Raport do zadania 1

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 <xc.h> - Główna biblioteka dla kompilatorów XC

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

#include <stdbool.h> - Biblioteka standardowa C dla typu bool (true/false

Definicje i konfiguracja

```
#define FCY 4000000UL
#define delay 500000
#define DEBOUNCE_DELAY 200000
#define buttonnext PORTDbits.RD6
#define buttonprev PORTDbits.RD13
```

#define FCY 4000000UL - Częstotliwość procesora 4 MHz

#define delay 500000 - Podstawowe opóźnienie

#define DEBOUNCE_DELAY 200000 - Opóźnienie dla eliminacji drgań styków

#define buttonnext PORTDbits.RD6 - Przycisk następny

#define buttonprev PORTDbits.RD13 - Przycisk poprzedni

Konfiguracja wejść/wyjść

```
    AD1PCFG = 0xFFFF;
    TRISA = 0x0000;
    TRISD = 0xFFFF;
```

AD1PCFG = 0xFFFF - Wszystkie piny analogowe ustawione jako cyfrowe

TRISA = 0x0000; - Port A jako wyjście (sterowanie LED)

TRISD = 0xFFFF; - Port D jako wejście (odczyt przycisków)

funk1() - 8-bitowy licznik binarny w górę

```
void funk1() {
    unsigned char licznik = 0;

    while (1) {
        LATA = licznik;
        __delay32(delay);

        if (!buttonnext || !buttonprev) {
            return;
        }
        licznik = licznik + 1;
    }
}
```

- Zlicza od 0 do 255 w systemie binarnym
- Zwiększa wartość licznika o 1 w każdej iteracji
- Wyświetla wynik na porcie LATA
- Stałe opóźnienie 500000 cykli
- Kończy działanie po wciśnięciu dowolnego przycisku

funk2() - 8-bitowy licznik binarny w dół

```
void funk2() {
    unsigned char licznik = 255;

    while (1) {
        LATA = licznik;
        __delay32(delay);

        if (!buttonnext || !buttonprev) {
            return;
        }
        licznik = licznik - 1;

    }
}
```

- Zlicza od 255 do 0 w systemie binarnym
- Zmniejsza wartość licznika o 1 w każdej iteracji
- Wyświetla wynik na porcie LATA
- Stałe opóźnienie 500000 cykli
- Kończy działanie po wciśnięciu dowolnego przycisku

funk3() - 8-bitowy licznik Gray w górę

```
void funk3() {
    unsigned char licznik_normalny = 0;
    unsigned char wynik_gray;

    while (1) {
        wynik_gray = licznik_normalny ^ (licznik_normalny >> 1);

        LATA = wynik_gray;
        __delay32(delay);

        if (!buttonnext || !buttonprev) {
            return;
        }
        licznik_normalny = licznik_normalny + 1;
    }
}
```

- Zlicza od 0 do 255 w kodzie Gray
- Konwersja do kodu Gray: wynik_gray = licznik_normalny ^ (licznik_normalny >> 1)
- W kodzie Gray tylko jeden bit zmienia się między kolejnymi liczbami
- Stałe opóźnienie 500000 cykli
- Kończy działanie po wciśnięciu dowolnego przycisku

funk4() - 8-bitowy licznik Gray w dół

```
void funk4() {
    unsigned char i;
    unsigned char kod_gray;
    i = 255;
    while (1) {
        kod_gray = i ^ (i >> 1);
        LATA = kod_gray;
        __delay32(delay);

        if (!buttonnext || !buttonprev) {
            return;
        }
        i = i - 1;
    }
}
```

- Zlicza od 255 do 0 w kodzie Gray
- Używa tej samej formuły konwersji co funk3()
- Zmniejsza licznik dziesiętny, następnie konwertuje do Gray
- Stałe opóźnienie 500000 cykli
- Kończy działanie po wciśnięciu dowolnego przycisku

funk5() - 2×4-bitowy licznik BCD w górę

```
void funk5() {
    unsigned char dziesiatki;
    unsigned char jednosci;
    unsigned char wynik_bcd;
    unsigned char licznik_dziesiatkowy = 0;

    while (1) {
        dziesiatki = licznik_dziesiatkowy / 10;
        jednosci = licznik_dziesiatkowy % 10;
        wynik_bcd = (dziesiatki << 4) | jednosci;
        LATA = wynik_bcd;
        __delay32(delay);

        if (!buttonnext || !buttonprev) {
            return;
        }
        licznik_dziesiatkowy++;

        if (licznik_dziesiatkowy > 99) {
            licznik_dziesiatkowy = 0;
        }
    }
}
```

- Zlicza od 0 do 99 w kodzie BCD (Binary Coded Decimal)
- Dzieli liczbę na dziesiątki i jedności
- Kodowanie BCD: wynik_bcd = (dziesiatki << 4) | jednosci
- Górne 4 bity reprezentują dziesiątki, dolne 4 bity jedności
- Po osiągnięciu 99 resetuje się do 0
- Stałe opóźnienie 500000 cykli

funk6() - 2×4-bitowy licznik BCD w dół

```
void funk6() {
    unsigned char dziesiatki;
    unsigned char jednosci;
    unsigned char bcd;
    unsigned char licznik = 99;

    while (1) {
        dziesiatki = licznik / 10;
        jednosci = licznik % 10;
        bcd = (dziesiatki * 16) + jednosci;
        LATA = bcd;
        __delay32(delay);

        if (!buttonnext || !buttonprev) {
            return;
        }
        licznik--;
        if (licznik == 255) {
            licznik = 99;
        }
    }
}
```

- Zlicza od 99 do 0 w kodzie BCD
- Kodowanie: bcd = (dziesiatki * 16) + jednosci równoważne z przesunięciem bitowym
- Po osiągnięciu 0 (które staje się 255 po dekrementacji unsigned char) resetuje się do 99
- Stałe opóźnienie 500000 cykli

funk7() - 3-bitowy "wężyk" lewo-prawo

```
void funk7() {
    unsigned char wezyk;
    unsigned char kierunek_w_prawo = 1;

    wezyk = 0b00000111;

    while (1) {
        LATA = wezyk;
        __delay32(delay);

        if (!buttonnext || !buttonprev) {
            return;
        }

        if (kierunek_w_prawo == 1) {
            wezyk = wezyk << 1;

            if (wezyk == 0b11100000) {
                kierunek_w_prawo = 0;
            }
        } else {
            wezyk = wezyk >> 1;

            if (wezyk == 0b00000111) {
                kierunek_w_prawo = 1;
            }
        }
    }
}
```

- Rozpoczyna z wzorem 0b00000111 (3 zapalene LED po prawej)
- Przesuwa wzór w lewo aż do 0b11100000
- Następnie zmienia kierunek i przesuwa w prawo
- Oscyluje między skrajnymi pozycjami

funk8() - Efekt kolejki

```
void funk8() {
    unsigned char stan_diod = 0;
    unsigned char pozycja;
    unsigned char maska;

    for (int numer_diody = 0; numer_diody < 8; numer_diody++) {
        maska = 1;

        for (int i = numer_diody + 1; i < 8; i++) {
            LATA = stan_diod | maska;
            maska = maska << 1;
            __delay32(delay);
            if (!buttonnext || !buttonprev) {
                return;
            }
        }

        stan_diod = stan_diod | maska;
        LATA = stan_diod;
        __delay32(delay);
        if (!buttonnext || !buttonprev) {
            return;
        }
    }
}
```

- Efekt kolejki - stopniowe zapalanie LED od prawej do lewej
- Zewnętrzna pętla iteruje przez pozycje LED (0-7)
- Wewnętrzna pętla zapala kolejne LED używając przesuwającej się maski
- Po każdej iteracji zewnętrznej pętli LED pozostaje zapalony
- Maska maska = 1 przesuwa się w lewo w każdej iteracji wewnętrznej pętli
- W końcowym efekcie światełka zostają zapalone
- Stałe opóźnienie 500000 cykli

funk9() - 6-bitowy generator pseudolosowy (LFSR)

```
void funk9() {
    unsigned char lfsr = 0b000001;
    while (1) {
        unsigned char wartosc = ((lfsr >> 5) & 1) ^
                                ((lfsr >> 4) & 1) ^
                                ((lfsr >> 3) & 1) ^
                                ((lfsr >> 1) & 1) ^
                                ((lfsr >> 0) & 1);


        lfsr = (lfsr >> 1) | (wartosc << 5);

        LATA = lfsr & 0b00111111;

        __delay32(delay);

        if (!buttonnext || !buttonprev)
            return;
    }
}
```

- Implementuje Linear Feedback Shift Register
- Początkowa wartość: 0b000001
- Sprzężenie zwrotne z bitów: 5, 4, 3, 1, 0 (XOR)
- ((lfsr >> 5) & 1) ^ ((lfsr >> 4) & 1) ^ ((lfsr >> 3) & 1) ^ ((lfsr >> 1) & 1) ^ ((lfsr >> 0) & 1)
- Nowy bit wstawiony na pozycję 5, reszta przesunięta w prawo
- Wyświetla tylko dolne 6 bitów: LATA = lfsr & 0b00111111
- Generuje sekwencję pseudolosową o okresie 63 (2^6 - 1)

Pętla while w main()

```
while (1) {
    switch (tryb) {
        case 1:
            funk1();
            break;
        case 2:
            funk2();
            break;
        case 3:
            funk3();
            break;
        case 4:
            funk4();
            break;
        case 5:
            funk5();
            break;
        case 6:
            funk6();
            break;
        case 7:
            funk7();
            break;
        case 8:
            funk8();
            break;
        case 9:
            funk9();
            break;
    }
```

Program działa w nieskończonej pętli, wykonując wybraną funkcję do momentu wciśnięcia przycisku, następnie sprawdza, który przycisk został wciśnięty i odpowiednio zmienia tryb.

Debouncing

```
        if (!buttonnext) {
            __delay32(DEBOUNCE_DELAY);
            if (!buttonnext) {
                tryb++;
                if (tryb > 9)
                    tryb = 1;
                while (!buttonnext);
                __delay32(DEBOUNCE_DELAY);
            }
        }

        if (!buttonprev) {
            __delay32(DEBOUNCE_DELAY);
            if (!buttonprev) {
                if (tryb == 1)
                    tryb = 9;
                else
                    tryb--;
                while (!buttonprev);
                __delay32(DEBOUNCE_DELAY);
            }
        }
    }
}
```

Przycisk "next" (RD6):

- Zwiększa tryb o 1
- Po trybie 9 wraca do trybu 1
- Implementuje debouncing z opóźnieniem 200000 cykli

Przycisk "prev" (RD13):

- Zmniejsza tryb o 1
- Z trybu 1 przechodzi do trybu 9
- Implementuje debouncing z opóźnieniem 200000 cykli