# 169224_zad_3 – raport z zadania 3

Jakub Budzich – nr albumu 169224

**Platforma:** MPLAB X IDE + XC16
**Mikrokontroler:** PIC24FJ128GA010

## Opis funkcjonalności programów

Program to alarm który po przekroczeniu nastawy świeci jedna dioda  i po 5 sekundach zaczynają się świecić wszystkie diody do momentu restartu przyciskiem lub zmniejszenia wartości nastawy na wartość mniejszą niż połowa

| Nr | Funkcja | Opis działania |
|---|---|---|
| 1 | void alarm() | Główna funkcja programu implementująca alarm o działaniu opisanym powyżej |
| 1a | ALARM_OFF | Alarm wyłączony - wszystkie diody zgaszone |
| 1b | ALARM_MRUGANIE | Mruganie pierwszej diody przez 5 sekund jako ostrzeżenie |
| 1c | ALARM_WSZYSTKIE | Wszystkie diody zapalone (do momentu restartu/zmiany nastawy) |

## Działanie kontrolera

### Regulacje alarmu wykorzystuje przetwornik ADC:

- Potencjometr podłączony do pinu AN5
- Wartość odczytywana w zakresie 0-1023 (10-bitowy ADC)
- Program uruchamia alarm jeśli wartość przekroczy 512
- Alarm jest resetowany za pomocą przycisku RD6 albo zmiany nastawy poniżej 512

## Wyjaśnienie poszczególnych lini kodu

0. Konfiguracja sprzętowa i inicjalizacja

```
#pragma config POSCMOD = NONE       // Primary Oscillator Select
#pragma config OSCIOFNC = ON        // Primary Oscillator Output Function
#pragma config FCKSM = CSDCMD       // Clock Switching and Monitor
#pragma config FNOSC = FRC          // Oscillator Select
#pragma config IESO = OFF           // Internal External Switch OverMode
#pragma config WDTPS = PS32768      // Watchdog Timer Postscaler
#pragma config FWPSA = PR128        // WDT Prescaler
#pragma config WINDIS = ON          // Watchdog Timer Window
#pragma config FWDTEN = OFF         // Watchdog Timer Enable
#pragma config ICS = PGx2           // Comm Channel Select
#pragma config GWRP = OFF           // General Code Segment WriteProtect
#pragma config GCP = OFF            // General Code Segment Code Protect
#pragma config JTAGEN = OFF         // JTAG Port Enable
```

- POSCMOD = NONE - wybór trybu oscylatora (brak zewnętrznego oscylatora)
- OSCIOFNC = ON - funkcja wyjścia oscylatora włączona
- FCKSM = CSDCMD - przełączanie zegara i monitor wyłączone
- FNOSC = FRC - wybór wewnętrznego oscylatora RC
- IESO = OFF - tryb przełączania między oscylatorami wyłączony
- WDTPS = PS32768, FWPSA = PR128, WINDIS = ON, FWDTEN = OFF - konfiguracja watchdog timera (wyłączony)
- ICS = PGx2 - wybór kanału komunikacyjnego dla debuggera
- GWRP = OFF, GCP = OFF - wyłączenie ochrony pamięci programu
- JTAGEN = OFF - wyłączenie portu JTAG
- Zdefiniowana jest częstotliwość kwarcu: _XTAL_FREQ 8000000 (8 MHz)

1. Stany alarmu

```
#define ALARM_OFF 0
#define ALARM_MRUGANIE 1
#define ALARM_WSZYSTKIE 2
```

Zdefiniowane są trzy możliwe stany alarmu:

- **ALARM_OFF (0) -** alarm wyłączony
- **ALARM_MRUGANIE (1) -** mruganie jedną diodą (ostrzeżenie)
- **ALARM_WSZYSTKIE (2) -** wszystkie diody zapalone (pełny alarm)

2. Zmienne globalne

```
volatile uint16_t wartosc_potencjometru = 0;  // wartosc odczytana z
potencjometru
volatile uint16_t nastawa_alarmowa = 512;     // nastawa alarmowa (polowa
zakresu 0-1023)
volatile uint8_t stan_alarmu = ALARM_OFF;     // aktualny stan alarmu
volatile uint32_t licznik_czasu = 0;          // licznik czasu do odmierzania
5 sekund
volatile uint32_t licznik_mrugania = 0;       // licznik do kontroli
czestotliwosci mrugania
volatile uint8_t mruganie_stan = 0;           // stan mrugania diody (0 -
zgaszona, 1 - zapalona)
```

- **wartosc_potencjometru** - przechowuje aktualną wartość odczytaną z przetwornika ADC (potencjometr)
- **nastawa_alarmowa** - próg, po przekroczeniu którego włącza się alarm (ustawiony na 512, czyli połowę zakresu 0-1023)
- **stan_alarmu** - aktualny stan systemu alarmowego
- **licznik_czasu** - licznik do odmierzania czasu 5 sekund w stanie ALARM_MRUGANIE
- **licznik_mrugania** - licznik określający częstotliwość mrugania diody
- **mruganie_stan** - określa, czy dioda jest aktualnie zapalona (1) czy zgaszona (0)

3. Czas i częstotliwość mrugania

```
#define CZAS_MRUGANIA 450 // calkowity czas fazy mrugania (ilosc iteracji)
#define CZESTOTL_MRUGANIA 40   // co ile iteracji zmienic stan diody
```

- CZAS_MRUGANIA - całkowity czas, przez który dioda będzie mrugać (450 iteracji przy delay(25) w main daje około 5 sekund)
- CZESTOTL_MRUGANIA - częstotliwość zmiany stanu diody (co 40 iteracji)

4. Funkcja delay do opóźnienia

```
void delay(uint32_t czas) {
    uint32_t i, j;
    for(i = 0; i < czas; i++) {
        for(j = 0; j < 100; j++) {
            asm("NOP");
        }
    }
}
```

- Wykorzystuje dwie zagnieżdżone pętle for
- **asm("NOP")** - instrukcja "No Operation", która nie wykonuje żadnej operacji, ale zajmuje jeden cykl procesora
- Parametr czas określa liczbę iteracji zewnętrznej pętli

5. Funkcja init( )

```
void initADC() {
    // Konfiguracja portu analogowego
    AD1PCFGbits.PCFG5 = 0;     // AN5 jako wejscie analogowe
    AD1CON1 = 0x00E0;          // SSRC = 111 zako?cz konwersje na samymkoncu
    AD1CON2 = 0;
    AD1CON3 = 0x1F3F;          // Czas probkowania = 31Tad, Tad = 64Tcy
    AD1CHS = 5;                // Wybor kana?u AN5 (potencjometr)
    AD1CON1bits.ADON = 1;      // Wlacz modul ADC
}
```

Funkcja **initADC** inicjalizuje przetwornik analogowo-cyfrowy **(ADC)**

- **AD1PCFGbits.PCFG5** = 0 - ustawia pin AN5 jako wejście analogowe
- **AD1CON1** = 0x00E0 - konfiguracja rejestru kontrolnego 1
- SSRC = 111 - automatyczne zakończenie próbkowania i rozpoczęcie konwersji
- **AD1CON2** = 0 - domyślna konfiguracja rejestru kontrolnego 2
- **AD1CON3** = 0x1F3F - konfiguracja rejestru kontrolnego 3
- Czas próbkowania = 31Tad
- Tad = 64Tcy (czas konwersji)
- **AD1CHS** = 5 - wybór kanału 5 (AN5) do konwersji
- **AD1CON1bits.ADON** = 1 - włączenie modułu ADC

6. Funkcja odczytu potencjometru

```
uint16_t czytajPotencjometr() {
    AD1CON1bits.SAMP = 1;      // Rozpocznij probkowanie
    __delay32(100);            // Daj czas na probkowanie
    AD1CON1bits.SAMP = 0;      // Rozpocznij konwersje
    while (!AD1CON1bits.DONE); // Czekaj na zakonczenie konwersji
    return ADC1BUF0;           // Zwroc wynik
}
```

- **AD1CON1bits.SAMP** = 1 - rozpoczyna próbkowanie
- **__delay32(100)** - opóźnienie dla zapewnienia odpowiedniego czasu próbkowania
- **AD1CON1bits.SAMP** = 0 - kończy próbkowanie i rozpoczyna konwersję
- **while (!AD1CON1bits.DONE)** - czeka na zakończenie konwersji
- **return ADC1BUF0** - zwraca wynik z rejestru bufora ADC

7. init( ) – funkcja inicjująca porty

```
void init() {
    AD1PCFG = 0xFFDF;          // Wszystkie piny cyfrowe oprocz AN5
    TRISA = 0x0000;            // Port A jako wyjscie
    TRISD = 0xFFFF;            // Port D jako wejscie

    // Konfiguracja przerwan od pinow
    CNPU1bits.CN15PUE = 1;     // Pull-up dla RD6 (przycisk wylaczenia alarmu)

    // Wlaczenie przerwan dla przyciskow
    CNEN1bits.CN15IE = 1;      // Wlacz przerwanie dla RD6

    IFS1bits.CNIF = 0;         // Wyczysc flage przerwania CN
    IEC1bits.CNIE = 1;         // Wlacz przerwania CN

    // Inicjalizacja ADC
    initADC();

    // Wszystkie diody poczatkowo wylaczone
    LATA = 0x0000;
}
```

- **AD1PCFG** = 0xFFDF - wszystkie piny są ustawione jako cyfrowe oprócz AN5 (bit 5 jest wyzerowany)
- **TRISA** = 0x0000 - ustawienie portu A jako wyjściowego (dla diod LED)
- **TRISD** = 0xFFFF - ustawienie portu D jako wejściowego (dla przycisków)

- **CNPU1bits.CN15PUE** = 1 - włączenie pull-up dla pinu RD6 (przycisk wyłączenia alarmu)
- **CNEN1bits.CN15IE** = 1 - włączenie przerwania Change Notification dla pinu RD6
- **IFS1bits.CNIF** = 0 - wyczyszczenie flagi przerwania CN
- **IEC1bits.CNIE** = 1 - włączenie przerwań CN
- **initADC()** - wywołanie funkcji inicjalizującej ADC
- **LATA** = 0x0000 - wyłączenie wszystkich diod na porcie A

8. _CNInterrupt( ) - obsługa przerwań przyciskami

```
void __attribute__((interrupt, no_auto_psv)) _CNInterrupt(void) {
    __delay32(200);

    // Sprawdzenie, czy przycisk RD6 zostal nacisniety (wylaczenie alarmu)
    if(PORTDbits.RD6 == 0) {
        stan_alarmu = ALARM_OFF;
        LATA = 0x0000;  // Wylacz wszystkie diody
    }

    while(PORTDbits.RD6 == 0);  // Czekaj na zwolnienie przycisku

    // Wyczyszczenie flagi przerwania
    IFS1bits.CNIF = 0;
}
```

- **__delay32(200)** - opóźnienie dla eliminacji drgań styków przycisku
- Sprawdza, czy przycisk **RD6** został naciśnięty (stan niski)
- Jeśli tak, ustawia **stan_alarmu** na **ALARM_OFF** i wyłącza wszystkie diody
- Pętla **while(PORTDbits.RD6 == 0)** czeka, aż przycisk zostanie zwolniony
- **IFS1bits.CNIF** = 0 - czyści flagę przerwania

9. Funkcja `alarm()`

```
 void alarm() {
    // Odczyt wartosci z potencjometru
    wartosc_potencjometru = czytajPotencjometr();

    // Sprawdzenie warunkow alarmu
    switch(stan_alarmu) {
        case ALARM_OFF:
            // Jesli wartosc przekroczyla nastawe, uruchom alarm (mruganie
jednej diody)
            if(wartosc_potencjometru > nastawa_alarmowa) {
                stan_alarmu = ALARM_MRUGANIE;
                licznik_czasu = 0;       // Zresetuj licznik czasu
                licznik_mrugania = 0;    // Zresetuj licznik mrugania
                mruganie_stan = 0;      // Rozpocznij od zgaszonej diody
            }
            break;

        case ALARM_MRUGANIE:
            // Zwieksz licznik czasu
            licznik_czasu++;
            licznik_mrugania++;

            // Mruganie jedna dioda co CZESTOTL_MRUGANIA cykli
            if(licznik_mrugania >= CZESTOTL_MRUGANIA) {
                licznik_mrugania = 0;

                if(mruganie_stan == 0) {
                    LATA = 0x0001;  // Zapal pierwsz? diode
                    mruganie_stan = 1;
                } else {
                    LATA = 0x0000;  // Zgas wszystkie diody
                    mruganie_stan = 0;
                }
            }

            // Po czasie okreslonym przez CZAS_MRUGANIA przejdz do stanu
ALARM_WSZYSTKIE
            if(licznik_czasu >= CZAS_MRUGANIA) {
                stan_alarmu = ALARM_WSZYSTKIE;
            }
```

```
            // Jesli wartosc spadla ponizej nastawy, wylacz alarm
            if(wartosc_potencjometru < nastawa_alarmowa) {
                stan_alarmu = ALARM_OFF;
                LATA = 0x0000;  // Wylacz wszystkie diody
            }
          break;

        case ALARM_WSZYSTKIE:
            LATA = 0x00FF;  // Zapal wszystkie diody

            // Jesli wartocs spadla ponizej nastawy, wylacz alarm
            if(wartosc_potencjometru < nastawa_alarmowa) {
                stan_alarmu = ALARM_OFF;
                LATA = 0x0000;  // Wylacz wszystkie diody
            }
            break;
    }
}
```

- Program najpierw odczytuje wartość z potencjometru
- Sprawdza następnie wartość alarmu i wykonuje następujące akcje
    - Stan **ALARM_OFF**
        - Jeśli wartość potencjometru przekracza nastawę alarmową, przechodzi do stanu **ALARM_MRUGANIE**
        - Resetuje liczniki czasu i mrugania
        - Ustawia stan mrugania na 0 (dioda zgaszona)
    - Stan **ALARM_MRUGANIE**
        - Zwiększa liczniki czasu i mrugania
        - Co **CZESTOTL_MRUGANIA** cykli zmienia stan diody (zapalona/zgaszona)
        - Jeśli licznik czasu osiągnie wartość **CZAS_MRUGANIA** (5 sekund), przechodzi do stanu **ALARM_WSZYSTKIE**
        - Jeśli wartość potencjometru spadnie poniżej nastawy, powraca do stanu ALARM_OFF i wyłącza diody
    - Stan **ALARM_WSZYSTKIE**
        - Zapala wszystkie diody (0x00FF)
        - Jeśli wartość potencjometru spadnie poniżej nastawy, powraca do stanu **ALARM_OFF** i wyłącza diody
        - Jeśli naciśnięty zostanie przycisk **RD6** to alarm przechodzi w stan **ALARM_MRUGANIE**

10. main( ) – główna funkcja programu

```
int main(void) {
    init();

    while(1) {
        alarm();

        delay(25);
    }

    return 0;
}
```

- Wywołuje funkcję init() do inicjalizacji mikrokontrolera
- Wchodzi w nieskończoną pętlę, w której
    - Wywołuje funkcję alarm() do obsługi systemu alarmowego
    - Wprowadza opóźnienie 25 jednostek ( co daje finalnie 5 sekund)

11. Cały kod programu:

```
/*
 * File:   main.c
 * Author: Jakub Budzich - 169224
 *
 * Created on 19 maj 2025, 08:13
 */
 #pragma config POSCMOD = NONE      // Primary Oscillator Select
 #pragma config OSCIOFNC = ON       // Primary Oscillator Output Function
 #pragma config FCKSM = CSDCMD      // Clock Switching and Monitor
 #pragma config FNOSC = FRC         // Oscillator Select
 #pragma config IESO = OFF          // Internal External Switch Over Mode

 #pragma config WDTPS = PS32768     // Watchdog Timer Postscaler
 #pragma config FWPSA = PR128       // WDT Prescaler
 #pragma config WINDIS = ON         // Watchdog Timer Window
 #pragma config FWDTEN = OFF        // Watchdog Timer Enable
 #pragma config ICS = PGx2          // Comm Channel Select
 #pragma config GWRP = OFF          // General Code Segment Write Protect
 #pragma config GCP = OFF           // General Code Segment Code Protect
 #pragma config JTAGEN = OFF        // JTAG Port Enable

 #define _XTAL_FREQ 8000000
 #include <xc.h>
 #include <libpic30.h>
 #include <stdlib.h>
```

```
 #include "p24FJ128GA010.h"

 // Definicje stan w alarmu
 #define ALARM_OFF 0
 #define ALARM_MRUGANIE 1
 #define ALARM_WSZYSTKIE 2

 volatile uint16_t wartosc_potencjometru = 0;  // wartosc odczytana z
potencjometru
 volatile uint16_t nastawa_alarmowa = 512;     // nastawa alarmowa (polowa
zakresu 0-1023)
 volatile uint8_t stan_alarmu = ALARM_OFF;     // aktualny stan alarmu
 volatile uint32_t licznik_czasu = 0;          // licznik czasu do odmierzania
5 sekund
 volatile uint32_t licznik_mrugania = 0;       // licznik do kontroli
czestotliwosci mrugania
 volatile uint8_t mruganie_stan = 0;           // stan mrugania diody (0 -
zgaszona, 1 - zapalona)

 // parametry do zarzadzania mruganiem jednej diody i czasem gdy wszystkie sie
zaswieca
 #define CZAS_MRUGANIA 450      // calkowity czas fazy mrugania (ilosc
iteracji)
                                // warto?? 450 daje 5 sekund przy delay(25) w
main
 #define CZESTOTL_MRUGANIA 40   // co ile iteracji zmienic stan diody


 // Funkcja opoznienia
 void delay(uint32_t czas) {
     uint32_t i, j;
     for(i = 0; i < czas; i++) {
         for(j = 0; j < 100; j++) {
             asm("NOP");
         }
     }
 }

 // Inicjalizacja ADC do odczytu potencjometru
 void initADC() {
     // Konfiguracja portu analogowego
     AD1PCFGbits.PCFG5 = 0;    // AN5 jako wejscie analogowe
     AD1CON1 = 0x00E0;         // SSRC = 111 zako?cz konwersje na samym koncu
     AD1CON2 = 0;
     AD1CON3 = 0x1F3F;         // Czas probkowania = 31Tad, Tad = 64Tcy
     AD1CHS = 5;               // Wybor kana?u AN5 (potencjometr)
     AD1CON1bits.ADON = 1;     // Wlacz modul ADC
```

```
}

// Funkcja do odczytu wartosci potencjometru
uint16_t czytajPotencjometr() {
    AD1CON1bits.SAMP = 1;     // Rozpocznij probkowanie
    __delay32(100);           // Daj czas na probkowanie
    AD1CON1bits.SAMP = 0;     // Rozpocznij konwersje
    while (!AD1CON1bits.DONE); // Czekaj na zakonczenie konwersji
    return ADC1BUF0;          // Zwroc wynik
}

// Inicjalizacja portow i przerwan
void init() {
    AD1PCFG = 0xFFDF;         // Wszystkie piny cyfrowe oprocz AN5
    TRISA = 0x0000;           // Port A jako wyjscie
    TRISD = 0xFFFF;           // Port D jako wejscie

    // Konfiguracja przerwan od pinow
    CNPU1bits.CN15PUE = 1;    // Pull-up dla RD6 (przycisk wylaczenia alarmu)

    // Wlaczenie przerwan dla przyciskow
    CNEN1bits.CN15IE = 1;     // Wlacz przerwanie dla RD6

    IFS1bits.CNIF = 0;        // Wyczysc flage przerwania CN
    IEC1bits.CNIE = 1;        // Wlacz przerwania CN

    // Inicjalizacja ADC
    initADC();

    // Wszystkie diody poczatkowo wylaczone
    LATA = 0x0000;
}

// Procedura obs?ugi przerwania przyciskami
void __attribute__((interrupt, no_auto_psv)) _CNInterrupt(void) {
    __delay32(200);

    // Sprawdzenie, czy przycisk RD6 zostal nacisniety (wylaczenie alarmu)
    if(PORTDbits.RD6 == 0) {
        stan_alarmu = ALARM_OFF;
        LATA = 0x0000;  // Wylacz wszystkie diody
    }

    while(PORTDbits.RD6 == 0);  // Czekaj na zwolnienie przycisku

    // Wyczyszczenie flagi przerwania
    IFS1bits.CNIF = 0;
```

```
 }

 void alarm() {
     // Odczyt wartosci z potencjometru
     wartosc_potencjometru = czytajPotencjometr();

     // Sprawdzenie warunkow alarmu
     switch(stan_alarmu) {
         case ALARM_OFF:
             // Jesli wartosc przekroczyla nastawe, uruchom alarm (mruganie
jednej diody)
             if(wartosc_potencjometru > nastawa_alarmowa) {
                 stan_alarmu = ALARM_MRUGANIE;
                 licznik_czasu = 0;       // Zresetuj licznik czasu
                 licznik_mrugania = 0;    // Zresetuj licznik mrugania
                 mruganie_stan = 0;      // Rozpocznij od zgaszonej diody
             }
             break;

         case ALARM_MRUGANIE:
             // Zwieksz licznik czasu
             licznik_czasu++;
             licznik_mrugania++;

             // Mruganie jedna dioda co CZESTOTL_MRUGANIA cykli
             if(licznik_mrugania >= CZESTOTL_MRUGANIA) {
                 licznik_mrugania = 0;

                 if(mruganie_stan == 0) {
                     LATA = 0x0001;  // Zapal pierwsz? diode
                     mruganie_stan = 1;
                 } else {
                     LATA = 0x0000;  // Zgas wszystkie diody
                     mruganie_stan = 0;
                 }
             }

             // Po czasie okreslonym przez CZAS_MRUGANIA przejdz do stanu
ALARM_WSZYSTKIE
             if(licznik_czasu >= CZAS_MRUGANIA) {
                 stan_alarmu = ALARM_WSZYSTKIE;
             }

             // Jesli wartosc spadla ponizej nastawy, wylacz alarm
             if(wartosc_potencjometru < nastawa_alarmowa) {
                 stan_alarmu = ALARM_OFF;
                 LATA = 0x0000;  // Wylacz wszystkie diody
```

```
            }
            break;

        case ALARM_WSZYSTKIE:
            LATA = 0x00FF;  // Zapal wszystkie diody

            // Jesli wartocs spadla ponizej nastawy, wylacz alarm
            if(wartosc_potencjometru < nastawa_alarmowa) {
                stan_alarmu = ALARM_OFF;
                LATA = 0x0000;  // Wylacz wszystkie diody
            }
            break;
    }
}

// Glowna funkcja programu
int main(void) {
    init();

    while(1) {
        alarm();

        delay(25);
    }

    return 0;
}
```