169224_zad_2 – raport z zadania 2

Jakub Budzich – nr albumu 169224

Platforma: MPLAB X IDE + XC16
Mikrokontroler: PIC24FJ128GA010

Opis funkcjonalności programów

Program oferuje **2 funkcje**, które użytkownik przełącza przy pomocy dwóch przycisków. Wszystkie funkcje posiadają zmienną prędkość działania sterowaną potencjometrem podłączonym do pinu AN5.

Nr	Funkcja	Opis działania
1	snake( )	"Wężyk" 3-bitowy poruszający się w lewo i prawo z prędkością regulowaną potencjometrem
2	binDown()	8-bitowy licznik binarny zliczający w dół od 255 do 0 z prędkością regulowaną potencjometrem

Przełączanie funkcji

Obsługa przycisków odbywa się przez przerwanie CNInterrupt():

- RD13 → poprzedni program
- RD6 → następny program
- Flaga "flaga = 1" przerywa aktualnie działającą funkcję

Regulacja prędkości wykorzystuje przetwornik ADC:

- Potencjometr podłączony do pinu AN5
- Wartość odczytywana w zakresie 0-1023 (10-bitowy ADC)
- Program dzieli zakres na 5 różnych poziomów prędkości

Wyjaśnienie poszczególnych lini kodu

0. Konfiguracja sprzętowa i inicjalizacja

- `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. Zmienne globalne

```
volatile uint16_t numer_programu = 1;
volatile uint8_t flaga = 0; // flaga informujaca o zmianie programu
volatile uint16_t predkosc = 0; // wartosc potencjometru
```

- **numer_programu**: wybór aktywnej funkcji
- **flaga**: zatrzymanie bieżącego działania (np. po naciśnięciu przycisku)
- **predkosc**: przechowuje wartość odczytaną z potencjometru (0-1023)

Wybranie volatile -> informuje kompilator, że jej wartość może być zmieniana w sposób, który nie jest bezpośrednio kontrolowany przez kod programu, np. przez zewnętrzne przerwania

2. Funkcja delay

```
// Funkcja opoznienie z regulacja predkosci
void delay(uint32_t podstawa) {
     uint32_t ilosc, i, j;

     // Przeliczenie opoznienia na podstawie odczytu z potencjometru
    // predkosc bedzie w zakresie 0-1023 (10-bitowy ADC)
    // i dzielmy na 5 roznych poziomow predkosci
     // 5 roznych predkosci bez zachowanego porządku aby było widoczne
że istnieje 5 różnych stanów
     if (predkosc < 205)         ilosc = podstawa /2;  // szybko
     else if (predkosc < 410)    ilosc = podstawa * 4;  // wolno
     else if (predkosc < 615)    ilosc = podstawa /4;  // jeszcze
szybciej niz przy 205 wartosci
     else if (predkosc < 820)    ilosc = podstawa * 12;  // najwolniej
     else                        ilosc = podstawa /8;  // najszybiciej
```

```
    for(i = 0; i < ilosc; i++) {
        for(j = 0; j < 100; j++) {
            asm("NOP");
        }
    }
}
```

- Funkcja przyjmuje podstawową wartość opóźnienia (podstawa)
- Całkowite opóźnienie jest przeliczane w zależności od wartości predkosc
- Wartość predkosc dzielona jest na 5 przedziałów:

< 205: opóźnienie = podstawa / 2 (szybko)

< 410: opóźnienie = podstawa * 4 (wolno)

< 615: opóźnienie = podstawa / 4 (szybciej niż przy < 205)

< 820: opóźnienie = podstawa * 12 (najwolniej)

= 820: opóźnienie = podstawa / 8 (najszybciej)

Opóźnienie realizowane jest przez zagnieżdżone pętle i instrukcję asm("NOP")

3. Funkcja `initADC()`

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

- Konfiguruje AN5 jako wejście analogowe (pin dla potencjometru)
- Ustawia parametry konwersji ADC:
- Tryb zakończenia konwersji (SSRC = 111)
- Czas próbkowania 31Tad, gdzie Tad = 64Tcy
- Wybór kanału 5 (AN5)
- Włączenie modułu ADC

4. Funkcja do odczytu wartości 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
}
```

- Rozpoczyna próbkowanie ustawiając bit SAMP
- Daje czas na próbkowanie przez opóźnienie
- Rozpoczyna konwersję przez wyzerowanie bitu SAMP
- Czeka na zakończenie konwersji (sprawdzając bit DONE)
- Zwraca odczytaną wartość z rejestru ADC1BUF0 (0-1023)

5. Funkcja init( )

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

     // Konfiguracja przerwan od pinow
     CNPU2bits.CN19PUE = 1;     // Pull-up dla RD13
     CNPU1bits.CN15PUE = 1;     // Pull-up dla RD6

     // Wlaczenie przerwan dla przyciskow
     CNEN1bits.CN15IE = 1;      // Wlacz przerwanie dla RD6
     CNEN2bits.CN19IE = 1;      // Wlacz przerwanie dla RD13

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

     // Inicjalizacja ADC
     initADC();
}
```

- Ustawia wszystkie piny jako cyfrowe z wyjątkiem AN5 (0xFFDF = 0b1111111111011111)
- Konfiguruje port A jako wyjście (dla diod LED)
- Konfiguruje port D jako wejście (dla przycisków)
- Włącza pull-upy dla przycisków RD13 i RD6
- Aktywuje przerwania Change Notification dla przycisków
- Wyczyszcza flagę przerwania i włącza przerwania CN
- Wywołuje funkcję initADC() do inicjalizacji przetwornika ADC)

6. Funkcja przerwania przycisków _CNInterrupt()

```
void __attribute__((interrupt, no_auto_psv)) _CNInterrupt(void) {
    __delay32(200);
    // Sprawdzenie, ktory przycisk zostal nacisniety
    // poprzedni program
    if(PORTDbits.RD13 == 0) {
        numer_programu--;
        flaga = 1;
    }
    // nastepny program
    else if(PORTDbits.RD6 == 0) {
        numer_programu++;
        flaga = 1;
    }

    while(PORTDbits.RD13 == 0 || PORTDbits.RD6 == 0);
    // Wyczyszczenie flagi
    IFS1bits.CNIF = 0;
}
```

- Obsługuje naciśnięcia przycisków RD6 i RD13
- Zmienia numer programu (`numer_programu++` lub `--`)
- Ustawia `flaga = 1`, co sygnalizuje zakończenie aktualnej funkcji
- Pętla `while()` czeka aż użytkownik puści przycisk
- Zmienia numer programu (inkrementacja lub dekrementacja)
- Ustawia flagę przerwania aktualnej funkcji
- Czeka na zwolnienie przycisków i czyści flagę przerwania

7. snake( )

```
void snake() {
   unsigned char wez = 0b00000111;
    uint16_t kierunek = 1;
    flaga = 0;

    while(!flaga) {
        // Odczyt potencjometru przed kazda iteracja
        predkosc = czytajPotencjometr();

        LATA = wez;
        delay(150); //delay okreslany wartoscia z potencjometru

        if (kierunek == 1) {
            wez <<= 1;
            if (wez > 0b01111000) {
               wez = 0b11100000;
                kierunek = -1;
            }
        }
        else if (kierunek == -1) {
            wez >>= 1;
            if (wez < 0b00000111) {
                wez = 0b00000111 << 1;
                kierunek = 1;
            }
        }
    }
}
```

- Inicjalizuje wez wartością 0b00000111 (3 aktywne bity "wężyka")
- Ustawia początkowy kierunek na 1 (prawo)
- Zeruje flagę zmiany programu
- Odczytuje aktualną wartość potencjometru i zapisuje w zmiennej predkosc
- Wyświetla aktualną pozycję wężyka na porcie A
- Wprowadza opóźnienie zależne od wartości potencjometru
- Aktualizuje pozycję wężyka w zależności od kierunku:
- Dla kierunku 1 (prawo): przesuwa bity w lewo, sprawdza czy osiągnięto prawą granicę, zmienia kierunek
- Dla kierunku -1 (lewo): przesuwa bity w prawo, sprawdza czy osiągnięto lewą granicę, zmienia kierunek
- Pętla kończy się po naciśnięciu przycisku (gdy flaga = 1)

8. binDown( )

```
void binDown(){
     unsigned char licznik = 255;
     flaga = 0;
     while(!flaga) {
         // program odczytuje przed kazda iteracja wartosc z potencjometru
         predkosc = czytajPotencjometr();

         LATA = licznik--;
         delay(150); //delay okreslany wartoscia z potencjometru
     }
 }
```

- Inicjalizuje licznik wartością 255 (maksymalna wartość 8-bitowa)
- Zeruje flagę zmiany programu
- Odczytuje aktualną wartość potencjometru i zapisuje w zmiennej predkosc
- Wyświetla aktualną wartość licznika na porcie A (diody LED)
- Dekrementuje licznik
- Wprowadza opóźnienie zależne od wartości potencjometru
- Pętla kończy się po naciśnięciu przycisku (gdy flaga = 1)

9. Funkcja `main()` - Główna pętla programu

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

    while(1) {
        if(numer_programu < 1) {
            numer_programu = 2;
         } else if(numer_programu > 2) {
            numer_programu = 1;
         }
         switch(numer_programu) {
            case 1:
                snake();
                break;
            case 2:
                binDown();
                break;
         }
     }
     return 0;
 }
```

- Wywołuje funkcję init() w celu inicjalizacji mikrokontrolera
- Wchodzi w nieskończoną pętlę:
- Sprawdza granice numeru programu (1-2)
- Używa instrukcji switch do wyboru właściwego programu
- Wywołuje odpowiednią funkcję (snake() lub binDown())
- Po powrocie z funkcji (co następuje po zmianie flagi), pętla kontynuuje działanie i sprawdza nowy numer programu

10. Cały kod programu:

```
/*
 * File:   main.c
 * Author: Jakub Budzich - 169224
 *
 * Created on 12 maj 2025, 08:17
 * Modified for potentiometer control
 */
#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"

volatile uint16_t numer_programu = 1;
volatile uint8_t flaga = 0; // flaga informujaca o zmianie programu
volatile uint16_t predkosc = 0; // wartosc potencjometru

// Funkcja opoznienie z regulacja predkosci
void delay(uint32_t podstawa) {
    uint32_t ilosc, i, j;
```

```
    // Przeliczenie opoznienia na podstawie odczytu z potencjometru
    // predkosc bedzie w zakresie 0-1023 (10-bitowy ADC)
    // i dzielmy na 5 roznych poziomow predkosci
    // 5 roznych predkosci bez zachowanego porządku aby było widoczne że
istnieje 5 różnych stanów
    if (predkosc < 205)         ilosc = podstawa /2;  // szybko
    else if (predkosc < 410)    ilosc = podstawa * 4;  // wolno
    else if (predkosc < 615)    ilosc = podstawa /4;  // jeszcze szybciej niz
przy 205 wartosci
    else if (predkosc < 820)    ilosc = podstawa * 12;  // najwolniej
    else                        ilosc = podstawa /8;  // najszybiciej

    for(i = 0; i < ilosc; 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 zakoncz konwersje na samym koncu
    AD1CON2 = 0;
    AD1CON3 = 0x1F3F;         // Czas probkowania = 31Tad, Tad = 64Tcy
    AD1CHS = 5;               // Wybor kanalu 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 wyj?cie
    TRISD = 0xFFFF;           // Port D jako wej?cie

    // Konfiguracja przerwan od pinow
```

```
    CNPU2bits.CN19PUE = 1;    // Pull-up dla RD13
    CNPU1bits.CN15PUE = 1;    // Pull-up dla RD6

    // Wlaczenie przerwan dla przyciskow
    CNEN1bits.CN15IE = 1;     // Wlacz przerwanie dla RD6
    CNEN2bits.CN19IE = 1;     // Wlacz przerwanie dla RD13

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

    // Inicjalizacja ADC
    initADC();
}

// Procedura obslugi przerwania przyciskami
void __attribute__((interrupt, no_auto_psv)) _CNInterrupt(void) {
    __delay32(200);
    // Sprawdzenie, ktory przycisk zostal nacisniety
    // poprzedni program
    if(PORTDbits.RD13 == 0) {
        numer_programu--;
        flaga = 1;
    }
    // nastepny program
    else if(PORTDbits.RD6 == 0) {
        numer_programu++;
        flaga = 1;
    }

    while(PORTDbits.RD13 == 0 || PORTDbits.RD6 == 0);
    // Wyczyszczenie flagi
    IFS1bits.CNIF = 0;
}

// 1. Snake wezyk od prawej do lewej
void snake() {
    unsigned char wez = 0b00000111;
    uint16_t kierunek = 1;
    flaga = 0;

    while(!flaga) {
        // Odczyt potencjometru przed kazda iteracja
        predkosc = czytajPotencjometr();

        LATA = wez;
        delay(150); //delay okreslany wartoscia z potencjometru
```

```
        if (kierunek == 1) {
            wez <<= 1;
            if (wez > 0b01111000) {
                wez = 0b11100000;
                kierunek = -1;
            }
        }
        else if (kierunek == -1) {
            wez >>= 1;
            if (wez < 0b00000111) {
                wez = 0b00000111 << 1;
                kierunek = 1;
            }
        }
    }
}

// 2. licznik wartosci binarnych w dol
void binDown(){
    unsigned char licznik = 255;
    flaga = 0;
    while(!flaga) {
        // program odczytuje przed kazda iteracja wartosc z potencjometru
        predkosc = czytajPotencjometr();

        LATA = licznik--;
        delay(150); //delay okreslany wartoscia z potencjometru
    }
}
// Glowna funkcja programu z wyborem programu
int main(void) {
    init();

    while(1) {
        if(numer_programu < 1) {
            numer_programu = 2;
        } else if(numer_programu > 2) {
            numer_programu = 1;
        }
        switch(numer_programu) {
            case 1:
                snake();
                break;
            case 2:
                binDown();
                break;
        }
```

```
    }
    return 0;
}
```