

169224_zad_1 – raport z zadania 1

Jakub Budzich – nr albumu 169224

Platforma: MPLAB X IDE + XC16

Mikrokontroler: PIC24FJ128GA010

Opis funkcjonalności programów

Program oferuje **9 funkcji**, które użytkownik przełącza przy pomocy dwóch przycisków:

Nr	Funkcja	Opis działania
1	binUP()	8-bitowy licznik binarny zliczający w górę od 0 do 255
2	binDOWN()	8-bitowy licznik binarny zliczający w dół od 255 do 0
3	grayUP()	Zliczanie w kodzie Graya (0 do 255)
4	grayDOWN()	Zliczanie w kodzie Graya (255 do 0)
5	bcdUP()	2x4-bitowy licznik BCD (dziesiątki i jedności) od 0 do 99
6	bcdDOWN()	Licznik BCD od 99 do 0
7	snake()	"Wąż" 3-bitowy poruszający się w lewo i prawo
8	kolejka()	Efekt kolejki zapalających się kolejno diod
9	losowe()	6-bitowy generator pseudolosowy oparty o LCG z maską 0x3F

Przetwarzanie funkcji

Obsługa odbywa się przez przerwanie CNInterrupt():

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

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 - przetwarzanie zegara i monitor wyłączone
- FNOSC = FRC - wybór wewnętrznego oscylatora RC
- IESO = OFF - tryb przetwarzania 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;
```

numer_programu: wybór aktywnej funkcji

flaga: zatrzymanie bieżącego działania (np. po naciśnięciu przycisku)

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

```
void delay(uint32_t ilosc) {
    uint32_t i, j;

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

Instrukcja asm("NOP") nic nie robi w kodzie ale zajmuje czas procesora przez co możemy uzyskać opóźnienie.

Zmienna ilosc jest wprowadzana przez użytkownika i jest to informacja ile razy wykonać ma petle for z zmienną i

3. Funkcja init()

```
void init() {
```

```

AD1PCFG = 0xFFFF;
TRISA = 0x0000; // Port A jako wyjście
TRISD = 0xFFFF; // Port D jako wejście

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

// Włączenie przerwan Change Notification dla przycisków
CNEN1bits.CN15IE = 1; // Włącz przerwanie dla RD6
CNEN2bits.CN19IE = 1; // Włącz przerwanie dla RD13

IFS1bits.CNIF = 0; // Wyczyść flagę przerwania CN
IEC1bits.CNIE = 1; // Włącz przerwania CN
}

```

- AD1PCFG = 0xFFFF – ustawienie wszystkich pinów jako cyfrowe
- CNPUx – włączenie pull-upów dla przycisków
- CNENx – aktywacja przerwań Change Notification
- IEC1bits.CNIE = 1 – włączenie przerwań CN
- IFS1bits.CNIF = 0 – wyczyszczenie flagi przerwania

4. Funkcja przerwania _CNInterrupt()

```

void __attribute__((interrupt, no_auto_psv)) _CNInterrupt(void) {
    __delay32(200);
    // Sprawdzenie, który przycisk został naciśnięty
    // poprzedni program
    if(PORTDbits.RD13 == 0) {
        numer_programu--;
        flaga = 1;
    }
    // następny program
    else if(PORTDbits.RD6 == 0) {
        numer_programu++;
        flaga = 1;
    }

    while(PORTDbits.RD13 == 0 || PORTDbits.RD6 == 0);
    // Wyczyść flagę
    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
- Czyści flagę przerwania

5. BinUp()

```
void binUP(){
    unsigned char licznik = 0;
    flaga = 0;
    while(!flaga) {
        LATA = licznik++;
        delay(750);
    }
}
```

- Inicjalizuje zmienną licznik wartością 0
- Resetuje flagę zmiany programu
- Wchodzi w pętlę, która będzie działać dopóki flaga pozostaje równa 0:
- Wyświetla aktualną wartość licznika na porcie A (diody LED)
- Inkrementuje licznik (z wykorzystaniem post-inkrementacji licznik++)
- Wprowadza opóźnienie 750 jednostek
- Licznik automatycznie wykonuje overflow z 255 na 0 ze względu na typ unsigned char (8 bitów)
- Pętla kończy się po naciśnięciu przycisku (gdy flaga zostanie ustawiona na 1)

6. BinDown()

```
void binDOWN(){
    unsigned char licznik = 255;
    flaga = 0;
    while(!flaga) {
        LATA = licznik--;
        delay(750);
    }
}
```

- Inicjalizuje zmienną licznik wartością 255 (maksymalna wartość 8-bitowa)
- Resetuje flagę zmiany programu
- Wchodzi w pętlę, która będzie działać dopóki flaga pozostaje równa 0:
- Wyświetla aktualną wartość licznika na porcie A (diody LED)
- Dekrementuje licznik (z wykorzystaniem post-dekrementacji licznik--)
- Wprowadza opóźnienie 750 jednostek

- Licznik automatycznie wykonuje underflow z 0 na 255 ze względu na typ `unsigned char` (8 bitów)
- Pętla kończy się po naciśnięciu przycisku (gdy flaga zostanie ustawiona na 1)

7. GrayUP()

```
void grayUP(){
    unsigned char licznik = 0;
    unsigned char grayCode;
    flaga = 0;

    while(!flaga) {
        grayCode = (licznik >> 1) ^ licznik;
        LATA = grayCode;
        licznik++;
        delay(750);
    }
}
```

- Inicjalizuje zmienną `licznik` wartością 0
- Deklaruje zmienną `grayCode` do przechowywania kodu Graya
- Resetuje flagę zmiany programu
- Wchodzi w pętlę, która będzie działać dopóki flaga pozostaje równa 0:
- Konwertuje binarną wartość licznika na kod Graya: `grayCode = (licznik >> 1) ^ licznik` (przesunięcie bitowe w prawo o 1 bit XOR z oryginalną wartością)
- Wyświetla uzyskany kod Graya na porcie A (diody LED)
- Inkrementuje licznik binarny
- Wprowadza opóźnienie 750 jednostek
- Pętla kończy się po naciśnięciu przycisku (gdy flaga zostanie ustawiona na 1)

8. grayDOWN()

```
void grayDOWN(){
    unsigned char licznik = 255;
    unsigned char grayCode;
    flaga = 0;

    while(!flaga) {
        grayCode = (licznik >> 1) ^ licznik;
        LATA = grayCode;
        licznik--;
        delay(750);
    }
}
```

- Inicjalizuje zmienną licznik wartością 255
- Deklaruje zmienną grayCode do przechowywania kodu Graya
- Resetuje flagę zmiany programu
- Wchodzi w pętlę, która będzie działać dopóki flaga pozostaje równa 0:
- Konwertuje binarną wartość licznika na kod Graya: $\text{grayCode} = (\text{licznik} \gg 1) \oplus \text{licznik}$
- Wyświetla uzyskany kod Graya na porcie A (diody LED)
- Dekrementuje licznik binarny
- Wprowadza opóźnienie 750 jednostek
- Pętla kończy się po naciśnięciu przycisku (gdy flaga zostanie ustawiona na 1)

9. bcdUP()

```
void bcdUP(){
    unsigned char dziesiątki = 0;
    unsigned char jedności = 0;
    flaga = 0;

    while(!flaga) {
        LATA = (dziesiątki << 4) | jedności;
        delay(750);

        jedności++;
        if (jedności > 9) {
            jedności = 0;
            dziesiątki++;
        }

        if (dziesiątki > 9) {
            dziesiątki = 0;
        }
    }
}
```

- Inicjalizuje zmienne dziesiątki i jedności wartościami 0
- Resetuje flagę zmiany programu
- Wchodzi w pętlę, która będzie działać dopóki flaga pozostaje równa 0:
- Łączy cyfry dziesiątek i jedności w jedną 8-bitową wartość BCD: $(\text{dziesiątki} \ll 4) \mid \text{jedności}$ (przesunięcie bitowe cyfry dziesiątek o 4 bity w lewo i połączenie operacją OR z cyfrą jedności)
- Wyświetla uzyskaną wartość BCD na porcie A (diody LED)

- Wprowadza opóźnienie 750 jednostek
- Inkrementuje cyfrę jedności
- Jeśli cyfra jedności przekroczy 9, resetuje ją do 0 i inkrementuje cyfrę dziesiątek
- Jeśli cyfra dziesiątek przekroczy 9, resetuje ją do 0
- Pętla kończy się po naciśnięciu przycisku (gdy flaga zostanie ustawiona na 1)

10. bcdDOWN()

```
void bcdDOWN() {
    unsigned char dziesiątki = 9;
    unsigned char jednosci = 9;
    flaga = 0;

    while(!flaga) {
        LATA = (dziesiątki << 4) | jednosci;
        delay(750);

        if (jednosci == 0) {
            jednosci = 9;
            dziesiątki--;
        } else {
            jednosci--;
        }

        if (dziesiątki == 0 && jednosci == 0) {
            dziesiątki = 9;
            jednosci = 9;
        }
    }
}
```

- Inicjalizuje zmienne dziesiątki wartością 9 i jednosci wartością 9 (startuje od 99)
- Resetuje flagę zmiany programu
- Wchodzi w pętlę, która będzie działać dopóki flaga pozostaje równa 0:
- Łączy cyfry dziesiątek i jedności w jedną 8-bitową wartość BCD: (dziesiątki << 4) | jednosci
- Wyświetla uzyskaną wartość BCD na porcie A (diody LED)
- Wprowadza opóźnienie 750 jednostek
- Obsługuje dekrementację:
 - Jeśli cyfra jedności jest równa 0, ustawia ją na 9 i dekrementuje cyfrę dziesiątek
 - W przeciwnym razie dekrementuje cyfrę jedności

- Jeśli obie cyfry osiągną 0, resetuje licznik do wartości 99
- Pętla kończy się po naciśnięciu przycisku (gdy flaga zostanie ustawiona na 1)

11. snake()

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

    while(!flaga) {
        LATA = wez;
        delay(750);

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

- Inicjalizuje zmienną wez wartością 0b00000111 (3 aktywne bity "wężyka")
- Inicjalizuje zmienną kierunek wartością 1 (początkowo wąż porusza się w prawo)
- Resetuje flagę zmiany programu
- Wchodzi w pętlę, która będzie działać dopóki flaga pozostaje równa 0:
- Wyświetla aktualną pozycję wężyka na porcie A (diody LED)
- Wprowadza opóźnienie 750 jednostek
- Aktualizuje pozycję wężyka:
 - Jeśli kierunek = 1, przesuwaj wąż w prawo (operacja <<)
 - Jeśli osiągnie prawą granicę (0b01111000), ustawia pozycję 0b11100000 i zmienia kierunek na -1
 - Jeśli kierunek = -1, przesuwaj wąż w lewo (operacja >>)
 - Jeśli osiągnie lewą granicę (0b00000111), zmienia kierunek na 1

- Pętla kończy się po naciśnięciu przycisku (gdy flaga zostanie ustawiona na 1)

12. kolejka()

```
void kolejka() {
    unsigned char kolejka = 0b00000000;
    unsigned char wzor;
    flaga = 0;

    while(!flaga) {
        // Reset kolejki po zapelnieniu
        kolejka = 0b00000000;

        for (uint16_t i = 0; i < 8 && !flaga; i++) {
            wzor = 0b00000001;
            for (uint16_t j = 0; j < 8 - i && !flaga; j++) {
                LATA = kolejka | wzor;
                delay(600);
                // Sprawdzenie czy wyjsc z petli
                if (flaga) {
                    return;
                }
                // Przesunięcie wzorku
                if (j != 7 - i)
                    wzor <<= 1;
            }
            // dodanie aktualnej diody do reszty kolejki
            kolejka |= wzor;
        }

        // Wszystkie diody sie swieca - wyswietl przez chwile
        LATA = 0b11111111;
        delay(1000);
        if (flaga) {
            return;
        }
    }
}
```

- Inicjalizuje zmienną kolejka wartością 0b00000000 (żadna dioda nie świeci)
- Deklaruje zmienną wzor dla aktualnego bitu "wjeżdżającego"
- Resetuje flagę zmiany programu
- Wchodzi w pętlę główną, która będzie działać dopóki flaga pozostaje równa 0:
- Resetuje kolejkę na początek każdego cyklu

- Przeprowadza 8 faz animacji (jedna dla każdej diody) za pomocą pętli zewnętrznej:
 - Inicjalizuje wzór jako 0b00000001 (najbardziej na prawo bit aktywny)
 - Przeprowadza animację "wjazdu" za pomocą pętli wewnętrznej:
 - Wyświetla aktualny stan kolejki połączony z wzorem: $\text{kolejka} \mid \text{wzor}$
 - Wprowadza opóźnienie 600 jednostek
 - Sprawdza czy nie wyjść z pętli (jeśli flaga zmiany programu jest ustawiona)
 - Przesuwa wzór w lewo, jeśli nie doszedł jeszcze do swojej docelowej pozycji
 - Po zakończeniu "wjazdu" dodaje wzór do kolejki: $\text{kolejka} \mid= \text{wzor}$
- Po wypełnieniu wszystkich diod, wyświetla pełny port (0b11111111) przez 1000 jednostek
- Pętla kończy się po naciśnięciu przycisku (gdy flaga zostanie ustawiona na 1)

13. losowe()

```
void losowe() {
    unsigned char lcg = 0x21; // warto?? pocz?tkowa
    const unsigned char a = 17; // mno?nik
    const unsigned char c = 43; // sta?a dodawana
    flaga = 0;

    while (!flaga) {
        LATA = lcg;           // wyj?cie na port
        delay(1000);
        //LCG z maskowaniem 6-bitowym
        lcg = (a * lcg + c) & 0x3F; // 0x3F -> maska 0b00111111
    }
}
```

- Inicjalizuje zmienną lcg wartością 0x21 (33 dziesiętnie) jako wartość początkową
- Definiuje stałe:
 - a = 17 - mnożnik
 - c = 43 - stała dodawana
- Resetuje flagę zmiany programu
- Wchodzi w pętlę, która będzie działać dopóki flaga pozostaje równa 0:
- Wyświetla aktualną wartość lcg na porcie A (diody LED)
- Wprowadza opóźnienie 1000 jednostek

- Oblicza następną wartość generatora liczbowego za pomocą formuły LCG: $lcg = (a * lcg + c) \& 0x3F$
 - Mnożenie aktualnej wartości przez współczynnik a (17)
 - Dodanie stałej c (43)
 - Zastosowanie maski bitowej 0x3F (0b00111111) dla zachowania tylko 6 najmłodszych bitów
- Pętla kończy się po naciśnięciu przycisku (gdy flaga zostanie ustawiona na 1)

14. Funkcja main() - Główna pętla programu

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

    while(1) {
        if(numer_programu < 1) {
            numer_programu = 9;
        } else if(numer_programu > 9) {
            numer_programu = 1;
        }
        switch(numer_programu) {
            case 1:
                binUP();
                break;
            case 2:
                binDOWN();
                break;
            case 3:
                grayUP();
                break;
            case 4:
                grayDOWN();
                break;
            case 5:
                bcdUP();
                break;
            case 6:
                bcdDOWN();
                break;
            case 7:
                snake();
                break;
            case 8:
                kolejka();
                break;
            case 9:
                losowe();
                break;
        }
    }
}
```

```

    }
}
return 0;
}

```

- Wywołuje funkcję `init()` w celu inicjalizacji mikrokontrolera
- Wchodzi w nieskończoną pętlę:
- Sprawdza granice numeru programu:
 - Jeśli `numer_programu < 1`, ustawia go na 9 (zapętlenie w dół)
 - Jeśli `numer_programu > 9`, ustawia go na 1 (zapętlenie w górę)
- Używa instrukcji `switch` do wyboru właściwego programu na podstawie `numer_programu`
- Wywołuje odpowiednią funkcję
- Po powrocie z funkcji (co następuje po zmianie flagi), pętla kontynuuje działanie i sprawdza nowy numer programu

15. Cały kod programu:

```

/*
 * File:   main.c
 * Author: Jakub Budzich - 169224
 *
 * Created on 24 marca 2025, 08:46
 */
#pragma config POSCMOD = NONE      // Primary Oscillator Select (HS
Oscillator mode selected)
#pragma config OSCIOFNC = ON       // Primary Oscillator Output Function
(OSC2/CLKO/RC15 functions as CLK0 (FOSC/2))
#pragma config FCKSM = CSDCMD      // Clock Switching and Monitor (Clock
switching and Fail-Safe Clock Monitor are disabled)
#pragma config FNOSC = FRC         // Oscillator Select (Primary Oscillator
with PLL module (HSPLL, ECPLL))
#pragma config IESO = OFF          // Internal External Switch Over Mode
(IESO mode (Two-Speed Start-up) disabled)

#pragma config WDTPS = PS32768     // Watchdog Timer Postscaler (1:32,768)
#pragma config FWPSA = PR128       // WDT Prescaler (Prescaler ratio of
1:128)
#pragma config WINDIS = ON         // Watchdog Timer Window (Standard
Watchdog Timer enabled,(Windowed-mode is disabled))
#pragma config FWDTEN = OFF        // Watchdog Timer Enable (Watchdog Timer
is disabled)
#pragma config ICS = PGx2          // Comm Channel Select (Emulator/debugger
uses EMUC2/EMUD2)

```

```

#pragma config GWRP = OFF          // General Code Segment Write Protect
(Writes to program memory are allowed)
#pragma config GCP = OFF           // General Code Segment Code Protect (Code
protection is disabled)
#pragma config JTAGEN = OFF        // JTAG Port Enable (JTAG port is
disabled)

#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
// funkcja opoznienie
void delay(uint32_t ilosc) {
    uint32_t i, j;

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

// Inicjalizacja portow i przerwan
void init() {
    AD1PCFG = 0xFFFF;
    TRISA = 0x0000; // Port A jako wyjście
    TRISD = 0xFFFF; // Port D jako wejście

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

    // Włączenie przerwan Change Notification dla przyciskow
    CNEN1bits.CN15IE = 1; // Włącz przerwanie dla RD6
    CNEN2bits.CN19IE = 1; // Włącz przerwanie dla RD13

    IFS1bits.CNIF = 0; // Wyczyść flagę przerwania CN
    IEC1bits.CNIE = 1; // Włącz przerwanie CN
}

// Procedura obsługi przerwania przyciskami
void __attribute__((interrupt, no_auto_psv)) _CNInterrupt(void) {
    __delay32(200);
}

```

```

// Sprawdzenie, który przycisk został naciśnięty
// poprzedni program
if(PORTDbits.RD13 == 0) {
    numer_programu--;
    flaga = 1;
}
// następny program
else if(PORTDbits.RD6 == 0) {
    numer_programu++;
    flaga = 1;
}

while(PORTDbits.RD13 == 0 || PORTDbits.RD6 == 0);
// Wyczyść flagę
IFS1bits.CNIF = 0;
}

//1. 8 bitowy licznik binarny zliczający w górę (0...255)
void binUP(){
    unsigned char licznik = 0;
    flaga = 0;
    while(!flaga) {
        LATA = licznik++;
        delay(750);
    }
}

//2. 8 bitowy licznik zliczający w dół (255...0)
void binDOWN(){
    unsigned char licznik = 255;
    flaga = 0;
    while(!flaga) {
        LATA = licznik--;
        delay(750);
    }
}

//3. 8 bitowy licznik w kodzie Graya zliczający w górę (repr. 0...255)
void grayUP(){
    unsigned char licznik = 0;
    unsigned char grayCode;
    flaga = 0;

    while(!flaga) {
        grayCode = (licznik >> 1) ^ licznik;
        LATA = grayCode;
        licznik++;
        delay(750);
    }
}

```

```
//4. 8 bitowy licznik w kodzie Graya zliczajacy w dol (repr. 255...0)
```

```
void grayDOWN(){  
    unsigned char licznik = 255;  
    unsigned char grayCode;  
    flaga = 0;  
  
    while(!flaga) {  
        grayCode = (licznik >> 1) ^ licznik;  
        LATA = grayCode;  
        licznik--;  
        delay(750);  
    }  
}
```

```
//5. 2x4 bitowy licznik w kodzie BCD zliczajacy w gore (0...99)
```

```
void bcdUP(){  
    unsigned char dziesiatki = 0;  
    unsigned char jednosci = 0;  
    flaga = 0;  
  
    while(!flaga) {  
        LATA = (dziesiatki << 4) | jednosci;  
        delay(750);  
  
        jednosci++;  
        if (jednosci > 9) {  
            jednosci = 0;  
            dziesiatki++;  
        }  
  
        if (dziesiatki > 9) {  
            dziesiatki = 0;  
        }  
    }  
}
```

```
//6. 2x4 bitowy licznik w kodzie BCD zliczajacy w dol (99...0)
```

```
void bcdDOWN() {  
    unsigned char dziesiatki = 9;  
    unsigned char jednosci = 9;  
    flaga = 0;  
  
    while(!flaga) {  
        LATA = (dziesiatki << 4) | jednosci;  
        delay(750);  
  
        if (jednosci == 0) {  
            jednosci = 9;  
            dziesiatki--;  
        }  
    }  
}
```

```

        dziesiatki--;
    } else {
        jednosci--;
    }

    if (dziesiatki == 0 && jednosci == 0) {
        dziesiatki = 9;
        jednosci = 9;
    }
}

//7. 3 bitowy wezyk poruszajacy sie lewo-prawo
void snake() {
    unsigned char wez = 0b00000111;
    uint16_t kierunek = 1;
    flaga = 0;

    while(!flaga) {
        LATA = wez;
        delay(750);

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

//8. Kolejka
void kolejka() {
    unsigned char kolejka = 0b00000000;
    unsigned char wzor;
    flaga = 0;

    while(!flaga) {
        // Reset kolejki po zapelnieniu
        kolejka = 0b00000000;
    }
}

```



```

    for (uint16_t i = 0; i < 8 && !flaga; i++) {
        wzor = 0b00000001;
        for (uint16_t j = 0; j < 8 - i && !flaga; j++) {
            LATA = kolejka | wzor;
            delay(600);
            // Sprawdzenie czy wyjsc z petli
            if (flaga) {
                return;
            }
            // Przesuniecie wzorku
            if (j != 7 - i)
                wzor <<= 1;
        }
        // dodanie aktualnej diody do reszty kolejki
        kolejka |= wzor;
    }

    // Wszystkie diody sie swieca - wyswietl przez chwile
    LATA = 0b11111111;
    delay(1000);
    if (flaga) {
        return;
    }
}

}

//9. 6 bitowy generator liczb pseudolosowych oparty o konfiguracje 11100111
void losowe() {
    unsigned char lcg = 0x21; // warto?? pocz?tkowa
    const unsigned char a = 17; // mno?nik
    const unsigned char c = 43; // sta?a dodawana
    flaga = 0;

    while (!flaga) {
        LATA = lcg; // wyj?cie na port
        delay(1000);
        //LCG z maskowaniem 6-bitowym
        lcg = (a * lcg + c) & 0x3F; // 0x3F -> maska 0b00111111
    }
}

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

    while(1) {
        if(numer_programu < 1) {

```

```
        numer_programu = 9;
    } else if(numer_programu > 9) {
        numer_programu = 1;
    }
    switch(numer_programu) {
        case 1:
            binUP();
            break;
        case 2:
            binDOWN();
            break;
        case 3:
            grayUP();
            break;
        case 4:
            grayDOWN();
            break;
        case 5:
            bcdUP();
            break;
        case 6:
            bcdDOWN();
            break;
        case 7:
            snake();
            break;
        case 8:
            kolejka();
            break;
        case 9:
            losowe();
            break;
    }
}
return 0;
}
```