Embedded_session_6

Embedded System Session 6 - Timers

Created: 2025-07-20
Author: Fares Hesham Mahmoud
Tags: AVR, Embedded system, Timer
Status: #Formatted

اللهمَّ علِّمنا ما ينفعنا، وانفعنا بما علمتنا، وزدنا علمًا. وافتح علينا فتحًا عظيمًا.

Timer Fundamentals

Basic Timing Concepts

Time Period/Clock Cycle

Time between two rising edges or two falling edges:

$$T = \frac{1}{f}$$

Where T = time period, f = frequency

Duty Cycle

The ratio between the ON time and the total time period:

$$\text{Duty Cycle} = \frac{\text{ON Time}}{\text{Time Period}} \times 100\%$$

System Timing Calculations

Core Formulas

Parameter	Formula	Example (8MHz, Prescaler=8)
System Time	$ST = \frac{1}{\text{system frequency}}$	$\frac{1}{8\times10^6} = 125 \text{ ns}$
Timer Frequency	$f_{timer} = \frac{f_{system}}{\text{Prescaler}}$	$\frac{8\times10^6}{8} = 1 \text{ MHz}$
Timer Time	$T_{timer} = \frac{\text{Prescaler}}{f_{system}}$	$\frac{8}{8\times10^6} = 1\ \mu\text{s}$

Parameter	Formula	Example (8MHz, Prescaler=8)
Tick Time	$T_{tick} = \frac{\text{Prescaler}}{f_{system}}$	$1\ \mu\text{s per tick}$

Timer Overflow Calculations

8-bit Timer Example

- Timer counts: 0 → 255 (255 tick times for counting)
- Overflow: 255 → 0 (1 tick time for overflow)
- Total: 256 tick times for complete overflow cycle

Time for Complete Overflow:

$$T_{oerfo} = \text{Tick Time} \times 2^{\text{esolution}}$$

8-bit Timer with 8MHz Clock, Prescaler=8

```
Tick Time = 8/(8×10⁶) = 1 µs
Time Overflow = 1 µs × 2⁸ = 256 µs
```

When timer overflows, it triggers the Timer peripheral ISR.

Creating Custom Timer Intervals

Multiple Overflow Method

To create longer delays, count multiple overflows:

$$oerfos = \frac{T_{reuired}}{T_{oerfo}}$$

Creating 1024 µs Interval

```
Required Time = 1024 µs
Overflow Time = 256 µs
Number of Overflows = 1024/256 = 4
```

```
ISR(TIMER0_OVF_vect) {
    static uint8_t counter = 0;
```

```
        counter++;
        if (counter == 4) {
            ADC_start();  // Execute desired action
            counter = 0;  // Reset counter
        }
    }
```

Preload Value Method

For fractional overflows, use preload values:

Custom Preload for Precise Timing

```
ISR(TIMER0_OVF_vect) {
    static uint8_t counter = 0;
    counter++;
    if (counter == 4) {
        ADC_start();
        counter = 0;
        TCNT0 = 192;  // Preload to get 0.25 overflow cycle
    }
}
```

Timer Design Principles

Timer Optimization Guidelines

- Lower $_{oerfos}$ = Better System Performance
- Less CPU Load: Fewer interrupts mean more time for main program
- More Precise Timing: Direct hardware timing vs. software counting
- Power Efficiency: Fewer wake-ups in low-power applications

Choosing the Right Prescaler

Prescaler	Advantages	Disadvantages	Use Case
Low (1, 8)	High resolution, precise timing	Frequent overflows, high CPU load	High-speed PWM, precise measurements

Prescaler	Advantages	Disadvantages	Use Case
Medium (64, 256)	Balanced resolution/CPU load	Moderate precision	General timing, periodic tasks
High (1024)	Low CPU load, long intervals	Low resolution	Slow periodic tasks, timeouts

Practical Timer Applications

PWM Generation

```
// Fast PWM Mode
TCCR0 = (1<<WGM01) | (1<<WGM00) | (1<<COM01) | (1<<CS01);
OCR0 = 128;  // 50% duty cycle (128/256)
```

Periodic Task Scheduling

```
// Timer setup for 1ms interrupts
void Timer_Init_1ms(void) {
    // CTC mode, prescaler 8
    TCCR0 = (1<<WGM01) | (1<<CS01);
    OCR0 = 124;  // For 1ms at 8MHz with prescaler 8
    TIMSK |= (1<<OCIE0);  // Enable compare match interrupt
}

ISR(TIMER0_COMP_vect) {
    // Called every 1ms
    static uint16_t ms_counter = 0;
    ms_counter++;

    if (ms_counter % 100 == 0) {
        // Every 100ms task
        LED_Toggle();
    }

    if (ms_counter >= 1000) {
        // Every 1s task
        ADC_StartConversion();
        ms_counter = 0;
    }
}
```

Timer Modes Overview

Mode	Description	Applications
Normal	Counts 0 to MAX, overflows to 0	Basic timing, overflow interrupts
CTC	Counts 0 to compare value, resets	Precise frequency generation
Fast PWM	Counts 0 to MAX, fast PWM output	Motor control, LED dimming
Phase Correct PWM	Counts up then down	High-quality PWM with centered pulses

Performance Considerations

⚠ **CPU Load Impact** ›

Each timer overflow interrupt consumes CPU cycles:

- Context Switching: ~10-20 cycles
- ISR Execution: Depends on code complexity
- Return Overhead: ~10-20 cycles

High-frequency interrupts can significantly impact main program execution!

Timer Load Calculation ›

```
Timer Frequency = 1 MHz
Overflow Rate = 1 MHz / 256 = 3.9 kHz
If ISR takes 50 cycles and CPU is 8 MHz:
CPU Load = (3.9k × 50) / 8M = 2.4%
```

Advanced Timer Techniques

Timer Chaining

```
// Use Timer0 overflow to increment a software counter
volatile uint32_t extended_timer = 0;
```

```
ISR(TIMER0_OVF_vect) {
    extended_timer++;  // Creates a 32-bit timer from 8-bit hardware
}
```

Accurate Delay Function

```
void delay_us(uint16_t microseconds) {
    // Assuming 1MHz timer clock
    TCNT1 = 0;
    while (TCNT1 < microseconds) {
        // Wait for timer to reach desired count
    }
}
```

References

- ATmega32 Timer/Counter Documentation
- AVR Timer Application Notes
- Real-Time Systems Design Principles