Lecture 6: Timers and Interrupts

Dr. Sarah Harris

CpE 301

Embedded System Design

February 7, 2017

Today’s Topics

- Timers
- Interrupts

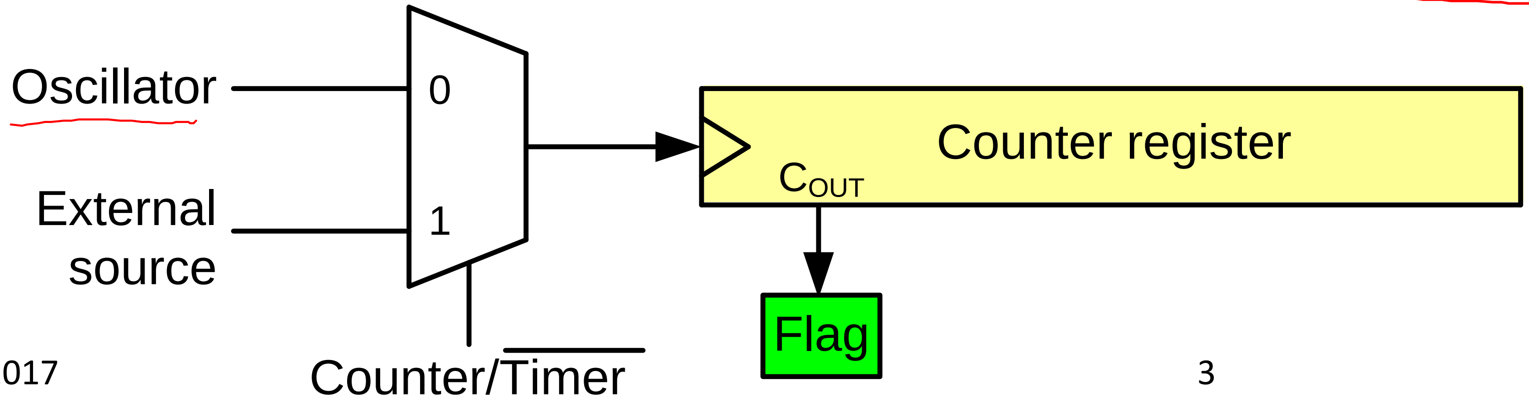
A generic timer/counter

- A timer is a counter
- Increments at each clock edge
- Clock source is either **internal oscillator** or **external source**
- When the counter overflows (i.e., $C_{out} = 1$), the overflow flag for that timer is set

(polling)

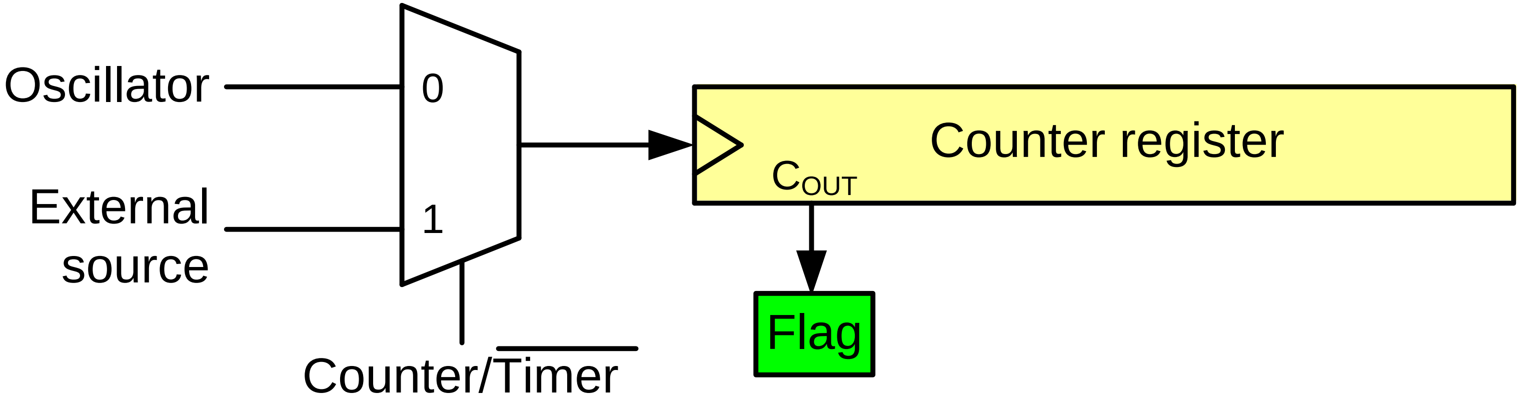
- A program can check the overflow flag manually or the flag can trigger an ***interrupt***

Note: the internal oscillator on ATmega328p is 8 MHz



Timer Uses

- Counting ✓
- Measuring a delay ✓
- Generating a waveform ✓
- “Capturing” a value (setting a flag when the timer reaches a certain value)



Timers in ATmega328p

- Two 8-bit timers
- One 16-bit timer

Timers in ATmega328p

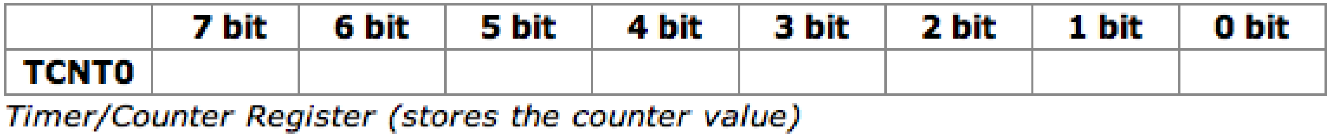
- Two 8-bit timers (Timer 0 and Timer 2)
- One 16-bit timer (Timer 1)

TIMER 0

Timer 0 Registers

- **TCNT0:** Value of the timer
- **TCCR0A:** Timer/counter control register 0 A
- **TCCR0B:** Timer/counter control register 0 B
- **TIFR0:** Timer flags register 0

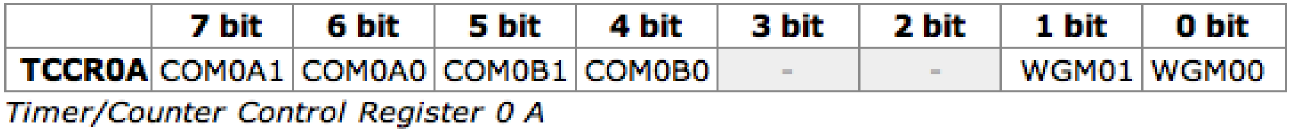
Timer/Counter 0 Register



	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCNT0								

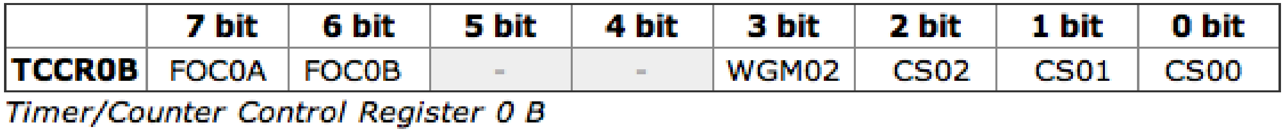
Timer/Counter Register (stores the counter value)

Timer 0 Control Registers



	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00

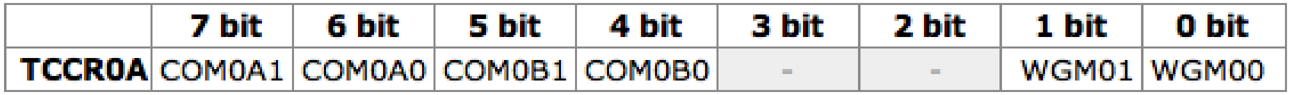
Timer/Counter Control Register 0 A



	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCCR0B	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00

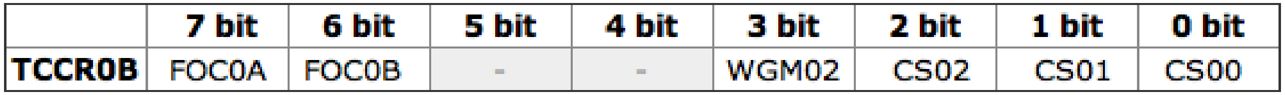
Timer/Counter Control Register 0 B

Timer 0 Control Registers



	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00

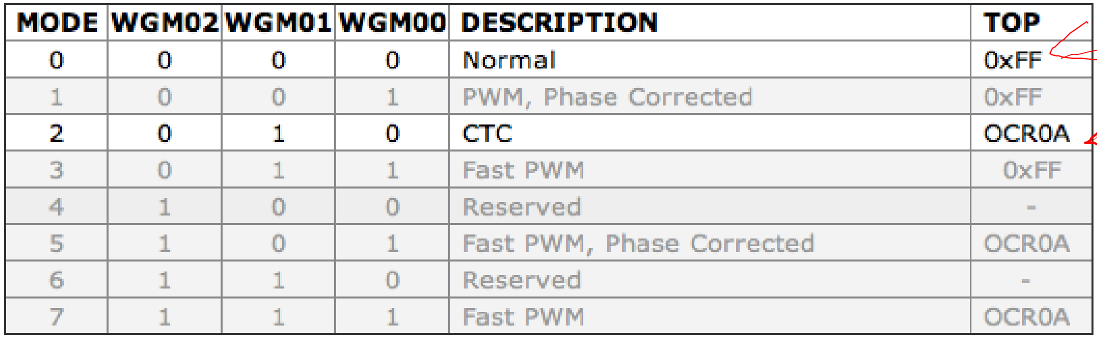
Timer/Counter Control Register 0 A



	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCCR0B	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00

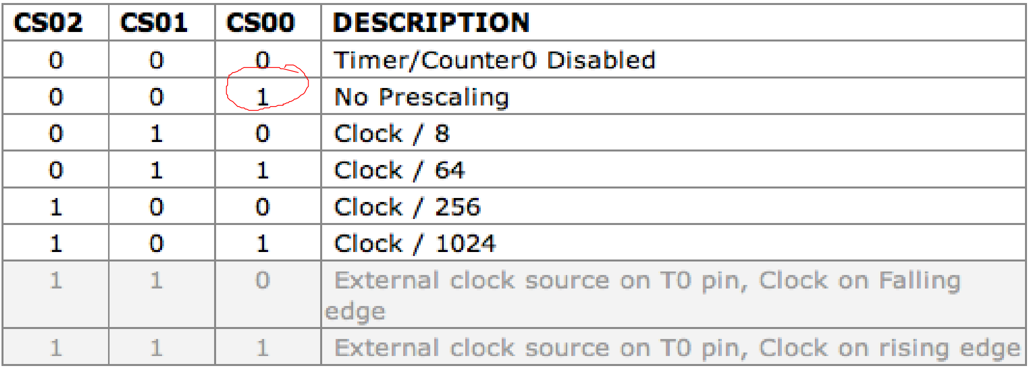
Timer/Counter Control Register 0 B

Tdelay



MODE	WGM02	WGM01	WGM00	DESCRIPTION	TOP
0	0	0	0	Normal	0xFF
1	0	0	1	PWM, Phase Corrected	0xFF
2	0	1	0	CTC	OCR0A
3	0	1	1	Fast PWM	0xFF
4	1	0	0	Reserved	-
5	1	0	1	Fast PWM, Phase Corrected	OCR0A
6	1	1	0	Reserved	-
7	1	1	1	Fast PWM	OCR0A

Waveform Generator Mode bits



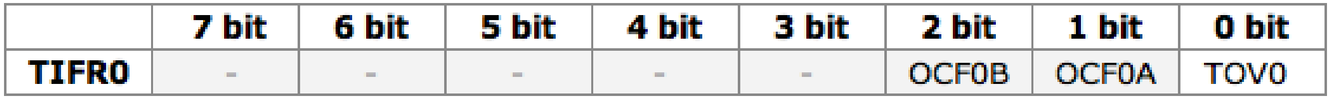
CS02	CS01	CS00	DESCRIPTION
0	0	0	Timer/Counter0 Disabled
0	0	1	No Prescaling
0	1	0	Clock / 8
0	1	1	Clock / 64
1	0	0	Clock / 256
1	0	1	Clock / 1024
1	1	0	External clock source on T0 pin, Clock on Falling edge
1	1	1	External clock source on T0 pin, Clock on rising edge

CS bits

f/8 = 1MHz

f/64 ...

Timer 0 Flags Register



	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TIFR0	-	-	-	-	-	OCF0B	OCF0A	TOV0

Timer/Counter Interrupt Flag Register

Timer Overflow for Timer 0

TOV0 = 1

when 0xFF → 0x100

Timer 0 Registers

- **TCNT0:** **Value** of the timer
- **TCCR0A:** Timer/counter **control register** 0 A
- **TCCR0B:** Timer/counter **control register** 0 B
- **TIFR0:** Timer **flags** register 0

Example 0

```
     ; Set up Timer0 and wait for overflow

     ; initialize Timer0
     LDI   R18,0        ;R18 = 0
     OUT   TCNT0,R18    ;load timer0 = 0                    1
     OUT   TCCR0A,R18   ;Timer0: normal mode, internal clock   2
     LDI   R17,0x1      ;Timer0: enabled, no prescalar    3
     OUT   TCCR0B, R17

     ; wait for overflow     4
AGAIN:
     IN    R20,TIFR0    ;read Timer0 flags register
     SBRS  R20,0        ;if overflow (TOV0) is set skip next instruction
     RJMP  AGAIN

     ; disable Timer0 and clear overflow flag (TOV0)
     LDI   R20,0x0      ;stop/disable Timer0
     OUT   TCCR0B,R20
     LDI   R20,0x1      ;clear Timer0 overflow flag (TOV0)
     OUT   TIFR0,R20
     ...
```

← write 1 to TOV0 to clear

Example 1

```
     ; Toggle PORTB.5 every time Timer0 overflows
     SBI   DDRB,5       ;PB.5 as an output
     LDI   R18,0        ;PB.5 = 0
     LDI   R16,0x20     ;R16 = 0x20
     OUT   PORTB,R18
BEGIN:
     OUT   TCNT0,R18    ;load timer0 = 0
     OUT   TCCR0A,R18   ;Timer0: normal mode, internal clock
     LDI   R17,0x1      ;Timer0: enabled, no prescalar
     OUT   TCCR0B, R17
AGAIN:
     IN    R20,TIFR0    ;read Timer0 flags register
     SBRS  R20,0        ;if overflow (TOV0) is set skip next instruction
     RJMP  AGAIN
     LDI   R20,0x0      ;stop/disable Timer0
     OUT   TCCR0B,R20
     LDI   R20,0x1      ;clear Timer0 overflow flag (TOV0)
     OUT   TIFR0,R20
     EOR   R18,R16      ;toggle bit 5 of R18
     OUT   PORTB,R18    ;toggle PB.5
     RJMP  BEGIN
```

0010 0000

bit 7 6 5 0

Example 1

```
     ; Toggle PORTB.5 every time Timer0 overflows
     SBI   DDRB,5       ;PB.5 as an output
     LDI   R18,0        ;PB.5 = 0
     LDI   R16,0x20     ;R16 = 0x20
     OUT   PORTB,R18
BEGIN:
     OUT   TCNT0,R18    ;load timer0 = 0
     OUT   TCCR0A,R18   ;Timer0: normal mode, internal clock
     LDI   R17,0x1      ;Timer0: enabled, no prescalar
     OUT   TCCR0B, R17
AGAIN:
     IN    R20,TIFR0    ;read Timer0 flags register
     SBRS  R20,0        ;if overflow (TOV0) is set skip next instruction
     RJMP  AGAIN
     LDI   R20,0x0      ;stop/disable Timer0
     OUT   TCCR0B,R20
     LDI   R20,0x1      ;clear Timer0 overflow flag (TOV0)
     OUT   TIFR0,R20
     EOR   R18,R16      ;toggle bit 5 of R18
     OUT   PORTB,R18    ;toggle PB.5
     RJMP  BEGIN
```

How often does PB.5 toggle? (assume 10 MHz clock)

Example 1

```
    ; Toggle PORTB.5 every time Timer0 overflows
    SBI   DDRB,5        ;PB.5 as an output
    LDI   R18,0         ;PB.5 = 0
    LDI   R16,0x20      ;R16 = 0x20
    OUT   PORTB,R18
BEGIN:
    OUT   TCNT0,R18     ;load timer0 = 0
    OUT   TCCR0A,R18    ;Timer0: normal mode, internal clock
    LDI   R17,0x1       ;Timer0: enabled, no prescalar
    OUT   TCCR0B, R17
AGAIN:
    IN    R20,TIFR0     ;read Timer0 flags register
    SBRS  R20,0         ;if overflow (TOV0) is set skip next instruction
    RJMP  AGAIN
    LDI   R20,0x0       ;stop/disable Timer0
    OUT   TCCR0B,R20
    LDI   R20,0x1       ;clear Timer0 overflow flag (TOV0)
    OUT   TIFR0,R20
    EOR   R18,R16       ;toggle bit 5 of R18
    OUT   PORTB,R18     ;toggle PB.5
    RJMP  BEGIN
```

How often does PB.5 toggle? (assume 10 MHz clock):

T_c = 1/10MHz = 0.1μs; 0x100*0.1μs = 25.6 μs

Example 1

```
          ; Toggle PORTB.5 every time Timer0 overflows
          SBI   DDRB,5       ;PB.5 as an output
          LDI   R18,0        ;PB.5 = 0
          LDI   R16,0x20     ;R16 = 0x20
          OUT   PORTB,R18
      BEGIN:
1         LDI   R18,0        ;PB.5 = 0
1         OUT   TCNT0,R18    ;load timer0 = 0
1         OUT   TCCR0A,R18   ;Timer0: normal mode, internal clock
1         LDI   R17,0x1      ;Timer0: enabled, no prescalar
1         OUT   TCCR0B, R17
      AGAIN:
1         IN    R20,TIFR0    ;read Timer0 flags register
(1)2      SBRS  R20,0        ;if overflow (TOV0) is set skip next instruction
(2)       RJMP  AGAIN
1         LDI   R20,0x0      ;stop/disable Timer0
1         OUT   TCCR0B,R20
1         LDI   R20,0x1      ;clear Timer0 overflow flag (TOV0)
1         OUT   TIFR0,R20
1         EOR   R18,R16      ;toggle bit 5 of R18
1         OUT   PORTB,R18    ;toggle PB.5
2         RJMP  BEGIN
```

How often does PB.5 toggle? (assume 10 MHz clock):

More accurate: (0x100+16)*0.1μs = 27.2 μs

Example 0 in C

```
// Set up Timer0 and wait for overflow

int main(void) {
    // initialize Timer0
    TCNT0  = 0;       // load timer0 = 0
    TCCR0A = 0;       // Timer0: normal mode, internal clock
    TCCR0B = 1;       // Timer0: enabled, no prescalar

    // wait for overflow
    while ( (TIFR0 & 0x1) == 0 ) ;

    // disable Timer0 and clear overflow flag (TOV0)
    TCCR0B = 0;       // stop/disable Timer0
    TIFR0  = 1;       // clear Timer0 overflow flag (TOV0)
    ...
}
```

Example 0: Alternate Code

(1<<CS00) (1<<CS01

```
     ; Set up Timer0 and wait for overflow

     ; initialize Timer0
     LDI   R18,0             ;R18 = 0
     OUT   TCNT0,R18         ;load timer0 = 0
     OUT   TCCR0A,R18        ;Timer0: normal mode, internal clock
     ;LDI R17, 0x1
     LDI   R17,(1<<CS00)     ;Timer0: enabled, no prescalar
     OUT   TCCR0B, R17

     ; wait for overflow
AGAIN:
     IN    R20,TIFR0     ;read Timer0 flags register
     SBRS  R20,0         ;if overflow (TOV0) is set skip next instruction
     RJMP  AGAIN

     ; disable Timer0 and clear overflow flag (TOV0)
     LDI   R20,0x0           ;stop/disable Timer0
     OUT   TCCR0B,R20
     ;LDI  R20,0x1           ;clear Timer0 overflow flag (TOV0)
     LDI   R20,(1<<TOV0)     ;clear Timer0 overflow flag (TOV0)
     OUT   TIFR0,R20
     ...
```

Example 0 in C: Alternate Code

```
// Set up Timer0 and wait for overflow

int main(void) {
    // initialize Timer0
    TCNT0  = 0;               // load timer0 = 0
    TCCR0A = 0;               // Timer0: normal mode, internal clock
    //  TCCR0B = 1;           // Timer0: enabled, no prescalar
    TCCR0B = (1 << CS00);     // Timer0: enabled, no prescalar


    // wait for overflow
    // while ( (TIFR0 & 0x1) == 0 )  ;
    while ( (TIFR0 & (1<<TOV0)) == 0 )  ;

    // disable Timer0 and clear overflow flag (TOV0)
    TCCR0B = 0;               // stop/disable Timer0
    // TIFR0  = 1;            // clear Timer0 overflow flag (TOV0)
    TIFR0  &= (1<<TOV0);      // clear Timer0 overflow flag (TOV0)
    ...
}
```

Example 2

Task: Toggle PB.5 every 15 µs. (assume 10 MHz clock)

$$T_c = \frac{1}{10\,\text{MHz}} = 0.1\,\mu s$$

$$15\,\mu s \cdot \frac{1\,\text{c.c}}{0.1\,\mu s} = \boxed{150\,\text{c.c.}}$$

c.c. = clock cycle

$$\text{initialize: } (256 - 150) = \boxed{106}$$

Example 2

Task: Toggle PB.5 every 15 µs. (assume 10 MHz clock)

How many clock cycles in 15 µs?

0.1 µs/clock cycle (Tc = 1/f = 1/10 MHz = 0.1 µs)
15 µs * (1 clock cycle/0.1 µs) = **150 clock cycles**

So… preload Timer0 with 0x100-150 = 256-150 (= **106**).
When it overflows 150 cycles will have passed

Example 2

Task: Toggle PB.5 every 15 µs. (assume 10 MHz clock)

```
// Set up Timer0 and wait for overflow
int main(void)
    DDRB  |=  (1<<5)          // PB.5 as an output
    PORTB &= ~(1<<5)          // PB.5 = 0
    while (1) {
        // initialize Timer0
        TCNT0  = 106;         // load timer0 = 106
        TCCR0A = 0;           // Timer0: normal mode, internal clock
        TCCR0B = (1 << CS00); // Timer0: enabled, no prescalar

        // wait for overflow
        while ( (TIFR0 & (1<<TOV0)) == 0 ) ;

        // disable Timer0 and clear overflow flag (TOV0)
        TCCR0B = 0;           // stop/disable Timer0
        TIFR0  &= (1<<TOV0);  // clear Timer0 overflow flag (TOV0)

        // toggle PB.5
        PORTB ^= (1<<5);
    }
}
```

Example 2

Task: Toggle PB.5 every 15 μs. (assume 10 MHz clock)

```
    ; Toggle PORTB.5 every 15 μs.
    SBI   DDRB,5          ;PB.5 as an output
    LDI   R18,0           ;PB.5 = 0
    LDI   R16,0x20        ;R16 = 0x20
    OUT   PORTB,R18
BEGIN:
    LDI   R19, 106        ;load Timer0 = 106
    OUT   TCNT0,R19
    OUT   TCCR0A,R18      ;Timer0: normal mode, internal clock
    LDI   R17,(1<<CS00)   ;Timer0: enabled, no prescalar
    OUT   TCCR0B, R17
AGAIN:
    IN    R20,TIFR0       ;read Timer0 flags register
    SBRS  R20,0           ;if overflow (TOV0) is set skip next instruction
    RJMP  AGAIN
    LDI   R20,0x0         ;stop/disable Timer0
    OUT   TCCR0B,R20
    LDI   R20,(1<<TOV0)   ;clear Timer0 overflow flag (TOV0)
    OUT   TIFR0,R20
    EOR   R18,R16         ;toggle bit 5 of R18
    OUT   PORTB,R18       ;toggle PB.5
    RJMP  BEGIN
```

Example 2

Task: Toggle PB.5 every 15 µs. (assume 10 MHz clock)

```
           ; Toggle PORTB.5 every 15 µs.
           SBI   DDRB,5           ;PB.5 as an output
           LDI   R18,0            ;PB.5 = 0
           LDI   R16,0x20         ;R16 = 0x20
           OUT   PORTB,R18
       BEGIN:
1          LDI   R19, 106         ;load Timer0 = 106
1          OUT   TCNT0,R19
1          OUT   TCCR0A,R18       ;Timer0: normal mode, internal clock
1          LDI   R17,(1<<CS00)    ;Timer0: enabled, no prescalar
1          OUT   TCCR0B, R17
       AGAIN:
1          IN    R20,TIFR0        ;read Timer0 flags register
(1)2       SBRS  R20,0            ;if overflow (TOV0) is set skip next instruction
(2)        RJMP  AGAIN
1          LDI   R20,0x0          ;stop/disable Timer0
1          OUT   TCCR0B,R20
1          LDI   R20,(1<<TOV0)    ;clear Timer0 overflow flag (TOV0)
1          OUT   TIFR0,R20
1          EOR   R18,R16          ;toggle bit 5 of R18
1          OUT   PORTB,R18        ;toggle PB.5
2          RJMP  BEGIN
```

Example 2

Task: Toggle PB.5 every 15 µs. (assume 10 MHz clock)

- Wanted to toggle every 15 µs (Tc = 30 µs, f = 33 kHz)
- But really toggled every:

  (150 + 16) clock cycles = 166 c.c.

  = 16.6 µs
- So, real Tc = 33.2 µs, f = 30.1 kHz

Example 2

Task: Toggle PB.5 every 15 µs. (assume 10 MHz clock)

- Wanted to toggle every 15 µs (Tc = 30 µs, f = 33 kHz)
- But really toggled every:

  (150 + 16) clock cycles = 166 c.c.

  = 16.6 µs
- So, real Tc = 33.2 µs, f = 30.1 kHz

How to fix this? (i.e., make it more accurate?)

Example 2

Task: Toggle PB.5 every 15 µs. (assume 10 MHz clock)

- Wanted to toggle every 15 µs (Tc = 30 µs, f = 33 kHz)
- But really toggled every:

  (150 + 16) clock cycles = 166 c.c.

  = 16.6 µs
- So, real Tc = 33.2 µs, f = 30.1 kHz

How to fix this? (i.e., make it more accurate?)

- Preload Timer so that it counts only (150-16 = **134** clock cycles) before overflowing. So load counter with: 256-134 = **122**

Example 2: more accurate

Task: Toggle PB.5 every 15 µs. (assume 10 MHz clock)

```
           ; Toggle PORTB.5 every 15 µs.
           SBI   DDRB,5            ;PB.5 as an output
           LDI   R18,0             ;PB.5 = 0
           LDI   R16,0x20          ;R16 = 0x20
           OUT   PORTB,R18
       BEGIN:
1          LDI   R19, 122          ;load Timer0 = 122
1          OUT   TCNT0,R19
1          OUT   TCCR0A,R18        ;Timer0: normal mode, internal clock
1          LDI   R17,(1<<CS00)     ;Timer0: enabled, no prescalar
1          OUT   TCCR0B, R17
       AGAIN:
1          IN    R20,TIFR0         ;read Timer0 flags register
(1)2       SBRS  R20,0             ;if overflow (TOV0) is set skip next instruction
(2)        RJMP  AGAIN
1          LDI   R20,0x0           ;stop/disable Timer0
1          OUT   TCCR0B,R20
1          LDI   R20,(1<<TOV0)     ;clear Timer0 overflow flag (TOV0)
1          OUT   TIFR0,R20
1          EOR   R18,R16           ;toggle bit 5 of R18
1          OUT   PORTB,R18         ;toggle PB.5
2          RJMP  BEGIN
```

Finding value to load into timer

1. Calculate the period of clock source.
   - Period = cycle time (T_c) = 1 / Frequency
     - E.g. For XTAL = 8 MHz ➔ T = 1/8MHz = 0.125us
     - E.g. For XTAL = 10 MHz ➔ T = 1/10 MHz = 0.1us
2. Divide the desired time delay by period of clock.
3. Perform 256 - n, where n is the decimal value from Step 2.
4. Set TCNT0 = 256 - n

Generating Large Delays

- Loops
- Prescaler
- Bigger counters

Example 3

Task: Toggle PB.5 every 1 second. (assume 8 MHz clock)

Solve using:

- Loops
- Prescaler
- Bigger counters

Example 3

Task: Toggle PB.5 every 1 second. (assume 10 MHz clock)

Solve using:

- **Loops**
- Tc = 1/10 MHz = 0.1 µs
- Each overflow event: 256*0.1 µs = 2.56 µs
- How many overflow events: 1 second/2.56 µs = 39062.5
- If choose **39062**: toggles every (39062*2.56 µs) = 0.9999872 seconds (Cycle time of PB.5 oscillation is twice that)
- Also need to take account of code overhead

Example 3

Task: Toggle PB.5 every 1 second. (assume 10 MHz clock)

Solve using loops:

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

int main(void) {
    DDRB = (1<<5);
    PORTB = (1<<5);// PB.5 = 1

    // initialize Timer0
    TCNT0  = 0;// load timer0 = 0
    TCCR0A = 0;// Timer0: normal mode, internal clock
    TCCR0B = 1;// Timer0: enabled, no prescalar
    while (1) {
        for (unsigned int i=0; i<39062; i++) {
          while ( (TIFR0 & 0x1) == 0 ) ; // wait for overflow
          TIFR0  = 1; // clear overflow flag (TOV0)
        }
        PORTB ^= (1<<5);  // Toggle PB.5
    }
}
```

Example 4

Task: Toggle PB.5 every 1 ms. (assume 10 MHz clock)

Solve using:

- Loops
- Prescaler
- Bigger counters

Example 4

Task: Toggle PB.5 every 1 ms. (assume 10 MHz clock)

Using Loops

- 1 ms / (0.1 µs/clock cycle) = 1 x 10^4 clock cycles
- 1 x 10^4 clock cycles / (256 +16 c.c./overflow event) = 36.7… overflow events
- So, 36 overflow events + 208 c.c.
- With 16 c.c. overhead: 36 overflow events + (208-16=192 c.c.). So for last iteration, load Timer0 with 256-192 = 64

Example 4

Task: Toggle PB.5 every 1 ms. (assume 10 MHz clock)

```
// Set up Timer0 and wait for overflow
int main(void) {
    DDRB  |=  (1<<5);           // PB.5 as an output
    PORTB &= ~(1<<5);           // PB.5 = 0
    while (1) {
        for (int i=0; i < 39; i++) {
            // initialize Timer0
            TCNT0  = 0;         // load timer0 = 0
            TCCR0A = 0;         // normal mode, internal clock
            TCCR0B = (1 << CS00); // enabled, no prescalar

            // wait for overflow
            while ( (TIFR0 & (1<<TOV0)) == 0 ) ;

            // disable Timer0 and clear overflow flag (TOV0)
            TCCR0B = 0;           // stop/disable Timer0
            TIFR0   &= (1<<TOV0); // clear Timer0 overflow flag
        }
        ... // finish up last few clock cycles (64)
```

Example 4

Task: Toggle PB.5 every 1 ms. (assume 10 MHz clock)

```
        ...
        // initialize Timer0
        TCNT0  = 64;     // load timer0 = 64
        TCCR0A = 0;      // normal mode, internal clock
        TCCR0B = (1 << CS00);// enabled, no prescalar

        // wait for overflow
        while ( (TIFR0 & (1<<TOV0)) == 0 ) ;

        // disable Timer0 and clear overflow flag (TOV0)
        TCCR0B = 0;          // stop/disable Timer0
        TIFR0  &= (1<<TOV0); // clear Timer0 overflow flag

        // toggle PB.5
        PORTB ^= (1<<5);
    }
}
```

Example 4

Task: Toggle PB.5 every 1 ms. (assume 10 MHz clock)

Solve using:

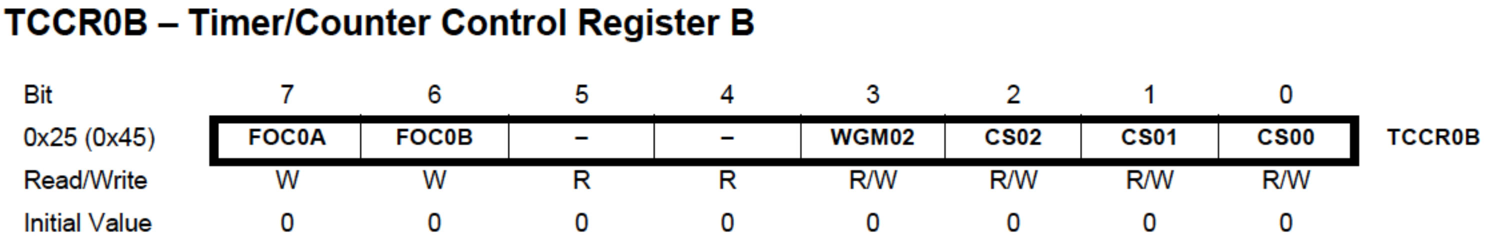
- Loops
- **Prescaler**
- Bigger counters

Example 4

Task: Toggle PB.5 every 1 ms. (assume 10 MHz clock)

Using Prescalar

Configuration Register B



TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

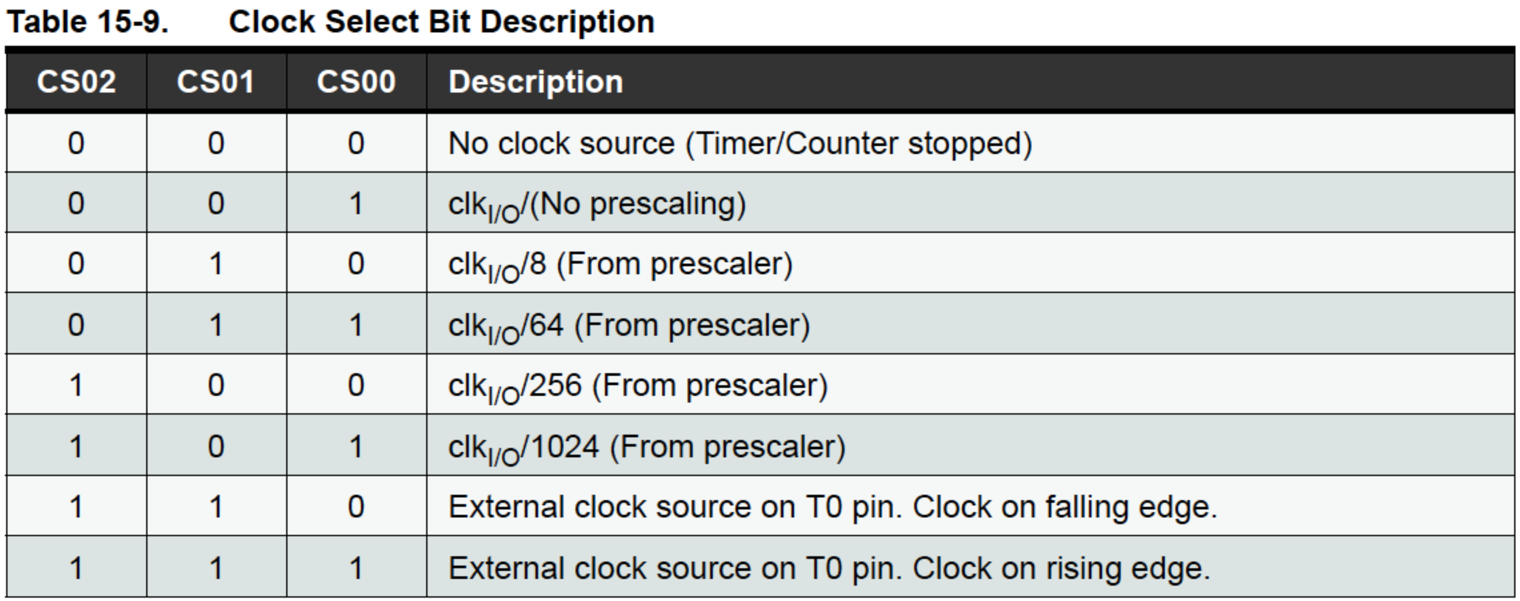


Table 15-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Example 4

Task: Toggle PB.5 every 1 ms. (assume 10 MHz clock)

Using Prescalar

- For example, use prescalar = 1024: f = 10 MHz/1024
  = 9.765 kHz

- Now, Tc = 1/9.765 kHz = 0.102 ms
- Number of c.c. in 1 ms is ~10 c.c.
- Could use a smaller prescalar…

Example 4

Task: Toggle PB.5 every 1 ms. (assume 10 MHz clock)

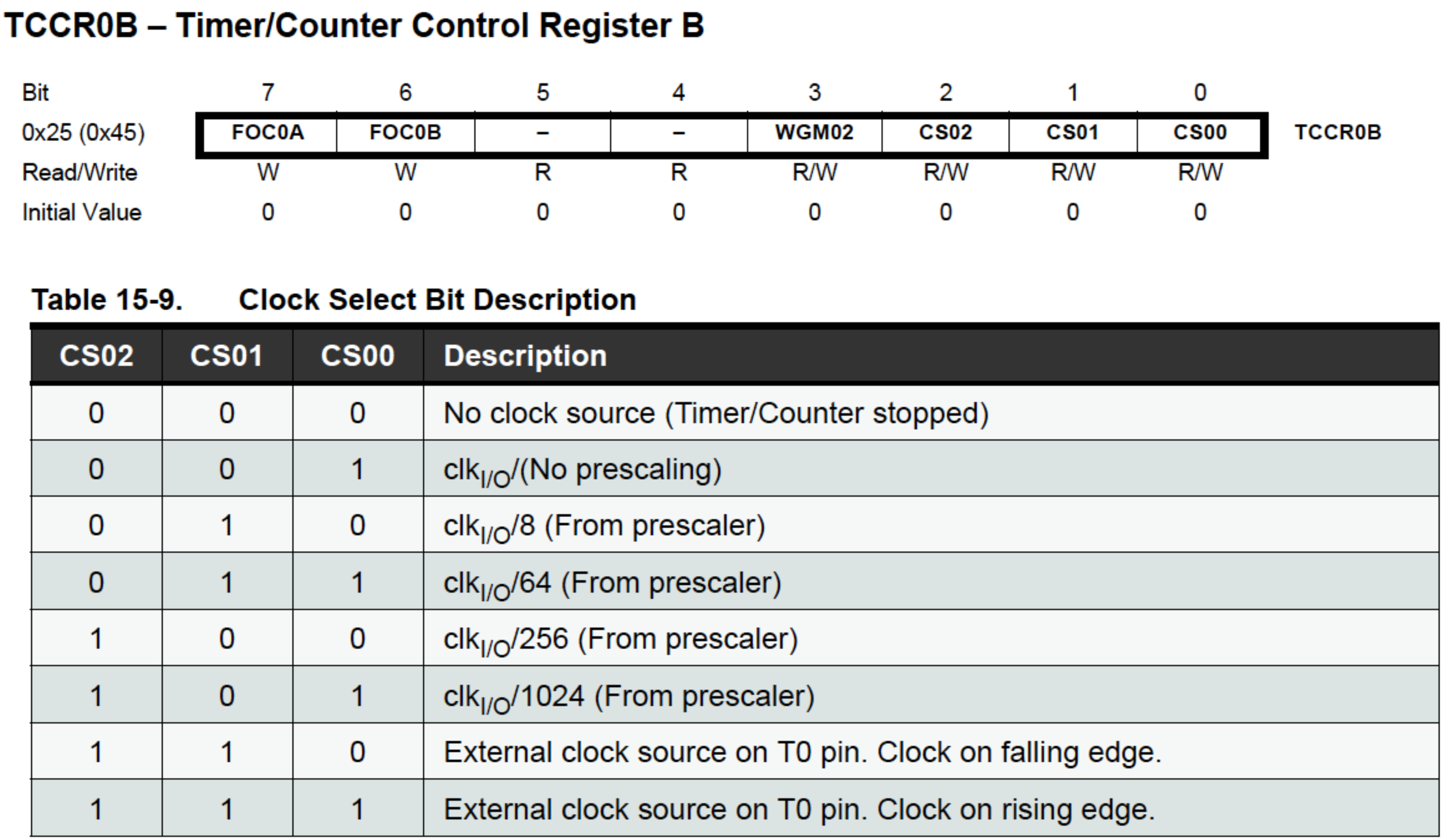
Using Prescalar

- For example, use prescalar = 64: f = 10 MHz/64
  = 156 kHz

- Now, Tc = 1/156 kHz = 6.4 µs
- Number of c.c. in 1 ms is: 1 ms/(6.4 µs/c.c.) = 156.25 c.c.

- So, set prescalar to **64** and Timer to: 256 – (156-16/64) = **100**

Configuration Register B



TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 15-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Example 4

Task: Toggle PB.5 every 1 ms. (assume 10 MHz clock)

Using Prescalar

```
// Set up Timer0 and wait for overflow
int main(void) {
    DDRB  |=  (1<<5);         // PB.5 as an output
    PORTB &= ~(1<<5);         // PB.5 = 0
    while (1) {
        // initialize Timer0
        TCNT0  = 0;           // load timer0 = 00
        TCCR0A = 0;           // Timer0: normal mode, internal clock
        // Timer0: enabled, prescalar = 64
        TCCR0B = (1 << CS01) | (1 << CS00);

        // wait for overflow
        while ( (TIFR0 & (1<<TOV0)) == 0 ) ;

        // disable Timer0 and clear overflow flag (TOV0)
        TCCR0B = 0;           // stop/disable Timer0
        TIFR0  &= (1<<TOV0);  // clear Timer0 overflow flag (TOV0)

        // toggle PB.5
        PORTB ^= (1<<5);
    }
}
```

Example 3

Task: Toggle PB.5 every 1 ms. (assume 10 MHz clock)

Solve using:

- Loops
- Prescaler
- **Bigger counters**

Example 3

Task: Toggle PB.5 every 1 ms. (assume 10 MHz clock)
Solve using **bigger counter: Timer 1 is 16-bits**

- 1 ms / (0.1 µs/clock cycle) = 1 x 10^4 clock cycles
- 10^4 = 0x2710 fits in 16 bits
- Initialize Timer 1 with 2^16 – 0x2710:
  0x10000-0x2710 = **0xD8F0**
- Again, could adjust this (add to it) to account for instruction overhead

Example 3

Task: Toggle PB.5 every 1 ms. (assume 10 MHz clock)

Solve using **bigger counter: Timer 1 is 16-bits**

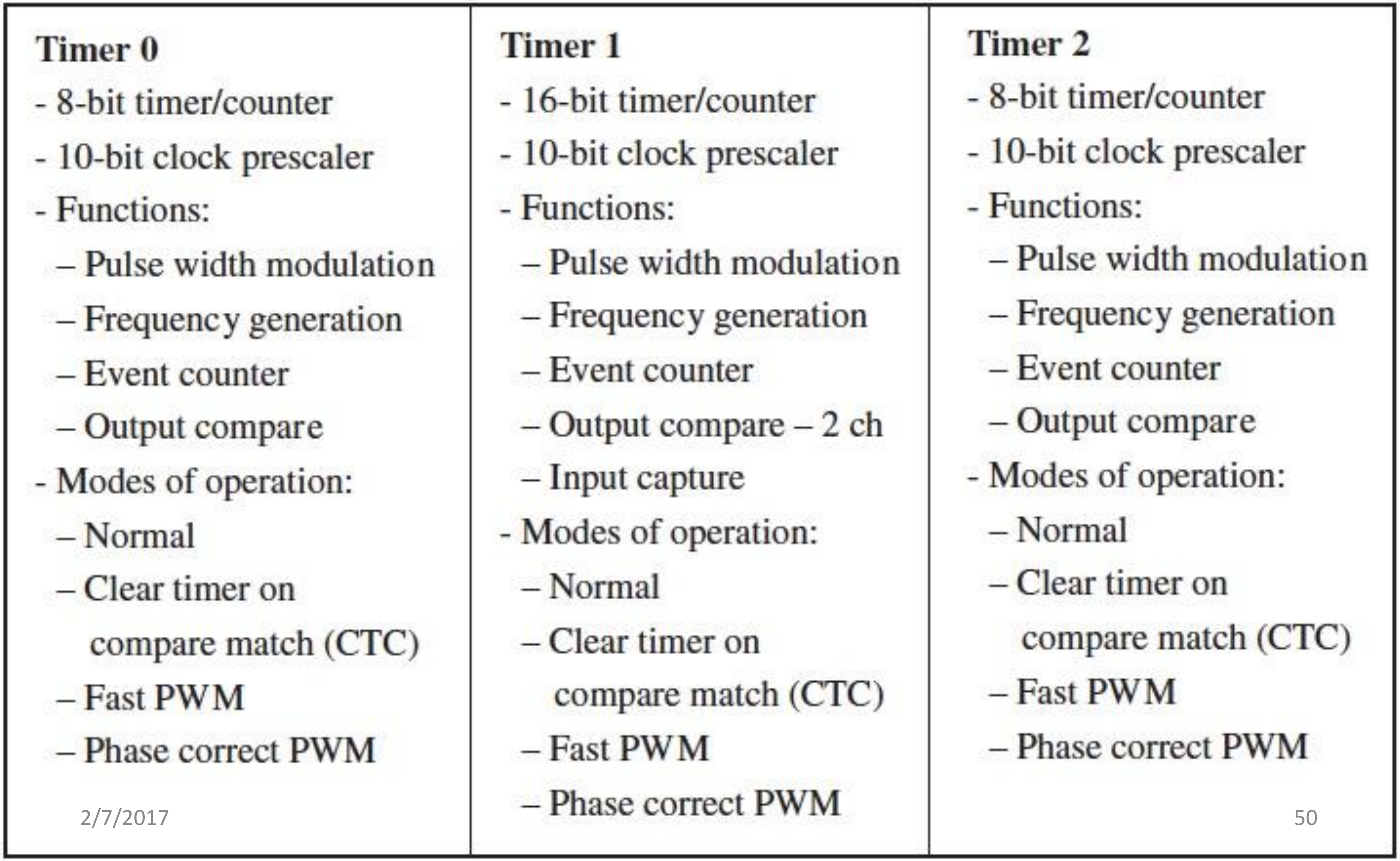
```
#include <avr/io.h>

// Set up Timer0 and wait for overflow
int main(void) {
  DDRB  |=  (1<<5);// PB.5 as an output
  PORTB &= ~(1<<5);// PB.5 = 0
  while (1) {
    // initialize Timer1
    TCNT1H = 0xD8;    // load Timer1 = 0xD8F0
    TCNT1L = 0xF0;
    TCCR1A = 0;    // Timer1: normal mode, internal clock
    TCCR1B = (1 << CS10);// Timer1: enabled, no prescalar

    // wait for overflow
    while ( (TIFR1 & (1<<TOV1)) == 0 ) ;
    TIFR1  &= (1<<TOV1);     // clear Timer1 overflow flag (TOV1)

    // toggle PB.5
    PORTB ^= (1<<5);
  }
}
```

Summary of Timers



Timer 0	Timer 1	Timer 2
- 8-bit timer/counter - 10-bit clock prescaler - Functions: – Pulse width modulation – Frequency generation – Event counter – Output compare - Modes of operation: – Normal – Clear timer on compare match (CTC) – Fast PWM – Phase correct PWM	- 16-bit timer/counter - 10-bit clock prescaler - Functions: – Pulse width modulation – Frequency generation – Event counter – Output compare – 2 ch – Input capture - Modes of operation: – Normal – Clear timer on compare match (CTC) – Fast PWM – Phase correct PWM	- 8-bit timer/counter - 10-bit clock prescaler - Functions: – Pulse width modulation – Frequency generation – Event counter – Output compare - Modes of operation: – Normal – Clear timer on compare match (CTC) – Fast PWM – Phase correct PWM

The difference between Timer0 and Timer2

- Timer0

CS02	CS01	CS00	Comment
0	0	0	Timer/Counter stopped
0	0	1	clk (No Prescaling)
0	1	0	clk / 8
0	1	1	clk / 64
1	0	0	clk / 256
1	0	1	clk / 1024
1	1	0	External clock (falling edge)
1	1	1	External clock (rising edge)

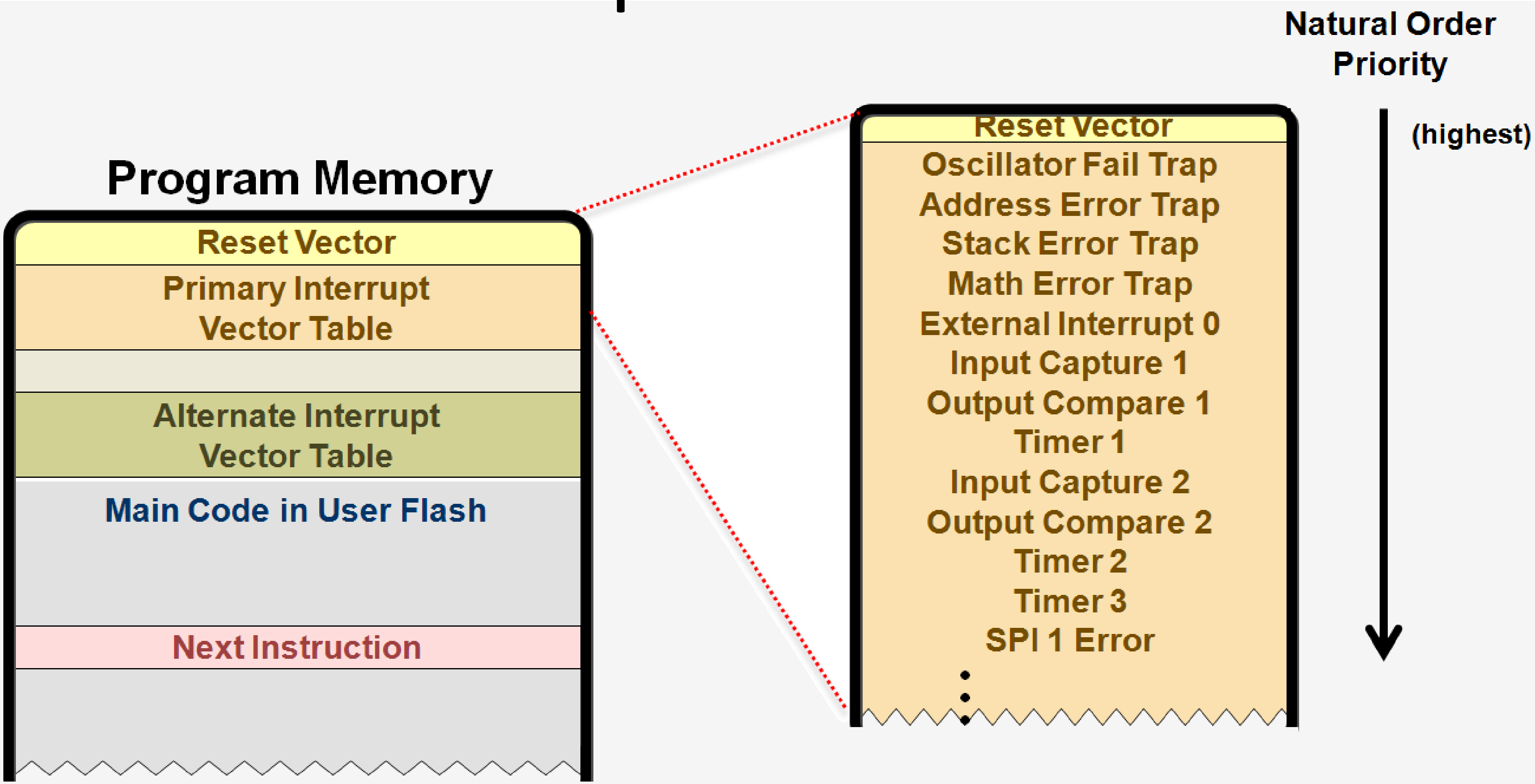
- Timer2

CS22	CS21	CS20	Comment
0	0	0	Timer/Counter stopped
0	0	1	clk (No Prescaling)
0	1	0	clk / 8
0	1	1	clk / 32
1	0	0	clk / 64
1	0	1	clk / 128
1	1	0	clk / 256
1	1	1	clk / 1024

Interrupts

- **Polling vs interrupts:**
    - **Polling:** checking a flag/variable until set
    - **Interrupt:** triggers “call” to function when flag is set. These functions are called: **interrupt service routines** (ISRs)
- Polling wastes valuable processor time

Interrupt Vector Table



http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html

Interrupt

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

// this ISR is run whenever a match occurs
ISR (TIMER1_COMPA_vect) {
     PORTB ^= (1 << 5);
}

int main(void) {
  DDRB |= (1 << 5);         // connect led to pin PC0
  // set up timer with prescaler = 64 and CTC mode
  TCCR1B |= (1 << WGM12)|(1 << CS11)|(1 << CS10);
  TCNT1 = 0;                // initialize counter
  OCR1A = 52999;            // initialize compare value
  TIMSK1 |= (1 << OCIE1A); // enable compare interrupt
  sei();                    // enable global interrupts

  // loop forever
  while(1) ;
}
```