169224_zad_5 – raport z zadania 5

Jakub Budzich – nr albumu 169224

Platforma: MPLAB X IDE + XC16
Mikrokontroler: PIC24FJ128GA010

Opis funkcjonalności programu

Program realizuje funkcjonalność zegara szachowego dla dwóch graczy z trybem SD (Sudden Death). Zegar odmierza czas każdemu graczowi osobno, a naciśnięcie przycisku jednego gracza powoduje przełączenie odmierzania czasu na drugiego gracza. Program umożliwia wybór czasu gry za pomocą potencjometru (5min, 3min, 1min), wyświetlanie pozostałego czasu dla obu graczy oraz automatyczne zakończenie gry w przypadku wyczerpania czasu przez któregoś z graczy. Informacje wyświetlane są na wyświetlaczu LCD dzięki plikom lcd.h i lcd.c. Start odliczania – gracz 1 kliknie – zaczyna się odliczanie 2 graczowi – analogicznie jak zacznie 2 gracz to czas zaczyna się liczyć 1 graczowi. Po zakończeniu stan Koniec powrót do wyboru przyciskiem RD6 lub RD13 dowolnie.

Stan	Funkcja	Opis działania
0	Wybór czasu	Ustawianie czasu gry potencjometrem - "Wybierz czas:"
1	Gracz 1	Odmierzanie czasu gracza 1, wyświetlanie obu czasów, zmiana stanu przyciskiem RD6
2	Gracz 2	Odmierzanie czasu gracza 2, wyświetlanie obu czasów, zmiana stanu przyciskiem RD13
4	Pauza	Zakończenie gry przez czas - "Wygral gracz X"

Obsługa przycisków

Obsługa przycisków odbywa się w funkcji sprawdz_przyciski(), wszystkie 4 przyciski są wykorzystywane do obsługi zegara szachowego

- **RD6 → Gracz 1** - Zakończenie ruchu gracza 1, start czasu gracza 2, lub start gry dla gracza 2
- **RD13 → Gracz 2** - Zakończenie ruchu gracza 2, start czasu gracza 1, lub start gry dla gracza 1
- **Reset po końcu gry** - Oba przyciski pozwalają na restart gry po jej zakończeniu

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

```
#include <stdio.h>       // do printf i sprintf
#include <stdlib.h>      // podstawowe funkcje
#include <xc.h>          // funkcje mikrokontrolera
#include <libpic30.h>    // do opoznien
#include "p24FJ128GA010.h"  // definicje pinow
#include "lcd.h"         // obsluga wyswietlacza z pliku
```

- **stdio.h**: biblioteka do funkcji printf i sprintf – do obsługi stringów na ekranie
- **stdlib.h**: podstawowe funkcje w c

- **xc.h**: plik z funkcjami mikrokontrolera na którym wykonywany jest program, zawiera on defincje mikrokontrolera dla kompilatora XC16
- **libpic30.h**: biblioteka do delay w której jest funkcja
- **p24FJ128GA010.h:** definicje pinów dla mikrokontrolera PIC24FJ128GA010
- **lcd.h:** plik nagłówkowy do obsługi wyświetlacza

```
// Deklaracja zegara systemowego
#define XTAL_FREQ 8000000
#define FCY 4000000

// DEKLARACJE FUNKCJI
void init_adc(void);
void czytaj_potencjometr(void);
void ustaw_urzadzenie(void);
void sprawdz_czas(void);
void pokaz_na_ekranie(void);
void resetuj_gre(void);
// Stany gry
#define STAN_WYBOR_CZASU 0      // wybieranie czasu gry
#define STAN_GRACZ1 1           // odmierza czas gracza 1
#define STAN_GRACZ2 2           // odmierza czas gracza 2
#define STAN_KONIEC 4           // koniec gry
```

- **XTAL_FREQ:** Definicja częstotliwości oscylatora - 8MHz (używana przez libpic30)
- **FCY:** Częstotliwość cyklu instrukcji - FCY = XTAL_FREQ/2 = 4MHz
- **Deklaracja funkcji**
- Stany gry opisane wcześniej w tabeli

1. Zmienne globalne

```
uint16_t czasy_opcje[] = {300, 180, 60}; // 5min, 3min, 1min
uint8_t opcje_ilosc = 3;
char* nazwy_czasow[] = {"5 min", "3 min", "1 min"};

// Zmienne globalne
volatile uint16_t czas_gracz1 = 0;      // czas pozostaly graczowi 1 (sekundy)
volatile uint16_t czas_gracz2 = 0;      // czas pozostaly graczowi 2 (sekundy)
volatile uint8_t stan_gry = STAN_WYBOR_CZASU;
volatile uint8_t aktywny_gracz = 1;     // 1 lub 2
volatile uint8_t wybrana_opcja = 1;     // domyslnie 3 min
volatile uint16_t odswiez_ekran = 1;
volatile uint16_t licznik_ms = 0;
volatile uint16_t ostatnia_sekunda = 0;
volatile uint8_t zwyciezca = 0;         // 1 lub 2 - kto wygral
```

- **czasy_opcje[]:** tablica z dostępnymi czasami gry w sekundach – 3 różne tryby 5min,3min,1min
- **opcje_ilosc:** liczba opcji trybów gry żeby potem zakres potencjometru podzielić przez tą zmienną
- **nazwy_czasow[]:** tablica z nazwami czasów do wyświetlania ich potem na ekranie
- **stan_gry:** aktualny stan gry od 0 do 5 – opisane w tabeli na początku pliku
- **aktywny_gracz:** który gracz ma aktualnie ruch i odliczany czas (gracz1 lub gracz2)
- **wybrana_opcja:** zmienna mająca wybrany czas gry do odliczania, domyślnie 3 min jakby potencjometr nie działał
- **odswiez_ekran:** flaga informująca o konieczności odswiezenia ekranu LCD
- **licznik_ms:** licznik milisekund używany w Timer1
- **ostatnia_sekunda:** znacznik czasu ostatniego odliczenia sekundy
- **zwyciezca:** zmienna w której jest zapisane kto wygrał – gracz 1 lub gracz 2, domyślnie 0 bo nikt nie wygral na początku programu

2. Funkcja przerwania `_T1Interrupt()`

```
 // Funkcja przerwania timera
void __attribute__((interrupt, auto_psv)) _T1Interrupt(void)
{
    IFS0bits.T1IF = 0;
    licznik_ms++;

    // Odczyt potencjometru co 100ms
    if (licznik_ms % 100 == 0 && stan_gry == STAN_WYBOR_CZASU) {
        czytaj_potencjometr();
    }
}
```

- Definicja procedury przerwania Timer1
- **auto_psv** - automatyczne zapisywanie rejestrów
- **IFS0bits.T1IF = 0;** - wyzerowanie flagi przerwania Timer1 (obowiązkowe w każdym przerwaniu)
- licznik_ms++ - inkrementacja licznika milisekund (wywoływane co 1ms)
- Co 100 ms podczas wyboru czasu odczytywany jest potencjometr

3. Przerwanie Change Notification `_CNInterrupt()`

```
void __attribute__((interrupt, no_auto_psv)) _CNInterrupt(void) {
    __delay32(FCY/100);  // debouncing 10ms

    // Przycisk gracza 1 (RD6)
    if(PORTDbits.RD6 == 0) {
```

```
        if (stan_gry == STAN_GRACZ1) {
            // Gracz 1 skonczyl ruch - teraz kolej gracza 2
            stan_gry = STAN_GRACZ2;
            aktywny_gracz = 2;
            ostatnia_sekunda = licznik_ms;
            odswiez_ekran = 1;
        }
        // START GRY: Gracz 1 startuje czas graczowi 2
        else if (stan_gry == STAN_WYBOR_CZASU) {
            czas_gracz1 = czasy_opcje[wybrana_opcja];
            czas_gracz2 = czasy_opcje[wybrana_opcja];
            stan_gry = STAN_GRACZ2;
            aktywny_gracz = 2;
            ostatnia_sekunda = licznik_ms;
            odswiez_ekran = 1;
        }
        // RESTART PO KONCU GRY
        else if (stan_gry == STAN_KONIEC) {
            resetuj_gre();
        }
    }

    // Przycisk gracza 2 (RD13)
    else if(PORTDbits.RD13 == 0) {
        if (stan_gry == STAN_GRACZ2) {
            // Gracz 2 skonczyl ruch - teraz kolej gracza 1
            stan_gry = STAN_GRACZ1;
            aktywny_gracz = 1;
            ostatnia_sekunda = licznik_ms;
            odswiez_ekran = 1;
        }
        // START GRY: Gracz 2 startuje czas graczowi 1
        else if (stan_gry == STAN_WYBOR_CZASU) {
            czas_gracz1 = czasy_opcje[wybrana_opcja];
            czas_gracz2 = czasy_opcje[wybrana_opcja];
            stan_gry = STAN_GRACZ1;
            aktywny_gracz = 1;
            ostatnia_sekunda = licznik_ms;
            odswiez_ekran = 1;
        }
        // RESTART PO KONCU GRY
        else if (stan_gry == STAN_KONIEC) {
            resetuj_gre();
        }
    }

    // Czekaj na zwolnienie przyciskow
    while(PORTDbits.RD6 == 0 || PORTDbits.RD13 == 0);
```

```
    // Wyczysc flage przerwania
    IFS1bits.CNIF = 0;
}
```

- **Debouncing**: 10ms opóźnienie dla eliminacji drgań styków
- **RD6 (Gracz 1)**: przełączenie z gracza 1 na gracza 2, start gry dla gracza 2, lub reset
- **RD13 (Gracz 2)**: przełączenie z gracza 2 na gracza 1, start gry dla gracza 1, lub reset
- **Blokowanie**: oczekiwanie na zwolnienie przycisków przed zakończeniem przerwania

4. Funkcja `init_adc()`

```
void init_adc(void)
{
    AD1PCFGbits.PCFG5 = 0;     // AN5 jako wejscie analogowe
    AD1CON1 = 0x00E0;
    AD1CON2 = 0;
    AD1CON3 = 0x1F3F;
    AD1CHS = 5;                // Kanal AN5
    AD1CON1bits.ADON = 1;      // Wlacz ADC
}
```

- Konfiguracja przetwornika ADC dla potencjometru
- **AN5** ustawiony jako wejście analogowe (pin **RB5**)
- Konfiguracja rejestrów ADC dla poprawnej pracy potencjometru

5. Funkcja `czytaj_potencjometr()`

```
void czytaj_potencjometr(void)
{
    AD1CON1bits.SAMP = 1;
    __delay32(100);
    AD1CON1bits.SAMP = 0;
    while (!AD1CON1bits.DONE);
    wartosc_potencjometru = ADC1BUF0;

    // Przelicz na opcje czasu (3 opcje)
    uint8_t nowa_opcja = (wartosc_potencjometru * opcje_ilosc) / 1024;
    if (nowa_opcja >= opcje_ilosc) nowa_opcja = opcje_ilosc - 1;

    if (nowa_opcja != wybrana_opcja) {
        wybrana_opcja = nowa_opcja;
```

```
        odswiez_ekran = 1;
    }
}
```

- Rozpoczęcie próbkowania ADC
- Oczekiwanie na zakończenie konwersji
- Przeliczanie wartości 0-1023 na 3 opcje czasu
- Aktualizacja wybranej opcji i odświeżanie ekranu LCD przy zmianie

6. Funkcja `main()`

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

    while (1) {
        sprawdz_czas();

        if (odswiez_ekran) {
            pokaz_na_ekranie();
            odswiez_ekran = 0;
        }

        __delay32(1000);
    }

    return 0;
}
```

- Inicjalizacja urządzenia
- W pętli która jest nieskończona
    - Sprawdzanie czy minęła sekunda i odliczanie czasu
    - Odświeżanie wyświetlacza przy zmianie stanu
    - Krótki delay dla stabilnego działania

7. Funkcja `ustaw_urzadzenie()`

```
void ustaw_urzadzenie(void)
{
    // Konfiguracja pinów
    TRISDbits.TRISD6 = 1;       // RD6 jako wejscie (gracz 1)
    TRISDbits.TRISD7 = 1;       // RD7 jako wejscie (reset)
    TRISAbits.TRISA7 = 1;       // RA7 jako wejscie (start)
    TRISDbits.TRISD13 = 1;      // RD13 jako wejscie (gracz 2)
```

```
    // Konfiguracja ADC
    AD1PCFG = 0xFFDF;           // Wszystkie cyfrowe oprócz AN5
    TRISBbits.TRISB5 = 1;       // RB5/AN5 jako wejscie analogowe

    // Inicjalizacja LCD
    LCD_Initialize();
    LCD_ClearScreen();

    // Konfiguracja timera na 1ms
    T1CON = 0;
    TMR1 = 0;
    PR1 = FCY/1000;             // 1ms
    T1CONbits.TCKPS = 0b00;     // Bez dzielnika
    IPC0bits.T1IP = 3;          // Priorytet przerwania
    IFS0bits.T1IF = 0;
    IEC0bits.T1IE = 1;
    T1CONbits.TON = 1;

    // Inicjalizacja ADC
    init_adc();

    // Wlacz przerwania
    INTCON1bits.NSTDIS = 0;

    // Ustaw domyslne czasy
    czas_gracz1 = czasy_opcje[wybrana_opcja];
    czas_gracz2 = czasy_opcje[wybrana_opcja];
}
```

- Konfiguracja **ADC** - wszystkie piny cyfrowe oprócz **AN5**
- Ustawienie **RB5** jako wejście analogowe dla potencjometru
- Konfiguracja Change Notification:
    - Pull-up resistory dla przycisków
    - Włączenie przerwań CN dla RD6 i RD13
- Konfiguracja Timer1 na przerwanie co 1ms
- **PR1 = FCY/1000** - okres 1ms przy częstotliwości 4MHz
- Włączenie przerwania i timera

8. Funkcje `sprawdz_czas()`

```
void sprawdz_czas(void)

{
    if ((stan_gry == STAN_GRACZ1 || stan_gry == STAN_GRACZ2) &&
        licznik_ms - ostatnia_sekunda >= 1000) {

        ostatnia_sekunda = licznik_ms;
```

```
        if (stan_gry == STAN_GRACZ1 && czas_gracz1 > 0) {
            czas_gracz1--;
            if (czas_gracz1 == 0) {
                // Gracz 1 przegral przez czas
                stan_gry = STAN_KONIEC;
                zwyciezca = 2;
            }
        } else if (stan_gry == STAN_GRACZ2 && czas_gracz2 > 0) {
            czas_gracz2--;
            if (czas_gracz2 == 0) {
                // Gracz 2 przegral przez czas
                stan_gry = STAN_KONIEC;
                zwyciezca = 1;
            }
        }

        odswiez_ekran = 1;
    }
}
```

- Sprawdzanie czy gra jest aktywna i czy minęła sekunda
- Odliczanie czasu dla aktywnego gracza
- Automatyczne zakończenie gry przy wyczerpaniu czasu
- Wyznaczenie zwycięzcy (drugi gracz wygrywa gdy pierwszemu skończy się czas)
- Używana jest różnica **licznik_ms - ostatnia_sekunda** dla pomiaru czasu

9. Funkcja `pokaz_na_ekranie()`

```
void pokaz_na_ekranie(void)

{
    char linia1[17], linia2[17];
    uint16_t min1, sek1, min2, sek2;

    LCD_ClearScreen();

    switch (stan_gry) {
        case STAN_WYBOR_CZASU:
            sprintf(linia1, "Wybierz czas:");
            sprintf(linia2, "-> %s <-", nazwy_czasow[wybrana_opcja]);
            break;

        case STAN_GRACZ1:
        case STAN_GRACZ2:
            min1 = czas_gracz1 / 60;
            sek1 = czas_gracz1 % 60;
            min2 = czas_gracz2 / 60;
            sek2 = czas_gracz2 % 60;
```

```
            // Wyswietl gracza 1 z oznaczeniem aktywnosci
            if (stan_gry == STAN_GRACZ1) {
                sprintf(linia1, "*Gracz1 %02d:%02d", min1, sek1);
            } else {
                sprintf(linia1, " Gracz1 %02d:%02d", min1, sek1);
            }

            // Wyswietl gracza 2 z oznaczeniem aktywnosci
            if (stan_gry == STAN_GRACZ2) {
                sprintf(linia2, "*Gracz2 %02d:%02d", min2, sek2);
            } else {
                sprintf(linia2, " Gracz2 %02d:%02d", min2, sek2);
            }
            break;

        case STAN_KONIEC:
            sprintf(linia1, "KONIEC GRY!");
            sprintf(linia2, "Wygral gracz %d", zwyciezca);
            break;
    }

    LCD_PutString(linia1, strlen(linia1));
    LCD_PutChar('\n');
    LCD_PutString(linia2, strlen(linia2));
}
```

- Funkcja odpowiedzialna za wyświetlanie aktualnego stanu na LCD
- **Stan wyboru czasu**: "Wybierz czas:" i aktualnie wybrana opcja
- **Stany gry**: czas obu graczy w formacie MM:SS z oznaczeniem aktywnego gracza (*)
- **Stan końca**: komunikat o zakończeniu i zwycięzcy
- Konwersja sekund na minuty i sekundy dla czytelnego wyświetlania

10. Funkcja `resetuj_gre()`

```
 void resetuj_gre(void)

{
    stan_gry = STAN_WYBOR_CZASU;
    aktywny_gracz = 1;
    zwyciezca = 0;
    czas_gracz1 = czasy_opcje[wybrana_opcja];
    czas_gracz2 = czasy_opcje[wybrana_opcja];
    odswiez_ekran = 1;
}
```

- przywraca wszystkie zmienne do stanu początkowego
- Ustawia stan na wybór czasu
- Zeruje zwycięzcę i ustawia gracza 1 jako aktywnego
- Przywraca czasy zgodnie z wybraną opcją
- Wymusza odświeżenie ekranu

11. Cały kod programu:

```
/*
 * File:   main.c
 * Author: Jakub Budzich - ISI1
 *
 * Created on May 26, 2025, 8:30
 */

#pragma config POSCMOD = NONE
#pragma config OSCIOFNC = ON
#pragma config FCKSM = CSDCMD
#pragma config FNOSC = FRC
#pragma config IESO = OFF
#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

#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#include <libpic30.h>
#include <string.h>
#include "lcd.h"

// Deklaracja zegara systemowego
#define XTAL_FREQ 8000000
#define FCY 4000000

// DEKLARACJE FUNKCJI
void init_adc(void);
void czytaj_potencjometr(void);
void ustaw_urzadzenie(void);
void sprawdz_czas(void);
void pokaz_na_ekranie(void);
void resetuj_gre(void);
```

```
// Stany gry
#define STAN_WYBOR_CZASU 0      // wybieranie czasu gry
#define STAN_GRACZ1 1           // odmierza czas gracza 1
#define STAN_GRACZ2 2           // odmierza czas gracza 2
#define STAN_KONIEC 4           // koniec gry

// Czasy gry (w sekundach)
uint16_t czasy_opcje[] = {300, 180, 60}; // 5min, 3min, 1min
uint8_t opcje_ilosc = 3;
char* nazwy_czasow[] = {"5 min", "3 min", "1 min"};

// Zmienne globalne
volatile uint16_t czas_gracz1 = 0;      // czas pozostaly graczowi 1 (sekundy)
volatile uint16_t czas_gracz2 = 0;      // czas pozostaly graczowi 2 (sekundy)
volatile uint8_t stan_gry = STAN_WYBOR_CZASU;
volatile uint8_t aktywny_gracz = 1;     // 1 lub 2
volatile uint8_t wybrana_opcja = 1;     // domyslnie 3 min
volatile uint16_t odswiez_ekran = 1;
volatile uint16_t licznik_ms = 0;
volatile uint16_t ostatnia_sekunda = 0;
volatile uint8_t zwyciezca = 0;         // 1 lub 2 - kto wygral

// ADC dla potencjometru
volatile uint16_t wartosc_potencjometru = 0;

// Funkcja przerwania timera
void __attribute__((interrupt, auto_psv)) _T1Interrupt(void)
{
    IFS0bits.T1IF = 0;
    licznik_ms++;

    // Odczyt potencjometru co 100ms
    if (licznik_ms % 100 == 0 && stan_gry == STAN_WYBOR_CZASU) {
        czytaj_potencjometr();
    }
}

// Przerwanie Change Notification - obsluga przyciskow
void __attribute__((interrupt, no_auto_psv)) _CNInterrupt(void) {
    __delay32(FCY/100);  // debouncing 10ms

    // Przycisk gracza 1 (RD6)
    if(PORTDbits.RD6 == 0) {
        if (stan_gry == STAN_GRACZ1) {
            // Gracz 1 skonczyl ruch - teraz kolej gracza 2
            stan_gry = STAN_GRACZ2;
            aktywny_gracz = 2;
```

```
            ostatnia_sekunda = licznik_ms;
            odswiez_ekran = 1;
        }
        // START GRY: Gracz 1 startuje czas graczowi 2
        else if (stan_gry == STAN_WYBOR_CZASU) {
            czas_gracz1 = czasy_opcje[wybrana_opcja];
            czas_gracz2 = czasy_opcje[wybrana_opcja];
            stan_gry = STAN_GRACZ2;
            aktywny_gracz = 2;
            ostatnia_sekunda = licznik_ms;
            odswiez_ekran = 1;
        }
        // RESTART PO KONCU GRY
        else if (stan_gry == STAN_KONIEC) {
            resetuj_gre();
        }
    }

    // Przycisk gracza 2 (RD13)
    else if(PORTDbits.RD13 == 0) {
        if (stan_gry == STAN_GRACZ2) {
            // Gracz 2 skonczyl ruch - teraz kolej gracza 1
            stan_gry = STAN_GRACZ1;
            aktywny_gracz = 1;
            ostatnia_sekunda = licznik_ms;
            odswiez_ekran = 1;
        }
        // START GRY: Gracz 2 startuje czas graczowi 1
        else if (stan_gry == STAN_WYBOR_CZASU) {
            czas_gracz1 = czasy_opcje[wybrana_opcja];
            czas_gracz2 = czasy_opcje[wybrana_opcja];
            stan_gry = STAN_GRACZ1;
            aktywny_gracz = 1;
            ostatnia_sekunda = licznik_ms;
            odswiez_ekran = 1;
        }
        // RESTART PO KONCU GRY
        else if (stan_gry == STAN_KONIEC) {
            resetuj_gre();
        }
    }

    // Czekaj na zwolnienie przyciskow
    while(PORTDbits.RD6 == 0 || PORTDbits.RD13 == 0);

    // Wyczysc flage przerwania
    IFS1bits.CNIF = 0;
}
```

```c
// Inicjalizacja ADC
void init_adc(void)
{
    AD1PCFGbits.PCFG5 = 0;    // AN5 jako wejscie analogowe
    AD1CON1 = 0x00E0;
    AD1CON2 = 0;
    AD1CON3 = 0x1F3F;
    AD1CHS = 5;               // Kanal AN5
    AD1CON1bits.ADON = 1;     // Wlacz ADC
}

// Odczyt potencjometru
void czytaj_potencjometr(void)
{
    AD1CON1bits.SAMP = 1;
    __delay32(100);
    AD1CON1bits.SAMP = 0;
    while (!AD1CON1bits.DONE);
    wartosc_potencjometru = ADC1BUF0;

    // Przelicz na opcje czasu (3 opcje)
    uint8_t nowa_opcja = (wartosc_potencjometru * opcje_ilosc) / 1024;
    if (nowa_opcja >= opcje_ilosc) nowa_opcja = opcje_ilosc - 1;

    if (nowa_opcja != wybrana_opcja) {
        wybrana_opcja = nowa_opcja;
        odswiez_ekran = 1;
    }
}

int main(void)
{
    ustaw_urzadzenie();

    while (1) {
        sprawdz_czas();

        if (odswiez_ekran) {
            pokaz_na_ekranie();
            odswiez_ekran = 0;
        }

        __delay32(1000);
    }

    return 0;
}
```

```
// Inicjalizacja urzadzenia
void ustaw_urzadzenie(void)
{
    // Konfiguracja ADC - wszystkie cyfrowe oprocz AN5
    AD1PCFG = 0xFFDF;
    TRISBbits.TRISB5 = 1;       // RB5/AN5 jako wejscie analogowe

    // Konfiguracja pinow jako wejscia
    TRISA = 0x0000;             // Port A jako wyjscie (dla LCD)
    TRISD = 0xFFFF;             // Port D jako wejscie (przyciski)

    // Konfiguracja Change Notification interrupts
    CNPU1bits.CN15PUE = 1;      // Pull-up dla RD6 (gracz 1)
    CNPU2bits.CN19PUE = 1;      // Pull-up dla RD13 (gracz 2)

    // Wlaczenie przerwan Change Notification
    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 LCD
    LCD_Initialize();
    LCD_ClearScreen();

    // Konfiguracja timera na 1ms
    T1CON = 0;
    TMR1 = 0;
    PR1 = FCY/1000 - 1;         // 1ms
    T1CONbits.TCKPS = 0b00;
    IPC0bits.T1IP = 3;          // Priorytet przerwania
    IFS0bits.T1IF = 0;
    IEC0bits.T1IE = 1;
    T1CONbits.TON = 1;

    // Inicjalizacja ADC
    init_adc();

    // Wlacz przerwania globalne
    INTCON1bits.NSTDIS = 0;

    // domyslne czasy
    czas_gracz1 = czasy_opcje[wybrana_opcja];
    czas_gracz2 = czasy_opcje[wybrana_opcja];
}
```

```c
// Sprawdzanie czasu
void sprawdz_czas(void)
{
    if ((stan_gry == STAN_GRACZ1 || stan_gry == STAN_GRACZ2) &&
        licznik_ms - ostatnia_sekunda >= 1000) {

        ostatnia_sekunda = licznik_ms;

        if (stan_gry == STAN_GRACZ1 && czas_gracz1 > 0) {
            czas_gracz1--;
            if (czas_gracz1 == 0) {
                // Gracz 1 przegral przez czas
                stan_gry = STAN_KONIEC;
                zwyciezca = 2;
            }
        } else if (stan_gry == STAN_GRACZ2 && czas_gracz2 > 0) {
            czas_gracz2--;
            if (czas_gracz2 == 0) {
                // Gracz 2 przegral przez czas
                stan_gry = STAN_KONIEC;
                zwyciezca = 1;
            }
        }

        odswiez_ekran = 1;
    }
}

// Wyswietlanie na ekranie
void pokaz_na_ekranie(void)
{
    char linia1[17], linia2[17];
    uint16_t min1, sek1, min2, sek2;

    LCD_ClearScreen();

    switch (stan_gry) {
        case STAN_WYBOR_CZASU:
            sprintf(linia1, "Wybierz czas:");
            sprintf(linia2, "-> %s <-", nazwy_czasow[wybrana_opcja]);
            break;

        case STAN_GRACZ1:
        case STAN_GRACZ2:
            min1 = czas_gracz1 / 60;
            sek1 = czas_gracz1 % 60;
            min2 = czas_gracz2 / 60;
            sek2 = czas_gracz2 % 60;
```

```
            // Wyswietl gracza 1 z oznaczeniem aktywnosci
            if (stan_gry == STAN_GRACZ1) {
                sprintf(linia1, "*Gracz1 %02d:%02d", min1, sek1);
            } else {
                sprintf(linia1, " Gracz1 %02d:%02d", min1, sek1);
            }

            // Wyswietl gracza 2 z oznaczeniem aktywnosci
            if (stan_gry == STAN_GRACZ2) {
                sprintf(linia2, "*Gracz2 %02d:%02d", min2, sek2);
            } else {
                sprintf(linia2, " Gracz2 %02d:%02d", min2, sek2);
            }
            break;

        case STAN_KONIEC:
            sprintf(linia1, "KONIEC GRY!");
            sprintf(linia2, "Wygral gracz %d", zwyciezca);
            break;
    }

    LCD_PutString(linia1, strlen(linia1));
    LCD_PutChar('\n');
    LCD_PutString(linia2, strlen(linia2));
}

// Reset gry
void resetuj_gre(void)
{
    stan_gry = STAN_WYBOR_CZASU;
    aktywny_gracz = 1;
    zwyciezca = 0;
    czas_gracz1 = czasy_opcje[wybrana_opcja];
    czas_gracz2 = czasy_opcje[wybrana_opcja];
    odswiez_ekran = 1;
}
```