Raport kodu dla mikrokontrolera PIC24FJ128GA010

Imię i Nazwisko: Jakub Dmochowski
Numer albumu: 169236

Platforma: MPLAB X IDE + XC16
Mikrokontroler: PIC24FJ128GA010

Przegląd funkcji

Nr	Funkcja	Opis
1	f1()	8-bitowy licznik binarny zliczający w górę od 0 do 255
2	f2()	8-bitowy licznik binarny zliczający w dół od 255 do 0
3	f3()	8-bitowy licznik w kodzie Graya (0 do 255)
4	f4()	8-bitowy licznik w kodzie Graya (255 do 0)
5	f5()	2×4-bitowy licznik BCD od 0 do 99
6	f6()	2×4-bitowy licznik BCD od 99 do 0
7	f7()	"Wężyk" 3-bitowy poruszający się w lewo i prawo
8	f8()	Kolejka
9	f9()	Generator pseudolosowy z pomocniczą funkcją helper9()

Konfiguracja sprzętowa

Program konfiguruje mikrokontroler w następujący sposób:

```
// PIC24FJ128GA010
// CONFIG2
#pragma config POSCMOD = XT
#pragma config OSCIOFNC = ON
#pragma config FCKSM = CSDCMD
#pragma config FNOSC = PRI
#pragma config IESO = ON
// CONFIG1
#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
```

Istotne jest również ustawienie watchdog timera oraz konfiguracja portów I/O:

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

AD1PCFG // Ustawienie wszystkich pinów jako cyfrowe

TRISA // Port A jako wyjście (do sterowania diodami LED)

Implementacja obsługi przycisków

Do obsługi przycisków wykorzystane są funkcje z plików buttons.h i buttons.c. W programie zaimplementowana została funkcja do eliminacji drgań styków przycisków:

```
bool isButtonPressedDebounced(BUTTON button) {
    if (BUTTON_IsPressed(button)) {
        __delay32(300000);
        if (BUTTON_IsPressed(button)) {
            while (BUTTON_IsPressed(button));
            __delay32(300000);
            return true;
        }
    }
    return false;
}
```

S3 - przełączanie do poprzedniej funkcji

S4 - przełączanie do następnej funkcji

Analiza kodu

Funkcja główna main()

Funkcja main() zawiera główną pętlę programu, która obsługuje wybór odpowiedniej funkcji za pomocą przycisków:

```
int main(void) {
    AD1PCFG = 0xFFFF;
    TRISA = 0x0000;
    char button = 0b00000001;

    while (1) {
        if (isButtonPressedDebounced(BUTTON_S3)) {
            button--;
            if (button == 0b00000000) button = 0b00001001;
        }
        else if (isButtonPressedDebounced(BUTTON_S4)) {
            button++;
            if (button == 0b00001010) button = 0b00000001;
        }

        switch(button) {
            case 1: f1(); break;
            case 2: f2(); break;
            case 3: f3(); break;
            case 4: f4(); break;
            case 5: f5(); break;
            case 6: f6(); break;
            case 7: f7(); break;
            case 8: f8(); break;
            case 9: f9(); break;
        }
    }
}
```

Zmienna button przechowuje numer aktualnie wybranej funkcji (1-9). Naciśnięcie przycisku S3 zmniejsza wartość, a S4 zwiększa, z odpowiednim zawijaniem wartości.

Funkcja f1() - Licznik binarny rosnący

```
int f1() {
    unsigned char value = 0b00000000;
    LATA = value;
    while(1){
        __delay32(1000000);
        LATA = ++value;
        if(BUTTON_IsPressed(BUTTON_S3) || BUTTON_IsPressed(BUTTON_S4)) return 0;
    }
}
```

Funkcja inicjalizuje licznik wartością 0, a następnie w nieskończonej pętli inkrementuje licznik i wyświetla jego wartość na diodach LED. Licznik automatycznie wykonuje overflow z 255 na 0 ze względu na typ unsigned char (8 bitów).

Funkcja f2() - Licznik binarny malejący

```
int f2(){
    unsigned char value = 0b11111111;
    LATA = value;
    while(1) {
        __delay32(1000000);
        LATA = --value;
        if(BUTTON_IsPressed(BUTTON_S3) || BUTTON_IsPressed(BUTTON_S4)) return 0;
    }
}
```

Funkcja działa podobnie jak f1(), ale rozpoczyna od wartości maksymalnej 255 i dekrementuje licznik.

Funkcja f3() - Kod Graya rosnący

```
int f3(){
    unsigned char value = 0b00000000;
    unsigned char gray = 0b00000000;
    LATA = gray;
    while(1){
        __delay32(1000000);
        value++;
        gray = value ^ (value >> 1);
        LATA = gray;
        if(BUTTON_IsPressed(BUTTON_S3) || BUTTON_IsPressed(BUTTON_S4)) return 0;
    }
}
```

Funkcja implementuje licznik w kodzie Graya, który jest specjalnym typem kodu binarnego, gdzie dwie kolejne wartości różnią się tylko jednym bitem. Konwersja z kodu binarnego na kod Graya odbywa się poprzez operację XOR (^) na liczbie i tej samej liczbie przesuniętej o 1 bit w prawo (value >> 1).

Funkcja f4() - Kod Graya malejący

```
int f4(){
    unsigned char value = 0b11111111;
    unsigned char gray = 0b11111111 ^ (0b11111111 >> 1);
    LATA = gray;
    while(1){
        __delay32(1000000);
        value--;
        gray = value ^ (value >> 1);
        LATA = gray;
        if(BUTTON_IsPressed(BUTTON_S3) || BUTTON_IsPressed(BUTTON_S4)) return 0;
    }
}
```

Funkcja działa podobnie jak f3(), ale rozpoczyna od wartości maksymalnej i dekrementuje licznik.

Funkcja f5() - Licznik BCD rosnący

```
int f5(){
    unsigned char value = 0;
    unsigned char bcd;
    while(1){
        __delay32(1000000);
        value++;
        if(value > 99) value = 0;
        bcd = ((value / 10) << 4) | (value % 10);
        LATA = bcd;
        if(BUTTON_IsPressed(BUTTON_S3) || BUTTON_IsPressed(BUTTON_S4)) return 0;
    }
}
```

Funkcja implementuje licznik w kodzie BCD (Binary-Coded Decimal), gdzie każda z cyfr dziesiętnych jest reprezentowana na 4 bitach. Starsze 4 bity reprezentują dziesiątki ((value / 10) << 4), a młodsze 4 bity jednostki (value % 10).

Funkcja f6() - Licznik BCD malejący

```
int f6(){
    unsigned char value = 99;
    unsigned char bcd = 0x99;
    while(1){
        __delay32(1000000);
        if(value == 0) value = 99;
        else value--;
        bcd = ((value / 10) << 4) | (value % 10);
        LATA = bcd;
        if(BUTTON_IsPressed(BUTTON_S3) || BUTTON_IsPressed(BUTTON_S4)) return 0;
    }
}
```

Funkcja działa podobnie jak f5(), ale rozpoczyna od wartości 99 i dekrementuje licznik.

Funkcja f7() – Wężyk

```
int f7(){
    unsigned char snake = 0b00000111;
    while(snake != 0b11100000){
        LATA = snake;
        snake <<= 1;
        __delay32(1000000);
        if(BUTTON_IsPressed(BUTTON_S3) || BUTTON_IsPressed(BUTTON_S4)) return 0;
    }
    LATA = snake;
    __delay32(1000000);
    while(snake != 0b00000111){
        snake >>= 1;
        LATA = snake;
        __delay32(1000000);
        if(BUTTON_IsPressed(BUTTON_S3) || BUTTON_IsPressed(BUTTON_S4)) return 0;
    }
    return f7();
}
```

Funkcja implementuje efekt "wężyka" - 3-bitowej sekwencji poruszającej się najpierw w prawo (snake <<= 1), a po osiągnięciu krawędzi - w lewo (snake >>= 1). Rekurencyjne wywołanie return f7() powoduje cykliczne powtarzanie efektu.

Funkcja f8() – Kolejka

```
int f8(){
    while(1){
        unsigned char value = 0b00000000;
        for(int i = 0; i < 8; i++){
            unsigned char temp = 0b00000001;
            for(int j = i + 1; j < 8; j++){
                LATA = value + temp;
                temp <<= 1;
                __delay32(1000000);
                if(BUTTON_IsPressed (BUTTON_S3) || BUTTON_IsPressed (BUTTON_S4)) return 0;
            }
            LATA = value + temp;
            value += temp;
            __delay32(1000000);
            if(BUTTON_IsPressed (BUTTON_S3) || BUTTON_IsPressed (BUTTON_S4)) return 0;
        }
    }
}
```

Funkcja tworzy efekt "kolejki" diod LED, gdzie każda kolejna dioda zapala się, a zapalona dioda pozostaje włączona. Implementacja wykorzystuje zagnieżdżone pętle i operacje przesunięcia bitowego. Zewnętrzna pętla iteruje po wszystkich 8 pozycjach, a wewnętrzna pętla zapala kolejne diody.

Funkcja f9() - Generator pseudolosowy

```
int f9(){
    unsigned char value = 0b01100100;
    while(1){
        value = helper9(value);
        LATA = value;
        __delay32(1000000);
        if(BUTTON_IsPressed(BUTTON_S3) || BUTTON_IsPressed(BUTTON_S4)) return 0;
    }
}

unsigned char helper9(unsigned char seed){
    unsigned char feedback = ((seed >> 6) & 1) ^ ((seed >> 5) & 1) ^
                             ((seed >> 4) & 1) ^ ((seed >> 1) & 1) ^ (seed & 1);
    return (feedback << 7) | (seed >> 1);
}
```

Funkcja implementuje generator liczb pseudolosowych oparty na liniowym rejestrze przesuwnym (LFSR - Linear Feedback Shift Register). Funkcja pomocnicza helper9() oblicza następny stan rejestru na podstawie aktualnego stanu, używając operacji XOR na wybranych bitach (6, 5, 4, 1 i 0). Wynik operacji XOR jest przesuwany na najstarszy bit, a reszta rejestru przesuwana w prawo.