



ВЕЛИКОТЪРНОВСКИ УНИВЕРСИТЕТ „СВ.СВ. КИРИЛ И
МЕТОДИЙ“
ФАКУЛТЕТ ИНФОРМАТИКА И МАТЕМАТИКА



КУРСОВА РАБОТА

по
учебната дисциплина
„Микропроцесорни системи“
по тема
“Хронометър”

РАЗРАБОТИЛ:

Сушко Матвей

Фак.№2309013835

Специалност: Компютърни науки

Курс: II

РЪКОВОДИТЕЛИ:

проф. Мирослав Бончев Бончев

Велико Търново

2024 - 2025 г.

Подход към разработка

При разработката на секундомера разделих функциите на три независими „потока“. Първият беше хардуерен таймер `Timer1`, настроен на 1 ms: той отговаря само за инкремента на глобалния брояч `msElapsed`, което гарантира равномерно протичане на времето, независимо от останалата логика. Вторият поток стана обработчик на прекъсвания при промяна на състоянието на пиновете (PCINT): там фиксирам фронтите на двата бутона и ги превръщам в флагове-събития `startEvent` и `resetEvent`. Благодарение на този подход натисканията се регистрират мигновено, но в ISR няма тежки операции — прекъсванията приключват за няколко микросекунди. Третият „поток“ е безкраен `loop()`: той дебаунсира събитията (чрез `LOCK_MS`), превключва състоянието `running`, нулира брояча, прерисува дисплея само при промяна на секундите и мига с LED.

Кодът беше изграден итеративно: първо изведох базовото отчитане на времето и показването на OLED; след това добавих реакция на бутоните и индикация „RUN“; накрая поставих двойна защита от трептене и случайно нулиране — таймаут `LOCK_MS` и условие `!running` при настройка на `resetEvent`.

Алгоритма

1. Инициализация (`setup`)

- 1.1. Настройваме изхода `LED_RUN` като изход, а бутоните `BTN_START`, `BTN_RESET` — като входове с подтяжка.
- 1.2. Инициализираме OLED дисплея, извеждаме „00:00“ и включваме екрана.
- 1.3. Конфигурираме хардуерния таймер T1 в режим CTC на честота $\approx 1\text{ kHz}$; разрешаваме неговото прекъсване (1 ms / тик).
- 1.4. Включваме пинови прекъсвания (PCINT) за двата бутона.
- 1.5. Разрешаваме глобални прекъсвания.

2. Прекъсване `Timer1` (всяка 1 ms)

- 2.1 Ако секундомерът е стартиран (`running == true`), увеличаваме глобалния брояч `msElapsed`.

3. Прекъсване PCINT (фронт на бутона)

3.1 При падане на `BTN_START` поставяме флаг `startEvent = true`.

3.2 При падане на `BTN_RESET`, ако хронометърът е спрял (`!running`), поставяме флаг `resetEvent = true`.

3.3 Актуализираме `prevState`, за да улавяме само фронтовете.

4. Главният цикъл (`loop`)

Безкрайно повтаряме следните стъпки:

4.1. Запазваме текущото време `nowMs = millis()`.

4.2. Обработка флага `startEvent`

4.2.1 Атомарно изчистваме флага.

4.2.2 Ако от момента на последното натискане е изминало повече от `LOCK_MS` (защита от трептене), инвертираме `running` и обновяваме `lastStartMs`.

4.3. Обработка на флаг `resetEvent`** (само кога `running == false`)

4.3.1 Атомарно изтриваме флаг и нулираме `msElapsed`.

4.3.2 Запазваме време на последен сброс `lastResetMs`.

4.4. Атомарно копираме `msElapsed` в локалната променлива `msCopy`.

4.5. Изчисляваме секундите `tSec = msCopy / 1000`.

4.5.1 Ако `tSec` са се променили в сравнение с `prevSec`, актуализираме `prevSec` и извикваме `drawTime`, извеждайки новите минути и секунди.

4.6. Управление на светодиода RUN**

4.6.1 Ако `running == true` — увеличаваме брояча `blink`.

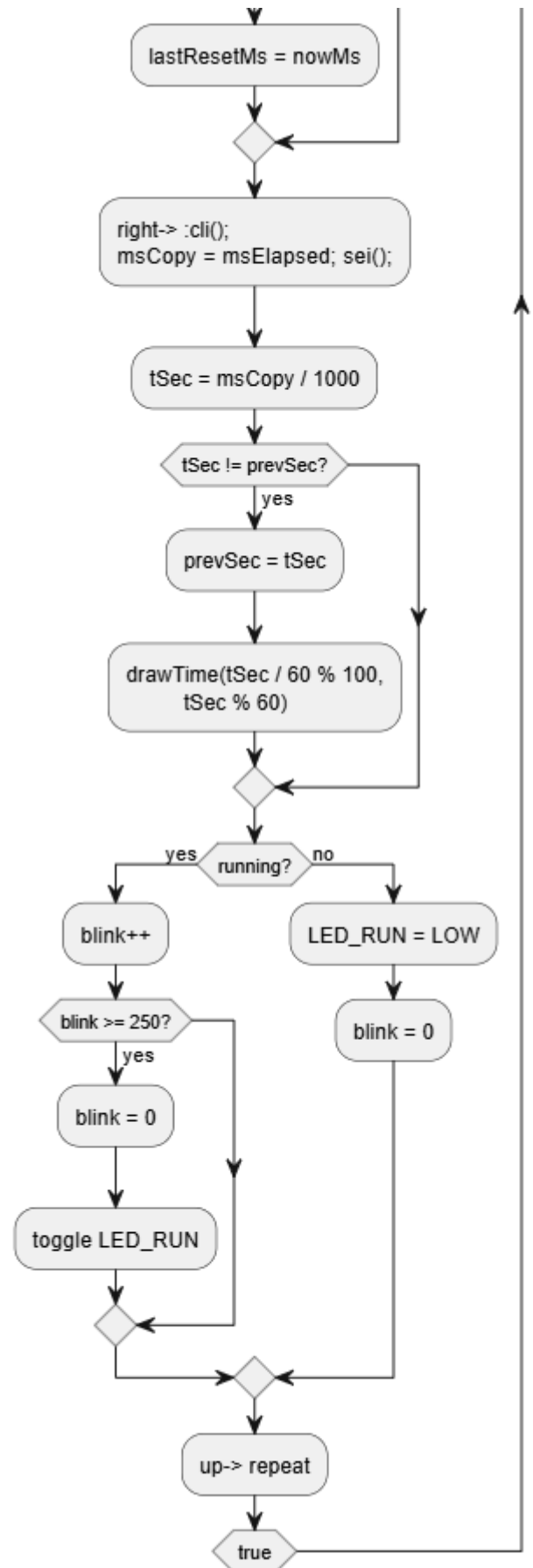
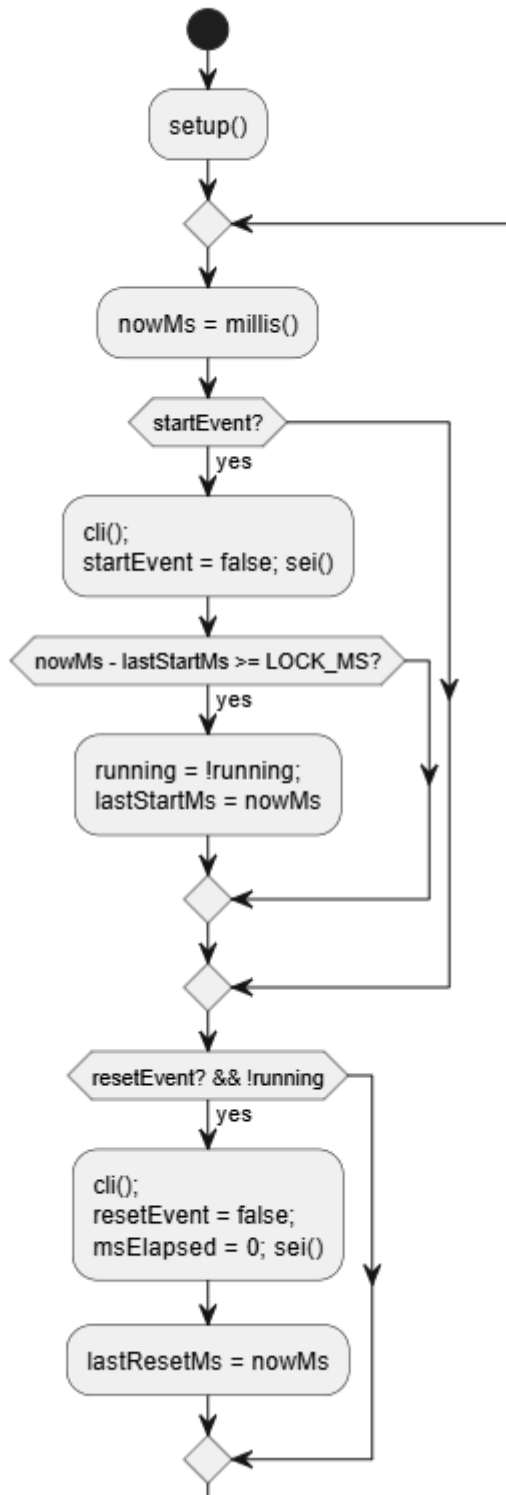
4.6.2 След 250 мс превключваме състоянието `LED_RUN` и нулираме `blink` (мигане 2 Гц).

4.6.3 Ако `running == false` — гасим `LED_RUN` и нулираме `blink = 0`.

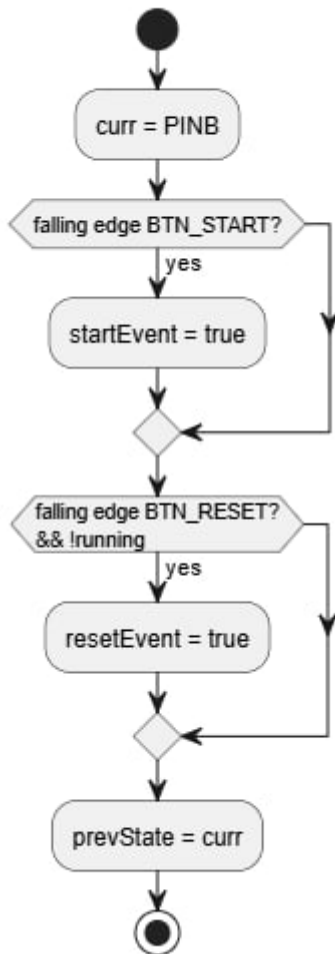
4.7. Преминаваме към началото на цикъла (стъпка 4.1).

Блок-схемите

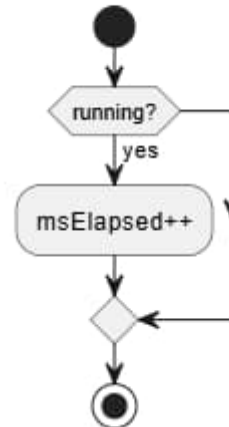
Chronometer - main loop (wide layout)



Pin-change ISR - button edges



Timer1 COMPA ISR - 1 ms tick



Самата програма

```

#include <avr/interrupt.h>
#include <Wire.h>
#include <Tiny4kOLED.h>

```

```

// ---- Хардуерни дефиниции -----
const byte LED_RUN = 1; // PB1 – индикаторен светодиод
const byte BTN_START = 4; // PB4 – бутон „Старт/Пауза“
const byte BTN_RESET = 3; // PB3 – бутон „Нулиране“

```

```

// ---- Глобални променливи -----
volatile uint32_t msElapsed = 0; // Натрупани милисекунди
volatile bool running = false; // Флаг – таймерът върви ли?

```

```

volatile bool startEvent = false; // Сигнал от прерывание за старт/пауза

```

```

volatile bool resetEvent = false; // Сигнал от прерывание за нулиране

uint32_t lastStartMs = 0;      // Последно време на натискане на старт
uint32_t lastResetMs = 0;      // Последно време на нулиране
const uint16_t LOCK_MS = 1000; // Софтуерен антидребен дребеж – 1 s

// ---- Позиция на цифрите на дисплея -----
const uint8_t X_MIN = 24;      // X-координата на минутите
const uint8_t X_SEC = X_MIN + 40; // X-координата на секундите
const uint8_t Y_TOP = 0;       // Y-координата – най-горен ред

// ---- Помощна функция: обновяване на едно поле -----
void updateField(uint8_t value, uint8_t x) {
    char buf[3];
    oled.setFontX2(FONT8X16P); // Двоен мащаб за по-едри цифри
    oled.setCursor(x, Y_TOP);
    oled.print(F(" ")); // Затриваме старото съдържание
    sprintf(buf, "%02u", value); // Форматираме с водеща нула
    oled.setCursor(x, Y_TOP);
    oled.print(buf); // Печатаме новата стойност
}

// ---- Рисуваме целия часовник (мин:сек) -----
void drawTime(uint8_t mm, uint8_t ss) {
    static uint8_t prevMM = 255, prevSS = 255; // Начални „невъзможни“ стойности
    static bool colonDone = false; // Рисуван ли е веднъж двоеточието

    if (mm != prevMM) { updateField(mm, X_MIN); prevMM = mm; }
    if (ss != prevSS) { updateField(ss, X_SEC); prevSS = ss; }

    // Двоеточието се рисува само веднъж – статично
    if (!colonDone) {
        oled.setFont(FONT8X16P);
        oled.setCursor(X_MIN + 32, 1);
        oled.print(':');
        colonDone = true;
    }
}

// ---- SETUP -----
void setup() {
    // Настройка на пиновете
    pinMode(LED_RUN, OUTPUT);
    pinMode(BTN_START, INPUT_PULLUP); // Вътрешен pull-up
    pinMode(BTN_RESET, INPUT_PULLUP);

    // Инициализация на OLED
    oled.begin();
    oled.clear();
    drawTime(0, 0); // Показваме „00:00“ при стартиране
}

```

```

oled.on();

// Таймер 1 – CTC режим, делител 64 000 → 1 kHz (1 ms) прекъсване
TCCR1 = (1 << CTC1) | (1 << CS12) | (1 << CS11) | (1 << CS10);
OCR1C = (F_CPU / 64000UL) - 1; // Числител за 1 ms
TIMSK |= (1 << OCIE1A);      // Разрешаваме прерыванията от таймера

// Прерывания по смяна на състоянието на бутоните (PCINT3, PCINT4)
GIMSK |= (1 << PCIE);
PCMSK |= (1 << PCINT3) | (1 << PCINT4);

sei(); // Глобално включване на прерыванията
}

// ---- LOOP -----
void loop() {
    uint32_t nowMs = millis(); // Текущо време за антидребежието

    // --- Обработка на събитие „Старт/Пауза“ ---
    if (startEvent) {
        cli(); startEvent = false; sei();      // Атомично изчистване
        if (nowMs - lastStartMs >= LOCK_MS) {   // Достатъчно ли време?
            running = !running;                 // Превключваме състоянието
            lastStartMs = nowMs;
        }
    }

    // --- Обработка на събитие „Нулиране“ ---
    if (resetEvent && !running) {               // Нулираме само ако е спрял
        cli(); resetEvent = false; msElapsed = 0; sei();
        lastResetMs = nowMs;
    }

    // --- Копие на msElapsed за безопасен достъп извън прерывания ---
    uint32_t msCopy; cli(); msCopy = msElapsed; sei();

    // --- Обновяване на дисплея веднъж в секунда ---
    static uint32_t prevSec = 0xFFFFFFFF;
    uint32_t tSec = msCopy / 1000;
    if (tSec != prevSec) {
        prevSec = tSec;
        drawTime(tSec / 60 % 100, tSec % 60);    // Минутите циклират през 00-99
    }

    // --- Мигалка на LED_RUN, докато часовникът върви ---
    static uint16_t blink = 0;
    if (running) {
        if (++blink >= 250) {                    // ~4 Hz при 1 ms loop()
            blink = 0;
            PINB |= _BV(LED_RUN);               // Toggle без digitalWrite
        }
    }
}

```

```

    }
} else {
    digitalWrite(LED_RUN, LOW);          // LED off в пауза
    blink = 0;
}
}

// ---- Прерывание от таймер 1 – тик на 1 ms -----
ISR(TIMER1_COMPA_vect) {
    if (running) ++msElapsed;           // Броим само ако работи
}

// ---- Прерывание при смяна на бутоните -----
ISR(PCINT0_vect) {
    static uint8_t prevState = 0xFF;    // Предишно състояние на PORTB
    uint8_t curr = PINB;               // Текущо четене

    // Пад на BTN_START → събитие старт/пауза
    if ((prevState & _BV(BTN_START)) && !(curr & _BV(BTN_START))) {
        startEvent = true;
    }

    // Пад на BTN_RESET → събитие нулиране (само ако не върви)
    if ((prevState & _BV(BTN_RESET)) && !(curr & _BV(BTN_RESET)) && !running) {
        resetEvent = true;
    }

    prevState = curr;
}

```