Timers and event counters

Timers: Why to use it?

There are three ways to create a delay in AVR:

1) A for loop (software) (16 MHz / 62.5 ns)

```
void delay_100ms (void)
{
      uint16_t i;
      for (i=0; i<16000;i++);
}
```

Total control over time = Cycles by instruction * number of instructions

Assembly

C + Compiler

Timers: Why to use it?

2) Predefined C functions(software):

_delay_ms()" , "_delay_us()" → "delay.h"

```
#include <util/delay.h>                    //delay loop functions
#include <avr/io.h>                        //standard AVR header

int main(void)
{
        void delay_ms(int d)               //delay in d microseconds
        {
                _delay_ms(d);
        }
        DDRB = 0xFF;                       //PORTA is output
        while (1){
                PORTB = 0xFF;
                delay_ms(10);
                PORTB = 0x55;
                delay_ms(10);
        }
        return 0;
}
```

Wrapper function

Timers: Why to use it?

3) Timers (Hardware):

- In general → to measure a given time interval

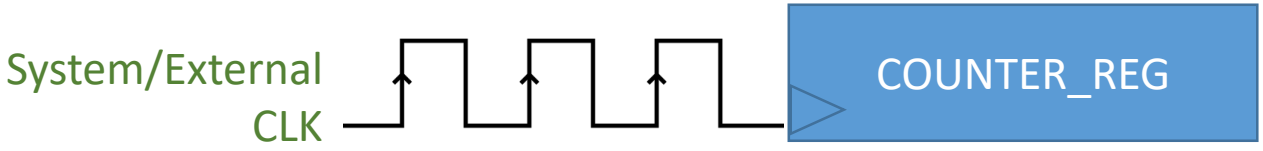
- Like alarm clock
  - Set a timer to trigger an interrupt when a time occurs.
  - Interrupt → run different code, or change a pin output.

- Timer is independent of the program execution **versus** a loop calling millis()

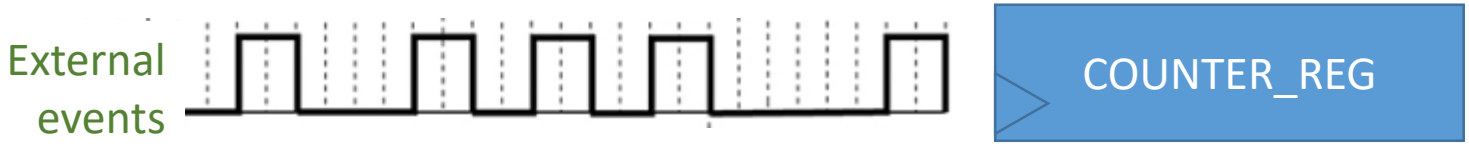
- The timer does that work while the code does other things.

Timers: How do they works?

- Timer/counters are binary counters → a register
- Timer counts clock cycles until a maximum value is reached
- When counter overflows → resets and back to zero.
- Two working modes:
  - Timer: detects the clock signal, it increases its counter by one.



  - Events counter: events to be counted are applied to the input, and the number of events occurring are counted.



Timers: Operation modes

- Normal mode : counts up to a maximum value and is reset to zero in the next timer clock cycle→
  Stopwatch -- > read the register in anytime

- Clear on Compare match (CTC)→ Compare a register (OCRx) loaded with a value [0 .. MaxVal], when a compare match occurs a flag is set and the count register can be cleared. → Alarm clock

- Fast PWM and Phase correct PWM → a waveform is generated according to a setup

- Input capture →a pin triggers the Reading of the timer value and saved it a a register. → Measure external pulses

Timers: Timer Event Notification

- Three types:
  - Polling
  - Interrupt

  A flag is setting (Software)

  - Automatic Reaction on Events

  To react on timer interrupt events on purely hardware basis without the need to execute code. In contrast to the two other solutions this happens in parallel to code execution and requires no processing time.

ATMega328 Timers

Timers in ATMEGA328 (Arduino UNO)

- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode Timer0 and Timer2
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode Timer1

 Data Sheet Complete DS40002061A-page 1

Source: Microchip

ATMega328 Timers: Normal mode

Timer0 : Normal Mode (Polling)

- 8-bit timer
- Each clock pulse (tick) increments the timer's counter (`TCNT0`) by one
- Count 0 (or another) up to 255 and overflows then rolls over
- When TCNT0 overflows a status flag in the Flag Register (TIFRn) is set.
- Mode present in whatever microcontroller

15.9.7 TIFR0 – Timer/Counter 0 Interrupt Flag Register

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	

Overflow flag

Timer0 : Normal Mode (Polling)

```
int main(void) {
 DDRB |= (1 << DDR2);                    // connected led to pin PB2
 timer0_init ();
while(1) {
  while((TIFR0 & (1<<TOV0))==0); // Wait for overflow flag
    PORTB ^= (1 << PORTB2);               // Toggle the LED
    TCNT0 = 0;                            // Re-start the TIMER0
    TIFR0 |= (1 << TOV0);                 // Clear timer0 flag
    }
  }
void timer0_init(){
 TCCR0B |= (1 << CS00); // Start the timer (More after)
 TCNT0 = 0;                        // Initialize counter
}
```

How much time has elapsed from 0x00 to 0xFF?

F_{sys} = 16 MHz → T_{sys} = 62,5 ns (1 count or tick)

1 count or tick → 62,5ns

(255+1) * 62,5ns = 16,00 µs

Timer0 : Normal Mode (Interrupt)

- Enable the OVERFLOW interrupt
- Count 0 (or another) up to 255 and overflows → Interrupt
- ISR timer executes any activity and reload the timer and clear TOV0 flag

TIMSK0 – Timer/Counter Interrupt Mask Register

Enable overflow interrupt

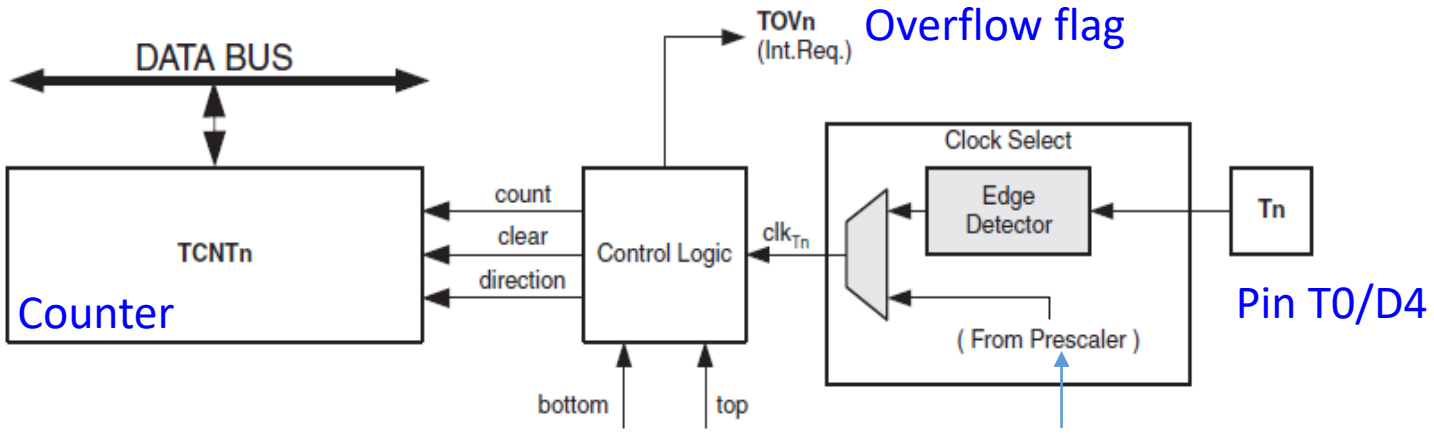
Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer0 : Normal Mode (Interrupt)

```
Inline void timer0_enaINT(){
     TIMSK0 |= (1 << TOIE0);  // initialize counter
}

ISR(TIMER0_OVF_vect){
  PORTB ^= (1 << PORTB2); // toggle led every 256 clk cycles
}

int main(void)
{
  DDRB |= (1 << DDB2);  // connect led to pin PB2
  timer0_init();        // initialize timer
  timer0_enaINT();      // enable overflow interrupt

  sei();

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

Timer0 : Normal Mode

Figure 15-2. Counter Unit Block Diagram



Signal description (internal signals):

count	Increment or decrement TCNT0 by 1.
direction	Select between increment and decrement.
clear	Clear TCNT0 (set all bits to zero).
clk_{Tn}	Timer/Counter clock, referred to as clk_{T0} in the following.
top	Signalize that TCNT0 has reached maximum value.
bottom	Signalize that TCNT0 has reached minimum value (zero).

Source: Microchip

Timer0 : Normal Mode (Prescaled)

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 Source: Microchip

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.

16 MHz/1024 = 15,6 KHz

1 tick or cycle → 1/15.6Khz = **64 us**

256 * 64 us/tick = **16.38 ms**

Too small to watch a LED blinking

Timer0 : Normal Mode (Prescaled)

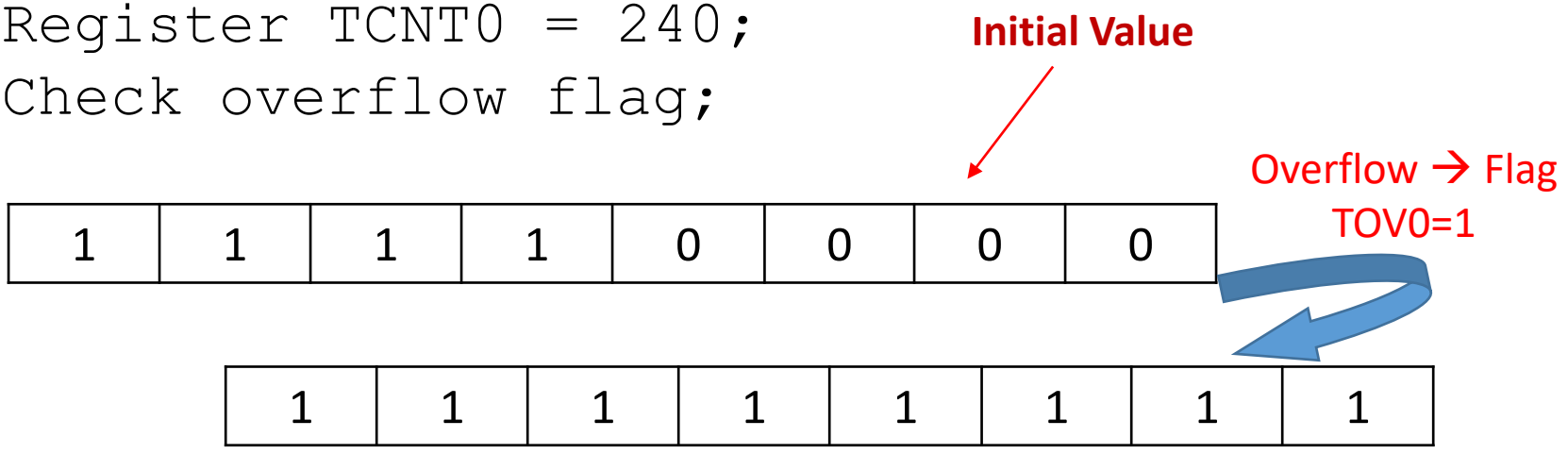
How to set a time lesser than the máximum?

Example:

No prescaler

Register TCNT0 = 240;

Check overflow flag;



How many CLK cycles have elapsed until the overflow?

15 clk-cycles (@16MHz Arduino UNO)

Elapsed time = 15*62,5ns = 930 ns