลา   |
|------|------|------|---|
| 0    | 0    | 0    | ไม่มีสัญญาณนาฬิกาถูกป้อนให้<br>ส่งผลให้วงจรนับ/จับเวลาหมายเลข $n$ หยุดทำงาน |
| 0    | 0    | 1    | $CLK_{I/O}$   |
| 0    | 1    | 0    | $CLK_{I/O}/8$   |
| 0    | 1    | 1    | $CLK_{I/O}/64$  |
| 1    | 0    | 0    | $CLK_{I/O}/256$   |
| 1    | 0    | 1    | $CLK_{I/O}/1024$  |
| 1    | 1    | 0    | วงจรนับ/จับเวลารับสัญญาณนาฬิกาที่ขอบขาลงของสัญญาณที่ขา $Tn$                 |
| 1    | 1    | 1    | วงจรนับ/จับเวลารับสัญญาณนาฬิกาที่ขอบขาขึ้นของสัญญาณที่ขา $Tn$               |





# TIMSK $n$ – Timer/Counter $n$ Interrupt Mask Register

| บิตที่ 7  | บิตที่ 6  | บิตที่ 5  | บิตที่ 4  | บิตที่ 3  | บิตที่ 2   | บิตที่ 1   | บิตที่ 0 |
|-----------|-----------|-----------|-----------|-----------|------------|------------|----------|
| ไม่ใช้งาน | ไม่ใช้งาน | ไม่ใช้งาน | ไม่ใช้งาน | ไม่ใช้งาน | OCIE $n$ B | OCIE $n$ A | TOIE $n$ |

- ◆ OCIE $n$ A (Timer/Counter  $n$  Compare Match A Interrupt Enable) ใช้เพื่อเปิดทาง/ปิดทางการตอบรับการขัดจังหวะเมื่อ TCNT $n$  = OCR $n$ A
- ◆ OCIE $n$ B (Timer/Counter  $n$  Compare Match B Interrupt Enable) ใช้เพื่อเปิดทาง/ปิดทางการตอบรับการขัดจังหวะเมื่อ TCNT $n$  = OCR $n$ B
- ◆ บิต TOIE $n$  (Timer/Counter  $n$  Overflow Interrupt Enable) ใช้เพื่อเปิดทาง/ปิดทางการตอบรับการขัดจังหวะเมื่อเกิดการล้นของค่าใน TCNT $n$





# TIFR<sub>n</sub> – Timer/Counter *n* Interrupt Flag Register

| บิตที่ 7  | บิตที่ 6  | บิตที่ 5  | บิตที่ 4  | บิตที่ 3  | บิตที่ 2           | บิตที่ 1           | บิตที่ 0         |
|-----------|-----------|-----------|-----------|-----------|--------------------|--------------------|------------------|
| ไม่ใช้งาน | ไม่ใช้งาน | ไม่ใช้งาน | ไม่ใช้งาน | ไม่ใช้งาน | OCF <sub>n</sub> B | OCF <sub>n</sub> A | TOV <sub>n</sub> |

- ◆ ตัวบ่งชี้ OCF<sub>n</sub>A ย่อมาจาก Timer/Counter *n* Output Compare Match A Flag ใช้บอกว่าค่าในเรจิสเตอร์ OCR<sub>n</sub>A มีค่าเท่ากับค่าในเรจิสเตอร์ TCNT<sub>n</sub>
- ◆ ตัวบ่งชี้ OCF<sub>n</sub>B ย่อมาจาก Timer/Counter *n* Output Compare Match B Flag ใช้บอกว่าค่าในเรจิสเตอร์ OCR<sub>n</sub>B มีค่าเท่ากับค่าในเรจิสเตอร์ TCNT<sub>n</sub>
- ◆ ตัวบ่งชี้ TOV<sub>n</sub> ย่อมาจาก Timer/Counter *n* Overflow Flag ใช้ในการบอกว่าค่าในเรจิสเตอร์ TCNT<sub>n</sub> เกิดการล้นและมีการวกกลับจากค่า 0xFF มาเป็นค่า 0x00



# แบบวิธีปกติของวงจรรนับ/จับเวลา0 และ 2

- ◆ ค่าในเรจิสเตอร์ TCNT $n$  จะเพิ่มขึ้นครั้งละหนึ่งค่า
- ◆ เมื่อค่าในเรจิสเตอร์ TCNT $n$  เกิดการล้นจากค่า 0xFF วงกลับเป็นค่า 0x00 จะทำให้ตัวบ่งชี้การขัดจังหวะ TOV $n$  เปลี่ยนค่าเป็นตรรกะสูง
- ◆ หลังจากนั้นเมื่อวงจรรนับ/จับเวลาได้สัญญาณนาฬิกาเข้ามาเพิ่มอีกรอบ ก็ จะเกิดการนับเพิ่มขึ้นที่เรจิสเตอร์ TCNT $n$  ต่อไปเรื่อยๆ
- ◆ จะต้องมีการคำนวณค่าเริ่มต้นของเรจิสเตอร์ TCNT $n$
- ◆ ในกรณีที่ให้บริการการขัดจังหวะจะต้องมีการตั้งค่าใน TCNT $n$  ใหม่

# แบบวิธีปกติของวงจรนับ/จับเวลา0 และ 2

$$TCNTn_{INITIAL} = 256 - \frac{CLK_{CPU} * S}{N}$$

- ◆  $TCNTn_{INITIAL}$  คือ ค่าเริ่มต้นที่จะต้องตั้งให้กับเรจิสเตอร์  $TCNTn$
- ◆  $CLK_{CPU}$  คือ ความถี่สัญญาณนาฬิกาที่ไมโครคอนโทรลเลอร์ใช้ในการทำงาน
- ◆  $S$  คือ ค่าเวลาในหน่วยวินาทีที่ต้องการจับเวลา
- ◆  $N$  คือ ค่าคงที่สำหรับป้อนให้วงจรพรีสเกลเลอร์ ใช้ในการหารความถี่ (เลือกค่าได้ 5 ค่า ได้แก่ 1, 8, 64, 256 และ 1024)





# แบบวิธีปรกติของวงจรนับ/จับเวลา0 และ 2

$$\text{Time} = (256 - \text{TCNT}n_{\text{INITIAL}}) * \frac{N}{\text{CLK}_{\text{CPU}}}$$

- ◆ Time คือ ค่าเวลาในหน่วยวินาทีที่สามารถจับเวลาได้
- ◆  $\text{TCNT}n_{\text{INITIAL}}$  คือ ค่าเริ่มต้นที่จะต้องตั้งให้กับเรจิสเตอร์  $\text{TCNT}n$
- ◆  $\text{CLK}_{\text{CPU}}$  คือ ความถี่สัญญาณนาฬิกาที่ไมโครคอนโทรลเลอร์ใช้ในการทำงาน
- ◆ N คือ ค่าคงที่สำหรับป้อนให้วงจรพรีสเกลเลอร์ ใช้ในการหารความถี่ (เลือกค่าได้ 5 ค่า ได้แก่ 1, 8, 64, 256 และ 1024)



# การเขียนโปรแกรมเพื่อให้วงจรนับ/จับเวลา 0 และ 2

## ทำงานในแบบวิธีปฏิบัติ

- ◆ กำหนดค่าเวลาในหน่วยวินาทีที่ต้องการจับเวลา
- ◆ คำนวณหาค่าเริ่มต้นของเรจิสเตอร์ TCNT $n$  และทำการตั้งค่าให้เรจิสเตอร์ TCNT $n$
- ◆ ทำการตั้งค่าควบคุมในเรจิสเตอร์ TCCR $n$ A และ TCCR $n$ B
- ◆ ตั้งค่าบิต TOIE $n$  ในเรจิสเตอร์ TIMSK $n$  ให้มีค่าตรรกะสูงเพื่อเปิดทางการตอบรับการขัดจังหวะจากวงจรนับ/จับเวลาหมายเลข  $n$
- ◆ เปิดทางการตอบรับการขัดจังหวะส่วนกลางโดยใช้คำสั่ง sei ในกรณีที่ใช้ภาษาแอสเซมบลี หรือใช้ฟังก์ชัน sei() ในกรณีที่ใช้ภาษาซี
- ◆ เขียนรoutines บริการการขัดจังหวะสำหรับวงจรนับ/จับเวลาหมายเลข  $n$





## ตัวอย่างที่ 4.1

- ◆ จงหาค่า  $N$  ที่เหมาะสมเพื่อตั้งค่าให้วงจรนับ/จับเวลาหมายเลขศูนย์ทำการจับเวลา 2.5 มิลลิวินาที
- ◆ กำหนดให้วงจรนับ/จับเวลานี้ทำงานในแบบวิธีปรกติ
- ◆ ไมโครคอนโทรลเลอร์ทำงานที่ความถี่ 1 เมกะเฮิรตซ์
- ◆ จงคำนวณหาค่าความคลาดเคลื่อนของค่าเวลาที่จับได้เมื่อใช้  $N$  ค่าต่าง ๆ







# ตัวอย่างที่ 4.1

$$TCNTn_{INITIAL} = 256 - \frac{CLK_{CPU} * S}{N}$$

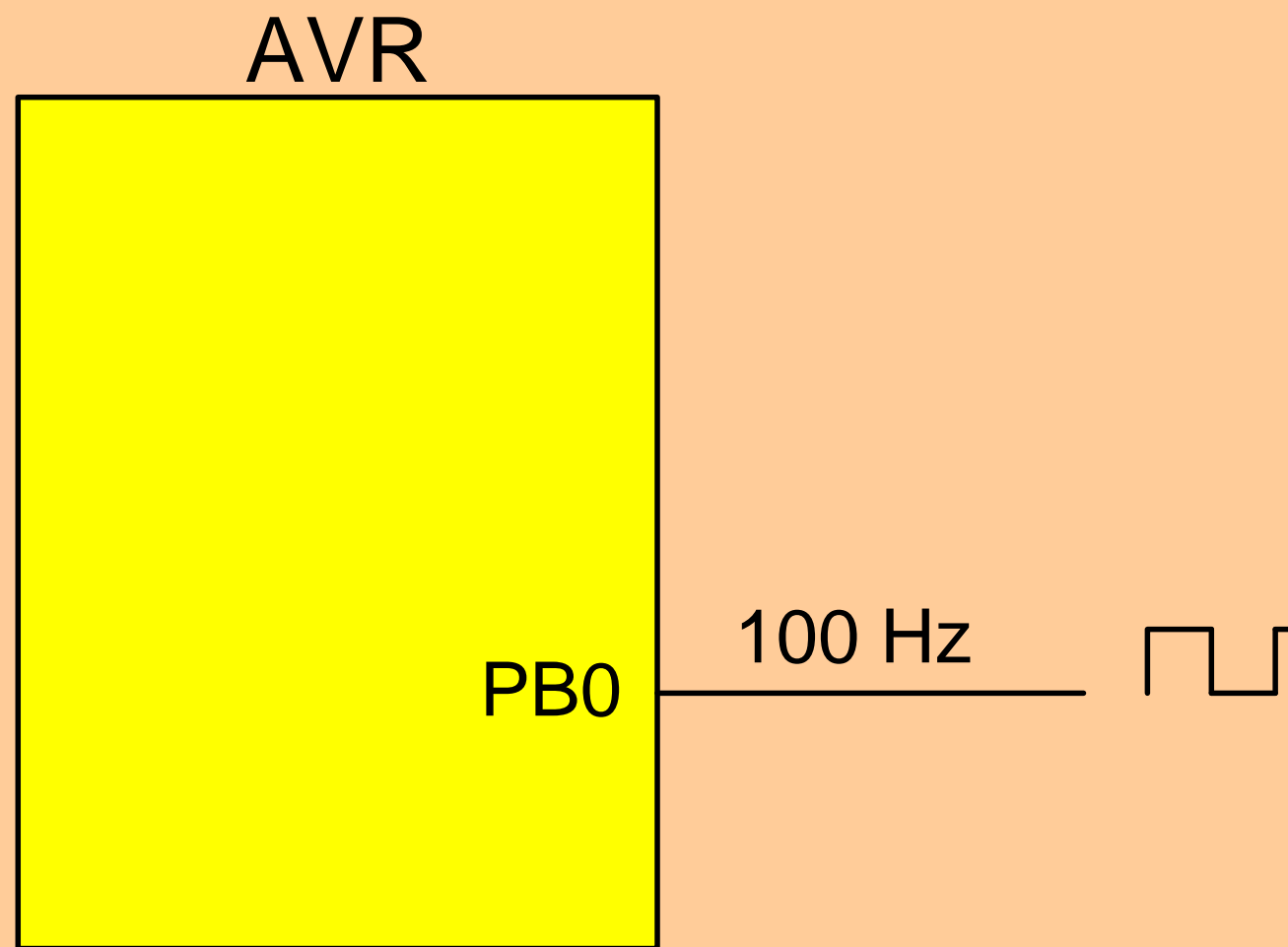
| N    | TCNT0 <sub>INITIAL</sub> |
|------|--------------------------|
| 1    | -2244.000                |
| 8    | -56.500                  |
| 64   | 216.938                  |
| 256  | 246.234                  |
| 1024 | 253.559                  |

| N    | TCNT0 | ค่าเวลาที่จับได้ | ความคลาดเคลื่อน |
|------|-------|------------------|-----------------|
| 64   | 217   | 2.496 mS         | -0.16 %         |
| 256  | 246   | 2.560 mS         | 2.40 %          |
| 1024 | 254   | 2.048 mS         | -18.08 %        |





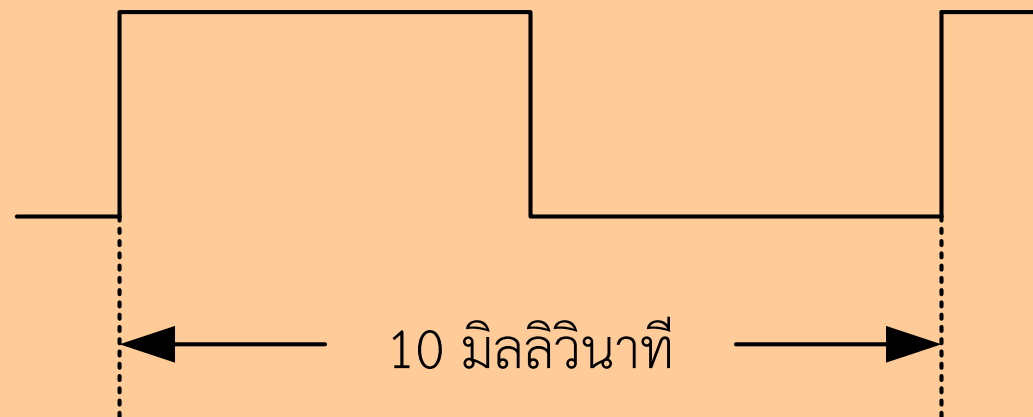
## ตัวอย่างที่ 4.2





## ตัวอย่างที่ 4.2

- ◆ จงเขียนโปรแกรมเพื่อส่งสัญญาณพัลส์รูปคลื่นสี่เหลี่ยม ความถี่ 100 เฮิรตซ์ ออกที่ขา PB0 ของตัวประมวลผล ATmega328P ซึ่งทำงานที่ความถี่สัญญาณนาฬิกา 1 เมกะเฮิรตซ์
- ◆ กำหนดให้ใช้วงจรนับ/จับเวลาหมายเลขศูนย์





## ตัวอย่างที่ 4.2

- ◆ คำนวณหาค่าเริ่มต้นของเรจิสเตอร์ TCNT0 ก่อน

$$\begin{aligned} \text{TCNT0}_{\text{INITIAL}} &= 256 - \frac{\text{CLK}_{\text{CPU}} * S}{N} \\ &= 256 - \frac{1 * 10^6 * 5 * 10^{-3}}{64} \\ &= 177.875 \end{aligned}$$

- ◆ คำนี้อาจจะต้องนำมาตั้งให้กับ TCNT0 และภายในรูทีนบริการ  
ขัดจังหวะจะต้องตั้งให้ TCNT0 เท่ากับค่านี้ทุกครั้งที่ทำงาน

การ





## ตัวอย่างที่ 4.2

|   |                            |  |
|---|----------------------------|--|
| 1 | #include <avr/io.h>        | //เรียกใช้คลังโปรแกรม io.h                                     |
| 2 | #include <avr/interrupt.h> | //เรียกใช้คลังโปรแกรมสำหรับการขัดจังหวะ                        |
| 3 | ISR(TIMERO0_OVF_vect)      | //ฟังก์ชันสนับสนุนการขัดจังหวะจากการล้นของค่าในวงจรรนับ/จับ-   |
| 4 | {                          | //-เวลาหมายเลขศูนย์  |
| 5 | TCNT0 = 178;               | //กำหนดค่าเริ่มต้นให้กับ TCNT0 ใหม่ทุกครั้งที่เกิดการขัดจังหวะ |
| 6 | PORTB ^= 0x01;             | //กลับเฉพาะบิตล่างของพอร์ต B ให้มีค่าเป็นตรงกันข้าม            |
| 7 | }                          | //สิ้นสุดขอบเขตของฟังก์ชันสนับสนุนการขัดจังหวะ                 |





## ตัวอย่างที่ 4.2

```
8  int main(void)                //ฟังก์ชันหลักของโปรแกรม
9  {
10     DDRB |= 0x01;              //กำหนดทิศทางให้พอร์ต B เป็นเอาต์พุตเฉพาะบิตล่างสุด คือ PBO
11     TIMSK0 = (1<<TOIE0);       //กำหนดให้ซีพียูตอบสนองการขัดจังหวะจากวงจรรนับ/จับเวลา0
12     TCCR0B = (1<<CS01 | 1<<CS00); //ตั้งให้ WGM02 เป็น 0 และ CS0[2:0] เท่ากับ 0112(หาร 64)
13     TCCR0A = 0x00;             //ตั้งให้บิต WGM0[1:0] มีค่า 002
14     TCNT0 = 178;               //กำหนดค่าเริ่มต้นให้กับ TCNT0 ด้วยค่าที่คำนวณไว้
15     sei();                     //สั่งให้ไมโครคอนโทรลเลอร์ตอบสนองการขัดจังหวะส่วนกลาง
16     while(1)                   //วนซ้ำการทำงานตลอดเวลาที่ยังเปิดเครื่อง (วนซ้ำไม่รู้จบ)
17     {
18                               //ไม่มีการทำงานใด ๆ ในส่วนนี้
19     }
20 }
```





# ตั้งค่า Timer0 อย่างไรให้เขียนโปรแกรมง่าย

- ◆  $TCCR0B = (0 \ll WGM02) \mid (0 \ll CS02) \mid (0 \ll CS01) \mid 1 \ll CS00);$
- ◆  $TCCR0A = (1 \ll WGM01) \mid (1 \ll WGM00);$
- ◆  $TCCR0A \mid= (1 \ll COM0A1);$
- ◆  $TCCR0A \mid= (0 \ll COM0A0) \mid (1 \ll COM0B1) \mid (0 \ll COM0B0);$







# แบบวิธี CTC ของวงจรรนับ/จับเวลาหมายเลข 0, 2

- ◆ ย่อมาจาก Clear Timer on Compare
- ◆ ค่าในเรจิสเตอร์ TCNT $n$  จะนับขึ้นจนถึงค่าที่ตั้งไว้ใน OCR $nA$  แล้วจึงกลับมาเริ่มนับจากศูนย์ใหม่อีกครั้ง
- ◆ ไม่จำเป็นต้องเขียนโปรแกรมเพื่อส่งบรรจุค่าเริ่มต้นให้กับเรจิสเตอร์ TCNT $n$  ใหม่ในกรณีที่บริการการขัดจังหวะ
- ◆ มักนิยมตั้งให้เรจิสเตอร์ TCNT $n$  มีค่าเริ่มต้นเป็นศูนย์
- ◆ มักนิยมใช้งานร่วมกับตัวสร้างรูปคลื่นสัญญาณ (Waveform Generator)



# แบบวิธี CTC ของวงจรรนับ/จับเวลาหมายเลข 0, 2

$$S_{CTCn} = \frac{(1 + OCRnA) * N}{CLK_{CPU}}$$

- ◆  $S_{CTCn}$  คือ เวลา (วินาที)
- ◆  $OCRnA$  คือ ค่าเริ่มต้นที่จะต้องตั้งให้กับเรจิสเตอร์  $OCRnA$
- ◆  $CLK_{CPU}$  คือ สัญญาณนาฬิกาที่ไมโครคอนโทรลเลอร์ใช้ในการทำงาน
- ◆  $N$  คือ ค่าคงที่สำหรับป้อนให้วงจรพรีสเกลเลอร์ ใช้ในการหาความถี่ (เลือกค่าได้ 5 ค่า ได้แก่ 1, 8, 64, 256 และ 1024)



# การเขียนโปรแกรมวงจรรนับ/จับเวลาหมายเลข 0, 2

## ให้ทำงานในแบบวิธี CTC

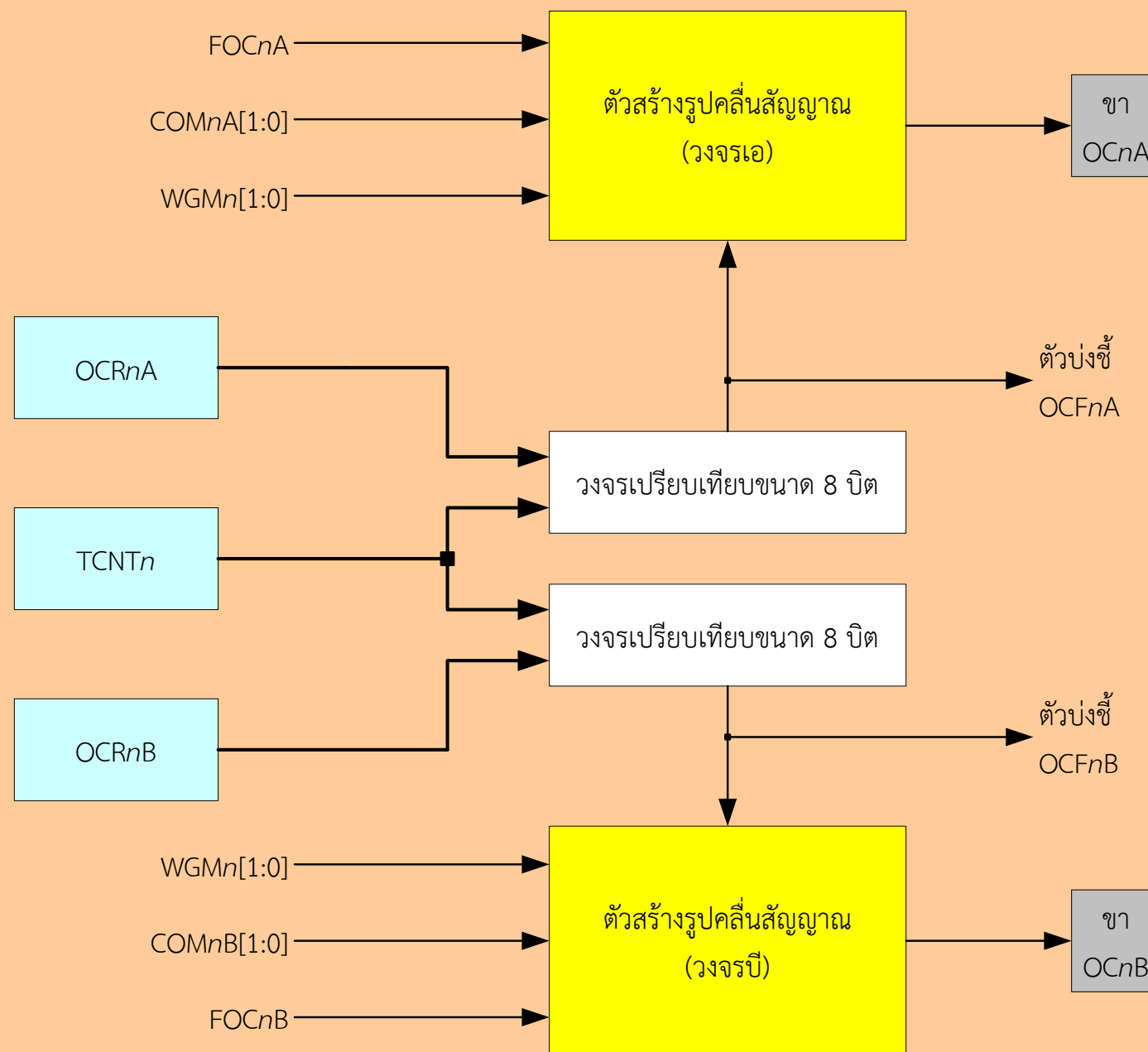


- ◆ กำหนดค่าเวลาในหน่วยวินาทีที่ต้องการจับเวลา จากนั้นนำค่าดังกล่าวมาคำนวณหาค่าเริ่มต้นของเรจิสเตอร์  $OCRnA$
- ◆ ตั้งค่าให้เรจิสเตอร์  $TCNTn$  ให้เท่ากับศูนย์
- ◆ ตั้งค่าควบคุมในเรจิสเตอร์  $TCCRnA$  และ  $TCCRnB$
- ◆ ตั้งค่าบิต  $OCIE_nA$  ในเรจิสเตอร์  $TIMSKn$  ให้มีค่าตรรกะสูงเพื่อเปิดทางการตอบรับการขัดจังหวะจากการเท่ากับค่าใน  $TCNTn$  และ  $OCRnA$
- ◆ เปิดทางการตอบรับการขัดจังหวะส่วนกลางโดยใช้คำสั่ง `sei` ในกรณีที่ใช้ภาษาแอสเซมบลี หรือใช้ฟังก์ชัน `sei()` ในกรณีที่ใช้ภาษาซี
- ◆ เขียนรoutinesบริการการขัดจังหวะสำหรับวงจรรนับ/จับเวลาหมายเลข  $n$





# ตัวสร้างรูปคลื่นสัญญาณ



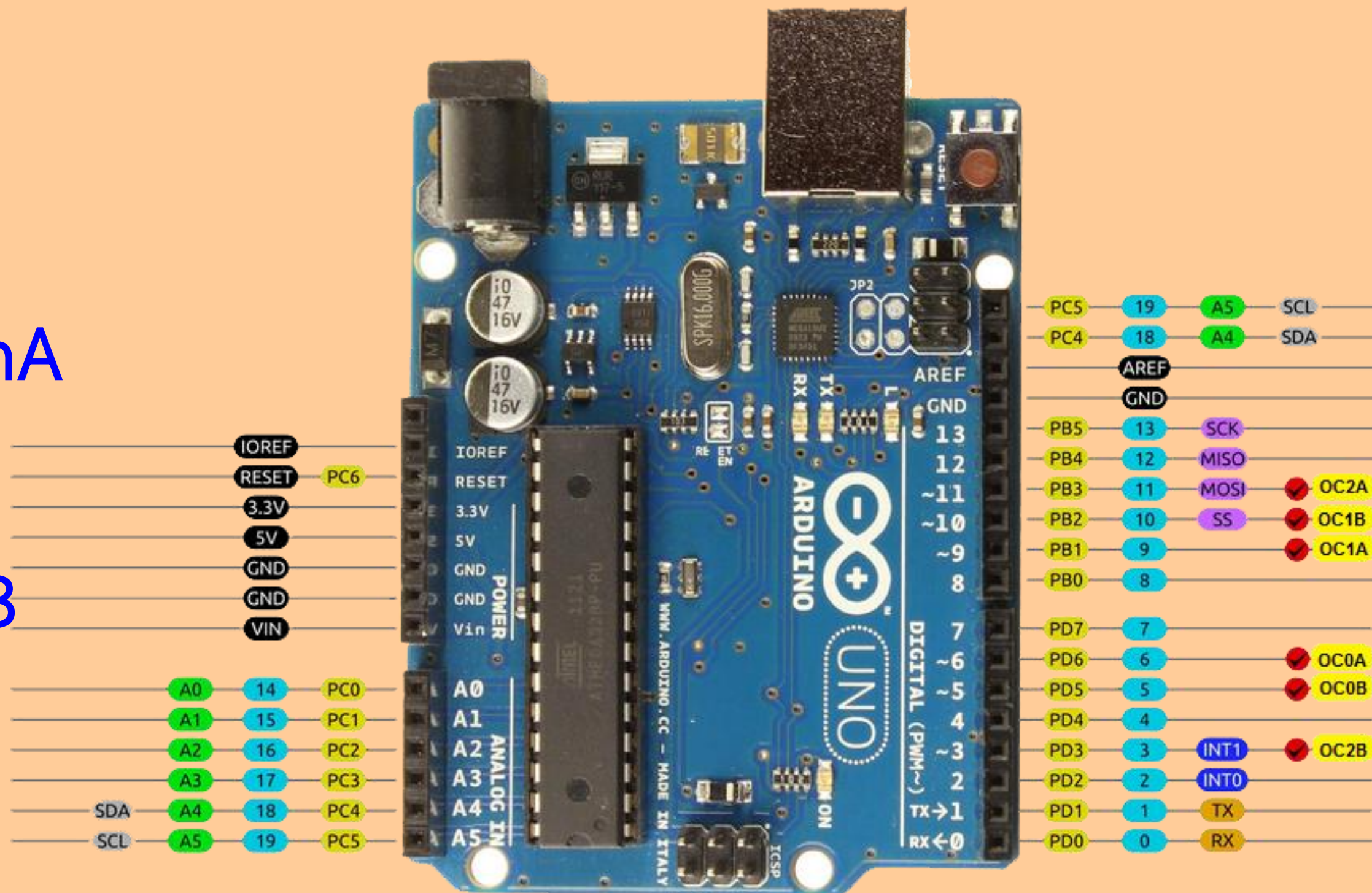




ขา OCnA

และ

OCnB



240-319

AVR

DIGITAL

ANALOG

POWER

SERIAL

SPI

I2C

PWM

INTERRUPT

CTC modes



# การตั้งค่าควบคุมตัวสร้างรูปคลื่นสัญญาณ

| บิตควบคุม |        | ความหมาย   |
|-----------|--------|--|
| COMnA1    | COMnA0 |  |
| 0         | 0      | ไม่มีการใช้งานขา OCnA ขาพอร์ตที่ตรงกับขา OCnA ทำหน้าที่รับส่งข้อมูลตามปกติ |
| 0         | 1      | กลับบิตที่ขา OCnA เป็นตรงกันข้ามเมื่อค่าใน TCNTn และ OCRnA เท่ากัน         |
| 1         | 0      | ตั้งให้ขา OCnA เป็นตรรกะต่ำเมื่อค่าใน TCNTn และ OCRnA เท่ากัน              |
| 1         | 1      | ตั้งให้ขา OCnA เป็นตรรกะสูงเมื่อค่าใน TCNTn และ OCRnA เท่ากัน              |

| บิตควบคุม |        | ความหมาย   |
|-----------|--------|--|
| COMnB1    | COMnB0 |  |
| 0         | 0      | ไม่มีการใช้งานขา OCnB ขาพอร์ตที่ตรงกับขา OCnB ทำหน้าที่รับส่งข้อมูลตามปกติ |
| 0         | 1      | กลับบิตที่ขา OCnB เป็นตรงกันข้ามเมื่อค่าใน TCNTn และ OCRnB เท่ากัน         |
| 1         | 0      | ตั้งให้ขา OCnB เป็นตรรกะต่ำเมื่อค่าใน TCNTn และ OCRnB เท่ากัน              |
| 1         | 1      | ตั้งให้ขา OCnB เป็นตรรกะสูงเมื่อค่าใน TCNTn และ OCRnB เท่ากัน              |







# กรณีใช้ตัวสร้างรูปคลื่นสัญญาณสร้างพัลส์

$$F_{OCnx} = \frac{CLK_{CPU}}{2 * N * (1 + OCRnx)}$$

- ◆  $F_{OCnx}$  คือ ค่าความถี่ที่กำเนิดจากขา OCnx
- ◆ OCRnA คือ ค่าที่จะต้องตั้งให้กับเรจิสเตอร์ OCRnA
- ◆  $CLK_{CPU}$  คือ สัญญาณนาฬิกาที่ไมโครคอนโทรลเลอร์ใช้ในการทำงาน
- ◆ N คือ ค่าคงที่สำหรับป้อนให้วงจรพรีสเกลเลอร์ ใช้ในการหารความถี่  
(เลือกค่าได้ 5 ค่า ได้แก่ 1, 8, 64, 256 และ 1024)





## ตัวอย่างที่ 4.3

- ◆ จงเขียนโปรแกรมเพื่อส่งสัญญาณพัลส์รูปคลื่นสี่เหลี่ยม ความถี่ 100 เฮิรตซ์ออกที่ขา PB0
- ◆ ใช้ตัวประมวลผล ATmega 328P ซึ่งทำงานที่ 8 MHz
- ◆ กำหนดให้ใช้วงจรนับ/จับเวลาหมายเลขศูนย์ทำงาน ในแบบวิธี CTC





## ตัวอย่างที่ 4.3

$$OCR0A = \frac{S_{CTC0} * CLK_{CPU}}{N} - 1$$

| N    | OCR0A     |
|------|-----------|
| 1    | 39,999.00 |
| 8    | 4,999.00  |
| 64   | 624.00    |
| 256  | 155.25    |
| 1024 | 38.06     |





## ตัวอย่างที่ 4.3

|   |                            |  |
|---|----------------------------|--|
| 1 | #include <avr/io.h>        | //เรียกใช้คลังโปรแกรม io.h                       |
| 2 | #include <avr/interrupt.h> | //เรียกใช้คลังโปรแกรมสำหรับการขัดจังหวะ          |
| 3 | ISR(TIMERO0_COMPA_vect)    | //ฟังก์ชันสนับสนุนการขัดจังหวะจากตัวบ่งชี้ OCF0A |
| 4 | {                          |  |
| 5 | PORTB ^= 0x01;             | //กลับบิตที่พอร์ต B บิตล่างสุดเป็นตรงกันข้าม     |
| 6 |                            |  |
| 7 | }                          |  |

เนื่องจากตัวสร้างรูปคลื่นสัญญาณมิได้เชื่อมต่อกับ PB0 จึงไม่สามารถใช้ Waveform gen เพื่อสร้างพัลส์ได้ จึงต้องใช้ ISR ในการกลับบิตของ PB0





## ตัวอย่างที่ 4.3

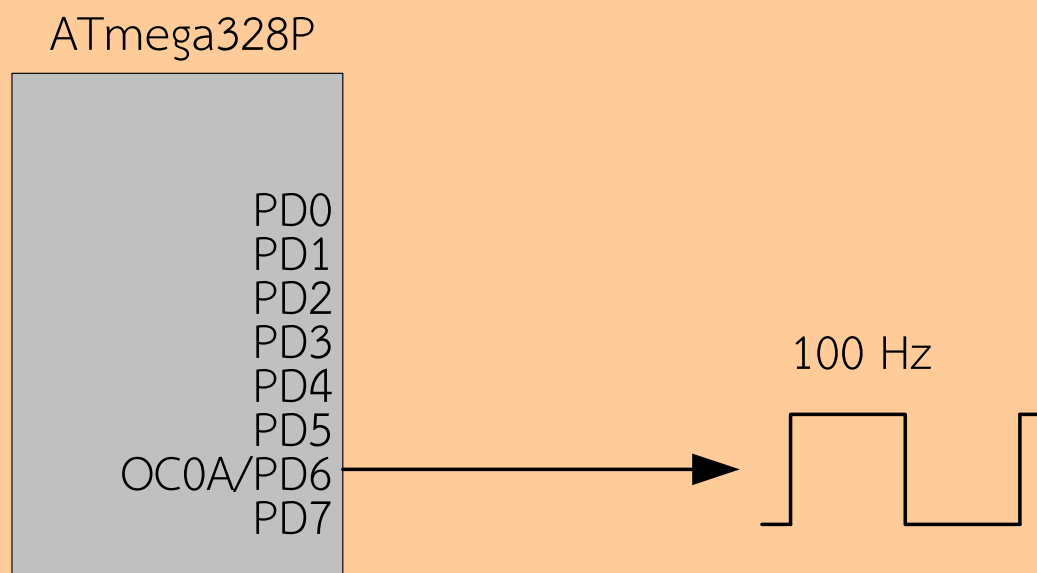
```
8  int main(void)                                //ฟังก์ชันหลักของโปรแกรม
9  {
10      DDRB |= 0x01;                             //ตั้งให้พอร์ต B เป็นเอาต์พุตเฉพาะบิตล่างสุด คือ PBO
11      TIMSK0 = (1<<OCIE0A);                     //อนุญาตให้เกิดการขัดจังหวะจากตัวบ่งชี้ OCF0A
12      TCCR0A = (1<<WGM01);                       //ตั้ง Timer0 ทำงานที่ CTC mode, WGM01=1, WGM00=0
13      TCCR0B = (1<<CS02|1<<CS00);                //หารความถี่ด้วย 1024 (CS02=1, CS01=0, CS00=1)
14      TCNT0 = 0;                                 //กำหนดให้ค่าเริ่มต้นของ TCNT0 เป็นศูนย์
15      OCR0A = 38;                                //ค่าเริ่มต้นของ OCR0A ซึ่งได้จากการคำนวณ
16      sei();                                     //สั่งให้ตัวประมวลผลตอบสนองการขัดจังหวะส่วนกลาง
17      while(1)                                   //วนซ้ำทำงานแบบไม่รู้จบ
18      {
19                                                  //ไม่มีการทำงานใด ๆ ในส่วนนี้
20      }
21 }
```





## ตัวอย่างที่ 4.4

- ◆ จงเขียนโปรแกรมส่งสัญญาณพัลส์รูปคลื่นสี่เหลี่ยมความถี่ 100 เฮิรตซ์ออกที่ขา OC0A
- ◆ ให้ใช้ตัวสร้างรูปคลื่นสัญญาณ (Waveform Generator) ของ วงจรนับ/จับเวลา หมายเลขศูนย์ และให้ทำงานในโหมด CTC
- ◆ ตัวประมวลผล ATmega328P ทำงานที่ความถี่ 8 MHz



# ตัวอย่างที่ 4.4

|    |                                  |  |
|----|----------------------------------|--|
| 1  | #include <avr/io.h>              | //เรียกใช้คลังโปรแกรม io.h                                   |
| 2  |                                  |  |
| 3  | int main(void)                   | //ฟังก์ชันหลักของโปรแกรม                                     |
| 4  | {                                | //เริ่มต้นขอบเขตของฟังก์ชันหลักของโปรแกรม                    |
| 5  | DDRD  = 1 << DDD6;               | //ตั้งให้พอร์ต D บิตที่ 6 เป็นเอาต์พุต                       |
| 6  | TCCR0A = (1<<COM0A0   1<<WGM01); | //COM0A[1:0] = 01 <sub>2</sub> , WGM0[1:0] = 10 <sub>2</sub> |
| 7  | TCCR0B = (1<<CS02   1<<CS00);    | //CS02=1, CS01=0, CS00=1, WGM02=0                            |
| 8  | TCNT0 = 0;                       | //ตั้งค่าเริ่มต้นให้ TCNT0 เริ่มนับจากศูนย์                  |
| 9  | OCR0A = 38;                      | //ค่า OCR0A ที่ได้จากการคำนวณ                                |
| 10 | while(1)                         | //วนซ้ำไม่รู้จบ  |
| 11 | {                                |  |
| 12 |                                  | //ไม่มีการทำงานใด ๆ ในส่วนนี้                                |
| 13 | }                                |  |
| 14 | }                                | //สิ้นสุดขอบเขตของฟังก์ชันหลักของโปรแกรม                     |





# การบ้าน



- ◆ จงเขียนโปรแกรมเพื่อส่งพัลส์ออกทางขา PC0 ความถี่ 350 Hz โดยใช้ Timer0 Overflow interrupt
- ◆ ให้เขียนด้วยภาษาซี
- ◆ เขียนเสร็จให้ทดสอบความถูกต้องด้วย Oscilloscope ในโปรแกรม Proteus
- ◆ ส่งที่ต้องส่งในตัวรายงาน
  - ◆ Code ภาษาซี และอธิบายการคำนวณค่าความถี่ที่ซีพียูสร้างได้
  - ◆ capture ภาพความถี่ที่วัดได้จากออสซิลโลสโคป
  - ◆ ส่งงานใน lms2 ภายในวันเวลาที่กำหนด

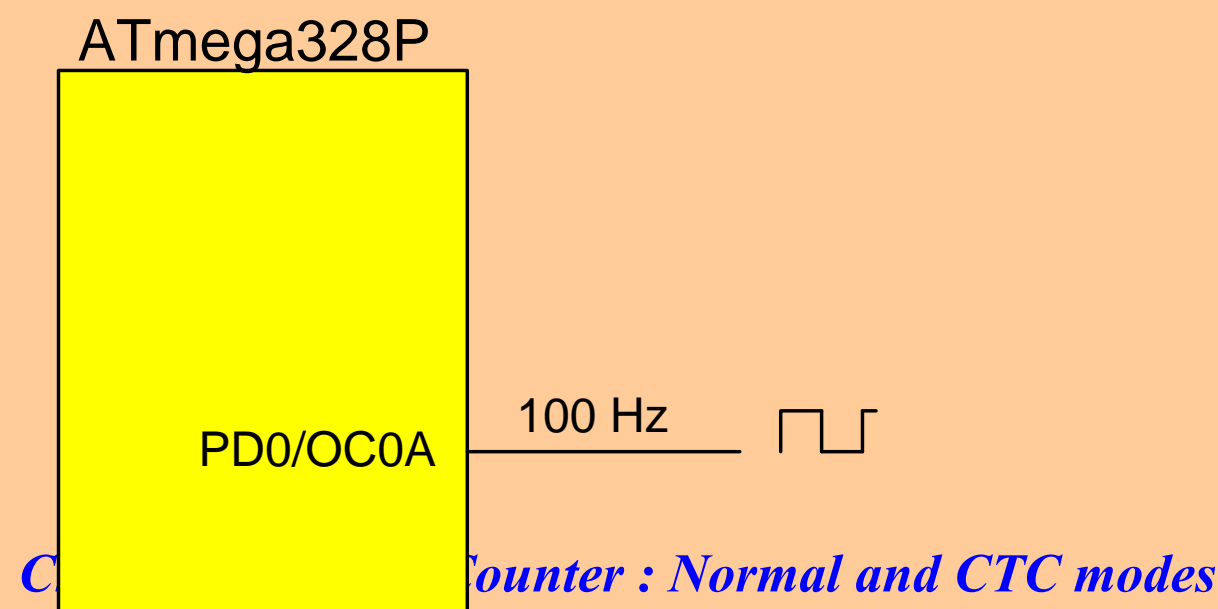






## ตัวอย่างที่ 4.5

- ◆ จงใช้ AVR ทำงานที่ 1 MHz สร้างสัญญาณพัลส์สี่เหลี่ยมความถี่ 100 Hz ออกทางขา OC0A ดังรูป
- ◆ กำหนดให้ใช้ Timer0 ในแบบวิธี CTC
- ◆ ไม่ต้องขัดจังหวะซีพียู แต่ใช้ตัวสร้างรูปคลื่นสัญญาณในการสร้างสัญญาณพัลส์แทน



## ตัวอย่างที่ 4.5

- ◆ ทำการหาค่าเริ่มต้นของ OCR0A ที่ทำให้ชิพียูทำการ Toggle ชิปียู ที่ขา OC0A ได้ความถี่ 100 Hz

$$F_{OC0A} = \frac{F_{clk\_io}}{2 * N * (1 + OCR0A)}$$
$$OCR0A = \frac{F_{clk\_io}}{2 * N * F_{OC0A}} - 1$$
$$= \frac{1 * 10^6}{2 * 64 * 100} - 1$$
$$= 77.125$$

- ◆ ดังนั้นต้องปัดเศษให้ค่าเริ่มต้นของ OCR0A เท่ากับ 77



## ตัวอย่างที่ 4.5 (ต่อ)

```
#include <avr/io.h>
#include <avr/interrupt.h>

void do_nothing(void)
{
}

//---1 MHz AVR is utilized

int main(void)
{
    TIMSK0 = 0x00; //--disable all interrupts of timer0
    TCCR0A = 0x42; //ctc mode and OC0A pin toggle
    TCNT0 = 0;
    OCROA = 77;
    TCCR0B = 0x03; // enable clock for Timer0 (using prescaler=64)

    DDRD |= (1<<DDD6); //PD6 pin is configured as output

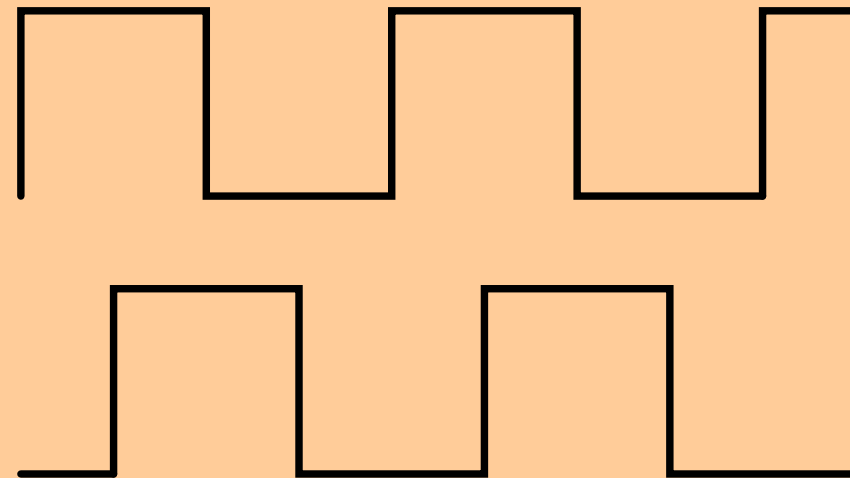
    while(1)
    {
        do_nothing();
    }
}

//(1<<CS02) | (1<<CS00)
```



# การบ้าน

- ◆ จงใช้ AVR สร้างสัญญาณพัลส์สี่เหลี่ยมความถี่ 60 Hz ออกทางขา PC1 และ PC2 โดยให้มีเฟสต่างกัน 90 องศา ดังรูป



- ◆ กำหนดให้ใช้ Timer0 ใน normal mode
- ◆ เขียนโปรแกรมใน AvrStudio และทดสอบการทำงานด้วย Proteus และส่งใน lms2





# Chapter 4 - part2

วงจรรนับ/จับเวลา (Counter/Timer)

ขนาด 16 บิต





# เรจิสเตอร์ควบคุมวงจรรนับ/จับเวลาหมายเลข 1

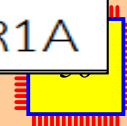
| บิตที่ 7 | บิตที่ 6 | บิตที่ 5 | บิตที่ 4 | บิตที่ 3 | บิตที่ 2 | บิตที่ 1 | บิตที่ 0 |                   |
|----------|----------|----------|----------|----------|----------|----------|----------|-------------------|
| COM1A1   | COM1A0   | COM1B1   | COM1B0   | -        | -        | WGM11    | WGM10    | เรจิสเตอร์ TCCR1A |
| ICNC1    | ICES1    | -        | WGM13    | WGM12    | CS12     | CS11     | CS10     | เรจิสเตอร์ TCCR1B |
| FOC1A    | FOC1B    | -        | -        | -        | -        | -        | -        | เรจิสเตอร์ TCCR1C |
| -        | -        | ICIE1    | -        | -        | OCIE1B   | OCIE1A   | TOIE1    | เรจิสเตอร์ TIMSK1 |
| -        | -        | ICIF1    | -        | -        | OCF1B    | OCF1A    | TOV1     | เรจิสเตอร์ TIFR1  |





# แบบวิธีการทำงานของวงจรรนับ/จับเวลาหมายเลข 1

| บิตควบคุมการทำงาน<br>WGM1[3:0] | ชื่อแบบวิธีการทำงาน  | ค่าสูงสุดที่นับได้ |
|--------------------------------|--|--------------------|
| 0000 <sub>2</sub>              | แบบวิธีปกติ  | 0xFFFF             |
| 0001 <sub>2</sub>              | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเฟสถูกต้องชนิด 8 บิต  | 0x00FF             |
| 0010 <sub>2</sub>              | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเฟสถูกต้องชนิด 9 บิต  | 0x01FF             |
| 0011 <sub>2</sub>              | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเฟสถูกต้องชนิด 10 บิต | 0x03FF             |
| 0100 <sub>2</sub>              | แบบวิธี CTC  | กำหนดใน OCR1A      |
| 0101 <sub>2</sub>              | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเร็วชนิด 8 บิต        | 0x00FF             |
| 0110 <sub>2</sub>              | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเร็วชนิด 9 บิต        | 0x01FF             |
| 0111 <sub>2</sub>              | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเร็วชนิด 10 บิต       | 0x03FF             |
| 1000 <sub>2</sub>              | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเฟสและความถี่ถูกต้อง  | กำหนดใน ICR1       |
| 1001 <sub>2</sub>              | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเฟสและความถี่ถูกต้อง  | กำหนดใน OCR1A      |
| 1010 <sub>2</sub>              | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเฟสถูกต้อง            | กำหนดใน ICR1       |
| 1011 <sub>2</sub>              | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเฟสถูกต้อง            | กำหนดใน OCR1A      |
| 1100 <sub>2</sub>              | แบบวิธี CTC  | กำหนดใน ICR1       |
| 1101 <sub>2</sub>              | ไม่มีการใช้งาน   | -                  |
| 1110 <sub>2</sub>              | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเร็ว                  | กำหนดใน ICR1       |
| 1111 <sub>2</sub>              | แบบวิธีการมอดูเลตความกว้างของพัลส์แบบเร็ว                  | กำหนดใน OCR1A      |





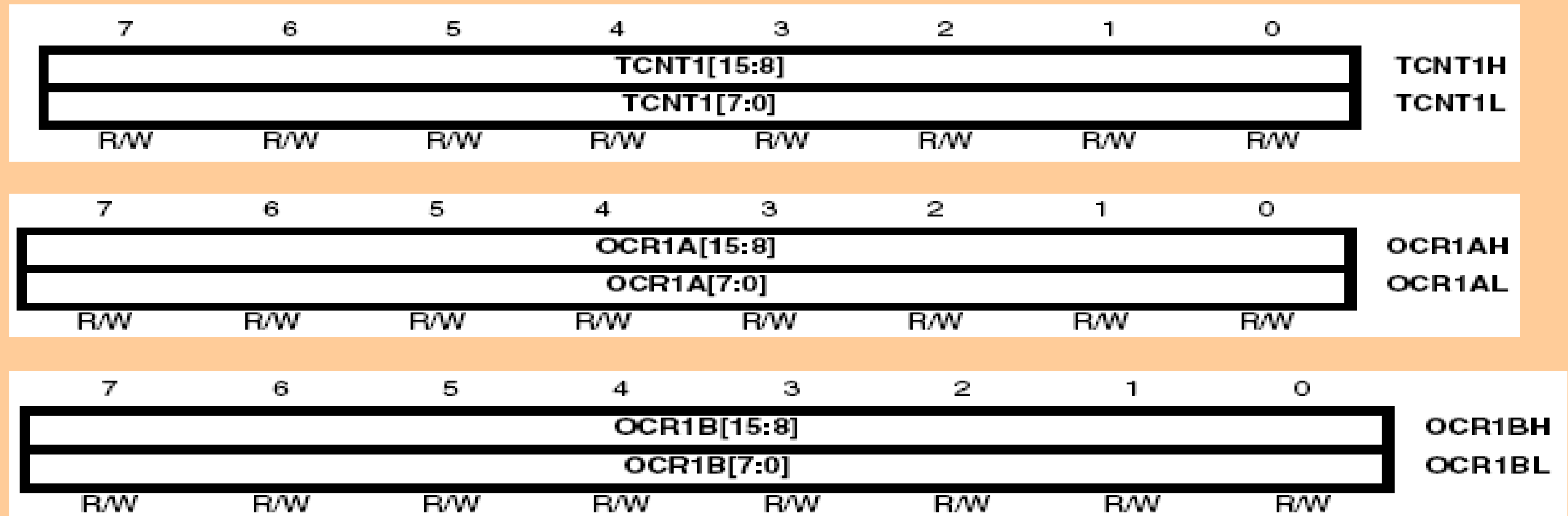


# CS12:CS10

| CS12 | CS11 | CS10 | ความหมาย  |
|------|------|------|---|
| 0    | 0    | 0    | ไม่มีสัญญาณนาฬิกาถูกป้อนให้ ส่งผลให้วงจรนับ/จับเวลาหมายเลขหนึ่งหยุดทำงาน  |
| 0    | 0    | 1    | วงจรนับ/จับเวลารับสัญญาณนาฬิกาของไมโครคอนโทรลเลอร์โดยตรง ไม่มีการหารความถี่โดยวงจรพรีสเกลเลอร์  |
| 0    | 1    | 0    | วงจรพรีสเกลเลอร์หารความถี่สัญญาณนาฬิกาของไมโครคอนโทรลเลอร์ด้วยค่า 8 แล้วส่งค่าความถี่ที่ได้ให้เป็นสัญญาณนาฬิกาของวงจรนับ/จับเวลาหมายเลขหนึ่ง    |
| 0    | 1    | 1    | วงจรพรีสเกลเลอร์หารความถี่สัญญาณนาฬิกาของไมโครคอนโทรลเลอร์ด้วยค่า 64 แล้วส่งค่าความถี่ที่ได้ให้เป็นสัญญาณนาฬิกาของวงจรนับ/จับเวลาหมายเลขหนึ่ง   |
| 1    | 0    | 0    | วงจรพรีสเกลเลอร์หารความถี่สัญญาณนาฬิกาของไมโครคอนโทรลเลอร์ด้วยค่า 256 แล้วส่งค่าความถี่ที่ได้ให้เป็นสัญญาณนาฬิกาของวงจรนับ/จับเวลาหมายเลขหนึ่ง  |
| 1    | 0    | 1    | วงจรพรีสเกลเลอร์หารความถี่สัญญาณนาฬิกาของไมโครคอนโทรลเลอร์ด้วยค่า 1024 แล้วส่งค่าความถี่ที่ได้ให้เป็นสัญญาณนาฬิกาของวงจรนับ/จับเวลาหมายเลขหนึ่ง |
| 1    | 1    | 0    | วงจรนับ/จับเวลารับสัญญาณนาฬิกาที่ขอบขาของขา T1  |
| 1    | 1    | 1    | วงจรนับ/จับเวลารับสัญญาณนาฬิกาที่ขอบขาขึ้นของขา T1  |



# เรจิสเตอร์นับของ Timer/Counter1



◆ ก่อนจะเข้าถึงเรจิสเตอร์ดังกล่าวจะต้องปิดทางการขัดจังหวะเสียก่อน



# การเข้าถึงเรจิสเตอร์ TCNT1

;---ตั้งค่าให้ TCNT1 เท่ากับ 0x02AB

ldi R31, 0x02

ldi R30, 0xAB

;-----สั่งให้ X เท่ากับเลขที่อยู่ของเรจิสเตอร์ TCNT1H

ldi XL, low(TCNT1H)

ldi XH, high(TCNT1H)

;-----สั่งให้ Y เท่ากับเลขที่อยู่ของเรจิสเตอร์ TCNT1L

ldi YL, low(TCNT1L)

ldi YH, high(TCNT1L)

;-----เขียนค่า 0x02 ลงสู่ TCNT1H

st X, R31

;-----เขียนค่า 0xAB ลงสู่ TCNT1L

st Y, R30

;---อ่านค่าจาก TCNT1 ให้กับเรจิสเตอร์ r31: r30

ld R31, X

ld R30, Y

unsigned int i;

...


//ตั้งค่าให้ TCNT1 เท่ากับ 0x02AB

TCNT1 = 0x02AB;

//อ่านค่าจาก TCNT1 ใส่ในตัวแปร i

i = TCNT1;

# การเขียนเรจิสเตอร์ TCNT1



```
;---ตั้งค่าให้ TCNT1 เท่ากับ 0x02AB
ldi    r31, 0x02
ldi    r30, 0xAB

;-----สั่งให้ X เท่ากับเลขที่อยู่ของเรจิสเตอร์ TCNT1H
ldi    XL, low(TCNT1H)
ldi    XH, high(TCNT1H)

;-----สั่งให้ Y เท่ากับเลขที่อยู่ของเรจิสเตอร์ TCNT1L
ldi    YL, low(TCNT1L)
ldi    YH, high(TCNT1L)

;-----เก็บค่าเดิมของตัวบ่งชี้การขัดจังหวะ (SREG)
in     r16, SREG

;-----ปิดทางการขัดจังหวะของไมโครคอนโทรลเลอร์
cli

;-----เขียนค่า 0x02 ลงสู่ TCNT1H
st     X, R31

;-----เขียนค่า 0xAB ลงสู่ TCNT1L
st     Y, R30


;-----คืนค่าสถานะเดิมของการขัดจังหวะ
out    SREG, r16
```

```
//---ฟังก์ชันสำหรับเขียนค่าลง TCNT1
void WRITE_TCNT1_SAFE(unsigned int i)
{
    unsigned char backup_sreg;
    //---เก็บค่าเดิมของ SREG เอาไว้ก่อน
    backup_sreg = SREG;

    //---ปิดทางการขัดจังหวะของไมโครคอนโทรลเลอร์
    cli();


    //---ตั้งค่าให้ TCNT1 เท่ากับ i
    TCNT1 = i;

    //---คืนค่าสถานะเดิมของการขัดจังหวะ
    SREG = backup_sreg;
}
```





# การอ่านเรจิสเตอร์ TCNT1



```
;---อ่านค่าจาก TCNT1 ให้กับเรจิสเตอร์ r31: r30
;-----สั่งให้ X เท่ากับเลขที่อยู่ของเรจิสเตอร์ TCNT1H
ldi    XL, low(TCNT1H)
ldi    XH, high(TCNT1H)
;-----สั่งให้ Y เท่ากับเลขที่อยู่ของเรจิสเตอร์ TCNT1L
ldi    YL, low(TCNT1L)
ldi    YH, high(TCNT1L)
;-----เก็บค่าเดิมของตัวบ่งชี้การขัดจังหวะ (SREG)
in     r16, SREG
;-----ปิดทางการขัดจังหวะของไมโครคอนโทรลเลอร์
cli
;-----อ่านค่าจาก TCNT1 ให้กับเรจิสเตอร์ r31: r30
ld     R31, X
ld     R30, Y
;-----คืนค่าสถานะเดิมของการขัดจังหวะ
out    SREG, r16
```

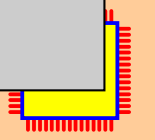

```
//---ฟังก์ชันสำหรับการอ่านค่าจาก TCNT1
unsigned int READ_TCNT1_SAFE(void)
{
    unsigned char backup_sreg;
    unsigned int i;
    //---เก็บค่าเดิมของ SREG เอาไว้ก่อน
    backup_sreg = SREG;

    //---ปิดทางการขัดจังหวะของไมโครคอนโทรลเลอร์
    cli();

    //อ่านค่าจาก TCNT1 ใส่ในตัวแปร i
    i = TCNT1;

    //---คืนค่าสถานะเดิมของการขัดจังหวะ
    SREG = backup_sreg;

    //---ส่งคืนค่า i ไปยังฟังก์ชันผู้เรียก
    return i;
}
```





# วงจรสร้างสัญญาณรูปคลื่นของวงจรรนับ/จับเวลาหมายเลข 1

## กรณีทำงานแบบวิธีปรกติและ CTC

| บิตควบคุม |        | ความหมาย   |
|-----------|--------|--|
| COM1x1    | COM1x0 |  |
| 0         | 0      | ไม่มีการใช้งานขาพอร์ตที่ตรงกับขา OC1x ให้ขาของพอร์ตทำหน้าที่รับส่งข้อมูลตามปรกติ |
| 0         | 1      | กลับบิตที่ขา OC1x เป็นตรงกันข้ามเมื่อค่าใน TCNT1 และ OCR1x เท่ากัน               |
| 1         | 0      | ตั้งให้ขา OC1x เป็นตรรกะต่ำเมื่อค่าใน TCNT1 และ OCR1x เท่ากัน                    |
| 1         | 1      | ตั้งให้ขา OC1x เป็นตรรกะสูงเมื่อค่าใน TCNT1 และ OCR1x เท่ากัน                    |





# Timer/Counter1 เมื่อทำงานในแบบวิธีปกติ

$$S_{\text{Timer1\_Normal}} = (65536 - \text{TCNT1}_{\text{INITIAL}}) * \frac{N}{\text{CLK}_{\text{CPU}}}$$

|                                       |   |
|---------------------------------------|---|
| เมื่อ $\text{TCNT1}_{\text{INITIAL}}$ | คือ ค่าเริ่มต้นที่จะต้องตั้งให้กับเรจิสเตอร์ TCNT1<br>ซึ่งต้องเป็นค่าจำนวนเต็มบวก และอยู่ในย่าน 0-65535 เท่านั้น          |
| $\text{CLK}_{\text{CPU}}$             | คือ ค่าความถี่สัญญาณนาฬิกาที่ไมโครคอนโทรลเลอร์ใช้ในการทำงาน   |
| $S_{\text{Timer1\_Normal}}$           | คือ คาบเวลาในหน่วยวินาทีที่ต้องการให้จับเวลาโดยใช้วงจรรนับ/จับเวลา<br>หมายเลขหนึ่งซึ่งทำงานในแบบวิธีปกติ                  |
| N                                     | คือ ค่าคงที่สำหรับป้อนให้วงจรรีเสทเลเตอร์ ใช้ในการหารความถี่<br>ซึ่งสามารถเลือกค่าได้ 5 ค่า ได้แก่ 1, 8, 64, 256 และ 1024 |



# Timer/Counter1 เมื่อทำงานในแบบวิธี CTC

$$S_{CTC1} = \frac{(1 + OCR1A) * N}{CLK_{CPU}}$$

เมื่อ

|             |  |
|-------------|--|
| $S_{CTC1}$  | คือ เวลา (วินาที) ที่สามารถจับโดยใช้วงจรนับ/จับเวลาหมายเลขหนึ่งในกรณีที่วงจรทำงานในแบบวิธี CTC                         |
| OCR1A       | คือ ค่าเริ่มต้นที่จะต้องตั้งให้กับเรจิสเตอร์ OCR1A   |
| $CLK_{CPU}$ | คือ สัญญาณนาฬิกาที่ไม่โครคอนโทรลเลอร์ใช้ในการทำงาน   |
| N           | คือ ค่าคงที่สำหรับป้อนให้วงจรพรีสเกลเลอร์ ใช้ในการหารความถี่ ซึ่งสามารถเลือกค่าได้ 5 ค่า ได้แก่ 1, 8, 64, 256 และ 1024 |



## ตัวอย่างที่ 4.8

- ◆ จงใช้ไมโครคอนโทรลเลอร์ AVR สำหรับทำการเปิด-ปิดหลอด LED ทุกๆ 1 นาที
- ◆ สมมติให้ใช้ชิพ ATmega328P ความเร็ว 8 MHz
- ◆ ให้ใช้ Timer ตัวใดก็ได้ของ AVR



## ตัวอย่างที่ 4.8 (ต่อ)

- ◆ ที่ 8 MHz ตัว Timer0 จับเวลาได้นานสุด

$$\begin{aligned} IP &= (256 - 0_{TCNT0}) * \frac{1024_N}{CPUclk} \\ &= \frac{256 * 1024}{8 * 10^6} \\ &= 32.768 \text{ } mS \end{aligned}$$

- ◆ ที่ 8 MHz ตัว Timer1 จับเวลาได้นานสุด

$$\begin{aligned} IP &= (65536 - 0_{TCNT1}) * \frac{1024_N}{CPUclk} \\ &= \frac{65536 * 1024}{8 * 10^6} \\ &= 8.3886 \text{ } sec \end{aligned}$$

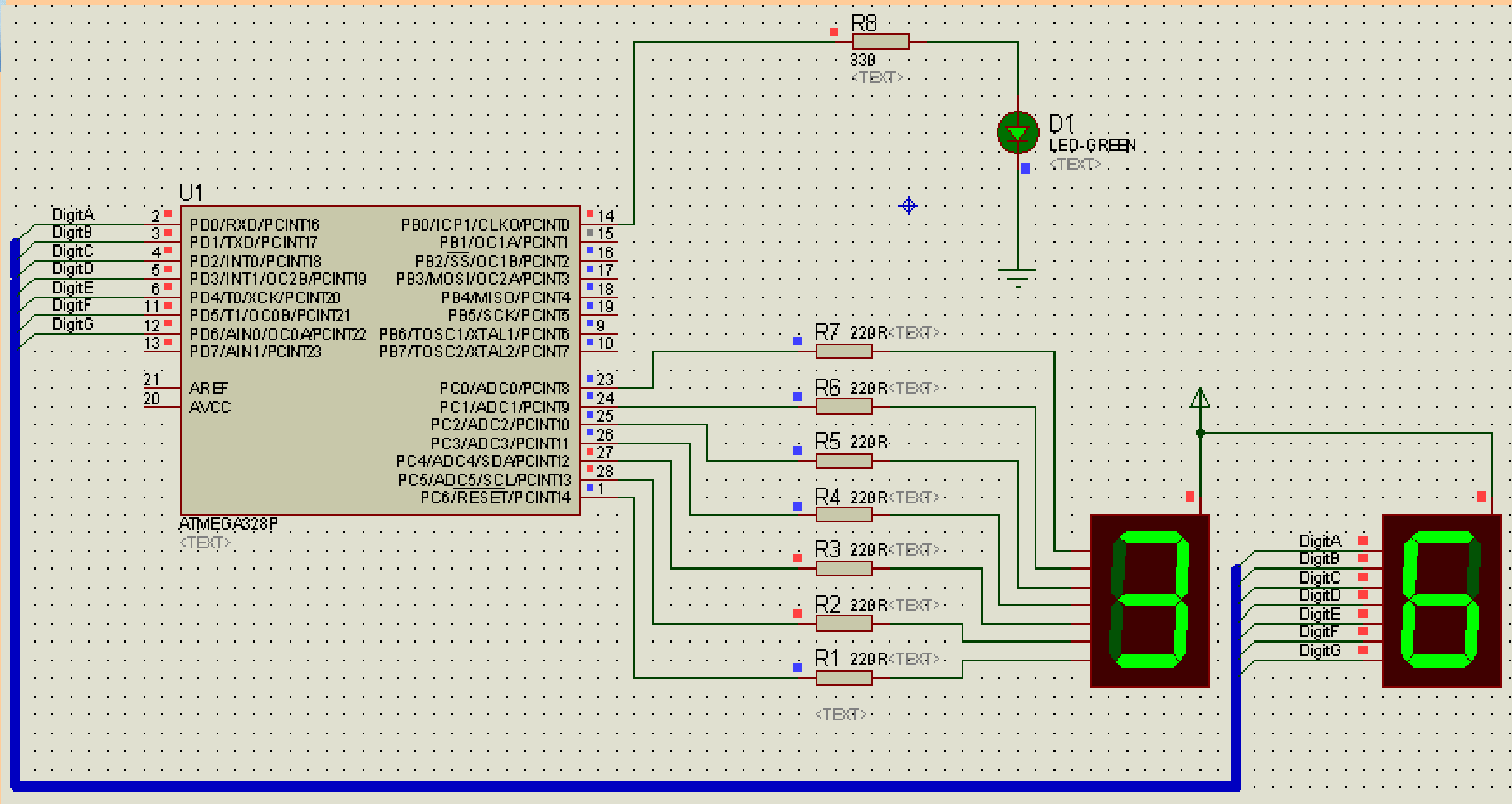


## ตัวอย่างที่ 4.8 (ต่อ)

- ◆ จะเห็นว่าการใช้ Timer1 และ Timer0 โดยตรงเพียงอย่างเดียวก็ไม่สามารถจับเวลานาน 1 นาทีได้
- ◆ ทางออกคือการใช้ Timer/Counter จับเวลาหลายๆ ครั้งและรวมเวลาเข้าด้วยกันให้ครบ 1 นาที
- ◆ สมมติใช้ Timer1 จับเวลา 1 วินาที จำนวน ก็หมายความว่าต้องใช้ Timer1 จับเวลาจำนวน 60 ครั้งจึงจะได้เวลาเท่ากับ 1 นาที



# ตัวอย่างที่ 4.8 (ต่อ)







# ตัวอย่างที่ 4.8 (ต่อ)

```
int main(void)
{
    TIME01 = 0;
    TIME10 = 0;

    DDRB  = 0b11111101;    //PB0= output, PB1=input
    DDRC  = 0b11111111;    //PC = output
    DDRD  = 0b11111111;    //PD = output

    PORTB = 0b11111110;    //when power-on, the LED is off
    PORTC = ~LOOKUPTB[0];
    PORTD = ~LOOKUPTB[0];

    //---set timer1 to interrupt the CPU every 1 second
    TCCR1A = 0x00;          //prescale factor = 256
    TCCR1B = 0x04;          //Timer 1 is run at normal mode
    TIMSK1 = 0x01; //enable timer1 overflow interrupt

    cli();
    TCNT1 = VALUE_FOR_TCNT1;
    sei();                  //enable global interrupt

    while(1)
    {
        do_nothing();
    }
}
```

```
ISR(TIMER1_OVF_vect)
{
    TCNT1 = VALUE_FOR_TCNT1;
    TIME01++;
    if (TIME01==10)
    {
        TIME10++;
        if (TIME10==6)
        {
            TIME10=0;
            PORTB = ~PORTB;
        }
        TIME01=0;
    }
    PORTC = ~LOOKUPTB[TIME10];
    PORTD = ~LOOKUPTB[TIME01];
}
```

```
#include <avr/io.h>
#include <avr/interrupt.h>
#define VALUE_FOR_TCNT1 34286

void do_nothing(void)
{
}

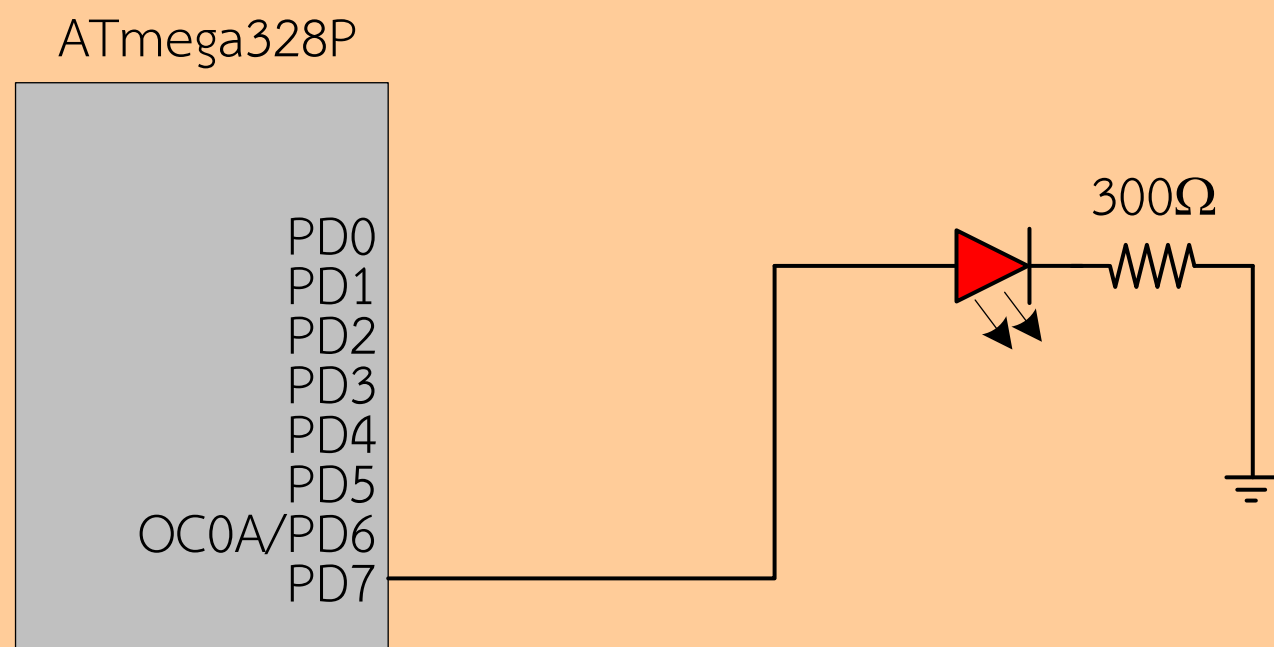
unsigned char TIME01, TIME10;

unsigned char LOOKUPTB[] = {
    0b00111111,
    0b00000110,
    0b01011011,
    0b01001111,
    0b01100110,
    0b01101101,
    0b01111101,
    0b00000111,
    0b01111111,
    0b01101111,
    0b01110111,
    0b01111100,
    0b00111001,
    0b01011110,
    0b01111001,
    0b01110001 };
```



## ตัวอย่างที่ 4.9

- ◆ จงเขียนโปรแกรมบน ATmega 328P เพื่อเปิดปิดหลอดแอลอีดีทุก ๆ หนึ่งนาทีก (เปิดหนึ่งนาทีกและปิดหนึ่งนาทีกสลับกัน)
- ◆ โดยสมมุติให้ตัวประมวลผลทำงานที่ความเร็ว 16 MHz





## ตัวอย่างที่ 4.9

$$S_{\text{Timer1\_Normal}} = (65536 - \text{TCNT1}_{\text{INITIAL}}) * \frac{N}{\text{CLK}_{\text{CPU}}}$$

| N    | TCNT1 <sub>INITIAL</sub> |
|------|--------------------------|
| 1    | -15,934,464              |
| 8    | -1,934,464               |
| 64   | -184,464                 |
| 256  | 3,036                    |
| 1024 | 49,911                   |





## ตัวอย่างที่ 4.9

```
1  #include <avr/io.h>           //ใช้คลังโปรแกรม io.h
2  #include <avr/interrupt.h>    //ใช้คลังโปรแกรม interrupt.h
3  uint8_t count;               //ประกาศตัวแปรส่วนกลางสำหรับใช้นับ
4  ISR(TIMER1_OVF_vect)         //ฟังก์ชันบริการการขัดจังหวะจากวงจรนับ/จับเวลาหมายเลขหนึ่ง
5  {                             //ทำงานเมื่อเกิดการล้นของค่าใน TCNT1
6      TCNT1 = 3036;            //บรรจุค่า 3036 ใหม่ลงไปใน TCNT1 ทุกครั้งที่ฟังก์ชันทำงาน
7      count++;                 //เพิ่มค่าในตัวแปร count ขึ้นหนึ่งค่า
8      if (count==60)           //หาก count เท่ากับ 60 แสดงว่าเวลาผ่านไปแล้ว 60 วินาที(1 นาที)
9      {                         //ให้ลบล้างค่าในตัวแปร count และกลับ PD7 เป็นตรงกันข้าม
10         count=0;              //ลบล้างค่าในตัวแปร count ให้กลับมาเป็นศูนย์
11         PORTD ^= 0x80;        //กลับบิต PD7 เป็นตรงกันข้ามจากค่าเดิม
12     }
13 }
```

//สิ้นสุดขอบเขตของฟังก์ชันบริการการขัดจังหวะ





## ตัวอย่างที่ 4.9

```
14 int main(void)           //ฟังก์ชันหลักของโปรแกรม
15 {                         //เริ่มต้นขอบเขตฟังก์ชันหลักของโปรแกรม
16     count = 0;           //ตั้งค่าเริ่มต้นให้ตัวแปร count เท่ากับศูนย์
17     DDRD = (1<<DDD7);    //ตั้งทิศทางให้บิตที่ 7 ของพอร์ต D ทำหน้าที่ส่งออก
18     PORTD = 0x80;        //เริ่มต้นให้บิตที่ 7 ของพอร์ต D เป็นตรรกะสูง (สั่งให้แอลอีดีติด)
19     TCCR1A = 0x00;       //ตั้งให้ WGM1[3:0]= 00002
20     TCCR1B = (1<<CS12);  //ตั้งให้ CS1[2:0] เท่ากับ 1002
21     TCNT1 = 3036;        //ตั้งค่าเริ่มต้นของ TCNT1 เท่ากับ 3036
22     TIMSK1 = (1<<TOIE1); //เปิดทางการขัดจังหวะของการล้นช่องค่าใน TCNT1
23     sei();               //เปิดทางการขัดจังหวะส่วนกลาง
24     while(1)             //วนซ้ำการทำงานไม่รู้จบ
25     {                   //ไม่มีการทำงานใด ๆ ในส่วนนี้
26     }
27 }                         //สิ้นสุดขอบเขตของฟังก์ชันหลัก
```



# ข้อสังเกตของเรจิสเตอร์ Timer Interrupt Flag

◆ เรจิสเตอร์ของชิพ AVR ต่างเบอร์กันจะมีชื่อไม่เหมือนกัน

◆ ATMEGA32 ใช้ TIFR

| Bit           | 7    | 6    | 5    | 4     | 3     | 2    | 1    | 0    |      |
|---------------|------|------|------|-------|-------|------|------|------|------|
|               | OCF2 | TOV2 | ICF1 | OCF1A | OCF1B | TOV1 | OCF0 | TOV0 | TIFR |
| Read/Write    | R/W  | R/W  | R/W  | R/W   | R/W   | R/W  | R/W  | R/W  |      |
| Initial Value | 0    | 0    | 0    | 0     | 0     | 0    | 0    | 0    |      |

◆ ATMEGA328P ใช้ TIFR0 และ TIFR1

| Bit           | 7 | 6 | 5 | 4 | 3 | 2     | 1     | 0    |       |
|---------------|---|---|---|---|---|-------|-------|------|-------|
| 0x15 (0x35)   | — | — | — | — | — | OCF0B | OCF0A | TOV0 | TIFR0 |
| Read/Write    | R | R | R | R | R | R/W   | R/W   | R/W  |       |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0     | 0     | 0    |       |

| Bit           | 7 | 6 | 5    | 4 | 3 | 2     | 1     | 0    |       |
|---------------|---|---|------|---|---|-------|-------|------|-------|
| 0x16 (0x36)   | — | — | ICF1 | — | — | OCF1B | OCF1A | TOV1 | TIFR1 |
| Read/Write    | R | R | R/W  | R | R | R/W   | R/W   | R/W  |       |
| Initial Value | 0 | 0 | 0    | 0 | 0 | 0     | 0     | 0    |       |





# ฉบับที่ 4

