Raport do zadania 5

Imię i nazwisko: Jakub Dmochowski

Numer Albumu: 169236

Konfiguracja sprzętowa

```
#pragma config POSCMOD = XT
#pragma config OSCIOFNC = ON
#pragma config FCKSM = CSDCMD
#pragma config FNOSC = PRI
#pragma config IESO = ON
```

POSCMOD = XT - wybór zewnętrznego kwarcu jako źródła zegara

OSCIOFNC = ON - funkcja wyjścia oscylatora włączona

FCKSM = CSDCMD - wyłączenie monitorowania zegara i przełączania

FNOSC = PRI - Podstawowe źródło zegara jako domyślne

IESO = ON - włączenie wewnętrznego/zewnętrznego przełączania oscylatora

Watchdog timer, debugowanie

```
#pragma config WDTPS = PS32768
#pragma config FWPSA = PR128
#pragma config WINDIS = ON
#pragma config FWDTEN = OFF
#pragma config ICS = PGx2
#pragma config GWRP = OFF
#pragma config GCP = OFF
#pragma config JTAGEN = OFF
```

WDTPS = PS32768 - Prescaler watchdog timera ustawiony na 1:32768

FWPSA = PR128 - Prescaler A watchdog timera ustawiony na 1:128

WINDIS = ON - Windowed watchdog timer wyłączony

FWDTEN = OFF - Watchdog timer całkowicie wyłączony

ICS = PGx2 - Komunikacja z debuggerem przez piny PGC2/PGD2

GWRP = OFF - Ochrona zapisu do pamięci programu wyłączona

GCP = OFF - Ochrona odczytu kodu wyłączona

JTAGEN = OFF - Interfejs JTAG wyłączony

Biblioteki

```
#include <xc.h>
#include <libpic30.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include "lcd.h"
#include "adc.h"
```

#include <xc.h> - główna biblioteka dla kompilatorów XC

#include <libpic30.h> - biblioteka dla PIC30, zawiera funkcje opóźnień

#include <stdbool.h> - biblioteka standardowa C dla typu bool

#include <stdio.h> - biblioteka standardowa C dla funkcji I/O

#include <string.h> - biblioteka standardowa C dla operacji na stringach

#include "lcd.h" - biblioteka do obsługi LCD

#include "adc.h" - biblioteka do obsługi ADC

Definicje i zmienne globalne

```
#define FCY 4000000UL
#define DEBOUNCE_DELAY 200000
#define PRZYCISK_GRACZ1 PORTDbits.RD6
#define PRZYCISK_GRACZ2 PORTDbits.RD13

unsigned int czas_g1 = 0;
unsigned int czas_g2 = 0;
unsigned char aktywny_gracz = 0;
unsigned int wybrany_czas = 300;

volatile bool przycisk_g1_flag = false;
volatile bool przycisk_g2_flag = false;
volatile bool przycisk_flag = false;
```

#define FCY 4000000UL - częstotliwość procesora 4 MHz

#define DEBOUNCE_DELAY_MS 2000 - opóźnienie dla eliminacji drgań styków

#define PRZYCISK_GRACZ1 PORTDbits.RD6 - przycisk gracza 1

#define PRZYCISK_GRACZ2 PORTDbits.RD13 - przycisk gracza 2

unsigned int czas_g1, czas_g2 - czas pozostały dla każdego gracza

unsigned char aktywny_gracz = 0; - Numer gracza, którego czas aktualnie jest odliczany

unsigned int wybrany_czas - czas wybrany przez potencjometr

volatile bool przycisk_g1_flag = false; - Flaga ktora sygnalizuje wcisniecie przez gracza 1

volatile bool przycisk_g2_flag = false; - Flaga ktora sygnalizuje wcisniecie przez gracza 2

volatile bool przycisk_flag = false; - Flaga która sygnalizuje wciśnięcie jakiegokolwiek przycisku

Przerwania

```
void __attribute__((interrupt, auto_psv)) _CNInterrupt(void) {
    if (!PRZYCISK_GRACZ1) {
        przycisk_g1_flag = true;
    }
    if (!PRZYCISK_GRACZ2) {
        przycisk_g2_flag = true;
    }
    przycisk_flag = true;
    IFS1bits.CNIF = 0;
}
```

void attribute((interrupt, auto_psv)) _CNInterrupt(void) - Funkcja obsługi przerwania od zmiany stanu pinów

if (!PRZYCISK_GRACZ1) - Sprawdzenie czy przycisk gracza 1 został naciśnięty

przycisk_g1_flag = true; - Ustawienie flagi sygnalizującej naciśnięcie przycisku gracza 1

if (!PRZYCISK_GRACZ2) - Sprawdzenie czy przycisk gracza 2 został naciśnięty

przycisk_g2_flag = true; - Ustawienie flagi sygnalizującej naciśnięcie przycisku gracza 2

przycisk_flag = true; - Ustawienie ogólnej flagi informującej o naciśnięciu przycisku

IFS1bits.CNIF = 0; - Wyzerowanie flagi przerwania Change Notification

Opóźnienie

```
void opoznienie_ms(unsigned int ms) {
    while (ms--) {
        __delay32(FCY / 1000);
    }
}
```

while (ms--) - Pętla wykonywana określoną liczbę razy

__delay32(FCY / 1000); - Opóźnienie na 1 milisekundę bazujące na częstotliwości procesora

Wyświetlanie czasu gry

```
void pokaz_czas() {
    char bufor[33];
    char wskaznik1 = (aktywny_gracz == 1) ? '>' : ' ';
    char wskaznik2 = (aktywny_gracz == 2) ? '>' : ' ';

    sprintf(bufor, "%cG1: %02u:%02u\n%cG2: %02u:%02u",
            wskaznik1, czas_g1 / 60, czas_g1 % 60,
            wskaznik2, czas_g2 / 60, czas_g2 % 60);
    LCD_ClearScreen();
    LCD_PutString(bufor, strlen(bufor));
}
```

char bufor[33] - bufor dla sformatowanego tekstu

char wskaznik1 = (aktywny_gracz == 1) ? '>' : ' ' - wskaźnik dla gracza 1

char wskaznik2 = (aktywny_gracz == 2) ? '>' : ' ' - wskaźnik dla gracza 2

sprintf(bufor, "%cG1: %02u:%02u\n%cG2: %02u:%02u", ...) - formatowanie tekstu:

- %c - znak wskaźnika
- %02u - liczba bez znaku, minimum 2 cyfry z zerami
- czas_g1 / 60 - minuty
- czas_g1 % 60 - sekundy
- \n - znak nowej linii między graczami

LCD_ClearScreen() - wyczyszczenie wyświetlacza LCD

LCD_PutString(bufor, strlen(bufor)) - wyświetlenie tekstu z obliczeniem długości

Wyświetlanie komunikatu

```
void komunikat(char* msg) {
    LCD_ClearScreen();
    LCD_PutString(msg, strlen(msg));
}
```

LCD_ClearScreen() - wyczyszczenie wyświetlacza

LCD_PutString(msg, strlen(msg)) - wyświetlenie komunikatu z automatycznym obliczeniem długości

Wyświetlanie czasu

```
void pokaz_ustawienia_czasu() {
    char bufor[33];
    sprintf(bufor, "Nastaw czas:\n%u:%02u min", wybrany_czas / 60, wybrany_czas % 60);
    LCD_ClearScreen();
    LCD_PutString(bufor, strlen(bufor));
}
```

bufor[33] - bufor dla sformatowanego tekstu

sprintf(bufor, "Nastaw czas:\n%u:%02u min", wybrany_czas / 60, wybrany_czas % 60)

- "Nastaw czas:" - tekst instrukcji w pierwszej linii
- %u - liczba bez znaku bez wypełnienia zerami (minuty)
- %02u - sekundy z wypełnieniem zerami do 2 cyfr
- "min" - jednostka czasu

Wyświetla aktualnie wybrany czas z potencjometru

Odczyt z potencjometru

```
unsigned int odczytaj_czas_z_potencjometru() {
    uint16_t pot = ADC_Read10bit(ADC_CHANNEL_5);
    if (pot < 341) {
        return 60;
    } else if (pot < 682) {
        return 180;
    } else {
        return 300;
    }
}
```

uint16_t pot = ADC_Read10bit(ADC_CHANNEL_5) - odczyt 10-bitowej wartości ADC

Mapowanie zakresu na 3 opcje czasowe:

- if (pot < 341) return 60 - 1 minuta
- else if (pot < 682) return 180 - 3 minuty
- else return 300 - 5 minut

Zwraca czas w sekundach

Konfiguracja przerwań

```
void konfiguruj_przerwania() {

    CNEN1bits.CN15IE = 1;
    CNEN2bits.CN19IE = 1;

    IPC4bits.CNIP = 2;
    IFS1bits.CNIF = 0;
    IEC1bits.CNIE = 1;
}
```

CNEN1bits.CN15IE = 1; - Włączenie przerwania dla pinu CN15 (RD6)

CNEN2bits.CN19IE = 1; - Włączenie przerwania dla pinu CN19 (RD13)

IPC4bits.CNIP = 2; - Ustawienie priorytetu przerwania na poziom 2

IFS1bits.CNIF = 0; - Wyczyszczenie flagi przerwania przed włączeniem

IEC1bits.CNIE = 1; - Włączenie obsługi przerwań

Pętla ustawienia czasu

```
void petla_ustawiania_czasu() {
    unsigned int poprzedni_czas = 0;

    komunikat("Ustaw czas");
    opoznienie_ms(1500);

    while (1) {
        wybrany_czas = odczytaj_czas_z_potencjometru();

        if (wybrany_czas != poprzedni_czas) {
            pokaz_ustawienia_czasu();
            poprzedni_czas = wybrany_czas;
        }

        if (przycisk_flag) {
            __delay32(DEBOUNCE_DELAY);

            if (przycisk_g1_flag && !PRZYCISK_GRACZ1) {
                czas_g1 = czas_g2 = wybrany_czas;
                przycisk_g1_flag = false;
                przycisk_flag = false;

                while (!PRZYCISK_GRACZ1) {
                    __delay32(10000);
                }
                __delay32(DEBOUNCE_DELAY);
                return;
            }

            if (przycisk_g2_flag && !PRZYCISK_GRACZ2) {
                czas_g1 = czas_g2 = wybrany_czas;
                przycisk_g2_flag = false;
                przycisk_flag = false;

                while (!PRZYCISK_GRACZ2) {
                    __delay32(10000);
                }
                __delay32(DEBOUNCE_DELAY);
                return;
            }

            przycisk_g1_flag = false;
            przycisk_g2_flag = false;
            przycisk_flag = false;
```

void petla_ustawiania_czasu() - Główna funkcja

unsigned int poprzedni_czas = 0; - Zmienna do śledzenia zmian potencjometru

komunikat("Ustaw czas"); - Wyświetlenie komunikatu

opoznienie_ms(1500); - Pauza 1.5 sekundy na przeczytanie

while(1)

wybrany_czas = odczytaj_czas_z_potencjometru(); - Ciągły odczyt pozycji potencjometru

if (wybrany_czas != poprzedni_czas) - Sprawdzenie czy pozycja potencjometru się zmieniła

- pokaz_ustawienia_czasu(); - Aktualizacja wyświetlacza z nową wartością czasu
- poprzedni_czas = wybrany_czas; - Zapamiętanie nowej wartości

if (przycisk_flag) - Sprawdzenie czy któryś przycisk został naciśnięty

- __delay32(DEBOUNCE_DELAY); - Opóźnienie eliminujące drgania styków

if (przycisk_g1_flag && !PRZYCISK_GRACZ1) - Potwierdzenie naciśnięcia przycisku gracza 1

- czas_g1 = czas_g2 = wybrany_czas; - Ustawienie czasu dla obu graczy na wybraną wartość
- while (!PRZYCISK_GRACZ1) - Oczekiwanie na zwolnienie przycisku
- return; - Wyjście z funkcji ustawiania czasu

Dla gracza 2 następuję identyczne działanie ustawiania gry

Inicjalizacja

```
void inicjalizacja() {
    TRISDbits.TRISD6 = 1;
    TRISDbits.TRISD13 = 1;
    TRISA = 0xFFFF;

    ADC_SetConfiguration(ADC_CONFIGURATION_DEFAULT);
    ADC_ChannelEnable(ADC_CHANNEL_5);
    LCD_Initialize();
}
```

void inicjalizacja() - Funkcja konfigurująca wszystkie peryferia mikrokontrolera

TRISDbits.TRISD6 = 1; - Konfiguracja pinu RD6 jako wejście dla przycisku gracza 1

TRISDbits.TRISD13 = 1; - Konfiguracja pinu RD13 jako wejście dla przycisku gracza 2

TRISA = 0xFFFF; - Cały port A skonfigurowany jako wejścia analogowe dla ADC

ADC_SetConfiguration(ADC_CONFIGURATION_DEFAULT); - Ustawienie domyślnej konfiguracji ADC

ADC_ChannelEnable(ADC_CHANNEL_5); - Włączenie kanału 5 ADC dla potencjometru

LCD_Initialize(); - Inicjalizacja wyświetlacza LCD

Główna pętla gry

```
void petla_gry() {
    unsigned int licznik_sekund = 0;

    aktywny_gracz = 0;
    pokaz_czas();

    komunikat("Nacisnij aby\nzaczac");
    opoznienie_ms(2000);
    pokaz_czas();

    while (1) {
        if (przycisk_flag) {
            __delay32(DEBOUNCE_DELAY);

            if (przycisk_g1_flag && !PRZYCISK_GRACZ1) {
                aktywny_gracz = 2;
                pokaz_czas();
                przycisk_g1_flag = false;

                while (!PRZYCISK_GRACZ1) {
                    __delay32(10000);
                }
                __delay32(DEBOUNCE_DELAY);
            }

            if (przycisk_g2_flag && !PRZYCISK_GRACZ2) {
                aktywny_gracz = 1;
                pokaz_czas();
                przycisk_g2_flag = false;

                while (!PRZYCISK_GRACZ2) {
                    __delay32(10000);
                }
                __delay32(DEBOUNCE_DELAY);
            }

            przycisk_flag = false;
        }

        licznik_sekund++;
        if (licznik_sekund >= 100) {
            licznik_sekund = 0;

                if (aktywny_gracz == 1 && czas_g1 > 0) {
                    czas_g1--;
                    pokaz_czas();
                } else if (aktywny_gracz == 2 && czas_g2 > 0) {
                    czas_g2--;
                    pokaz_czas();
                }

                if (czas_g1 == 0 && aktywny_gracz == 1) {
                    komunikat("Gracz 1 \nPRZEGRAL");
                    break;
                }
                if (czas_g2 == 0 && aktywny_gracz == 2) {
                    komunikat("Gracz 2 \nPRZEGRAL!");
                    break;
                }
            }

            opoznienie_ms(10);
        }
    }
```

void petla_zegaru() – Główna funkcja zegaru szachowego

unsigned int licznik_sekund = 0; - Licznik do odmierzania sekund opiera sie na iteracji

aktywny_gracz = 0; - Na początku żaden gracz nie ma aktywnego zegara

komunikat("Nacisnij aby\nzaczac"); - Kominukat aby zacząć

opoznienie_ms(2000); - Pauza na przeczytanie instrukcji

while (1)

if (przycisk_flag) - Sprawdzenie czy został naciśnięty któryś przycisk

if (przycisk_g1_flag && !PRZYCISK_GRACZ1) - Naciśnięcie przycisku gracza 1

aktywny_gracz = 2; - Przełączenie zegara na gracza 2 kiedy gracz 1 skończył ruch

pokaz_czas(); - Aktualizacja wyświetlacza

licznik_sekund++; - Inkrementacja licznika dla odmierzania czasu

if (licznik_sekund >= 100) - Po 100 iteracjach pętli

licznik_sekund = 0; - Zerowanie licznika

if (aktywny_gracz == 1 && czas_g1 > 0) - Jeśli aktywny jest gracz 1 i ma jeszcze czas

czas_g1--; - Odejmowanie jednej sekundy z czasu gracza 1

pokaz_czas(); - Aktualizacja wyświetlacza

if (czas_g1 == 0 && aktywny_gracz == 1) - Sprawdzenie czy graczowi 1 skończył się czas

komunikat("Gracz 1 \nPRZEGRAL"); - Wyświetlenie komunikatu o przegranej

break; - Wyjście z pętli gry

opoznienie_ms(10); - Krótkie opóźnienie między iteracjami pętli

Dla gracza 2 działanie jest identyczne, lecz potem z gracza 2 przełącza na gracza 1

Petla main

```c
int main(void) {
    inicjalizacja();
    konfiguruj_przerwania();

    petla_ustawiania_czasu();

    komunikat("Czas ustawiony");
    opoznienie_ms(2000);

    petla_zegaru();

    while (1) {
        opoznienie_ms(1000);
    }

    return 0;
}
```

int main(void) - Punkt startowy programu

inicjalizacja(); - Skonfigurowanie wszystkich peryferiów mikrokontrolera

konfiguruj_przerwania(); - Włączenie obsługi przycisków przez przerwania

petla_ustawiania_czasu(); - Przejście do fazy ustawiania początkowego czasu gry

komunikat("Czas ustawiony"); - Potwierdzenie zakończenia ustawiania

opoznienie_ms(2000); - Pauza przed rozpoczęciem gry

petla_zegaru(); - Rozpoczęcie właściwej gry z zegarem szachowym

while (1) - Nieskończona pętla po zakończeniu gry

- opoznienie_ms(1000); - Program pozostaje aktywny, ale nic nie robi

return 0; - Koniec programu