Akademia Górniczo-Hutnicza
w Krakowie
Instytut Elektroniki
WIET

Laboratorium
Technika Mikroprocesorowa 2

Ćwiczenie 2

Interfejs użytkownika

Autor: Mariusz Sokołowski

wer. 28.09.2021

1. WSTĘP

1.1. CEL

Celem ćwiczenia jest:

- zapoznanie się z podstawowymi urządzeniami interfejsu użytkownika – podłączanie, konfiguracja, programowanie, użytkowanie:
  - alfanumeryczny wyświetlacz ciekłokrystaliczny (LCD),
  - panel dotykowy,
  - klawiatura.
- konfiguracja portów we/wy, do współpracy z diodami LED – port wyjściowy,
- konfiguracja portów we/wy, do współpracy z przyciskami – port wejściowy
- uruchomienie przykładowego programu, pokazującego współpracę urządzeń wejściowych i wyjściowych ze sobą.

1.2. WYMAGANIA

Sprzętowe:

- komputer klasy PC, spełniający wymagania sprzętowe aplikacji KEIL v5,
- zestaw FRDMKL05Z

Programowe:

- system operacyjny Windows 7 lub wyższy (wszystkie instrukcje powstały w oparciu o Windows 7 Pro x64),
- środowisko Keil / uVision 5 MDK-ARM

Doświadczenie:

- podstawowa umiejętność obsługi komputera klasy PC,
- podstawowa znajomość systemów operacyjnych rodziny Windows,

Literatura:

- KL05 Sub-Family Reference Manual, Freescale Semiconductor
- Kinetis L Peripheral Module Quick Reference, Freescale Semiconductor
- KL05 Sub-Family Reference Manual, Freescale Semiconductor
- Joseph Yiu, The Definitive Guide to the ARM Cortex-M0, Elsevier, 2011
- PCF8574 - Remote 8-bit I/O expander for I2C-bus, Data Sheet, PHILIPS
- HD44780U (LCD-II) – Dot Matrix Liquid Crystal Display Controller/Driver, Data Sheet, Hitachi

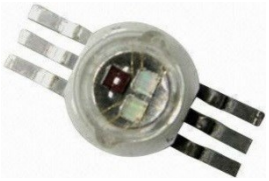
Interfejs użytkownika powinien pozwalać na „dialog” człowieka z maszyną. Urządzenia, które pomagają maszynie komunikować się z człowiekiem to układy **wyjściowe**, oddziałowujące przede wszystkim na nasze zmysły wzroku, słuchu, dotyku i temperatury. Jak maszyna się pali, to również na nasz zmysł powonienia. Przykładem może być: monitor ekranowy, wyświetlacz LCD, wyświetlacz i diody LED, wskaźniki elektromechaniczne, głośnik, element wibracyjny, grzałka. Natomiast urządzenia, które pomagają nam w przekazywaniu informacji maszynie to układy **wejściowe**, które po odpowiednim przetworzeniu informacji na cyfrowy sygnał elektryczny (napięciowy), pozwalają człowiekowi wprowadzać dane, komendy czy decyzje. Przykładem może być: klawiatura (pojedyncze przyciski), panel dotykowy, tablet graficzny, myszka, przetwornik obrazu (kamera), mikrofon.

2. URZĄDZENIA WYJŚCIOWE

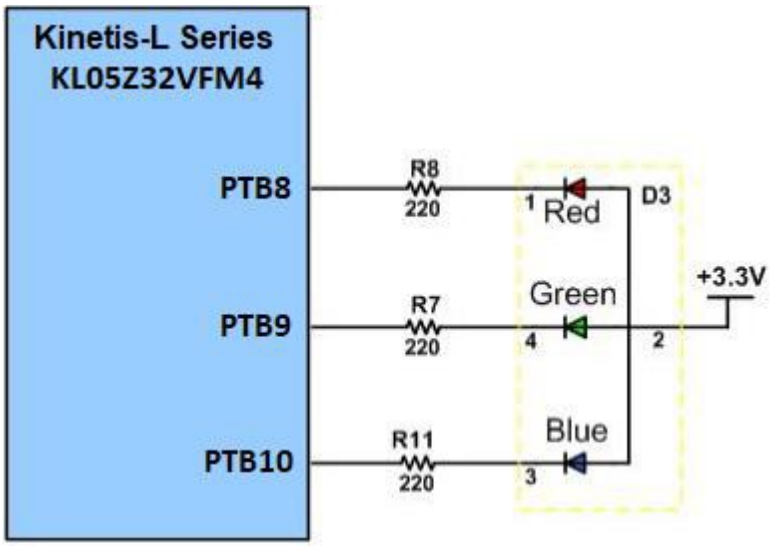
W tym ćwiczeniu poznamy dwa urządzenia wyjściowe, obydwa wizualne: diody świecące (LED) i alfanumeryczny wyświetlacz LCD (podobny jak w telefonie).

2.1. DIODY LED

Na płytce KL05Z jest umieszczona trójkolorowa dioda LED (Rys. 14). Jest to pojedynczy element, zawierający trzy, niezależne diody: czerwoną (R), zieloną (G) i niebieską (B) – Rys. 1.

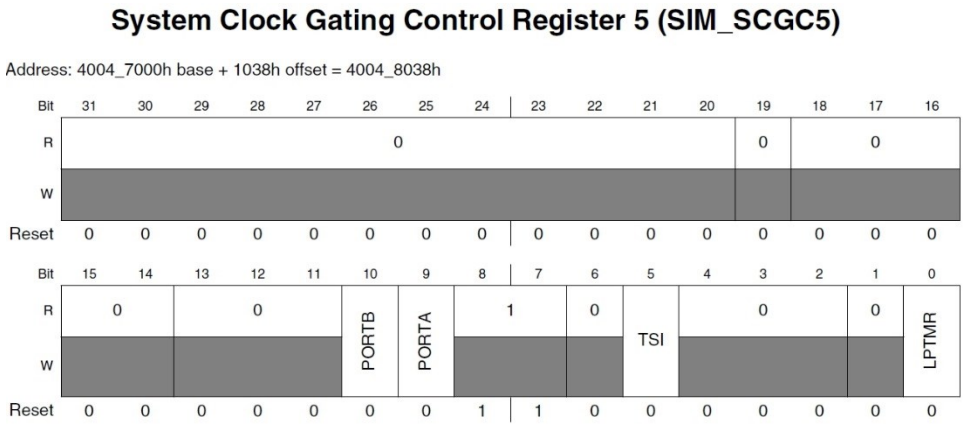


Rys. 1 Budowa diody LED RGB



Rys. 2 Schemat podłączenia diody LED RGB do mikrokontrolera KL05Z32VFM4

Katody, za pośrednictwem rezystorów, zabezpieczających przed przeciążeniem diod, są podłączone do kolejnych pinów (końcówek) portu B (GPIO). Jak wiemy, wszystkie urządzenia we/wy, zawarte w mikrokontrolerze, są domyślnie wyłączone. Dlatego, aby skorzystać z portu B należy go najpierw włączyć. Ponieważ wszystkim zarządzają rejestry, które dzielą się na: konfiguracyjne, statusowe (stanu) i danych, więc należy odwołać się do odpowiedniego rejestru konfiguracyjnego. Jest nim rejestr SCGC5, w bloku SIM (ang. **S**ystem **I**ntegration **M**odule). Należy ustawić wartość „1” w polu PORTB.



Rys. 3 Mapa rejestru SCGC5

W bibliotece MKL05Z4.h, blokiem SIM rządzi struktura typu SIM_Type. Jest to zbiór zmiennych całkowitych, o kwalifikatorze ***volatile***.

```
/* ----------------------------------------------------------------------------
   -- SIM Peripheral Access Layer
   ---------------------------------------------------------------------------- */

/*!
 * @addtogroup SIM_Peripheral_Access_Layer SIM Peripheral Access Layer
 * @{
 */

/** SIM - Register Layout Typedef */
typedef struct {
  __IO uint32_t SOPT1;                             /**< System Options Register 1, offset: 0x0 */
  __I  uint32_t SOPT1CFG;                          /**< SOPT1 Configuration Register, offset: 0x4 */
       uint8_t RESERVED_0[4092];
  __IO uint32_t SOPT2;                             /**< System Options Register 2, offset: 0x1004 */
       uint8_t RESERVED_1[4];
  __IO uint32_t SOPT4;                             /**< System Options Register 4, offset: 0x100C */
  __IO uint32_t SOPT5;                             /**< System Options Register 5, offset: 0x1010 */
       uint8_t RESERVED_2[4];
  __IO uint32_t SOPT7;                             /**< System Options Register 7, offset: 0x1018 */
       uint8_t RESERVED_3[8];
  __I  uint32_t SDID;                              /**< System Device Identification Register, offset: 0x1024 */
       uint8_t RESERVED_4[12];
  __IO uint32_t SCGC4;                             /**< System Clock Gating Control Register 4, offset: 0x1034 */
  __IO uint32_t SCGC5;                             /**< System Clock Gating Control Register 5, offset: 0x1038 */
  __IO uint32_t SCGC6;                             /**< System Clock Gating Control Register 6, offset: 0x103C */
  __IO uint32_t SCGC7;                             /**< System Clock Gating Control Register 7, offset: 0x1040 */
  __IO uint32_t CLKDIV1;                           /**< System Clock Divider Register 1, offset: 0x1044 */
       uint8_t RESERVED_5[4];
  __IO uint32_t FCFG1;                             /**< Flash Configuration Register 1, offset: 0x104C */
  __I  uint32_t FCFG2;                             /**< Flash Configuration Register 2, offset: 0x1050 */
       uint8_t RESERVED_6[4];
  __I  uint32_t UIDMH;                             /**< Unique Identification Register Mid-High, offset: 0x1058 */
  __I  uint32_t UIDML;                             /**< Unique Identification Register Mid Low, offset: 0x105C */
  __I  uint32_t UIDL;                              /**< Unique Identification Register Low, offset: 0x1060 */
       uint8_t RESERVED_7[156];
  __IO uint32_t COPC;                              /**< COP Control Register, offset: 0x1100 */
  __O  uint32_t SRVCOP;                            /**< Service COP Register, offset: 0x1104 */
} SIM_Type;
```

Rys. 4 Definicja typu strukturowego SIM_Type, w bibliotece MKL05Z4.h

```
/* IO definitions (access restrictions to peripheral registers) */
/**
    \defgroup CMSIS_glob_defs CMSIS Global Defines

    <strong>IO Type Qualifiers</strong> are used
    \li to specify the access to peripheral variables.
    \li for automatic generation of peripheral register debug information.
*/
#ifdef __cplusplus
  #define   __I     volatile             /*!< Defines 'read only' permissions */
#else
  #define   __I     volatile const       /*!< Defines 'read only' permissions */
#endif
#define     __O     volatile             /*!< Defines 'write only' permissions */
#define     __IO    volatile             /*!< Defines 'read / write' permissions */

/* following defines should be used for structure members */
#define     __IM     volatile const      /*! Defines 'read only' structure member permissions */
#define     __OM     volatile            /*! Defines 'write only' structure member permissions */
#define     __IOM    volatile            /*! Defines 'read / write' structure member permissions */

/*@} end of group Cortex-M0+ */
```

Rys. 5 Definicja kwalifikatorów ***volatile*** w bibliotece core_cm0plus.h

```
/* SIM - Peripheral instance base addresses */
/** Peripheral SIM base address */
#define SIM_BASE                                 (0x40047000u)
/** Peripheral SIM base pointer */
#define SIM                                      ((SIM_Type *)SIM_BASE)
/** Array initializer of SIM peripheral base pointers */
#define SIM_BASES                                { SIM }
```

Rys. 6 Deklaracja struktury SIM, w bibliotece MKL05Z4.h

Proszę zauważyć, że struktura SIM jest zadeklarowana jako wskaźnik, więc odwoływanie się do jej elementów będzie następujące:

SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK; // Włącz port B

W bibliotece MKL05Z4.h, w sekcji „SIM Register Masks", są zdefiniowane różne pomocne maski i makra, pozwalające na bardziej przejrzystą budową programu. Przykładowa maska bitu PORTB, rejestru SCGC5, posiada wartość „1" na pozycji 10, więc powyższa instrukcja ustawi właśnie wartość „1" w pozycji PORTB.

Następnym etapem jest wybór funkcji, którą ma spełniać pin, sterujący odpowiednią diodą. Dokonuje się tego w rejestrze PCRn, bloku PORTB, gdzie „n" to numer pinu, którego dotyczy rejestr.

Pin Control Register n (PORTx_PCR*n*)

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	*	*	*	0	*	0	*	0	*	*	*

Rys. 7 Mapa rejestru PCRn

W bibliotece MKL05Z4.h, blokiem PORTx rządzi struktura typu PORT_Type. Jest to zbiór zmiennych całkowitych, o kwalifikatorze ***volatile***.

```
/* ----------------------------------------------------------------------------
   -- PORT Peripheral Access Layer
   ---------------------------------------------------------------------------- */

/*!
 * @addtogroup PORT_Peripheral_Access_Layer PORT Peripheral Access Layer
 * @{
 */

/** PORT - Register Layout Typedef */
typedef struct {
  __IO uint32_t PCR[32];                           /**< Pin Control Register n, array offset: 0x0, array step: 0x4 */
  __O  uint32_t GPCLR;                             /**< Global Pin Control Low Register, offset: 0x80 */
  __O  uint32_t GPCHR;                             /**< Global Pin Control High Register, offset: 0x84 */
       uint8_t RESERVED_0[24];
  __IO uint32_t ISFR;                              /**< Interrupt Status Flag Register, offset: 0xA0 */
} PORT_Type;
```

Rys. 8 Definicja typu strukturowego PORT_Type, w bibliotece MKL05Z4.h

```
/* PORT - Peripheral instance base addresses */
/** Peripheral PORTA base address */
#define PORTA_BASE                               (0x40049000u)
/** Peripheral PORTA base pointer */
#define PORTA                                    ((PORT_Type *)PORTA_BASE)
/** Peripheral PORTB base address */
#define PORTB_BASE                               (0x4004A000u)
/** Peripheral PORTB base pointer */
#define PORTB                                    ((PORT_Type *)PORTB_BASE)
/** Array initializer of PORT peripheral base pointers */
#define PORT_BASES                               { PORTA, PORTB }
```

Rys. 9 Deklaracja struktur PORTA i PORTB, w bibliotece MKL05Z4.h

Szukamy w wierszach tabeli (Tab. 1) pinów PTB8, PTB9 i PTB10, a w kolumnie szukamy identycznie nazywającej się funkcji (kolumna ALT1).

KL05Z (32QFx)	Pin Name	On Board	IO header	DEFAULT	ALT0	ALT1	ALT2	ALT3
1	PTB6/IRQ_2/LPTMR0_ALT3		D6	DISABLED		PTB6/IRQ_2/LPTMR0_ALT3	TPM0_CH3	TPM_CLKIN1
2	PTB7/IRQ_3		D7	DISABLED		PTB7/IRQ_3	TPM0_CH2	
3	VDD	VDD	3V3	VDD	VDD			
4	VREFH	VREFH	AREF	VREFH	VREFH			
5	VREFL	VREFL	—	VREFL	VREFL			
6	VSS	VSS	GND	VSS	VSS			
7	PTA3	32 KHz XTAL	—	EXTAL0	EXTAL0	PTA3	I2C0_SCL	I2C0_SDA
8	PTA4/LLWU_P0	32 KHz XTAL	—	XTAL0	XTAL0	PTA4/LLWU_P0	I2C0_SDA	I2C0_SCL
9	PTA5/LLWU_P1/RTC_CLK_IN		D10	DISABLED		PTA5/LLWU_P1/RTC_CLK_IN	TPM0_CH5	SPI0_SS_b
10	PTA6/LLWU_P2		D12	DISABLED		PTA6/LLWU_P2	TPM0_CH4	SPI0_MISO
11	PTB8	Red LED	A0	ADC0_SE11	ADC0_SE11	PTB8	TPM0_CH3	
12	PTB9	Green LED	A1	ADC0_SE10	ADC0_SE10	PTB9	TPM0_CH2	
13	PTB10	Blue LED	D8	ADC0_SE9/TSI0_IN7	ADC0_SE9/TSI0_IN7	PTB10	TPM0_CH1	
14	PTB11		D9	ADC0_SE8/TSI0_IN6	ADC0_SE8/TSI0_IN6	PTB11	TPM0_CH0	
15	PTA7/IRQ_7/LLWU_P3		D11	ADC0_SE7/TSI0_IN5	ADC0_SE7/TSI0_IN5	PTA7/IRQ_7/LLWU_P3	SPI0_MISO	SPI0_MOSI
16	PTB0/IRQ_8/LLWU_P4		D13	ADC0_SE6/TSI0_IN4	ADC0_SE6/TSI0_IN4	PTB0/IRQ_8/LLWU_P4	EXTRG_IN	SPI0_SCK
17	PTB1/IRQ_9		D1	ADC0_SE5/TSI0_IN3/DAC0_OUT/CMP0_IN3	ADC0_SE5/TSI0_IN3/DAC0_OUT/CMP0_IN3	PTB1/IRQ_9	UART0_TX	UART0_RX
18	PTB2/IRQ_10/LLWU_P5		D0	ADC0_SE4/TSI0_IN2	ADC0_SE4/TSI0_IN2	PTB2/IRQ_10/LLWU_P5	UART0_RX	UART0_TX
19	PTA8		A2	ADC0_SE3/TSI0_IN1	ADC0_SE3/TSI0_IN1	PTA8		
20	PTA9		A4	ADC0_SE2/TSI0_IN0	ADC0_SE2/TSI0_IN0	PTA9		
21	PTA10/IRQ_12	Accel INT2	D4	DISABLED	TSI0_IN11	PTA10/IRQ_12		
22	PTA11/IRQ_13		D2	DISABLED	TSI0_IN10	PTA11/IRQ_13		
23	PTB3/IRQ_14	Accel I2C	D15	DISABLED		PTB3/IRQ_14	I2C0_SCL	UART0_TX
24	PTB4/IRQ_15/LLWU_P6	Accel I2C	D14	DISABLED		PTB4/IRQ_15/LLWU_P6	I2C0_SDA	UART0_RX
25	● PTB5/IRQ_16		D3	● NMI_b	ADC0_SE1/CMP0_IN1	PTB5/IRQ_16	TPM1_CH1	NMI_b
26	PTA12/IRQ_17		D5	ADC0_SE0/CMP0_IN0	ADC0_SE0/CMP0_IN0	PTA12/IRQ_17/LPTMR0_ALT2	TPM1_CH0	TPM_CLKIN0
27	PTA13	Touch Slider	—	TSI0_IN9	TSI0_IN9	PTA13		
28	PTB12	Touch Slider	—	TSI0_IN8	TSI0_IN8	PTB12		
29	PTB13		A5	ADC0_SE13	ADC0_SE13	PTB13	TPM1_CH1	RTC_CLKOUT
30	PTA0/IRQ_0/LLWU_P7	SWD_CLK	A3	SWD_CLK	ADC0_SE12/CMP0_IN2	PTA0/IRQ_0/LLWU_P7	TPM1_CH0	SWD_CLK
31	PTA1/IRQ_1/LPTMR0_ALT1	RESET	RESET	RESET_b		PTA1/IRQ_1/LPTMR0_ALT1	TPM_CLKIN0	RESET_b
32	PTA2	SWD_DIO	—	SWD_DIO		PTA2	CMP0_OUT	SWD_DIO

Tab. 1 Tabela ze skróconym opisem funkcji pinów

W pole MUX, rejestru PCRn, wpisujemy cyfrę stojącą przy skrócie ALT (u nas 1). Struktura PORTB jest zadeklarowana jako wskaźnik, więc odwoływanie się do jej elementów będzie następujące:

PORTB->PCR[8] |= PORT_PCR_MUX(1);

PORTB->PCR[9] |= PORT_PCR_MUX(1);

PORTB->PCR[10] |= PORT_PCR_MUX(1);

Korzystamy z makra, zdefiniowanego w bibliotece MKL05Z4.h, w sekcji „PORT Register Masks".

Kolej na rolę, jaką ma odgrywać każdy pin. Mają być wyjściem. Tu zadanie swoje wypełnia rejestr PDDR, w bloku GPIO (ang. **G**eneral-**P**urpose **I**nput/**O**utput).

Port Data Direction Register (GPIOx_PDDR)

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

Rys. 10 Mapa rejestru PDDR

Każdy bit tego rejestru rządzi odpowiednim bitem (pinem) portu. Aby pin pracował jako cyfrowe wyjście, należy w odpowiadającej mu pozycji rejestru PDDR wpisać „1".

W bibliotece MKL05Z4.h, blokiem GPIO rządzi struktura typu GPIO_Type. Jest to zbiór zmiennych całkowitych, o kwalifikatorze ***volatile.***

```
/* ----------------------------------------------------------------------------
   -- GPIO Peripheral Access Layer
   ---------------------------------------------------------------------------- */

/*!
 * @addtogroup GPIO_Peripheral_Access_Layer GPIO Peripheral Access Layer
 * @{
 */

/** GPIO - Register Layout Typedef */
typedef struct {
  __IO uint32_t PDOR;                              /**< Port Data Output Register, offset: 0x0 */
  __O  uint32_t PSOR;                              /**< Port Set Output Register, offset: 0x4 */
  __O  uint32_t PCOR;                              /**< Port Clear Output Register, offset: 0x8 */
  __O  uint32_t PTOR;                              /**< Port Toggle Output Register, offset: 0xC */
  __I  uint32_t PDIR;                              /**< Port Data Input Register, offset: 0x10 */
  __IO uint32_t PDDR;                              /**< Port Data Direction Register, offset: 0x14 */
} GPIO_Type;
```

Rys. 11 Definicja typu strukturowego GPIO_Type, w bibliotece MKL05Z4.h

```
/* GPIO - Peripheral instance base addresses */
/** Peripheral PTA base address */
#define PTA_BASE                                 (0x400FF000u)
/** Peripheral PTA base pointer */
#define PTA                                      ((GPIO_Type *)PTA_BASE)
/** Peripheral PTB base address */
#define PTB_BASE                                 (0x400FF040u)
/** Peripheral PTB base pointer */
#define PTB                                      ((GPIO_Type *)PTB_BASE)
/** Array initializer of GPIO peripheral base pointers */
#define GPIO_BASES                               { PTA, PTB }
```

Rys. 12 Deklaracja struktur PTA i PTB, w bibliotece MKL05Z4.h

Struktura PTB jest zadeklarowana jako wskaźnik, więc odwoływanie się do jej elementów będzie następujące:

PTB->PDDR |= (1<<8)|(1<<9)|(1<<10); //Ustaw na 1 bity 8, 9 i 10 – wyjścia

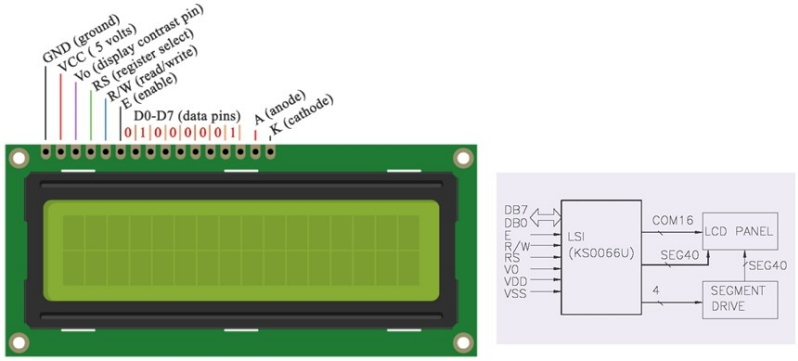
Porównać z programem w instrukcji do Ćw.1.

Teraz, gdy na odpowiednim bicie portu ustawimy wartość „0”, to dołączona do niego dioda zaświeci, a gdy zapiszemy wartość „1” to zgaśnie. Ustawianiem wartości poszczególnych pinów w porcie zajmują się cztery rejestry ze struktury GPIO (Rys. 11):

- PDOR – 32-u bitowy rejestr, którego każdy bit rządzi odpowiadającym mu bitem portu. Wpisanie wartości „1”, na pozycję danego bitu, powoduje ustawienie wartości „1” na odpowiedniej końcówce portu. Z wartością „0” jest podobnie.
- PSOR – 32-u bitowy rejestr, którego każdy bit rządzi odpowiadającym mu bitem portu. Wpisanie wartości „1”, na pozycję danego bitu, powoduje ustawienie wartości „1” na odpowiedniej końcówce portu. Wpisanie wartości „0” nie powoduje żadnych zmian. Aktywna jest tylko wartość „1”.
- PCOR - 32-u bitowy rejestr, którego każdy bit rządzi odpowiadającym mu bitem portu. Wpisanie wartości „1”, na pozycję danego bitu, powoduje ustawienie wartości „0” na odpowiedniej końcówce portu. Wpisanie wartości „0” nie powoduje żadnych zmian. Aktywna jest tylko wartość „1”.
- PTOR – 32-u bitowy rejestr, którego każdy bit rządzi odpowiadającym mu bitem portu. Wpisanie wartości „1”, na pozycję danego bitu, powoduje ustawienie wartości przeciwnej do poprzedniej, na odpowiedniej końcówce portu. Wpisanie wartości „0” nie powoduje żadnych zmian. Aktywna jest tylko wartość „1”.

2.2. WYŚWIETLACZ LCD

Diody LED niosą ze sobą bardzo mało informacji dla użytkownika. Wielkości alfanumeryczne można by przekazywać chyba jedynie alfabetem Morse’a. Dlatego wymyślono różnorakiego rodzaju wyświetlacze: siedmiosegmentowe (tylko cyfry i niektóre litery), wierszowe alfanumeryczne, monitory ekranowe, itp. Na rysunku Rys. 13 jest pokazany popularny, dwuwierszowy, 16-znakowy, alfanumeryczny wyświetlacz LCD, sterowany układem HD44780U.



Rys. 13 Moduł wyświetlacza LCD wraz ze schematem blokowym

Moduł posiada 8-bitową, dwukierunkową szynę danych DB0 ÷ DB7 oraz, uwzględniając jeszcze dodatkowy bit, sterujący podświetlaniem BL, 4-bitową szynę sterującą. Razem 12 bitów. W systemach takich, jak KL05Z, każda końcówka jest na wagę złota, a tu taka rozrzutność. Z tego też względu, można ograniczyć liczbę końcówek (wyświetlacza) do dwóch, kosztem szybkości, stosują do komunikacji port szeregowy, w tym wypadku I^2C. Wyświetlacz, jako wyjściowy interfejs komunikacji z użytkownikiem, nie musi być bardzo szybki. Redukcji dokonano, stosując 8-bitowy ekspander I^2C, który będzie pośredniczył pomiędzy portem szeregowym (I^2C) a równoległym portem układu HD44780U. Układ ten ma możliwość ograniczenia długości swojej szyny danych do 4-ech bitów, co razem z bitami sterującymi daje liczbę 8-miu bitów, czyli dokładnie tyle, ile ma ekspander. Nie będziemy wchodzić dalej z butami w wewnętrzną strukturę układu, aby co słabszy nerwowo student nie padł na palpitację. Będziemy wykorzystywać gotową bibliotekę, która pozwoli wyświetlać różne cuda na kiju.

Aby z niego skorzystać, należy go najpierw podłączyć do płytki KL05Z, wykorzystując dołączone kabelki (oddzieramy z taśmy tasiemkę zawierającą cztery kolorowe kable), wg poniższego rysunku (Rys. 14) i tabeli (Tab. 2).

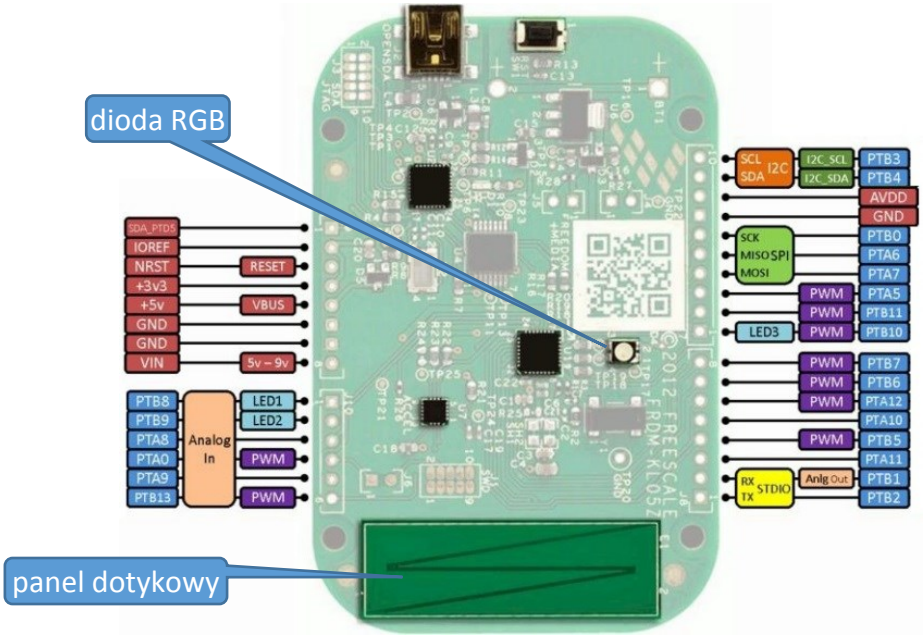
Podłączeń dokonujemy przy odłączonym kablu USB!!! Proszę parę razy sprawdzić poprawność połączeń przed podłączeniem kabla USB!!!

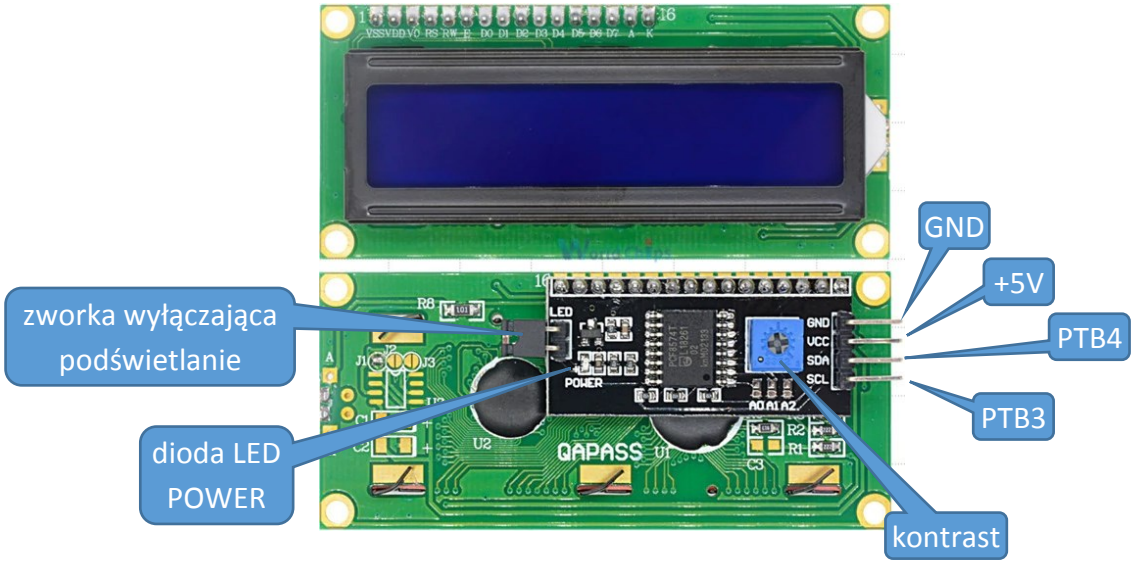
FRDM-KL05	LCD
PTB3	SCL
PTB4	SDA
+5V	VCC
GND	GND

Tab. 2 Tabela podłączeń dla wyświetlacza LCD

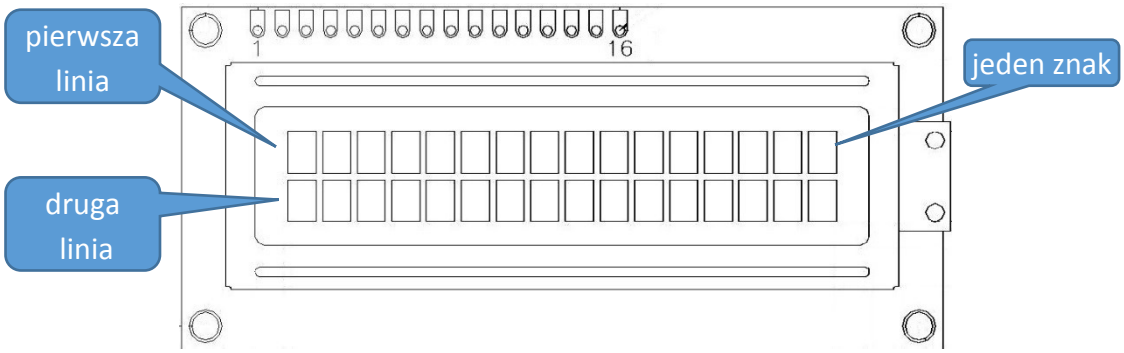
Po podłączeniu zasilania (kabel USB), z tyłu wyświetlacza LCD, powinna się świecić czerwona dioda LED (POWER) – Rys. 14. Po uruchomieniu projektu, w którym jest włączane podświetlanie ekranu, wyregulować widoczność znaków wg własnych upodobań. Dokonuje się tego za pomocą śrubokręta, delikatnie obracając w lewo lub w prawo potencjometrem „kontrast” (Rys. 14).

Na module LCD jest również dostępna zworka, wyłączająca sprzętowo podświetlanie wyświetlacza, ale jej nie ruszamy.





Rys. 14 Podłączanie wyświetlacza LCD do płytki KL05Z



Rys. 15 Rozkład wyświetlanych linii

Wyświetlacz posiada dwie linie, z czego każda zawiera 40 znaków. Aktualnie jest wyświetlanych 16 znaków w linii (Rys. 15), reszta jest niewidoczna, chyba że zaczniemy posuwać tekst w prawo albo w lewo.

2.2.1. BIBLIOTEKA WYŚWIETLACZA LCD

Bibliotekę tworzą dwa zbiory: *LCD1602.c* i *LCD1602.h*. Zawierają podstawowe funkcje inicjujące pracę portów komunikacyjnych oraz funkcje, pozwalające wyświetlać informacje alfanumeryczne. Ponieważ komunikacja z mikrokontrolerem odbywa się za pomocą magistrali I^2C, więc do poprawnego działania są potrzebne również zbiory zarządzające tym portem: *i2c.c* i *i2c.h*. Piątym zbiorem jest *frdm_bsp.h*, który definiuje pewne stałe i makra, wykorzystywane w trakcie realizacji ćwiczeń. Wszystkie zbiory są zebrane w jednym archiwum (*Biblioteki_1.zip*), które należy rozpakować do folderu każdego projektu używającego w/w funkcji. Zbiory nagłówkowe (**.h*) należy dołączać do projektów, dyrektywą „***#include***".

2.2.1.1. FUNKCJE BIBLIOTECZNE

W nawiasach są podane typy i liczba argumentów. Po wyświetleniu ostatniego znaku, kursor ustawia się w następnym polu. Kursor może być widoczny lub nie.

- ✓ LCD1602_Init() – inicjuje pracę wyświetlacza. Musi być wywołana na początku programu.
- ✓ LCD1602_Backlight(uint8_t state) – sterowanie podświetlaniem:
  state=TRUE świeci,
  state=FALSE nie świeci.
- ✓ LCD1602_SetCursor(uint8_t col, uint8_t row) – ustawia kursor w linii „row", na pozycji znakowej „col". Od tego miejsca odbywać się będzie wyświetlanie znaków, wywołanych funkcjami wyświetlającymi.
- ✓ LCD1602_ClearAll() – czyści cały wyświetlacz i ustawia kursor w pierwszej linii, na pierwszej pozycji znakowej.
- ✓ LCD1602_Print(char *str) – wyświetla ciąg znaków str. Jeśli jest wyświetlana tablica znaków, wypełniana „na piechotę", to ostatnim znakiem powinno być zero ('\0').
- ✓ LCD1602_Blink_Off_Cursor_On() – pokazuje kursor w aktualnej pozycji.

3. URZĄDZENIA WEJŚCIOWE

W tym ćwiczeniu poznamy dwa urządzenia wejściowe: panel dotykowy i klawiaturę.

3.1. PANEL DOTYKOWY

Płytka KL05Z posiada w swoich zasobach panel dotykowy (Rys. 14). Podpięty jest do specjalnego interfejsu, który przetwarza dotkniętą palcem pozycję na odpowiedni sygnał elektryczny, a następnie na cyfrowy odpowiednik, który jest do naszej dyspozycji. Jak go zinterpretujemy to zależy od nas i naszego programu. Tak jak w przypadku wyświetlacza, będą dostępne biblioteki z funkcjami inicjującymi prace pola oraz funkcją pozwalającą pozyskiwać dane na temat dotkniętej pozycji.

3.1.1. BIBLIOTEKA PANELU DOTYKOWEGO

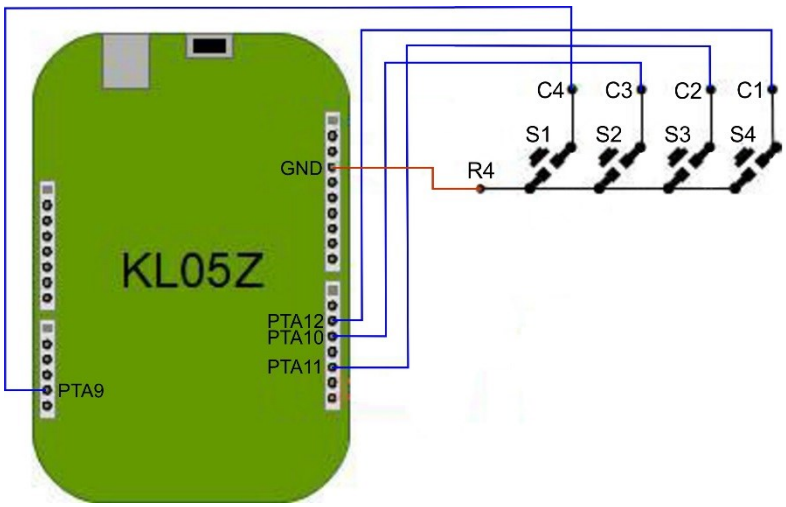
Bibliotekę tworzą dwa zbiory: *tsi.c* i *tsi.h*. Wszystkie zbiory są zebrane w jednym archiwum (*Biblioteki_2.zip*), które należy rozpakować do folderu każdego projektu używającego w/w funkcji. Zbiór nagłówkowy (**.h*) należy dołączać do projektów, dyrektywą „***#include***”.

3.1.2. FUNKCJE BIBLIOTECZNE

- ✓ TSI_Init() – konfiguruje i inicjuje pracę pola dotykowego.
- ✓ uint8_t TSI_ReadSlider () – zwraca liczbę z przedziału 0 ÷ 100, która niesie informację o pozycji dotkniętej palcem. Zero to skrajna lewa pozycja, 100 skrajna prawa.

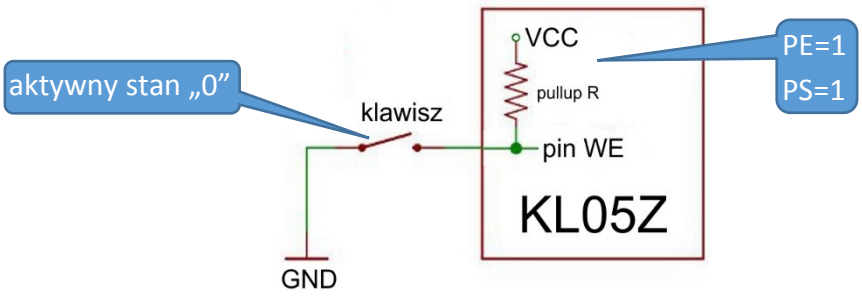
3.2. KLAWIATURA

Klawiaturę będą realizować tylko cztery klawisze: S1 ÷ S4, podłączone do płytki KL05Z wg schematu z rysunku Rys. 16.



Rys. 16 Podłączenie klawiatury w wersji uproszczonej

Każdy klawisz pracuje w konfiguracji z rysunku Rys. 17.



Rys. 17 Konfiguracja pracy pojedynczego klawisza

No i znowu trzeba „poczarować” znanymi już nam rejestrami. Tym razem w grę wchodzi port A. W rejestrze SCGC5, w bloku SIM, ustawiamy bit PORTA na wartość „1” (Rys. 3):

SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK; // Włącz port A

Analizujemy tabelę Tab. 1, odnajdując piny PTA9 ÷ PTA12 i ich funkcję jako piny portu (ALT1). W pole MUX, rejestru PCRn, wpisujemy cyfrę stojącą przy skrócie ALT (u nas 1). Ponieważ piny te będą pracować jako wejścia, współpracujące z klawiszami zwiernymi (Rys. 17), więc przy okazji należy również zainteresować się bitami PE i PS, w rejestrze PCRn (Rys. 7). Bit PE włącza możliwość użycia rezystora dodatkowego, a bit PS wybiera, jak ten rezystor ma być podłączony. Ponieważ bity te są w tym samym rejestrze i dotyczą tej samej funkcji, więc można je „hurtowo” ustawić za pomocą jednej instrukcji.

```
PORTA->PCR[9] |= PORT_PCR_MUX(1);
PORTA->PCR[10] |= PORT_PCR_MUX(1);
PORTA->PCR[11] |= PORT_PCR_MUX(1);
PORTA->PCR[12] |= PORT_PCR_MUX(1);
PORTA->PCR[9] |= PORT_PCR_PE_MASK | PORT_PCR_PS_MASK;
PORTA->PCR[10] |= PORT_PCR_PE_MASK | PORT_PCR_PS_MASK;
PORTA->PCR[11] |= PORT_PCR_PE_MASK | PORT_PCR_PS_MASK;
PORTA->PCR[12] |= PORT_PCR_PE_MASK | PORT_PCR_PS_MASK;
```

Znowu skorzystaliśmy z pomocnych masek i makr, zdefiniowanych w bibliotece MKL05Z4.h, w sekcji „PORT Register Masks”.

Na koniec zostało już tylko ustawienie roli tych pinów, jako wejścia. Ponieważ po sygnale RESET, domyślne ustawienie bitów rejestru PDDR (Rys. 10), zarządzającego rolami pinów, to „0” (czyli wejście), więc nie musimy tego robić jeszcze raz.

Klawiatura jest gotowa do użytku.

Do czytania stanu klawiatury służy rejestr PDIR, ze struktury GPIO (Rys. 11):

- PDIR – 32-u bitowy rejestr, którego wartość każdego bitu odzwierciedla stan odpowiadającego mu pinu. Rejestr ten jest tylko do odczytu.

4. ĆWICZENIE

Rozpakować zbiór *Interfejsy.zip*. Uruchomić projekt *Interfejsy.uvprojx*. Na początku pojawia się ekran powitalny. Program czeka na wciśnięcie klawisza S1. Po jego wciśnięciu, na ekranie pojawia się wartość początkowa pozycji palca na panelu dotykowym (--), czyli nic jeszcze nie dotknięto. Program czeka na dwa zdarzenia: wciśnięcie klawisza S2 lub/i dotknięcie panelu dotykowego. W przypadku pierwszego, dioda czerwona LED (R) zmienia swój stan świecenia na przeciwny, czyli jeśli na początku była zgaszona, to teraz świeci. Kolejne wciskanie tego klawisza gasi, zapala, gasi, zapala, itd. Przytrzymanie klawisza S2 w stanie wciśniętym, powoduje, że dioda mruga. W przypadku dotknięcia panelu dotykowego, na wyświetlaczu pojawi się wartość liczbowa pozycji (od 0 do 100) palca. Stan ten będzie pokazywany do czasu nowego dotknięcia. Zamiast dotykania, można palcem przesuwać po polu.

1.1. ZADANIE

- Wyjaśnić, dlaczego w czasie trzymania klawisza S2 w pozycji wciśniętej, dioda mruga?
- Zrealizować identyczną funkcję, co dla diody czerwonej, dla diody zielonej (G) i niebieskiej (B). Diodą zieloną ma sterować klawisz S3, a niebieską S4.