

Basic registers of timers

- Upon reset, $TCNT_n = 0$. Then, it counts up with each clock pulse
- The content of the timer/counter can be accessed with the $TCNT_n$ register. It is possible to also load a value into the counter.
- The TOV_n flag allows to know when the $TCNT_n$ overflows
- The $TCCR_n$ controls the mode of operation of the timer. For example, it allows to either work as timer or as a counter by loading a proper value into the $TCCR0$
- The OCR_n register is used to compare with the content of the $TCNT_n$. When they are equal the OCF_n flag will be set.

Timers in the ATmega328

Timers in the ATmega328

ATmega328 has three times:

- 8-bit TC0—It is also known as **TCNT0** (Timer/Counter 0).
- 16-bit TC1 —It is also known as **TCNT1** (Timer/Counter 0).
- 8-bit TC2 —It is also known as **TCNT2** (Timer/Counter 2).

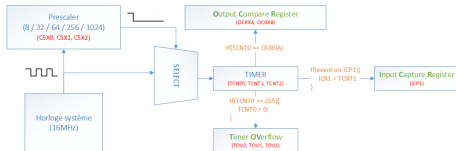
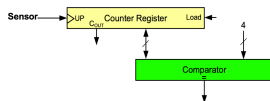


FIG 3. AVR Timer diagram

EXAMPLE: Counting

- The counter register can be used to record the frequency of an event
- The counter register stores the current value of a counter. Any time a certain even occurs, the value of the counter is increased by one.
- For example, this would allow us to know how often a wheel has turned during a given time
- AVR MCU comes with 8-bit and 16-bit timers.
- The 8-bit counters can count to $2^8 - 1 = 255$ whilst the 16-bit counter can count to $2^{16} - 1 = 65,536$.



EXAMPLE: Calculating a car's speed

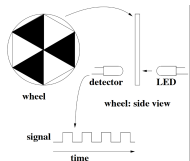


FIG 4. A wheel equipped with a simple encoder. The encoder wheel is partitioned into regions that are alternating opaque and transparent. When the wheel is turning at a constant velocity, the detector observes a digital signal of some frequency and a duty cycle of 50%. A counter can be used to count the number of times the detector transitions from high to low; thus, allow to estimate how many times the wheel has turned

Example: Making an asynchronous delay

- Each clock pulse increments the timer's counter by one, the timer measures intervals in periods of one on the input frequency:

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

- For instance, if we supply a 100Hz signal to a timer, our period becomes:

$$\begin{aligned} \text{Timer Resolution} &= \frac{1}{\text{Input Frequency}} \\ &= \frac{1}{100\text{Hz}} \\ &= .01\text{s} \end{aligned}$$

- In this case, the period is $T = .01\text{s}$, thus our timer will be able to measure times that are a multiple of this duration.

Example: Making an asynchronous delay

- Objective: flash an LED at 10Hz on an AVR MCU running at 1MHz clock.
- At 1MHz, one tick is $1\mu\text{s}$, so flashing an LED at 10Hz requires 50ms of delay
- Algorithm 1 shows a pseudocode required to drive the LED:

Algorithm 1: Flashing an LED at 10Hz with a timer

Result: Initialize the LED DDR to output

Initialize the timer hardware;

while *While forever do*

if *COUNTER_VALUE* > 0.05 **then**

 Reset counter;

 Turn on LED;

end

end

Example: Making an asynchronous delay

- But what is the counter values of 50ms?
- Reminder: 8-bit counter store up to 255 and 16-bit counter up to 65535
- For 1MHz clock, each tick corresponds to $1\mu\text{s}$.
- Thus, for 50ms (i.e., $1/20\text{seconds}$)

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

- We need a counter that can count up to 49999.

```
#include <avr/io.h>
int main (void){
    DDRB |= (1 << 0);
    // Set up timer
    TCCR1B |= (1 << CS10);
    while(1){
        //Check if the counter is full
        if (TCNT1 >= 49999){
            PORTB ^= (1 << PB0);
            // Reset timer value
            TCNT1 = 0;
        }
    }
}
```

LISTING 1: Flashing an LED at 10Hz with a timer

The end