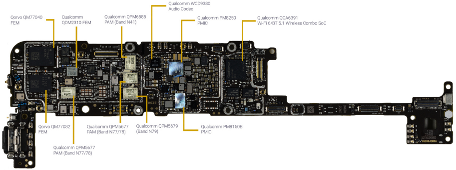
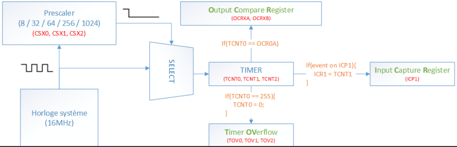


8-bit and 16 bit AVR timers

Kizito NKURIKIYEYEZU, Ph.D.

8-bit timers in the ATMega328

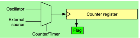
- An ATMega228 has two 8-bit timers/counters (Timer0 and timer2)
- In the general parlance, TCNT0 and TCNT2) are nearly identical.
- the two timers are 8 bit timers—can count from 0 to 255
- The TCNT0 register hold the timer Count and it is incremented on every timer "tick". If the timer is turned on it ticks from 0 to 255 and overflows

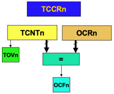


8-bit timer/counter registers

There exists 4 registers: TCNT0, OCR0, TCCR0, ASSR[1], [2].

- TCNT0: (timer/counter register)
  - The 8-bit counter itself
  - Holds the present value of count
  - Upon reset, zero. It counts up with each pulse.
- OCR0: (output compare register): this register is always compared against TCNT0
- TCCR0: (timer/counter 0 control register): determines the mode of operation
- ASSR: (asynchronous status register): coordinates writing to





8 bit timer programming

TCCR0 —Timer/Counter Control Register

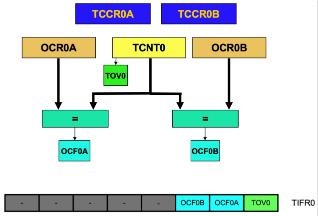


FIG 2. 8-bit registers in TIMER0

The Timer/Counter Register TCNTx

- There are two counter registers for each timer, namely TCNT0 (Timer/Counter 0), TCNT1L, TCNT1H and TCNT2.
- Since the Timer1 is 16 bit, it needs two registers. This lecturer only focuses on Timer0 and 2
- The Timer/Counter Control Registers is responsible for controlling the timer

TAB 1. TCCR0 —Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

FOC0 D7 Force compare match: This is a write-only bit, which can be used while generating a wave. Writing 1 to it causes the wave generator to act as if a compare match had occurred.

WGM00, WGM01

D6	D3	Timer0 mode selector bits
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM, phase correct
1	1	Fast PWM

COM01:00 D5 D4 Compare Output Mode:
These bits control the waveform generator.

CS02:00

D2	D1	D0	Timer0 clock selector
0	0	0	No clock source (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 source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

TAB 2. TCCR2 —Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

FOC2 D7 Force compare match: a write-only bit, which can be used while generating a wave. Writing 1 to it causes the wave generator to act as if a compare match had occurred.

WGM20, WGM21

D6	D3	Timer2 mode selector bits
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM, phase correct
1	1	Fast PWM

COM21:20 D5 D4 Compare Output Mode:
These bits control the waveform generator.

CS22:20

D2	D1	D0	Timer2 clock selector
0	0	0	No clock source (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

CS02:CS00—Timer0 clock source bits

CS02:CS00 (Timer0 clock source): These bits in the TCCR0 register are used to choose the clock source

- If CS02:CS00 = 000, then the counter is stopped.
- If CS02–CS00 have values between 001 and 101, the oscillator is used as clock source and the timer/counter acts as a timer. In this case, the timers are often used for time delay generation

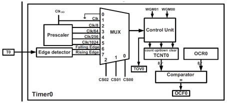


FIG 3. TIMER0 Timer0 clock source

TAB 3. Setting the prescaler with the clock select bits for Timer2

CS22	CS21	CS20	Description
0	0	0	No Clock Source
0	0	1	System Clock
0	1	0	Prescaler = 8
0	1	1	Prescaler = 32
1	0	0	Prescaler = 64
1	0	1	Prescaler = 128
1	1	0	Prescaler = 256
1	1	1	Prescaler = 1024

Resolution

Resolution—Smallest amount of time that a timer can measure. This is the inverse of the timer clock frequency. For example, at 16MHz clock

$$resolution = \frac{1}{16MHz/N} = \frac{N}{16MHz} = N \times 62.5ns \tag{1}$$

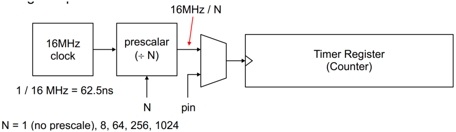


FIG 4. Timer, clock and prescaler

Range

Range—maximum amount of time that a timer can measure.

$$range = resolution \times 2^n \tag{2}$$

(where n is the number of bits in the timer)

TAB 4. Timer resolution and range calculation

Prescale (N)	Resolution	Timer 1 Range	Timer 0 or 2 Range
1	0.0625us	4.096ms	16us
8	0.5us	32.768ms	128us
64	4us	262.144ms	1.024ms
256	16us	1.048576s	4.096ms
1024	64us	4.194304s	16.384ms

Note: Timer 2 has additional prescale values of 32 and 128.

CS02:CS00—Examples

- Find the value for TCCR0 if we want to program Timer0 in Normal mode, no prescaler. Use AVR's crystal oscillator for the clock source.

Solution:

TCCR0 =	0	0	0	0	0	0	0	1
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

- Find the timer's clock frequency and its period for various AVR-based systems, with the following crystal frequency of (A) 10MHz, (B) 8MHz and (C) 1MHz. Assume that no prescaler is used.

Solution:

(a) F = 10 MHz and T = 1/10 MHz = 0.1 µs
(b) F = 8 MHz and T = 1/8 MHz = 0.125 µs
(c) F = 1 MHz and T = 1/1 MHz = 1 µs

WGM01:00 —working mode bits

TIMER0 can work in 4 different modes

- Normal, ($WGM00 : WGM01 = 00$)
- Clear timer on compare(CTC), ($WGM00 : WGM01 = 01$)
- Phase correct PWM, ($WGM00 : WGM01 = 10$)
- Fast PWM, ($WGM00 : WGM01 = 11$)

TAB 5. Overview of the settings for timer0[3]

Mode	WGM22	WGM21	WGM20	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag set on
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, phase correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	PWM, phase correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

[3] https://wolles-elektronikkiste.de/en/timer-and-pwm-part-1-8-bit-timer0-2

AVR TIMER2

- AVR's TIMER2 is an 8-bit timer and works mostly like TIMER0

Bit	7	6	5	4	3	2	1	0
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

FIG 5. TCCR2 —Timer/Counter Control Register Register

- However, it can also be used as real-time counter as follows:
  - A 32.768kHz crystal is connected to TOSC1 and TOSC2 pins of the MCU
  - The AS2 bit of the Asynchronous Status Register (ASSR) is set

Bit	7	6	5	4	3	2	1	0
	-	-	-	-	AS2	TCN2UB	OCR2UB	TCR2UB

AS2 When it is zero, Timer2 is clocked from $clk_{I/O}$. When it is set, Timer2 works as RTC.

FIG 6. The Asynchronous Status Register

TIFR —Timer Interrupt Flag Register register

The TIFR register contains the flags of different timers

Bit	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
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

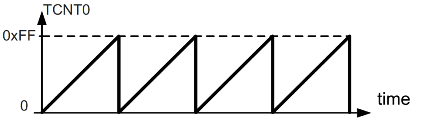
TOV0	D0	Timer0 overflow flag bit 0 = Timer0 did not overflow. 1 = Timer0 has overflowed (going from \$FF to \$00).
OCF0	D1	Timer0 output compare flag bit 0 = compare match did not occur. 1 = compare match occurred.
TOV1	D2	Timer1 overflow flag bit
OCF1B	D3	Timer1 output compare B match flag
OCF1A	D4	Timer1 output compare A match flag
ICF1	D5	Input Capture flag
TOV2	D6	Timer2 overflow flag
OCF2	D7	Timer2 output compare match flag

TOV0 —TIMER0 overflow flag

- The TOV0 flag is set when the counter overflows, going from 255 to 0.
- When the timer rolls over from 255 to 0, the TOV0 flag is set to 1 and it will **remain set until the software clears it**
- To clear this flag, we need to write 1 to it (and not zero, please remember this). This strange rules applies to all flags in all AVR MCUs.

Normal mode

- In this mode, the content of the timer/counter increments with each clock.
- It counts up until it reaches its max of 0xFF (i.e., 255).
- When it rolls over from 0xFF to 0x00, it sets high a flag bit called TOV0. This timer flag can be monitored



Steps to program Timer0 in Normal mode

```
1. Load the TCNT0 register with the initial count
   value.
2. Load the value into the TCCR0 register,
   indicating which mode (8-bit or 16-bit) is to be
   used and the prescaler option. When you select
   the clock source, the timer/counter starts to
   count, and each tick causes the content of the
   timer/counter to increment by 1.
3. Keep monitoring the timer overflow flag (TOV0) to
    see if it is raised. Get out of the loop when
   TOV0 becomes high.
4. Stop the timer by disconnecting the clock source
5. Clear the TOV0 flag for the next round.
6. Go back to Step 1 to load TCNT0 again.
```

```c
#include <avr/io.h>
/*F=(32768)/(2^8 * 64 * 2) = 1 blinks per sec*/
#define F_CPU 32768UL
int main(){
  uint8_t count=0;
  DDRB    |= (1<<PB1)
  ASSR  |= (1<<AS0); //use ext oscillator
  TCCR0 |= (1<<CS00); //normal mode, no prescaling
  while(1) {
    while (! (TIFR & (1<<TOV0))){/*Wait until
       overflow occurs*/}
      TIFR |= (1<<TOV0); //clear by writing a one to
         TOV0
      count++;  //extend counter
      if((count % 64) == 0){//toggle PB0 every 64
         overflows
        PORTB ^= (1<<PB1);
        }
```


Generating larger time delays

The delay depends on two factors that are beyond the control of the programmers:

- The crystal frequency
- The timer's 8-bit register.
- It is possible to use a prescaler of the TCCR0 to increase the delay
- The prescaler allows to divide the clock by a factor of 8 to 1024

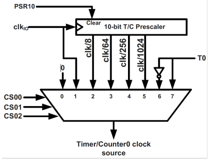
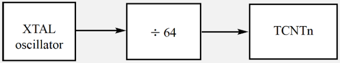


FIG 7. Prescaler of the TCCR0

Generating larger time delays—Example

Find the timer's clock frequency and its period for various AVR-based systems, with the following crystal frequencies. Assume that a prescaler of 1:64 is used: (A) 8MHz, (B) 16MHz, and (C) 10MHz

Solution:



(a) $1/64 \times 8$ MHz = 125 kHz due to 1:64 prescaler and T = 1/125 kHz = 8 µs
(b) $1/64 \times 16$ MHz = 250 kHz due to prescaler and T = 1/250 kHz = 4 µs
(c) $1/64 \times 10$ MHz = 156.2 kHz due to prescaler and T = 1/156 kHz = 6.4 µs

Generating larger time delays—Example

Find the value for TCCR0 if we want to program Timer0 in Normal mode with a prescaler of 64 using internal clock for the clock source.

Solution:

From Figure 5 we have TCCR0 = 0000 0011; XTAL clock source, prescaler of 64.

TCCR0 =	0	0	0	0	0	0	1	1
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

Generating larger time delays—Example

Assume XTAL = 8 MHz.

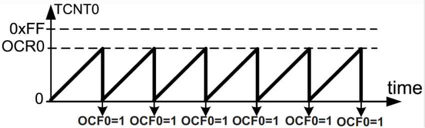
1. Find the clock period fed into Timer0 if a prescaler option of 1024 is chosen.
2. Show what is the largest time delay we can get using this prescaler option and Timer0.

(a) 8 MHz × 1/1024 = 7812.5 Hz due to 1:1024 prescaler and T = 1/7812.5 Hz = 128 ms = 0.128 ms

(b) To get the largest delay, we make TCNT0 zero. Making TCNT0 zero means that the timer will count from 00 to 0xFF, and then roll over to raise the TOV0 flag. As a result, it goes through a total of 256 states. Therefore, we have delay = (256 – 0) × 128 μs = 32,768 μs = 0.032768 seconds.

CTC mode programming

- Withthe CTC mode, the OCR0 register is used (refer to previous slides)
- Unlike the normal, the timer counts until the content of the TCNT0 register is equal to the content in OCR0
- At this point, the timer is cleared and the 0CF0 flag of the TIFR register is set



CTC mode example

Assuming $XTAL = 8MHz$, how would you generate a delay of 1 ms?

As XTAL = 8 MHz, the different outputs of the prescaler are as follows:

Prescaler	Timer Clock	Timer Period	Timer Value
None	8 MHz	1/8 MHz = 0.125 μs	1 ms/0.125 μs = 8000
8	8 MHz/8 = 1 MHz	1/1 MHz = 1 μs	1 ms/1 μs = 1000
64	8 MHz/64 = 125 kHz	1/125 kHz = 8 μs	1 ms/8 μs = **125**
256	8 MHz/256 = 31.25 kHz	1/31.25 kHz = 32 μs	1 ms/32 μs = **31.25**
1024	8 MHz/1024 = 7.8125 kHz	1/7.8125 kHz= 128 μs	1 ms/128 μs = **7.8125**

From the above calculation we can only use the options Prescaler = 64, Prescaler = 256, or Prescaler = 1024. We should use the option Prescaler = 64 since we cannot use a decimal point. To wait 125 clocks we should load OCR0 with 125 – 1 = 124.

16 bit timer programming

AVR's 16-bit timers

- TIMER1 is 16 bit-wide
- In contrast to timer 0 or timer 2, timer 1 is a 16-bit timer/counter. Because of this, you can use it for longer counting sequences. The counting extent is between 0x0000 and 0xFFFF.
- Since Timer1 is a 16-bit timer its 16-bit register is split into two bytes. These are referred to as TCNT1L—Timer1 low byte and TCNT1H —Timer1 high byte

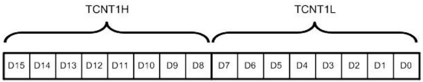


FIG 8. Timer1 High and Low Registers

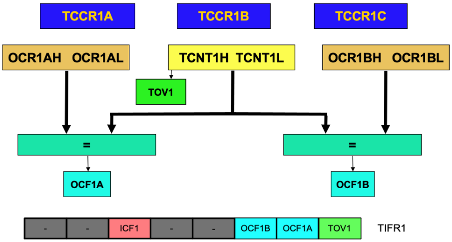


FIG 9. AVR TIMER1's registers

AVR's 16 bit timers

- Timer1 also has two control registers named TCCR1A (Timer/counter 1 control register) and TCCR1B.
- The TOV1 (timer overflow) flag bit goes HIGH when overflow occurs
- Timer1 also has the prescaler options of 1:1, 1:8, 1:64,1:256 and 1:1024
- There are two OCR registers in Timer1: OCR1A and OCR1B.
- There are two separate flags for each of the OCR registers, which act independently of each other.
- When $TCNT1 = OCR1A$, the OCF1A flag will be set on the next timer clock.
- When $TCNT = OCR1B$, the OCF1B flag will be set on the next clock.
- As Timer1 is a 16-bit timer, the OCR registers are 16-bit registers as well and they are made of two 8-bit registers. For example, OCR1A is made of OCR1AH (OCR1A high byte)

Example

Using CTC mode, calculate the value that should be loaded in the counter to generate a delay of 8 ms. Assume XTAL = 8 MHz.

Solution:

As XTAL = 8 MHz, the different outputs of the prescaler are as follows:

Prescaler	Timer Clock	Timer Period	Timer Value
None	8 MHz	1/8 MHz = 0.125 µs	8 ms / 0.125 µs = 64 k
8	8 MHz/8 = 1 MHz	1/1 MHz = 1 µs	8 ms / 1 µs = 8000
32	8 MHz/32 = 250 kHz	1/250 kHz = 4 µs	8 ms / 4 µs = 2000
64	8 MHz/64 = 125 kHz	1/125 kHz = 8 µs	8 ms / 8 µs = 1000
128	8 MHz/128 = 62.5 kHz	1/62.5 kHz = 16 µs	8 ms / 16 µs = 500
256	8 MHz/256 = 31.25 kHz	1/31.25 kHz = 32 µs	8 ms / 32 µs = **250**
1024	8 MHz/1024 = 7.8125 kHz	1/7.8125 kHz= 128 µs	8 ms / 128 µs = **62.5**

From the above calculation we can only use options Prescaler = 256 or Prescaler = 1024. We should use the option Prescaler = 256 since we cannot use a decimal point. To wait 250 clocks we should load OCR2 with 250 – 1 = 249.

The TIFR register

Bit	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
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

TOV0	D0	Timer0 overflow flag bit 0 = Timer0 did not overflow. 1 = Timer0 has overflowed (going from $FF to $00).
OCF0	D1	Timer0 output compare flag bit 0 = compare match did not occur. 1 = compare match occurred.
TOV1	D2	Timer1 overflow flag bit
OCF1B	D3	Timer1 output compare B match flag
OCF1A	D4	Timer1 output compare A match flag
ICF1	D5	Input Capture flag
TOV2	D6	Timer2 overflow flag
OCF2	D7	Timer2 output compare match flag

FIG 10. The TIFR register contains the TOV1, OCF1A, and OCF1B flags

The TCCR1B —Timer 1 Control Register

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	**TCCR1B**
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ICNC1	D7	Input Capture Noise Canceler 0 = Input Capture is disabled. 1 = Input Capture is enabled.
ICES1	D6	Input Capture Edge Select 0 = Capture on the falling (negative) edge 1 = Capture on the rising (positive) edge
	D5	Not used
WGM13:WGM12	D4 D3	Timer1 mode

Timer1 clock selectors

CS12:CS10	D2D1D0	Timer1 clock selector
	0 0 0	No clock source (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 source on T1 pin. Clock on falling edge.
	1 1 1	External clock source on T1 pin. Clock on rising edge.

Timer1 working modes

Mode	WGM13	WGM12	WGM11	WGM10	Timer/Counter Mode of Operation	Top	Update of OCR1x	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

Normal Mode

- simplest mode
- count up to 0xFFFF and wrap around to 0x0000
- no clear is ever performed
- TOV flag is set when the wrap around occurs (overflow)
- to reset TOV, must execute ISR or clear flag manually
- no output pins are used

Clear Timer on Compare Match (CTC) Mode

- resolution of counter is manipulated by output compare register A (OCRnA) or input capture register (ICRn)
- counter is cleared to zero when its value equals either ICRn or OCRnA
- TOP is defined by ICRn or OCRnA
- interrupt can be generated at compare point
- output pins (OCnx) can be utilized
- toggle, set, or clear on match

Example 1

```
/*blink frequency=(16,000,000)/(2^16 * 64 * 2)=1.91
  cycles/sec*/
#include <avr/io.h>
#define F_CPU 16000000UL
int main(){
      DDRB |= (1<<PB0); //set port B bit zero to
         output
      TCCR1A = 0x00; //normal mode
      TCCR1B = (1<<CS11) | (1<<CS10); //use clk/64
      TCCR1C = 0x00; //no forced compare
      while(1) {
      if (TIFR & (1<<TOV1)) { //if overflow bit TOV1
          is set
      TIFR |= (1<<TOV1); //clear it by writing a one
          to TOV1
      PORTB ^= (1<<PB0); //toggle PB0 each time this
          happens
```


Example 2—Toggle PB0 on compare match

```
/*blink frequency ~=(16,000,000)/(2^15*64*2)=3.8
  cycles/sec*/
#include <avr/io.h>
int main(){
      DDRB !=(1<<PB0);
      //ctc mode, toggle on compare match
      TCCR1A |= (1<<COM1A0);
      //use OCR1A as source for TOP, use clk/64
      TCCR1B = (1<< WGM12) | (1<<CS11) | (1<<CS10);
      TCCR1C = 0x00; //no forced compare
      OCR1A = 0x7FFF; //compare at half of 2^16
      while(1) {
        if (TIFR & (1<<OCF1A)) {//if output compare
           flag is set
        PORTB ^= (1<<PB0); //toggle PB0 each time
```

Examples

Example 1—Blink an LED at 10Hz

```
Set up LED hardware
Set up timer

WHILE forever
  IF timer value IS EQUAL TO OR MORE THAN 1/20
     sec
    THEN Reset counter
    Toggle LED
  END IF
END WHILE
```

LISTING 4: Pseudocode for flashing an LED at 10Hz

Example 1—Blink an LED at 10Hz

We need to know the count value needed for a 10 Hz delay (i.e., 1/20 delay)

$$
\begin{aligned}
\text{Target Timer Count} &= \frac{1}{\text{Target Frequency}} \Big/ \frac{1}{\text{Timer Clock Frequency}} - 1 \\
&= \frac{1}{20} \Big/ \frac{1}{1000000} - 1 \\
&= \frac{.05}{0.000001} - 1 \\
&= 50000 - 1 \\
&= 49999
\end{aligned}
$$

- AVR MCU have an internal frequency $F_C PU = 1MHz$
- The count is updated each timer input clock tick, thus it takes one tick
- The timer needs to count to 49999 before 120th of a second has elapsed.

Example 1—Blink an LED at 10Hz

```c
# include <avr /io.h>
int main (void){
  DDRB |= (1 << 0);
  TCCR1B |= (1 << CS10 ); // set up timer
  while(1){
    // true when count matches 1/20 of a second
    if ( TCNT1 >= 49999){
      PORTB ^= (1 << 0);
      // Reset timer value
      TCNT1 = 0;
    }
  }
}
```

LISTING 5: Code for flashing an LED at 10Hz

Example 2—Blink an LED at 1Hz

- What if we needed a longer delay, e.g., 1 sec?
- AVR MCU have a prescaler that can be used to reduce the clock frequency.

$$\text{Timer Resolution} = \frac{1}{\text{Input Frequency}/\text{Prescale}} = \frac{\text{Prescale}}{\text{Input Frequency}}$$

- At $F_{CPU} = 1MHz$ the resolutions with a prescaler are as follows:

Prescaler Value	Resolution ($F_{CPU} = 1$ MHz)
1	1 µs
8	8 µs
64	64 µs
256	256 µs
1024	1024 µs

Example 2—Blink an LED at 1Hz

With a pre-scaler, the needed count is computed as:

$$\text{Target Timer Count} = \left(\frac{1}{\text{Target Frequency}} \Big/ \frac{\text{Prescale}}{\text{Input Frequency}} \right) - 1$$

Which can be rearranged as

$$\text{Target Timer Count} = \left(\frac{\text{Input Frequency}}{\text{Prescale} \times \text{Target Frequency}} \right) - 1$$

Example 2—Blink an LED at 1Hz

If we need a prescaler that gives a 1Hz delay at $F_{CPU} = 1MHz$, then the counter is computed as follows:

Prescaler Value	Target Timer Count
1	999999
8	124999
64	15624
256	3905.25
1024	975.5625

- Prescaler 256 and 1024 do not divide evenly; thus are discounted because they would lead to poor precision
- Only prescaler 1,8,64 are eligible.
- Prescaler 1,8 give very large values that cannot fit in any timer counter
- Only prescaler 64—with a count of 15624—allow a 1Hz delay

Example 2—Blink an LED at 1Hz

```
#include <avr /io.h>
int main ( void ){
  DDRB |= (1 << 0);
  // Set up timer at Fcpu /64
  TCCR1B |= ((1 << CS10 ) | (1 << CS11 ));
  while(1){
    // true when count matches 1 second
    if ( TCNT1 >= 15624){
      PORTB ^= (1 << 0); // Toggle the LED
      TCNT1 = 0; // Reset timer value
    }
  }
}
```

LISTING 6: Code for flashing an LED at 1Hz

Longer timer delays

- What if you needed a 10 second or 1 minute or even 1 year delay?
- Solution: create a sort of prescaler and increment a variable each time that period is reached, and only act after the counter is reached a certain value.

```
Set up LED hardware
Set up timer
Initialise counter to 0
WHILE forever
  IF timer value IS EQUAL TO 1 sec THEN
    Increment counter
    Reset timer
    IF counter value IS EQUAL TO 60 seconds
      THEN
      Toggle LED
    END IF
  END IF
```



```c
#include <avr/io.h>
int main ( void ){
  unsigned char elapsed_seconds = 0;
  DDRB |= (1 << 0);
  TCCR1B |= ((1 << CS10 ) | (1 << CS11 ));
  while(1){
    if ( TCNT1 >= 15624){/*true when count
       matches 1 second*/
      TCNT1 = 0; /*reset timer value*/
      elapsed_seconds ++;
      if(elapsed_seconds==60) {/*if one min is
         elapsed*/
        elapsed_seconds = 0; /*reset counter
           variable*/
        PORTB ^= (1 << 0);
      }
    }
  }
}
```

LISTING 8: 1 minute timer delay

The end