SPRAWOZDANIE

SYSTEMY WBUDOWANE

Obsługa układów we/wy.

Konfiguracja modułu PIO.

IMIĘ I NAZWISKO: Jacek Wójcik

NUMER ĆWICZENIA: 9

Grupa laboratoryjna: 10

Data wykonania ćwiczenia: 15.12.2021

Spis treści

2. Zadanie 1 - kod

2.1 PIO_library.h

```
#include <targets\AT91SAM7.h> // Dołączam biliotekę dla zestawu w
labolatorium

/* Funkcja do realizowania opóźnień
*  ms - liczba milisekund, jakie trzeba odczekać
*  Nic nie zwraca
*/
void time_delay(unsigned int ms);

/* Funkcja do włączania i wyłączania zegara dla peryferiów
*  pio_pcer - włączenie (1) lub wyłączenie (0) zegara
*  a_b - kontroler peryferiów: PIOA (0) lub PIOB(1)
*  Nic nie zwraca
*/
void PIO_clock_enable(unsigned int pio_pcer, unsigned int a_b);

/* Funkcja do włączania i wyłączania obsługi pinów w kontrolerze PIOB
*  pin_number - numer pinu do włączenia w kontrolerze PIOB
*  enable - włączenie (1) lub wyłączenie (0) danego pinu
*  Nic nie zwraca
*/
void PIOB_enable(unsigned int pin_number, unsigned int enable);

/* Funkcja do ustawiania pinów w kontrolerze PIOB jako wejście lub wyjście
*  pin_number - numer pinu do skonfigurowania w kontrolerze PIOB
*  enable - ustawianie pinu jako wyjście (1) lub wejście (0)
*  Nic nie zwraca
*/
void PIOB_output_enable(unsigned int pin_number, unsigned int enable);

/* Funkcja do ustawiania stanu pinów w kontrolerze PIOB
*  pin_number - numer pinu do skonfigurowania w kontrolerze PIOB
*  enable - ustawianie pinu w stan wysoki (1) lub niski (1)
*  Nic nie zwraca
*/
void PIOB_output_state(unsigned int pin_number, unsigned int enable);

/* Funkcja do negowania stanu pinów w kontrolerze PIOB
*  pin_number - numer pinu do zanegowania
*  Nic nie zwraca
*/
void PIOB_output_negate(unsigned int pin_number);
```

```c
/*  Funkcja do pobierania stanu pinu z kontrolera B
*   SW_number - numer pinu, którego stan będę pobierał
*   Zwraca liczbę oznaczającą stan wysoki (1) lub stan niski (0)
*/
unsigned int SW_odczyt(unsigned int SW_number);

/*  Funkcja do debouncingu przycisków
*   SW_number - numer pinu, którego stan będę sprawdzał
*   Nic nie zwraca
*/
void SW_czytaj(unsigned int SW_number);
```

2.2 PIO_library.c

```
#include "PIO_library.h"    // Dołączam moją bibliotekę

/*  Funkcja do realizowania opóźnień
*   ms - liczba milisekund, jakie trzeba odczekać
*   Nic nie zwraca
*/
void time_delay(unsigned int ms)
{
    // Pierwsza pętla odliczająca milisekundy
    for( volatile int aa=0;aa<=ms;aa++)
    {
        // Druga pętla odliczająca jedną milisekundę
        for( volatile int bb=0;bb<=3000;bb++)
        {
            __asm__("NOP");
        }
    }
}

/*  Funkcja do włączania i wyłączania zegara dla peryferiów
*   pio_pcer - włączenie (1) lub wyłączenie (0) zegara
*   a_b - kontroler peryferiów: PIOA (0) lub PIOB(1)
*   Nic nie zwraca
*/
void PIO_clock_enable(unsigned int pio_pcer, unsigned int a_b)
{
    if(pio_pcer==1)
    {
        // Włączam kontroler ustawiając bit nr. 2 dla PIOA lub 3 dla PIOB
        PMC_PCER = 1 << a_b + 2;
    }

    if(pio_pcer==0)
    {
        // Wyłączam kontroler ustawiając bit nr. 2 dla PIOA lub 3 dla PIOB
        PMC_PCDR = 1 << a_b + 2;
    }
}

/*  Funkcja do włączania i wyłączania obsługi pinów w kontrolerze PIOB
*   pin_number - numer pinu do włączenia w kontrolerze PIOB
*   enable - włączenie (1) lub wyłączenie (0) danego pinu
*   Nic nie zwraca
*/
void PIOB_enable(unsigned int pin_number, unsigned int enable)
{
```

```
    if(enable == 1)
    {
        // Włączam obsługę pinu ustawiając bit odpowiadający danemu pinowi
        PIOB_PER = 1 << pin_number;
    }

    if(enable == 0)
    {
        // Wyłączam obsługę pinu ustawiając bit odpowiadający danemu pinowi
        PIOB_PDR = 1 << pin_number;
    }
}

/*  Funkcja do ustawiania pinów w kontrolerze PIOB jako wejście lub wyjście
*   pin_number - numer pinu do skonfigurowania w kontrolerze PIOB
*   enable - ustawianie pinu jako wyjście (1) lub wejście (0)
*   Nic nie zwraca
*/
void PIOB_output_enable(unsigned int pin_number, unsigned int enable)
{
    if(enable == 0)
    {
        // Ustawiam pin jako wejście
        PIOB_ODR = 1 << pin_number;
    }

    if(enable == 1)
    {
        // Ustawiam pin jako wyjście
        PIOB_OER = 1 << pin_number;
    }
}

/*  Funkcja do ustawiania stanu pinów w kontrolerze PIOB
*   pin_number - numer pinu do skonfigurowania w kontrolerze PIOB
*   enable - ustawianie pinu w stan wysoki (1) lub niski (1)
*   Nic nie zwraca
*/
void PIOB_output_state(unsigned int pin_number, unsigned int enable)
{
    if(enable == 0)
    {
        // Ustawiam pin w stan niski
        PIOB_CODR = 1 << pin_number;
    }

    if(enable == 1)
    {
```

```
        // Ustawiam pin w stan wysoki
        PIOB_SODR = 1 << pin_number;
    }
}

/*  Funkcja do negowania stanu pinów w kontrolerze PIOB
*   pin_number - numer pinu do zanegowania
*   Nic nie zwraca
*/
void PIOB_output_negate(unsigned int pin_number)
{
    // Oznaczam piny, których stan będę zmieniał
    PIOB_OWER = 1 << pin_number;
    // Neguję stan pinu
    PIOB_ODSR ^= 1 << pin_number;
}

/*  Funkcja do pobierania stanu pinu z kontrolera B
*   SW_number - numer pinu, którego stan będę pobierał
*   Zwraca liczbę oznaczającą stan wysoki (1) lub stan niski (0)
*/
unsigned int SW_odczyt(unsigned int SW_number)
{
    // Zwracam wartość bitu, który odpowiada danemu pinowi
    return PIOB_PDSR & 1 << SW_number;
}

/*  Funkcja do debouncingu przycisków
*   SW_number - numer pinu, którego stan będę sprawdzał
*   Nic nie zwraca
*/
void SW_czytaj(unsigned int SW_number)
{
    time_delay(75); // Czekam, aż stan przycisku się ustabilizuje
    // Czekam tak długo, jak przycisk będzie wciśnięty
    while((PIOB_PDSR & 1 << SW_number) == 0);
}
```

2.3 main.c

```c
#include "PIO_library.h"    // Dołączam moją bibliotekę

#define LCD_BL 20   // Makro dla czytelności
#define PIOB 1      // Makro dla czytelności
#define ENABLE 1    // Makro dla czytelności
#define DISABLE 0   // Makro dla czytelności

int main() // Funkcja główna
{
    // Włączam zegar dla kontrolera PIOB
    PIO_clock_enable(ENABLE,PIOB);
    // Włączam obsługę pinu 20 w kontrolerze PIOB
    PIOB_enable(LCD_BL, ENABLE);
    // Ustawiam pin 20 jako wyjście
    PIOB_output_enable(LCD_BL, ENABLE);

    while(1) // Nieskończona pętla
    {
        // Włączam podświetlenie ekranu
        PIOB_output_state(LCD_BL, ENABLE);
        time_delay(500); // Czekam 500ms
        // Wyłączam podświetlenie ekranu
        PIOB_output_state(LCD_BL, DISABLE);
        time_delay(500); // Czekam 500ms
    }
    return 0;
}
```

3. Zadanie 1 - opis

To zadanie polegało na napisaniu biblioteki do obsługi kontrolera PIO i wykorzystaniu jej do wykonania akcji migania podświetleniem ekranu LCD poprzez włączanie go i wyłączanie co 500 ms.

Biblioteka znajduje się w plikach “**PIO_library.h**” oraz “**PIO library.c**” i opis wszystkich jej funkcji, ich argumenty oraz wartości przez nie zwracane znajduje się na każdą deklaracją i definicją funkcji zarówno w pliku “**PIO library.c**” jak i pliku “**PIO_library.h**”. Rejestry, jakich użyłem do realizacji zadań tych funkcji są opisane w moim poprzednim sprawozdaniu. Poniżej zamieszczę skrót najważniejszych informacji:

- **time_delay** - służy do opóźniania o około 1 sekundę
- **PIO_clock_enable** - służy do włączania i wyłączania zegara dla peryferiów: kontrolera PIOA i PIOB
- **PIOB_enable** - służy do włączania i wyłączania obsługi poszczególnych pinów
- **PIOB_output_enable** - służy do ustawiania pinów jako wejście lub wyjście
- **PIOB_output_state** - służy do ustawiania pinów w stan wysoki lub niski
- **PIOB_output_negate** - służy do negowania stanu pinów
- **SW_odczyt** - służy do pobrania stanu przycisku
- **SW_czytaj** - służy do czekania aż użytkownik puści przycisk.

W celu wykonania akcji migania podświetleniem wyświetlacza LCD musiałem najpierw włączyć zegar dla kontrolera B funkcją **PIO_clock_enable**, jako argument podając PIOB, następnie musiałem włączyć obsługę pinu podłączonego do podświetlenia funkcją **PIOB_enable**, a potem wystarczyło tylko ustawić ten pin jako wyjście funkcją **PIOB_output_enable**. W nieskończonej pętli while() najpierw funkcją **PIOB_output_state** włączałem podświetlenie, następnie funkcją **time_delay** czekam 500ms, następnie znowu funkcją **PIOB_output_state** wyłączam podświetlenia, a następnie znowu czekam 500ms.

Żeby zachować czystość kodu postanowiłem dla części najważniejszych stałych zdefiniować aliasy za pomocą słowa kluczowego “**define**”. Dzięki temu mogę “czytać” mój kod i nie muszę się zastanawiać co znaczą poszczególne liczby. Jest to także bardzo użyteczne, gdy chcemy wytłumaczyć kod komuś, może go nie znać.

4. Zadanie 2 - kod

```
#include "PIO_library.h"    // Dołączam moją bibliotekę

#define LCD_BL 20       // Makro dla czytelności
#define BUTTON_SW1 24   // Makro dla czytelności
#define BUTTON_SW2 25   // Makro dla czytelności
#define PIOB 1          // Makro dla czytelności
#define ENABLE 1        // Makro dla czytelności
#define DISABLE 0       // Makro dla czytelności

int main() // Funkcja główna
{
    // Włączam zegar dla kontrolera PIOB
    PIO_clock_enable(ENABLE,PIOB);
    // Włączam obsługę pinu 20 w kontrolerze PIOB
    PIOB_enable(LCD_BL, ENABLE);
    // Ustawiam pin 20 jako wyjście
    PIOB_output_enable(LCD_BL, ENABLE);

    while(1) // Nieskończona pętla
    {
        // Jeżeli przycisk SW1 jest wciśnięty
        if(SW_odczyt(BUTTON_SW1)
        {
            // Włączam podświetlenie ekranu
            PIOB_output_state(LCD_BL, ENABLE);
            // Czekam aż użytkownik puści przycisk
            SW_czytaj(BUTTON_SW1);
        }
        // Jeżeli natomiast przycisk SW2 jest wciśnięty
        else if(SW_odczyt(BUTTON_SW2)
        {
            // Wyłączam podświetlenie ekranu
            PIOB_output_state(LCD_BL, DISABLE);
            // Czekam aż użytkownik puści przycisk
            SW_czytaj(BUTTON_SW2);
        }
    }
    return 0;
}
```

5. Zadanie 2 - opis

W tym zadaniu należało wykorzystać utworzoną bibliotekę do kontroli podświetlenia ekranu LCD za pomocą przycisków SW1 i SW2. Naciśnięcie przycisku SW1 miało włączać podświetlenie, a naciśnięcie przycisku SW2 miało je wyłączać. Żeby podczas jednoczesnego naciśnięcia przycisków nie doszło do migotania ekranu, należało zastosować instrukcję "**if {} else if{}**", i czekać, aż dany przycisk zostanie puszczony, żeby naciśnięcie drugiego przycisku nie spowodowało migotania ekranu.

W tym celu najpierw włączyłem kontroler PIOB funkcją **PIOB_output_state**. Następnie musiałem przygotować pin 20 do pracy, stosują identyczne kroki jak w zadaniu 1, tj. wywołać funkcję **PIOB_enable**, a następnie **PIOB_output_enable**.

W nieskończonej pętli while() najpierw sprawdzałem za pomocą funkcji **SW_odczyt** czy przycisk SW_1 został wciśnięty. Jeżeli tak, to włączałem podświetlenie funkcją **PIOB_output_state** i czekałem aż użytkownik puści przycisk funkcją **SW_czytaj**. Jeżeli przycisk SW_1 nie został wciśnięty, sprawdzałem czy został wciśnięty przycisk SW_2. Jeżeli tak, to analogicznie wyłączałem podświetlenie za pomocą funkcji **PIOB_output_state** i czekałem aż użytkownik puści przycisk za pomocą funkcji **SW_czytaj**.

Tak samo jak w zadaniu 1, postanowiłem oznaczyć część ważnych stałych za pomocą aliasów w celu zwiększenia czytelności kodu.

6. Zadanie 3 - kod

```
#include "PIO_library.h"    // Dołączam moją bibliotekę

#define LCD_BL 20       // Makro dla czytelności
#define BUTTON_SW1 24   // Makro dla czytelności
#define BUTTON_SW2 25   // Makro dla czytelności
#define PIOB 1          // Makro dla czytelności
#define ENABLE 1        // Makro dla czytelności
#define DISABLE 0       // Makro dla czytelności

int main() // Funkcja główna
{
    // Włączam zegar dla kontrolera PIOB
    PIO_clock_enable(ENABLE,PIOB);
    // Włączam obsługę pinu 20 w kontrolerze PIOB
    PIOB_enable(LCD_BL, ENABLE);
    // Ustawiam pin 20 jako wyjście
    PIOB_output_enable(LCD_BL, ENABLE);

    while(1) // Nieskończona pętla
    {
        // Jeżeli przycisk SW1 jest wciśnięty
        if(SW_odczyt(BUTTON_SW1)
        {
            // Zmieniam stan podświetlenia ekranu
            PIOB_output_negate(LCD_BL, ENABLE);
            // Czekam aż użytkownik puści przycisk
            SW_czytaj(BUTTON_SW1);
        }
    }
    return 0;
}
```

7. Zadanie 3 - opis

W tym zadaniu należało wykorzystać utworzoną bibliotekę do kontroli podświetlenia ekranu LCD za pomocą przycisku SW1. Naciśnięcie przycisku SW1 miało zmieniać stan podświetlenia. Ważne było, żeby nie dopuścić do sytuacji, gdzie po przytrzymaniu przycisku ekran zaczynał migotać. W tym celu po każdym naciśnięciu przycisku należało czekać, aż dany przycisk zostanie puszczony, żeby naciśnięcie drugiego przycisku nie spowodowało migotania ekranu.

W tym celu najpierw włączyłem kontroler PIOB funkcją **PIOB_output_state**. Następnie musiałem przygotować pin 20 do pracy, stosują identyczne kroki jak w zadaniu 1 i 2, tj. wywołać funkcję **PIOB_enable**, a następnie **PIOB_output_enable**.

W nieskończonej pętli while() najpierw sprawdzałem za pomocą funkcji **SW_odczyt** czy przycisk SW_1 został wciśnięty. Jeżeli tak, to funkcją **PIOB_output_negate** zmieniałem stan pinu odpowiedzialnego za podświetlenie ekranu i czekałem aż użytkownik puści przycisk funkcją **SW_czytaj**.
Tak samo jak w zadaniu 1 i 2, postanowiłem oznaczyć część ważnych stałych za pomocą aliasów w celu zwiększenia czytelności kodu.

Dzięki temu, że czekałem aż użytkownik puści przycisk funckją SW_czytaj, nie dopuściłem do sytuacji, gdzie podczas długotrwałego trzymania wciśniętego przycisku SW1 dochodziło do migotania ekranu, ponieważ program cały czas odczytywał naciśnięcia przycisku i zmieniał stan podświetlenia ekranu.

8. Wnioski

Podczas tych laboratoriów nauczyłem się następujących rzeczy:

- ➔ Sterowniki tworzy się w celu usprawnienia tworzenia aplikacji, ponieważ obsługę danego modułu (może to być port USB, kontroler PIO lub port RS232) można oddzielić od faktycznego kodu aplikacji i zamknąć np. w bibliotece. Dzięki temu można wielokrotnie użyć tego samego kodu, a w przypadku chęci poprawy lub zmiany sterownika należy zmienić tylko kod w jednym miejscu a nie w kilku, co ogranicza ryzyko powstawania błędów, ponieważ jest tylko jedno miejsce, w którym możemy popełnić błąd.
- ➔ Modyfikowanie stanu danego rejestru zazwyczaj nie odbywa się bezpośrednio, ale poprzez ustawienie odpowiedniego bitu rejestru odpowiedzialnego za ustawianie lub czyszczenie odpowiedniego bitu w danym rejestrze.

  Przykładem takiego rejestru może być rejestr **PMC** (“Peripheral Clock Controller”), który służy do włączania zasilania do poszczególnych peryferiów. Do ustawiania bitów służy rejestr **PMC_PCER** (“Peripheral Clock Enable”), a do czyszczenia bitów służy rejestr **PMC_PCDR** (“Peripheral Clock Disable”). Żeby wyczyścić lub ustawić dany bit w rejestrze PMC należy ustawić bit w rejestrze **PMC_PCER** lub **PMC_PCDR** na 1. Ustawienie bitu na 0 w tych dwóch rejestrach nie ma żadnego efektu.
- ➔ Jednak są od tego wyjątki, jak np. rejestr **PIOB_ODSR** (“Output Data Status Register”), który pozwala na manipulację stanem wyjścia pinów poprzez zapisanie tam logicznego 0 (stan niski) lub 1 (stan wysoki).
- ➔ W celu sterowania pinem jako wyjście (tj. ustawianie go w stan wysoki lub niski) należy włączyć zegar dla odpowiedniego kontrolera, włączyć obsługę danego pinu i ustawić ten pin jako wyjście.
- ➔ Jednak okazuje się, że w celu odczytu stanu z danego pinu należy jedynie włączyć zegar dla odpowiedniego kontrolera.
- ➔ Możemy sterować wyjściem na dwa sposoby:
  - ◆ Za pomocą funkcji **PIOB_SODR** i **PIOB_CODR** (“Set Output Data Register” i “Clear Output Data Register”). Żeby ustawić stan danego pinu należy ustawić odpowiadający mu bit w:
    - **PIOB_SODR** w celu ustawienia pinu w stan wysoki
    - **PIOB_CODR** w celu ustawienia pinu w stan niski
  - ◆ Za pomocą manipulacji rejestrem **PIOB_ODSR** (“Output Data Status Register”). Żeby ustawić stan danego pinu należy najpierw odmaskować odpowiadający mu bit w rejestrze **PIOB_OWER** (“Output Write Enable Register”), a następnie w rejestrze **PIOB_ODSR** należy ustawić bit reprezentujący dany pin w stan logicznego 0 (stan niski) lub 1 (stan wysoki).
- ➔ Makrodefinicje są bardzo dobrym sposobem na zwiększenie czytelności kodu, co może pomóc gdy wrócimy do niego po dłuższym czasie, lub gdy chcemy go pokazać komuś, kto nigdy go nie widział.