Akademia Górniczo-Hutnicza
w Krakowie
Katedra Elektroniki
WIET

Laboratorium TM2

Ćwiczenie 2

Podstawy programowania mikrokontrolerów w języku C

Autor: Paweł Russek

wer. 16.10.2020

1. Cel ćwiczenia

W trakcie poniższego ćwiczeniu student zapozna się z charakterystycznymi dla programowania mikrokontrolerów elementami języka C. Instrukcja zakłada jednak, że student zna już ogólne podstawy programowania w tym języku.

Dodatkowo, instrukcja ma za zadanie przedstawienie płytki ewaluacyjnej FRDM-KL05Z, której głównym elementem jest mikrokontroler Kinetis-L MKL05Z32VFM4 firmy NXP. Bardziej szczegółowo zostaną omówione bloki wejścia/wyjścia ogólnego zastosowania tego mikrokontrolera.

Dodatkowo, po wykonaniu przedstawionych w instrukcji ćwiczeń, student będzie wiedział jak korzystać ze środowisko programowania mikrokontrolerów uVision5. Student nauczy się jak tworzyć projekt, wprowadzać kod, kompilować i uruchamiać aplikację na mikrokontrolerze.

2. Język C dla mikrokontrolerów

2.1. Heksadecymalny zapis liczb

W programach przeznaczonych dla mikrokontrolerów wartości liczbowe bardzo często są zapisywane w kodzie heksadecymalnym. Wartości heksadecymalne (nazywane również liczbami hex) są liczbami zapisanymi w szesnastkowym kodzie pozycyjnym. Kod ten używa szesnastu symboli, którymi najczęściej są cyfry od 0 do 9 i litery od A, B, C, D, E, F (lub a, b, c, d, e, f), które reprezentują wartości od 10 do 15. W języku C liczby te są przedstawiane jako ciąg symboli od 0 do F poprzedzonych prefiksem „0x". Na przykład: 0xC2, 0x2ad1.

Konwersja wartości z formatu hex do dziesietnego jest w praktyce potrzebna bardzo rzadko. Jednak, student powinien wiedzieć jak to robić. Częściej występuje potrzeba konwersji liczb binarnych do heksadecymalnych i odwrotnie. Przeprowadzenie konwersji „hex do bin" i „bin do hex" jest bardzo proste jeżeli zauważyć, że każdej liczbie heksadecymalnej odpowiada czterobitowa sekwencja binarna.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Z przedstawioną powyżej tablicą konwersja „hex do bin" jest realizowana poprzez dekompozycję liczby na pojedyncze cyfry hex:

0x9ABC			
9	A	B	C
1001	1010	**1011**	1100
0b**1001**1010**1011**1100			

Ćwiczenia 1.2

A. Przeprowadź konwersję do wartości binarnej: 0xAD, 0x23B

B. Przeprowadź konwersję do wartości heksadecymalnej: 0b11011011, 0b1000100111

2.2. Typy danych dla systemów wbudowanych

W podręcznikach programowania w ANSI języku C najczęściej spotykanymi typami danych całkowitoliczbowych są: char, short, int, long. Rozmiar typu int zazwyczaj odpowiada rozmiarowi rejestrów procesora dla którego kompilator generuje kod maszynowy. Przykładowo, int ma rozmiar dwóch bajtów dla procesorów 16 bitowych, a dla procesorów 32 bitowych ma rozmiar czterech bajtów. Ten brak ujednolicenia int w C często prowadzi do problemów z przenośnością kodów. Co więcej, również to czy typ char jest traktowany liczbą ze znakiem czy bez również zależy od kompilatora. Standard C99, który jest powszechnie stosowany w programowaniu mikrokontrolerów rozwiązuje te problemy poprzez wprowadzenie dodatkowych typów danych. W standardzie ISO C99, nowe typy danych w swojej nazwie, wprost określają swój rozmiar bitowy oraz to czy są liczbami ze znakiem czy bez. Na przykład:

Data type	**Size**	**Range**
int8_t	1 byte	-128 to 127
uint8_t	1 byte	0 to 255
int16_t	2 byte	-32,768 to 32,767
uint16_t	2 byte	0 to 65,535
int32_t	4 byte	-2,147,483,648 to 2,147,483,647
uint32_t	4 byte	0 to 4,294,967,295

2.3. Operacje bitowe w C

Jedną z najważniejszych mocnych stron języka C, jeżeli chodzi o programowanie mikrokonotrolerów, są zaimplementowane operacje manipulowania bitami. Każdy programista języka C jest dobrze zaznajomiony z operacjami logicznymi AND (&&), OR (II) i NOT (!). Jednak wielu programistów, programujących aplikacje działające pod kontrolą systemu operacyjego , nie zna sposobu stosowania bitowych operatorów logicznych i przesunięć: AND(&), OR(|), EX-OR(^), inwersja(~), przesuń w prawo (>>), przesuń w lewo (<<). Logiczne operatory bitowe są szeroko stosowane w programowaniu systemów wbudowanych i kontrolno-pomiarowych. Operatory bitowe są realizowane niezależnie na poszczególnych parach odpowiadających sobie bitów. Na przykład:

0x54 & 0x0F daje w wyniku 0x04 // AND

0x40 | 0x86 daje w wyniku 0xC6 // OR

0x54 ^ 0x78 daje w wyniku 0x2C // XOR

~ 0x55 daje w wyniku in 0xAA //Inwersja 0x55

W rezultacie, operator OR może być użyty do ustawiania wybranych bitów, operator AND może być użyty do skasowania wybranych bitów, a operator XOR do zmiany ich wartości na wartość przeciwną.

Na przykład:

VAR1=VAR1 | 0x01 //Ustawia bit b0 zmiennej VAR1

VAR2=VAR2 | 0x08 /Ustawia bit b3 zmiennej VAR1

VAR3=VAR3&(~0x04) //Zeruje bit b2 zmiennej VAR3

VAR4=VAR4^0x80 //Przestawia bit b7 zmiennej VAR4

Dodatkowo, język C dostarcza operatory przesunięć bitowych. Operator przesuń w prawo (>>) przesuwa bity w prawo o wybraną liczbę pozycji. Operator przesuń w lewo (<<) przesuwa bity w lewo o wybraną liczbę pozycji.

Na przykład:

VAR1=0x01<<3 //VAR1=0x08

VAR1=0xD7>>2 //VAR1=0x35

W programowaniu mikrokontrolerów logiczne operacje bitowe i są wykorzystywane do łatwego manipulowania stanem pojedynczych bitów wybranej zmiennej.

Na przykład:

VAR1=VAR1 | (1<<5) //Ustawia bit b5

VAR2=VAR2 & ~(1<<7) //Kasuj bit b7

VAR3=VAR3 ^ ((1<<3)|(1<<6)) //Przestaw bity b3 i b6

Warto odnotować, że operacje bitowe również mogą być łączone z operatorem przypisania (analogicznie jak przypisanie VAR=VAR+1 może być zastąpione zapisem VAR+=1)

Na przykład:

VAR1|= (1<<5) //Ustaw bit b5

VAR2&= ~(1<<7) //Kasuj bit b7

VAR3^= (1<<3) |(1<<6)//Przestaw bity b3 i b6

2.4. Programowanie wskaźników

Tym co szczególnie odróżnia język C od innych języków programowania, to wskaźniki, których rozumienie i umiejętność programowania jest umiejętnością konieczną w przypadku każdego programisty systemów wbudowanych. Jak wie każdy kto ukończył chociaż podstawowy kurs programowania w C, wskaźniki reprezentują adresy zmiennych w pamięci. Z drugiej jednak strony, dzisiejsze systemy komputerowe, pracujące pod kontrolą współczesnych systemów operacyjnych, w skuteczny sposób zapewniają ochronę pamięci i izolują programistę od pamięci fizycznej. Powoduje to, że bezpośrednie ustawianie wskaźników nie jest ani możliwe, ani konieczne w przypadku programowania aplikacji pracujących dla PC. Sytuacja jest odmienna w przypadku programowania aplikacji mikrokontrolerów pracujących samodzielnie i bez systemu operacyjnego. W tym przypadku programista często chce bezpośrednio modyfikować wybrane adresy w przestrzeni adresowej mikrokontrolera.

Na przykład:

uint32_t mem_addr = 0xFFFF2000; //Deklaracja wskaźnika, ustawionego na adres 0xFFFF2000;*

**mem_addr = 0xBA8A0070; //Ustawia wartość 0xBA8A0070 pod adresem 0xFFFF2000;*

Powyższe dwie linie kodu mogą być zastąpione jedną linią kodu:

**((uint32_t*) 0xFFFF2000)= 0xBA8A0070;*

Ze względu na przejżystość kodu, czasami może być również zapisanie tego samego w jeszcze inny sposób:

*#define MY_MEM_LOCATION *((uint32_t*) 0xFFFF2000)*

MY_MEM_LOCATION=0xBA8A0070; //ustawia wartość 0xBA8A0070 pod adresem 0xFFFF2000

MY_MEM_LOCATION|=(1<<27); //Ustawia bit b27 pod adresem 0xFFFF2000;

MY_MEM_LOCATION&=~((1<<4) | (1<<23)); //Czyści bity b4 i b23 pod adresem 0xFFFF2000;

2.5. Niekończąca się pętla

Bardzo ważnym elementem kodu aplikacji pracującej pod kontrolą mikrokontrolera jest niekończąca się pętla. Kiedy na mikrokontrolerze nie uruchomiono systemu operacyjnego aplikacja nigdy nie może się skończyć, bo w tym przypadku procesor wyszedłby poza zakres ważnego obszaru pamięci programu i doszłoby do awarii systemu. Z tego powodu programiści mikrokontrolerów zawsze umieszczeją w kodzie niekończące się pętle.

Przykład niekończącej się pętli w C:

```
main(){
        /* Tutaj umieścić kod programu do jednorazowego uruchomienia */
        while(1){
                /* Tutaj umieścić kod programu do niekończącego się powtarzania */
        };
}
```

2.6. Słowo kluczowe „volatile”

W języku C, programiści oznaczają zadeklarowane zmienne jako „*volatile*” (ulotne) chcąc im nadać specjalne właściwości, które mają dla kompilatora znaczenie w procesie optymalizacji kodu. Najkrócej mówiąc, określenie zmiennej jako „*volatile*” wyłącza pewne metody optymalizacji. Słowo kluczowe „*volatile*” dodane przy deklaracji zmiennej oznacza, że dana zmienna może zmieniać swoją wartość w sposób niezwiązany z wykonywaniem przez procesor kodu programu. W tym ćwiczeniu będziemy używali „*volatile*”, aby oznaczyć zmienne związane z rejestrami sprzętowymi (a nie z pamięcią). Jeżeli kompilator wie, że dana zmienna lub wskaźnik nie odnoszą się do pamięci, ale do rejestru sprzętowego modyfikowanego przez zdarzenia zewnętrzne, nie będzie próbował usuwać niepotrzebnego kodu programu.

Przykład:

Example:

```
#define MY_MEM_LOCATION  *((uint32_t*) 0xFFFF2000)
main(){
        MY_MEM_LOCATION=0x55555555;
        while(1);
}
```

W powyższym kodzie lokacja pod adresem MY_MEM_LOCATION jest zapisywana, ale nigdy nie jest odczytywana dlatego kompilator może usunąć taki kod w procesie optymalizacji.

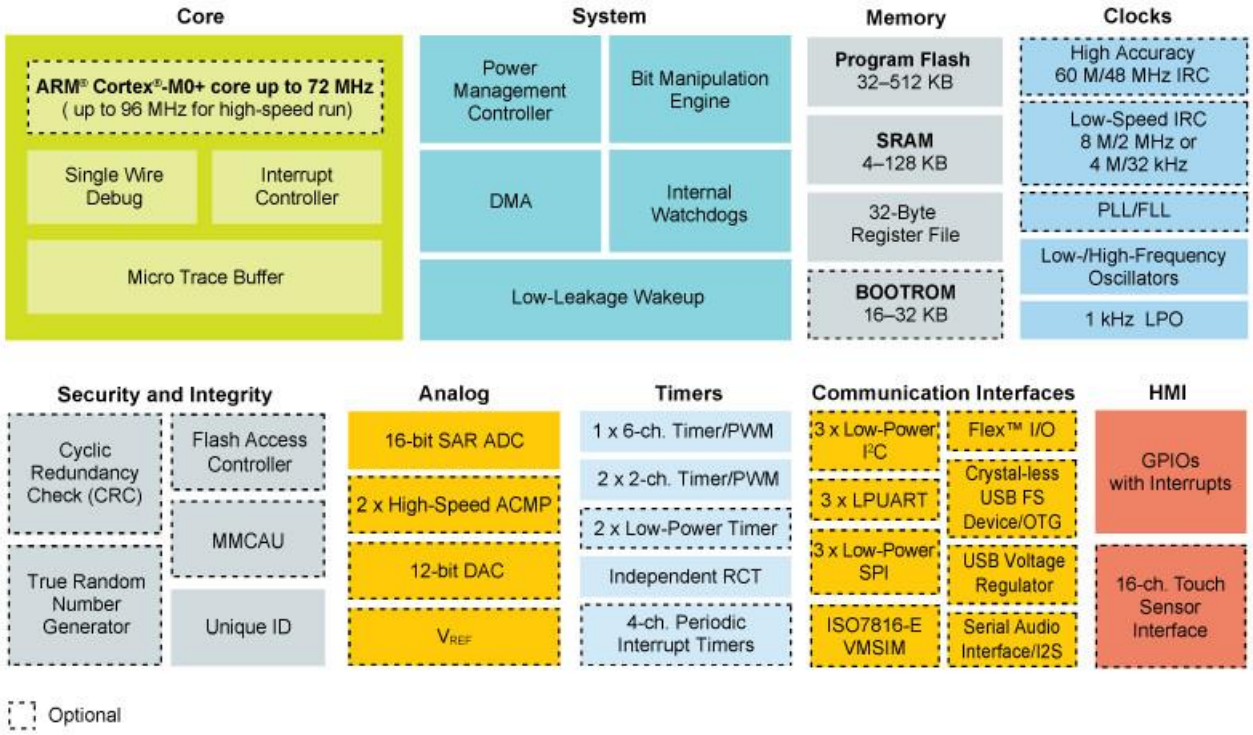
```
#define MY_MEM_LOCATION  *((volatile uint32_t*) 0xFFFF2000)
main(){
        MY_MEM_LOCATION=0x55555555;
        while(1);
}
```

Po dodaniu słowa kluczowego „*volatile*” optmalizacja kodu nie zostanie przeprowadzona.

Istnieje więcej przypadków oprócz deklaracji rejestrów sprzętowych, które wymagają użycia „*volatile”*, ale nie będą one omawiane na potrzeby tego ćwiczenia.

3. Rdzeń ARM Cortex-M0+

Układ MKL05Z32VFM4 produkowany przez firmę NXP to mikrokontroler zbudowany w oparciu o powszechnie stosowany rdzeń ARM z rodziny Cortex-M0+. Mikrokontroler MKL05Z32VFM4 posiada 32 kilobajtów (32KB) pamięci programu w postaci pamięci nieulotnej flash, 4 kB wewnętrznej pamięci danych SRAM oraz dużą liczbę układów peryferyjnych WE/WY.



Schemat blokowy mikrokontrolera z serii Kinetis L KLxx

ARM to mikrokontroler 32 bitowy, mający przestrzeń adresową o rozmiarze 4 GB (gigabajty). Przestrzeń adresowa pamięci i układów WE/WY są współdzielone.

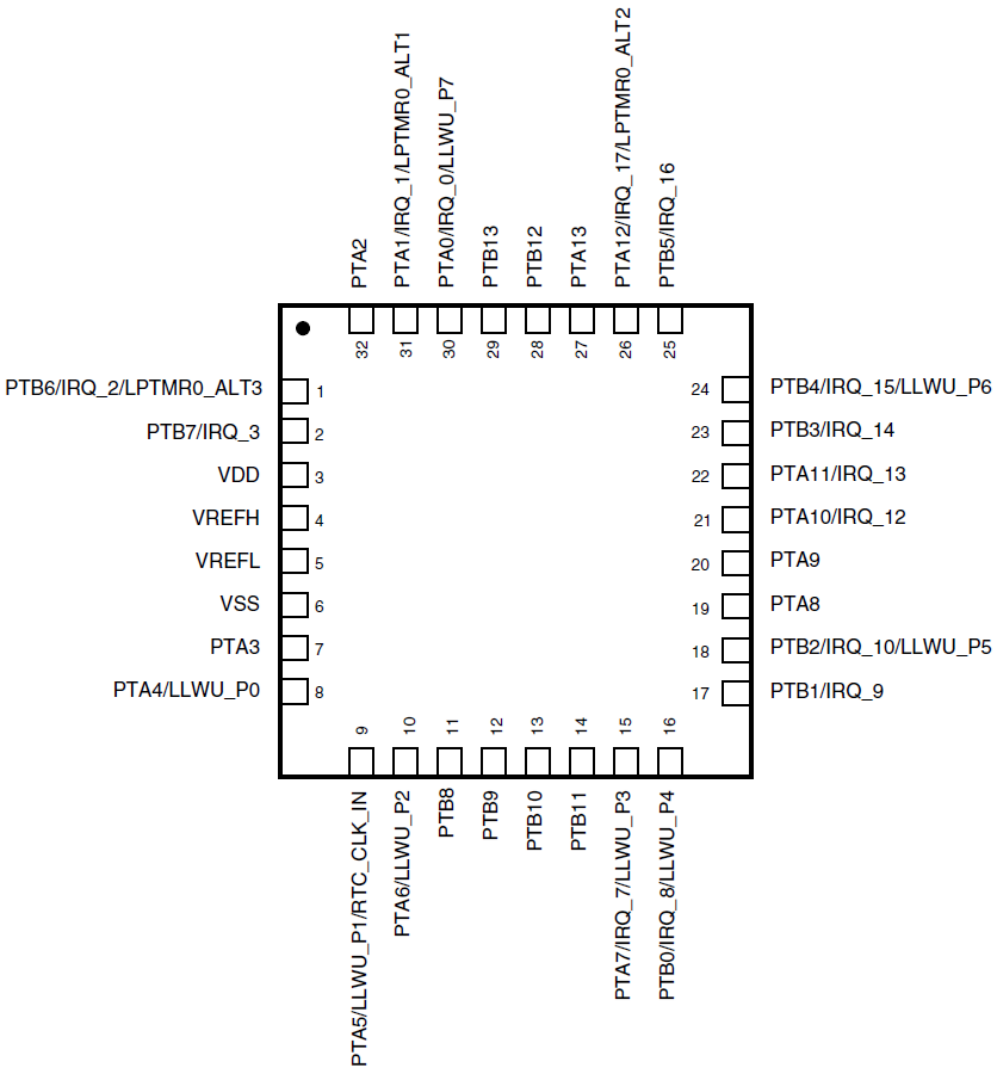
Przestrzeń pamięci	Rozmiar	Zakres
Flash	32KB	0x00000000 do 0x00007FFF
SRAM	4KB	0x1FFFFC00 do 0x1FFFFFFF
I/O	Urządzenia peryferyjne	0x40000000 do 0x400FFFFF

Mapa pamięci mikrokontrolera MKL05Z32VFM4 (32 KB FLASH memory)

Pamięć Flash o rozmiarze 32kB jest używana na kod wykonywanego programu. Do przechowywania zmiennych programu i stos wykorzystywane jest 4 kB pamięci SRAM. Układy peryferyjne WE/WY (takie jak układy czasowe, przetworniki AC/CA, układy komunikacyjne) są umieszczone w przestrzeni pamięciowej począwszy od adresu 0x40000000.

4. Układy WE/WY ogólnego stosowania (General Purpose IO: GPIO)

Podczas gdy pamięć jest wykorzystywana przez CPU do przechowywania kodu programu i zmiennych, układy peryferyjne WE/WY są wykorzystywane do komunikacji z urządzeniami zewnętrznymi. Mikrokontrolery zawsze oferują projektantowi układy ogólnego stosowania (ang. General Purpose IO: GPIO), które są wykorzystywane do komunikacji z prostymi i nie wymagającymi dużej szybkości obsługi układami zewnętrznymi takimi jak na przykład diody LED i przełączniki. W mikrokontrolerach firmy NXP z rodziny Kinetis-L porty GPIO są nazwane zgodnie z kolejnymi literami alfabetu A, B, C, itd. Każdy port ma 32 bity oznaczone kolejno PTA0-PTA31, PTB0-PTB31, itp. Trzeba podkreślić, że nie wszystkie porty są implementowane w każdym układzie rodziny Kintis-L. Również nie wszystkie bit są wyprowadzone na końcówki mikrokontrolera (tzw. piny), ponieważ nie byłoby to możliwe z uwagi na ograniczoną ich liczbę. Mikrokontroler MKL05Z32VFM4 stosowany w tym ćwiczeniu ma dwa porty GPIO: A i B.



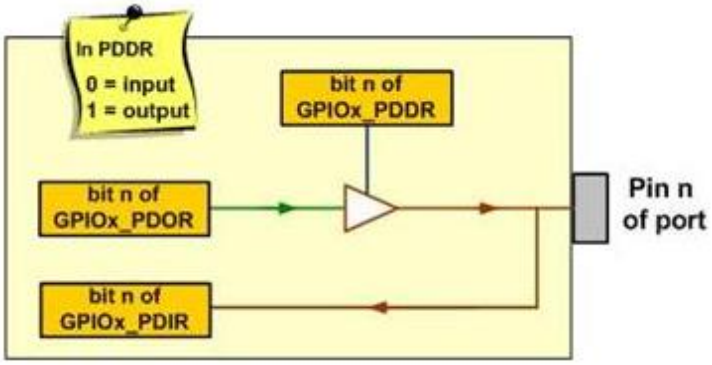
Piny układu mikrokontrolera KL05Z32VFM4 w obudowie 32 QFN

Adresy bazowe portów PTA i PTB w przestrzeni adresowej mikrokontrolera to:

GPIO Port A	0x400F F000
GPIO Port B	0x400F F040

W tabeli podano adres bazowy, który rozpoczyna przestrzeń pamięci w której kolejno umieszczonych jest wiele rejestrów kontrolujących pracę potrów GPIO. Każdy rejestr ma swój unikalny adres w przestrzeni pamięciowej.

Zasadniczo każdy mikrokontroler musi mieć minimum dwa rejestry przypisane do portu GPIO. Te rejestry to rejestr danych (Data Register) i rejestr kierunku (Direction Register). Direction Register konfiguruje port jako wejście albo wyjście. Kiedy rejestr kierunku jest już poprawnie skonfigurowany, odczyt lub zapis do Data Register powoduje odpowiednio odczyt stanów pinów portu lub odpowiednio ich wysterowanie (zgodnie z rysunkiem poniżej).



Uproszczona podstawowa struktura GPIO

Przedstawiony wcześniej Data Register jest reprezentowany na rysunku przez dwa porty: Port Data Output Register (GPIOx_PDOR) i Port Data Output Register (GPIOx_PDOR). W mikrokontrolerach Kinetis L Port Data Output Register jest umieszczony w przestrzeni adresowej modułu GPIO pod adresem o przesunięciu (offest) 0x0000 w stosunku do adresu bazowego portu.

Address: Base address + 0h offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PDO																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

GPIOx_PDOR field descriptions

Field	Description
31–0 PDO	Port Data Output Register bits for un-bonded pins return a undefined value when read. 0 Logic level 0 is driven on pin, provided pin is configured for general-purpose output. 1 Logic level 1 is driven on pin, provided pin is configured for general-purpose output.

Analogicznie, adres rejestru kierunku GPIO Direction Register (GPIOx_PDDR) jest umieszczony pod adresem o offsecie 0x0014 w stosunku do adresu bazowego portu GPIO mikrokontrolera Kinetis L.

Bit rejestru kierunku musi być ustawiony na 0, aby skonfigurować bit portu jako wejście i 1, aby odpowiedni pin portu pracował jako wyjście.

Address: Base address + 14h offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PDD																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

GPIOx_PDDR field descriptions

Field	Description
31–0 PDD	Port Data Direction Configures individual port pins for input or output. 0 Pin is configured as general-purpose input, for the GPIO function. 1 Pin is configured as general-purpose output, for the GPIO function.

4.1. Alternatywne funkcje pinów mikrokontrolera

Zależnie od swojej konfiguracji, każdy pin układu Kinetis-L może pełnić różne funkcje na potrzeby różnych układów peryferyjnych, włączając w to GPIO. Mozliwość używania jednej końcówki układu do różnych funkcji mikrokontrolera jest nazywana multipleksacją. Gdyby układ nie oferował multipleksacji musiałby mieć kilkaset pinów, aby obsłużyć wszystkie wewnętrzna układy peryferyjne. Przykładowo, wybrany pin może być użyty jako proste cyfrowe WE/WY typu GPIO, albo wejście analogowe przetwornika AC, lub wyjście interfejsu I2C. Oczywiście nie wszystkie funkcje są dostępne równocześnie, dlatego programista musi być pewny ze każdemu pinowi przypisano tylko jedna funkcje w każdym momencie pracy. Funkcja jaką pełni pin jest ustawiana przez programistę w rejestrze PCR (Pin Control Register). Rejestr PORTx_PCRn (Portx Pin Control) to rejestr specjalnego przeznaczenia w Kinetis L, który pozwala na wybór alternatywnych funkcji (alternate functions) pinów. Do każdego pinu portów A i B przypisano jeden osobny rejestr PORTx_PCRn ('x' może być literą A lub B, n jest liczbą 0-31. Jak wspomniano porty A i B mają 32 bity.

Address: Base address + 0h offset + (4d × i), where i=0d to 31d

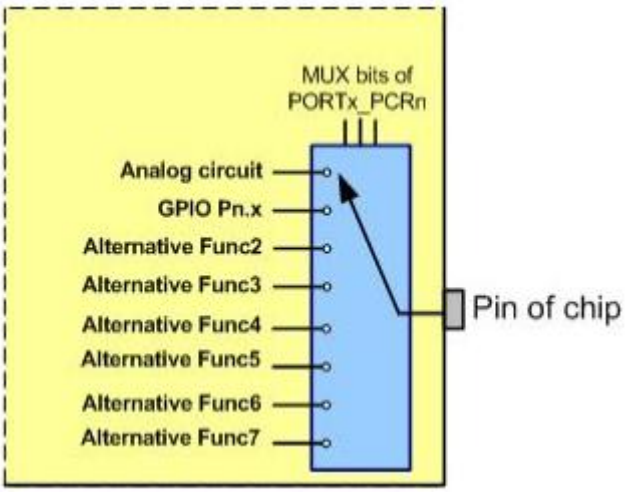
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0							ISF	0				IRQC			
W								w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0					MUX			0	DSE	0	PFE	0	SRE	PE	PS
W																
Reset	0	0	0	0	0	x*	x*	x*	0	x*	0	x*	0	x*	x*	x*

* Notes:
- x = Undefined at reset.

PORTx_PCR*n* field descriptions

W tym momencie interesuje nas 3-bitowe pole oznaczone na rysunku powyżej jako MUX. Aby skonfigurować pin jako GPIO musimy wybrać opcję MUX=0b001 (001 binarnie).



Rejestry PCR nie znajdują się w przestrzeni adresowej modułu GPIOx, ale w przestrzeni modułu PORTx Kinetis L, ale w innym obszarze pamięci:

Adres bazowy PORTA	0x4004_9000
Adres bazowy PORTB	0x4004_A000

Jak już podkreślano, każda końcówka portów A i B ma swój rejestr PORTx_PCRn. Aby otrzymać adres rejestru PORTx_PCRn odpowiedzialnego za konfigurację wybranego pinu należy obliczyć:

Adres bazowy PORTx + (4 × n), gdzie n=0d to 31

Przykładowo, adres rejestru PORTB_PCR18 to 0x4004_A000 + (18 × 4) =
0x4004_A000 + (0x12 << 2) = 0x4004_A048

Uwaga. '<<' to operator logicznego przesunięcia w lewo.

Ćwiczenie 4.1

Uzupełnij poniższą tabelę podając adresy rejestrów

Nazwa rejestru	Adres
GPIOB_PDOR	
GPIOB_PDDR	
PORTB_PCR8	
PORTB_PCR9	
PORTB_PCR10	

5. Włączanie zegara systemowego

Rejestr System Clock Gating Control Register 5 w module System Integration Moduke (SIM_SCGC5) jest używany przez Kinetis L do sterowania włączaniem i wyłączaniem sygnału zegarowego poszczególnych modułów peryferyjnych mikrokontrolera. Jeżeli moduł peryferyjny nie jest używany powinien być odłączony od sygnału zegarowego, aby umożliwić oszczędzanie energii. Zegary modułów GPIO w Kinetis L są sterowane poprzez ustawienie bitów 9 i 10 w SIM_SCGC5

Address: 4004_7000h base + 1038h offset = 4004_8038h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0												0		0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0		0			PORTB	PORTA	1		0	TSI	0			0	LPTMR
W																
Reset	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0

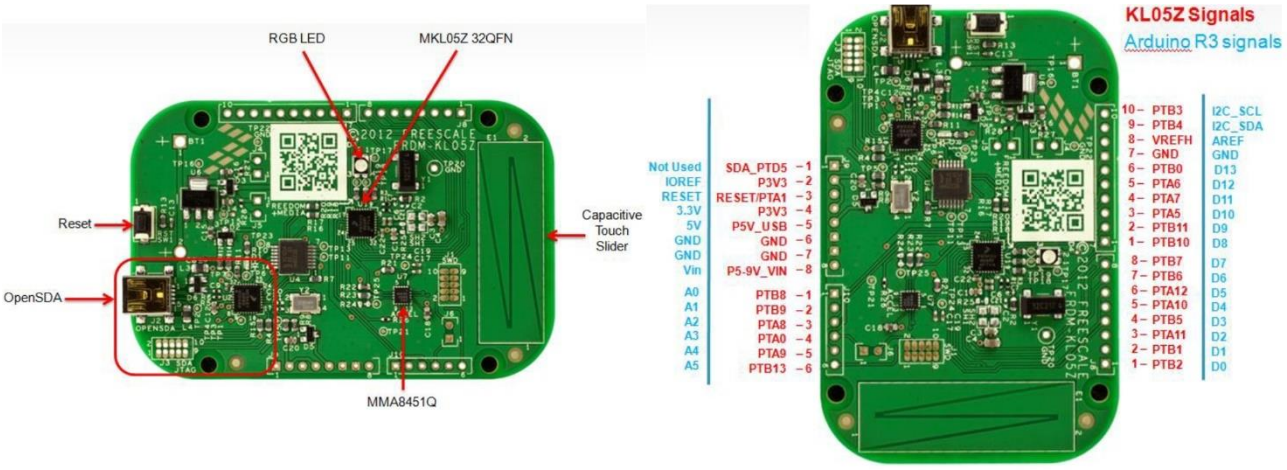
SIM_SCGC5 field descriptions

Dla przykładu aby włączyć zegar portu B (PTB), należy ustawić na 1 bit b10 rejestru SIM_SCGC5. Aby to zrobić można na przykład wykonać operację logiczną OR rejestru SIM_SCGC5 i wartości 0x400 (0b0100 0000 0000). Operacja OR zapewni, że wszystkie bity poza bitem b10 pozostaną niezmienione.

SIM_SCGC5|=(1<<10);

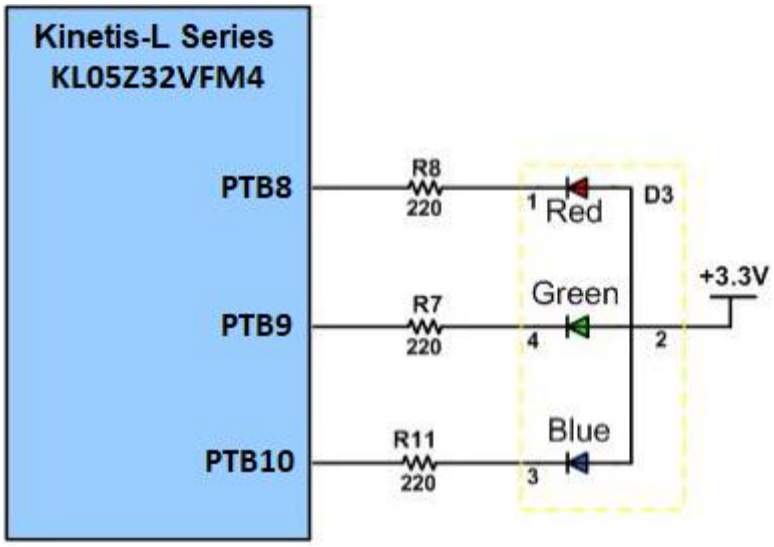
6. Zestaw uruchomieniowy FRDM-Kl05Z

Zestaw uruchomieniowy FRDM-KL05Z to tania płytka służąca do rozwoju oprogramowania dla mikrokontrolera Kinetis L MKL05Z32VFM4. Końcówki płytki KL05Z mają nazwy pinów portów GPIOx mikrokontrolera. Na przykład pierwszy pin portu A jest oznaczony jako PTA0 na złączu płytki.



6.1. Podłączenie trójkolorowej diody LED na FRDM-KL05Z

Zestaw FRDM-KL05Z wyposażony jest w trójkolorowa diodę LED podłączoną do PTB8 (czerwony), PTB9 (zielony), and PTB10 (niebieski).



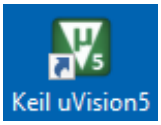
Podłączenie trójkolorowej diody LED na płytce KL05Z

7. Oprogramowanie Keil uVision5

W tym punkcie ćwiczenia w końcu możemy zacząć robić coś praktycznego. Rozpoczniemy od uruchomienia środowiska programowania mikrokontrolerów Keil uVision5, a następnie zbudujemy pierwszą aplikację na mikrokontrolerze. Środowisko uVision5 firmy Keil (ARM-Keil) umożliwia programowanie i uruchamianie aplikacji na mikrokontrolery z rodziny ARM.

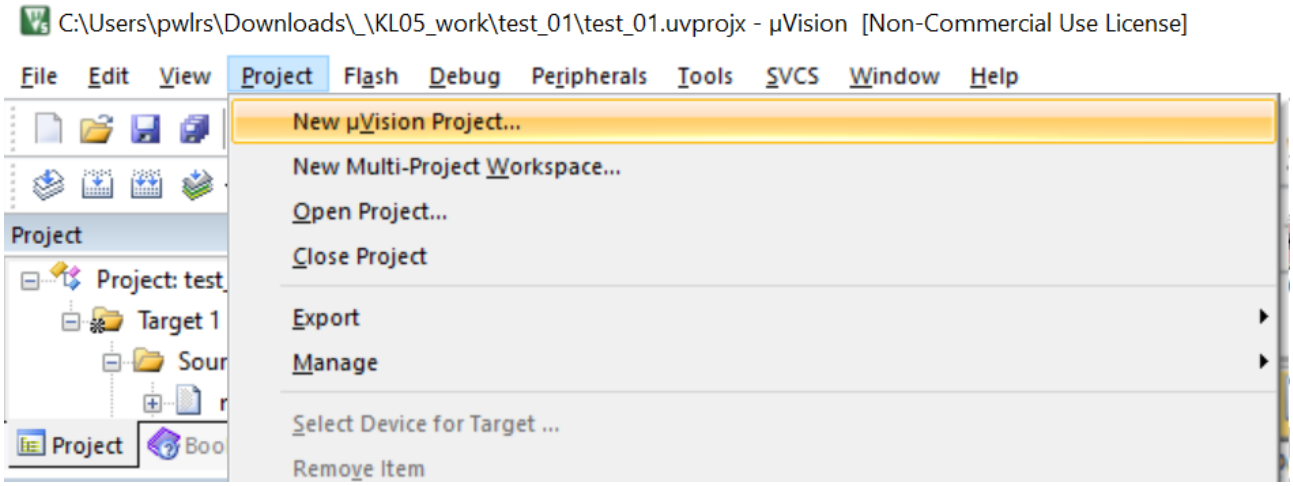
Exercise 7.1

Uruchom uVision5 i stwórz nowy projekt dla MKL05Z32VFM4.



1. Na swoim PC, kliknij ikonę uVision umieszczoną na Pulpicie albo w Menu Start:

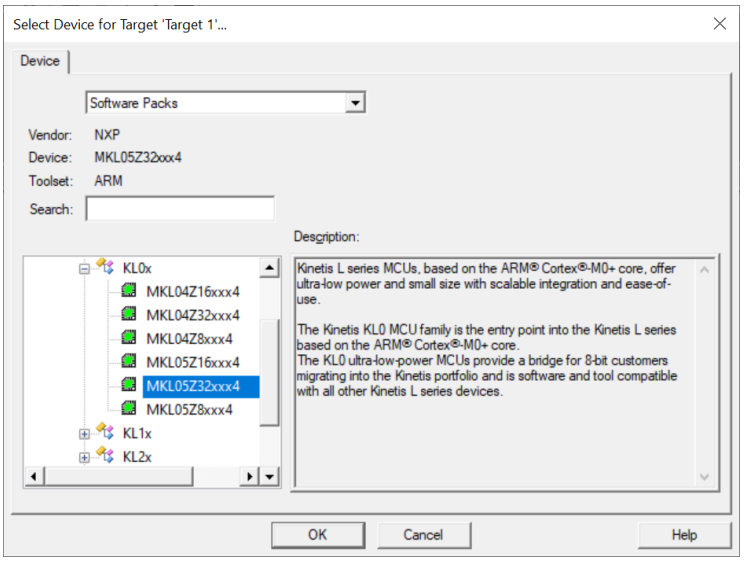
2. Z menu wybierz "Project -> New uVision Project ... "



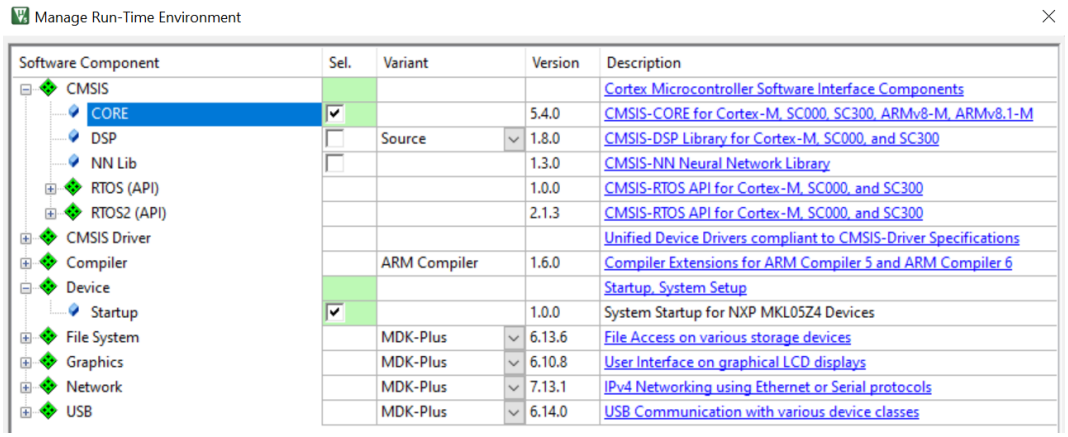
Uwaga. Z powodów praktycznych dobrze jest, aby każda grupa projektowa miała założony własny, imienny katalog, w którym będą umieszczane podkatalogi z kolejnymi projektami/ćwiczeniami.

3. Na nowy projekt stwórz katalog <nazwisko>/rgb_led. Następnie utwórz nowy projekt o nazwie **rgb_led.** Ostatecznie powinna powstać ścieżka i plik C:\MDK-ARM**my_name**\rgb_led\rgb_led.uvproj

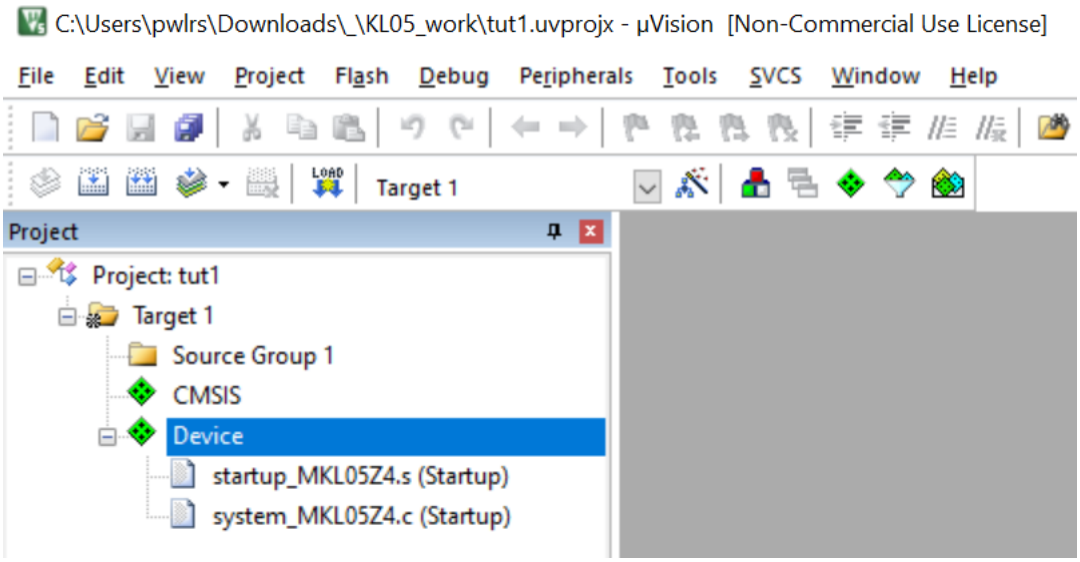
4. Z listy wybierz mikrokontroler MKL05Z32xxx4



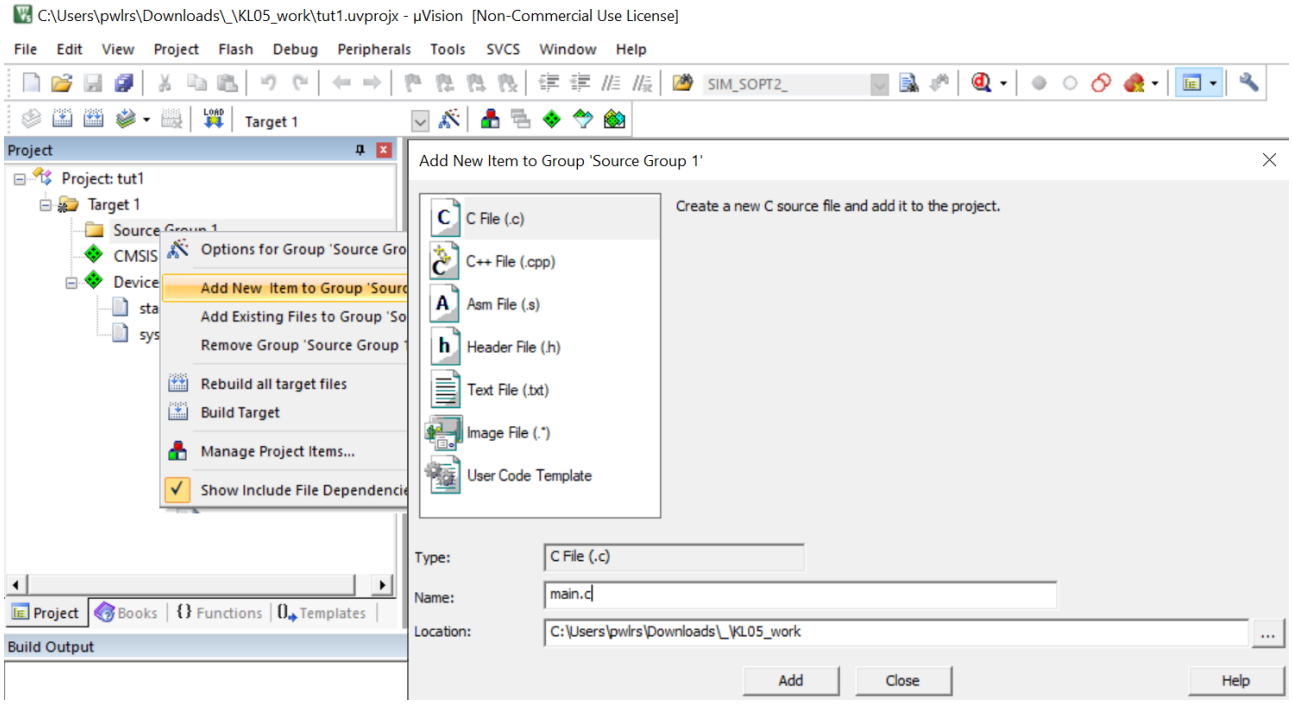
5. Następnie powinieneś wybrać biblioteki i startowe pliki systemowe potrzebne do uruchomienia aplikacji. Wybierz: **CMSIS->CORE i Device->Startup.** Kliknij OK. Powinno pojawić się główne okno uVision5.



6. Zlokalizuj pliki z kodem startowym (Startup) w oknie “Project”.



7. Kliknij prawym przyciskiem myszy na “Source Group” i wybierz “Add new item to group...”. Utwórz nowy plik źródłowy “main.c”. Kliknij “Add”.



8. Wklej kod podany poniżej do pliku „main.c”.

```
/*----------------------------------------------------------------------------
* Name:    main.c
* Purpose: RGB LED experiments for FRDM-KL05Z board
* Author:  Student
*----------------------------------------------------------------------------*/

#include "MKL05Z4.h" /*Device header*/

#define RED_LED_POS 8
#define GREEN_LED_POS 9
#define BLUE_LED_POS 10

#define SCGC5 (*((volatile uint32_t*) 0x????????))  //Fill with the proper address
#define PDOR (*((volatile uint32_t*) 0x????????))  //Fill with the proper address
#define PDDR (*((volatile uint32_t*) 0x????????))  //Fill with the proper address
#define PCR8 (*((volatile uint32_t*) 0x????????))  //Fill with the proper address

void delay_ms( int n) {
volatile int i;
volatile int j;
                for( i = 0 ; i < n; i++)
                for(j = 0; j < 3500; j++) {}
}
```

```
int main()
{
		SCGC5 |=(1<<10); /* Enable clock for GPIO B */

		PCR8 |=(1<<8);  /* MUX config. Set Pin 8 of PORT B as GPIO */

		PDDR|=(1<<RED_LED_POS); /* Set pin 8 of GPIO B as output */

		while(1){
			PDOR&=~(1<<RED_LED_POS); /* Turn on RED LED */
			delay_ms(500); /* Wait */
			PDOR|=(1<<RED_LED_POS); /* Turn off RED LED */
			delay_ms(500); /* Wait */
		}

}
```

9. W podanym kodzie uzupełnij adresy rejestrów zgodnie z ćwiczeniem 4.1.

10. Skompiluj kod, wciskając F7 lub wybierając ikonę 'Build'

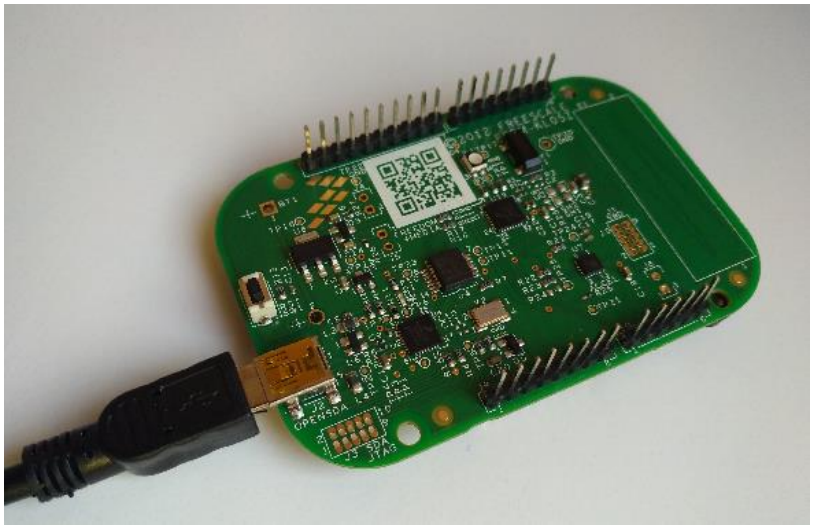
11. Upewenij, że twój kod skompilował się bez błedów. Sprawdź logi w oknie"Build Output". Powinno być:

.\rgb_led.axf" - 0 Error(s)

8. Wgranie pliku binarnego aplikacji do FRDM-KL05Z

Ćwiczenie 8.1

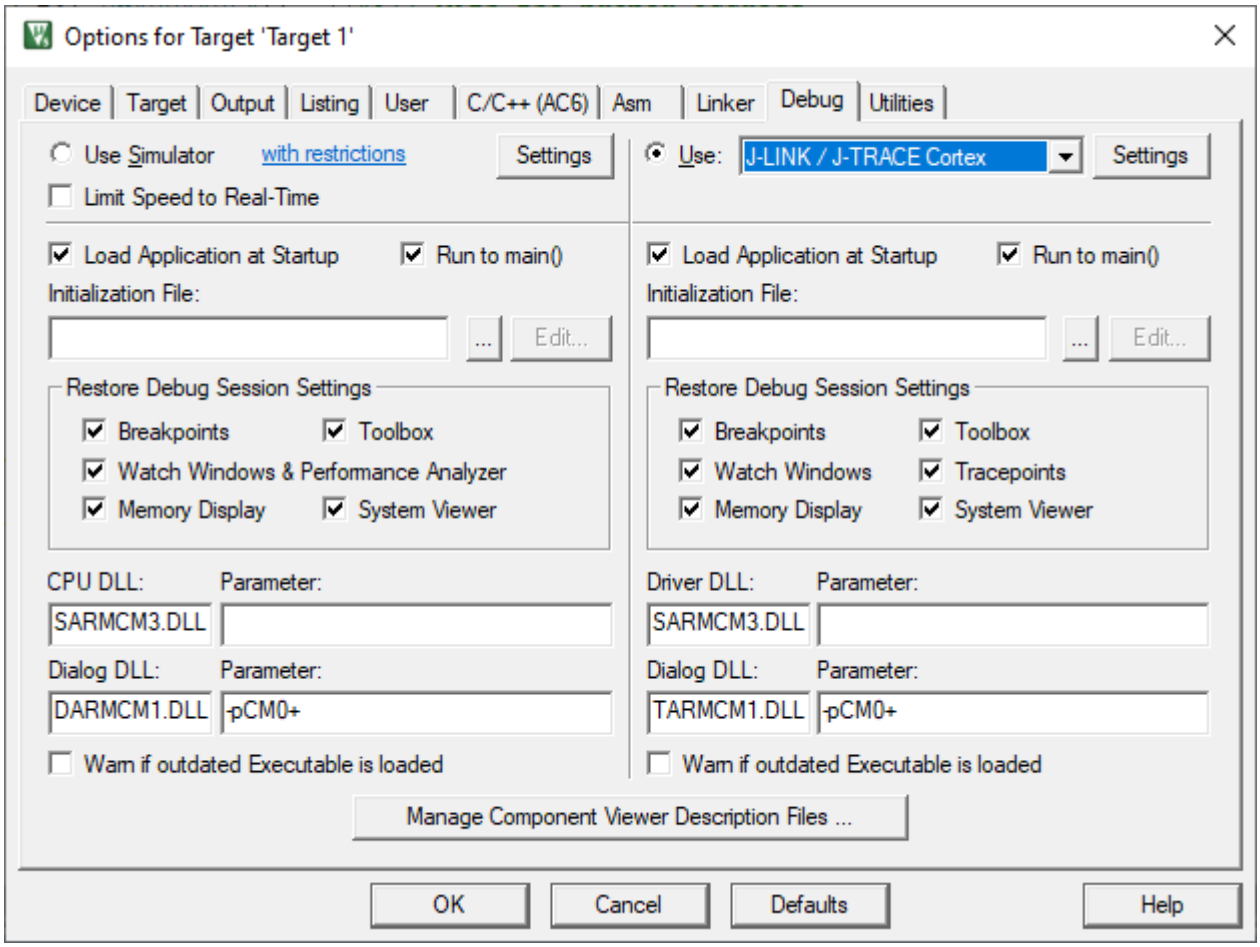
1. PodłączF RDM-KL05Z do portu USB swojego PC. Po stronie płytki FRDM-KL05Z użyj portu **OpenSDA** .



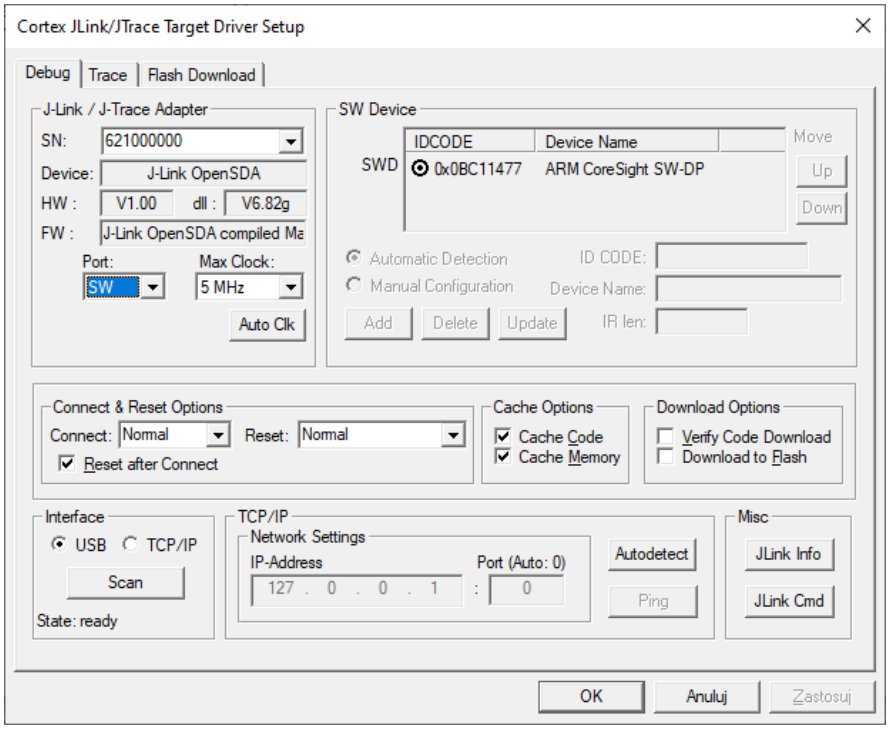
OpenSDA to nazwa jaką firma NXP (Freescale) nadała interfejsowi CMSIS-DAP firmy ARM. CMSIS-DAP to standard obsługujący interfejsy słuzace do debugowania aplikacji dla mikrokontrolerów.

2. Wybierz „Options for Target"  lub ALT-F7, a następnie wybierz zakładkę „Debug".

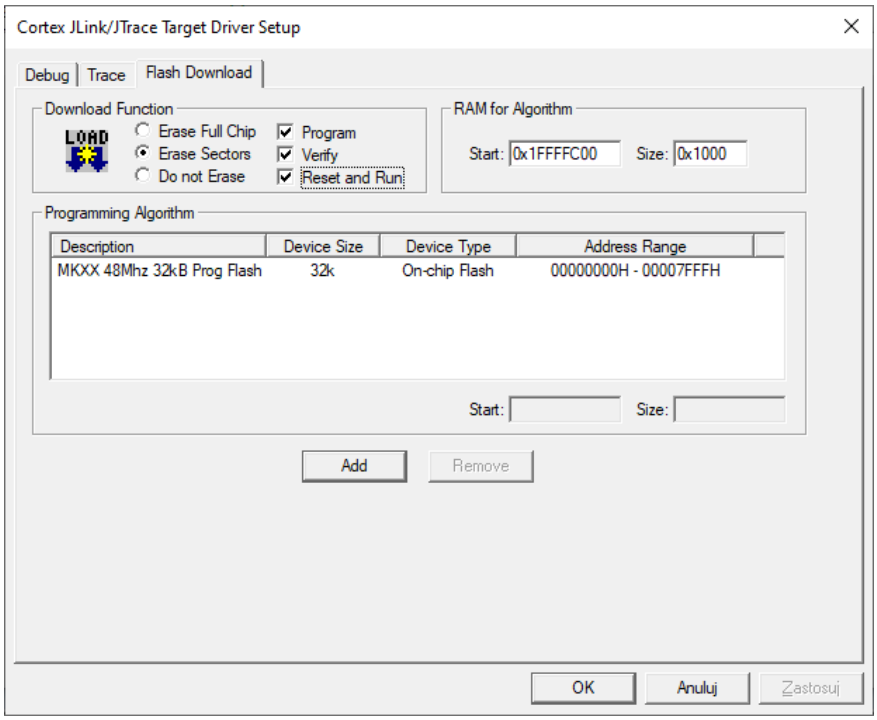
3. Z rozwijalnej listy wybierz “J-LINK/J-TRACE Cortex”.



4. Wybierz „Settings” i w następnym oknie opcję “SW” z listy „Port”.



5. Dodatkowo, wybierz opcję „Reset and Run” w zakładce „Flash Download”. Dzięki temu aplikacja automatycznie uruchomi się po wgraniu na płytkę.



6. Aby zamknąć okno “Options for target”, kolejno kliknij dwa razy OK .

7. Załadowanie zbudowanego pliku binarnego do mikrokontrolera odbywa się po wciśnięciu ikony „Download”: 

8. Zaobserwuj sposób migania diody na twojej płytce.

9. Plik nagłówkowy „MKL05Z4.h"

Jak do tej pory programowanie mikrokontrolerów może wydawać Ci się bardzo żmudne. Wyszukiwanie adresu rejestrów sterujących pracą peryferiów jest pracochłonne i nużące. Na szczęście w normalnej praktyce nie będziesz tego musiał robić samodzielnie, ponieważ wszystkie potrzebne adresy rejestrów są już zdefiniowane w pliku nagłówkowym „MKL05Z4.h”. Jedyne czego potrzebujesz, to nauczyć się jak determinować jak nazywa się definicja odpowiadająca potrzebnemu Ci rejestrowi. Popatrz na kod poniżej i zobacz jak wcześniej uruchomiony kod aplikacji „rgb_led” wygląda w przypadku użycia definicji z nagłówka „MKL05Z4.h”.

```
/*----------------------------------------------------------------------------
 * Name:    main.c
 * Purpose: RGB LED experiments for FRDM-KL05Z board
 * Author:  Student
 *----------------------------------------------------------------------------*/

 #include "MKL05Z4.h" /*Device header*/
```

```
#define RED_LED_POS 8
#define GREEN_LED_POS 9
#define BLUE_LED_POS 10

void delay_ms( int n) {
volatile int i;
volatile int j;
        for( i = 0 ; i < n; i++)
        for(j = 0; j < 3500; j++) {}
}

int main()
{
        SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK; /* Enable clock for GPIO B */
        PORTB->PCR[8] |= PORT_PCR_MUX(1);  /* MUX config. Set Pin 8 of PORT B as GPIO */

        PTB->PDDR |= (1<<RED_LED_POS); /* Set pin 8 of GPIO B as output */

        while(1){
                PTB->PDOR&=~(1<<RED_LED_POS); /* Turn on RED LED */
                delay_ms(500); /* Wait */
                PTB->PDOR|=(1<<RED_LED_POS); /* Turn off RED LED */
                delay_ms(500); /* Wait */
        }
}
```

Jak zapewne zauważyłeś, aby lepiej zorganizować dostęp do rejestrów mikrokontrolera, MKL05Z4 wykorzystuje mechanizm wskaźników, struktur i zestawu makr. W kodzie użyto wskaźników do trzech struktur: SIM, POTRTB i PTB, które odpowiadają odpowiednio trzem modułom mikrokontrolera (SIM, PORT i GPIO) . Zdefiniowane struktury zawierają wszystkie rejestry modułu do których programista dostaje się korzystając z symbolu '→' (dostępu do struktury podanej jako wskaźnik).

Na przykład SIM->SCGC5 użyto do zaadresowania rejestru SCGC5 w module SIM (System Integration Module). Podobnie PTB->PDDR użyto aby uzyskać dostęp do rejestru PDDR w module GPIOB. itp, itd.

Maska SIM_SCGC5_PORTB_MASK pozwala na ustawienie bitu PORTB w rejestrze SCGC5 modułu SIM. W MKL05Z4.h została ona zdefiowana jako:

```
#define         SIM_SCGC5_PORTB_MASK            0x400u
```

Jak możesz zauważyć, faktycznie maska zawiera aktywny jedynie bit 10, który odpowiada za zegar modułu port B.

Zauważ również, że rejestry PORTx_PCRn tworzą tablicę w strukturze PORTx (w naszym wypadku PORTB). Na przykład PORTB->PCR[8] to wskaźnik na rejestr PORTB_PCR8.

Dodatkowo w kodzie wykorzystano makro PORT_PCR_MUX(x), które ustawia bity pola MUX w rejestrze PCR - odpowiedzialne za sterowanie multiplekserem w GPIO.

Ćwiczenie 9.1

Jeżeli jesteś dużym entuzjastą języka C, to ciekawym doświadczeniem będzie dokładniejsze przestudiowanie w jaki sposób zostały zdefiniowane adresy rejestrów w pliku „MKL05Z4.h”.

Ćwiczenie 9.2

Skompiluj i uruchom nową wersję kodu dla aplikacji rgb_led.

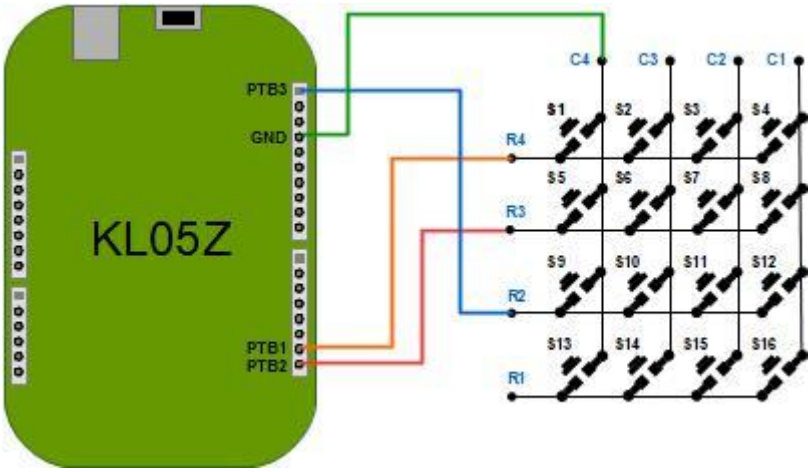
Następnie zmodyfikuj kod w taki sposób, aby zaprogramować miganie diody według sekwencji swojego pomysłu. **Użyj wszystkich trzech kolorów diody.**

10. Wykorzystanie GPIO jako wejścia

W celu przetestowanie modułów GPIO w funkcji wejścia wykorzystamy zewnętrzną klawiaturę matrycową, której przyciski posłużą do podawania stanów logicznych na wejścia mikrokontroera.

Ćwiczenie 10.1

Użyj przewodów do połączenia moduł FRDM KL05Z z klawiaturą matrycową zgodnie ze schematem.



Ćwiczenie 10.2

Skompiluj i uruchom kod dla aplikacji rgb_button zamieszczony poniżej. Wciskanie przycisku S1 powinno na przemian włączać i wyłączać diodę czerwoną.

Następnie zmodyfikuj kod w taki sposób, aby przyciski S1, S2 i S3 włączały i wyłączały odpowiednio diodę czerwoną, zieloną i niebieską.

```
/*----------------------------------------------------------------------------
 * Name:    main.c
 * Purpose: RGB & BUTTON experiments for FRDM-KL05Z board
 * Author:  Student
 *----------------------------------------------------------------------------*/

 #include "MKL05Z4.h" /*Device header*/
```

```
#define RED_LED_POS 8
#define GREEN_LED_POS 9
#define BLUE_LED_POS 10

#define BUTTON_1_POS 1
#define BUTTON_2_POS 2
#define BUTTON_3_POS 3

void delay_ms( int n) {
volatile int i;
volatile int j;
        for( i = 0 ; i < n; i++)
        for(j = 0; j < 3500; j++) {}
}

int main()
{
        SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK; /* Enable clock for GPIO B */
        PORTB->PCR[RED_LED_POS] |= PORT_PCR_MUX(1);  /* Set Pin 8 MUX as GPIO */

        PORTB->PCR[BUTTON_1_POS] |= PORT_PCR_MUX(1);  /* Set Pin 1 MUX as GPIO */
        // Enable pull-up resitor on Pin 1
        PORTB->PCR[BUTTON_1_POS] |= PORT_PCR_PE_MASK | PORT_PCR_PS_MASK;

        PTB->PDDR |= (1<<RED_LED_POS); /* Set RED LED pin as output */
        PTB->PSOR |= (1<<RED_LED_POS); /* Turn off RED LED */

        while(1){
                if( ( PTB->PDIR & (1<<BUTTON_1_POS) ) ==0 ){ /* Test if button pressed */
                        PTB->PTOR|=(1<<RED_LED_POS); /* Toggle RED LED */
                        while( ( PTB->PDIR & (1<<BUTTON_1_POS) ) == 0 ) /* Wait for release */
                                delay_ms(100); /* Debouncing */
                }
        }
}
```

Ćwiczenie 10.3

Zmodyfikuj powyższy kod w taki sposób, aby trzy przyciski S1, S2 i S3 włączały i wyłączały odpowiednio diodę czerwoną, zieloną i niebieską.

Zaproponuj kod który będzie oferował minimalny rozmiar kodu maszynowego programu. Rozmiar kod programu możesz odczytać z raportu linkera jako blok progamu Code.