

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СИКОРСЬКОГО»**

**КАФЕДРА КОНСТРУЮВАННЯ ЕЛЕКТРОННО-ОБЧИСЛЮВАЛЬНОЇ
АПРАТУРИ**

ЗВІТ

з лабораторної роботи № 3
по курсу «Обчислювальні та МП засоби в РЕА-2»

Виконав:
студент гр. ДК-82
Дмитрук О.О.

Перевірів:
ст. викладач
Бондаренко Н.О.

Завдання

1. Реалізувати виведення однорозрядного числа на семисигментному індикаторі з програмним перетворенням числа в код індикатора.
Зменшення/збільшення числа натисканням кнопок.

Теоретична частина

Розглянемо теорію, необхідну для виконання лабораторної роботи.

Щоб працювати з портами вводу/виводу (GPIO) мікроконтролеру STM32 потрібно виконати певні дії, а саме:

1. Ввімкнути тактування портів
2. Налаштувати порти Input / Output
3. Записувати в регістр ODR якісь значення (У випадку Output)
Зчитувати з регістру IDR певні записані в нього значення (Налашт. Input)

В кожного периферійного блоку є своя базова адреса, по якій можна звертатись до певних регістрів налаштування. На рис.1. зображена карта пам'яті мікроконтр. STM32F4, з яким ми будемо працювати.

Boundary address	Peripheral	Bus	Register map
0x5000 0000 - 0x5003 FFFF	USB OTG FS	AHB2	Section 22.16.6: OTG_FS register map on page 755
0x4002 6400 - 0x4002 67FF	DMA2	AHB1	Section 9.5.11: DMA register map on page 198
0x4002 6000 - 0x4002 63FF	DMA1		
0x4002 3C00 - 0x4002 3FFF	Flash interface register		Section 3.8: Flash interface registers on page 60
0x4002 3800 - 0x4002 3BFF	RCC		Section 6.3.22: RCC register map on page 137
0x4002 3000 - 0x4002 33FF	CRC		Section 4.4.4: CRC register map on page 70
0x4002 1C00 - 0x4002 1FFF	GPIOH		Section 8.4.11: GPIO register map on page 164
0x4002 1000 - 0x4002 13FF	GPIOE		
0x4002 0C00 - 0x4002 0FFF	GPIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		

Рис.1. Карта пам'яті SET32F4

Всю інформацію про регістри налаштування даного мікроконтр. можна найти в reference manual-i.

Щоб ввімкнути тактування портів, необхідно звернутись до блоку RCC (Reset Clock Control) що має базову адресу 0x40023800

Тепер потрібно звернутись до регістру `RCC_AHB1ENR` який відповідає за ввімкнення тактування `GPIOx`, який має зміщення відносно базової адреси `0x30`.

6.3.9 RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: `0x30`

Reset value: `0x0000 0000`

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									DMA2EN	DMA1EN	Reserved				
									rw	rw					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCEN	Reserved				GPIOH EN	Reserved		GPIOEEN	GIPOD EN	GPIOC EN	GPIOB EN	GPIOA EN
			rw					rw			rw	rw	rw	rw	rw

Bit 0 **GPIOAEN**: IO port A clock enable

Set and cleared by software.

0: IO port A clock disabled

1: IO port A clock enabled

Рис.2. Регістр тактування `GPIOx`

Як бачимо, щоб ввімкнути тактування певного `GPIO`, необхідно встановити 1 в певному зарезервованому біті. Це можна легко зробити звернувшись по базовій адресі + зміщення, і виконати “побітове або” з маскою ($1 \ll \text{BIT}$) де `BIT` – номер біту певного порту.

Тепер розберемось з налаштуванням `GPIOx` на ввід/вивід. Для цього існує регістр `GPIOx_MODER` (тобто в кожного порту є такий регістр)

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..E and H)

Address offset: `0x00`

Reset values:

- `0x0C00 0000` for port A
- `0x0000 0280` for port B
- `0x0000 0000` for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits $2y:2y+1$ **MODERy[1:0]**: Port x configuration bits ($y = 0..15$)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

Рис.3. Регістр налаштування `GPIOx`

Записавши в даний регістр в певні біти певні данні, ми зможемо налаштувати певні піни на наступні режими:

1. 00 – Режим читання – тобто пін буде зчитувати сигнал і записувати його в регістр IDR по кожному активному фронту тактового сигналу
2. 01 – Режим запису – пін буде генерувати цифрову 1 або 0 (3.3в або 0в)
3. 10 – Режим, в якому пін буде налаштований під певні периферії, такі як SPI, I2C і т.п.
4. 11 – Аналоговий режим – пін налаштований на генер. ШИМ сигналу і т.п.

Розберемось з регістрами GPIOx_ODR та GPIO_IDR

8.4.5 GPIO port input data register (GPIOx_IDR) (x = A..E and H)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy**: Port input data (y = 0..15)

These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

8.4.6 GPIO port output data register (GPIOx_ODR) (x = A..E and H)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)

These bits can be read and written by software.

Note: For atomic bit set/reset, the ODR bits can be individually set and reset by writing to the GPIOx_BSRR register (x = A..E and H).

Рис.4. Регістри даних IDR та ODR

За допомогою регістру ODR ми можемо генерувати сигнали на піни GPIOx. Також з даного регістру можна зчитувати.

Регістр IDR працює лише на читування, і необхідний щоб в нього по кожному актив. фронту сигналу синхр. записувались дані з певних пінів GPIO.

Можна ідентифікувати нажаття кнопки, що спричинить запис в певний біт регістру 0 або 1, в залежності від того як підключена кнопка.

Також існує досить багато регістрів, такі як BSSR, за допомогою якого також можна встановлювати та скидати певні біти регістру ODR (перші 16 біт відповідають за встановлення, старші 16 за скидання), GPIOx_PUPDR, за допомогою якого можна підтянути пін до землі або до живлення, OTYPER, яким можна налаштувати пін як OpenDrain або Push-pull, за необхідності узгодження за напругою пінів. Сильно заглиблюватись в дані регістри не будемо, так як для виконання завдання вони не потрібні.

Також зазначу, що в STM32F4 є можливість дебагу в Real time, де можливо перевірити вміст всіх регістрів, і взагалі коректність програми. Для виводу регістрів на екран (в фреймворці Keil) при виконанні дебагу, потрібно перейти view → system viewer, і там вибрати необхідний регістр.

Базову теорію розглянули, тепер можемо приступити до роботи.

Виконання роботи

Щоб реалізувати поставлене завдання, необхідно попередньо визначитись з портами вводу / виведення, на які будемо подавати лог. сигнали для семисигментного індикатору. Для цього було обрано перші 7 пінів порту GPIOB. Для кнопок було обрано 13 і 4 піни порту GPIOC.

Перейдемо до написання програми.

Для початку оприділимо всі необхідні адреси, а саме адресу регістру ввімкнення тактування портів, регістру ODRx, IDR_x, MODER_x.

```

1  #pragma once
2  #include <stdio.h>
3
4  typedef unsigned long      uint32;
5  typedef unsigned long long uint64;
6
7  //REG ENABLE CLOCK
8  #define RCC_AHB1_ENABLE_REG (uint32*) 0x40023830
9
10 //REG ENABLE IN/OUT
11 // #define GPIOA_MODE_REG      (uint32*) 0x40020000
12 #define GPIOB_MODE_REG      (uint32*) 0x40020400
13 #define GPIOC_MODE_REG      (uint32*) 0x40020800
14 // #define GPIOB_BSRR          (uint32*) 0x40020418
15
16 // #define RCC_AHB1_GPIOA_CLOCK 0 //номер біту для ініц.тактування порта A
17 #define RCC_AHB1_GPIOB_CLOCK 1 //номер біту для ініц.тактування порта B
18 #define RCC_AHB1_GPIOC_CLOCK 2 //номер біту для ініц.тактування порта C
19
20 //reg output and input
21 // #define GPIOA_IN_DATA_REG    (uint32*) 0x40020010 //input register A
22 #define GPIOC_IN_DATA_REG    (uint32*) 0x40020810 //input register C
23
24 #define GPIOB_OUT_DATA_REG    (uint32*) 0x40020414 //output register B
25

```

Рис.5. Визначення адрес

Далі визначимо константи, котрі містять код для формування сигналів семисигм. індикатору.

```

26 //constants
27 #define ZERO 63
28 #define ONE 6
29 #define TWO 91
30 #define THREE 79
31 #define FOUR 102
32 #define FIVE 109
33 #define SIX 125
34 #define SEVEN 7
35 #define EIGHT 127
36 #define NINE 111
37

```

рис.6. Визначення констант для форм. сигнл. семисиг. індикатору

```

43 //functions
44 #define GPIOx_SET_MODE_OUTPUT(port, pin) \
45     do { \
46         *GPIO##port##_MODE_REG |= (1 << (pin * 2)); \
47         *GPIO##port##_MODE_REG &= ~(1 << (pin * 2 + 1)); \
48     } while(0)
49
50 #define GPIOx_SET_MODE_INPUT(portl, pinl) \
51     do { \
52         *GPIO##portl##_MODE_REG &= ~(1 << (pinl * 2)); \
53         *GPIO##portl##_MODE_REG &= ~(1 << (pinl * 2 + 1)); \
54     } while(0)
55
56
57 int ButtonPress (const int but, int *but_flag);
58 int ButtonReleased (const int but, int *but_flag);
59 void delay (long sec);
60 void fixNumber(int *num);
61 void switchNum(int num);
62 void debugLed(long sec);

```

Рис.7. Визначення функцій для ініціалізації пінів портів, прототипи функцій обробки кнопок, і т.п.

Перейдемо до головної функції main.

```

11 //SETUP
12 //Clock enable.....
13 //RCC_AHB1_ENABLE_REG |= (1 << 0); //включили тактування порту A
14 *RCC_AHB1_ENABLE_REG |= (1 << 1); //включили тактування порту B
15 *RCC_AHB1_ENABLE_REG |= (1 << 2); //включили тактування порту C
16
17 //port mode.....
18 //GPIOA
19
20 //GPIOC
21 GPIOx_SET_MODE_INPUT(C,4);
22 GPIOx_SET_MODE_INPUT(C,13);
23
24 //GPIOB
25 GPIOx_SET_MODE_OUTPUT(B,0);
26 GPIOx_SET_MODE_OUTPUT(B,1);
27 GPIOx_SET_MODE_OUTPUT(B,2);
28 GPIOx_SET_MODE_OUTPUT(B,3);
29 GPIOx_SET_MODE_OUTPUT(B,4);
30 GPIOx_SET_MODE_OUTPUT(B,5);
31 GPIOx_SET_MODE_OUTPUT(B,6);
32 GPIOx_SET_MODE_OUTPUT(B,7);
33
34 //DEBUG SevenSeg. LEDs
35 //debugLed(sec);

```

Рис.8. Ініціалізація змінних, ввімкнення тактування портів, їхнє налаштування

На рис.8. представлено початок виконання програми, де проводиться ініціалізація змінних, необхідних для обробки натискання кнопок, збереження

числа і т.п. Починаючи зі строчки 14 виконується ввімкнення тактування портів В та С. Тіло функції зображено на рис.7. Виконуємо побітове або вмісту адреси регістру, яку ми оприділили на рис. 5 з маскою одиниці здвинутої вліво на певну кількість біт.

Починаючи з строки 21 виконується налаштування регістру MODERx.

Аналогічна процедура, що і з регістром RCC_AHB1, але тепер потрібно змінити 2 біти, для пінів порту С на зчитування, тобто *00*, для пінів порту В на запис *01*

Далі буде представлений нескінченний цикл, в якому виконується обробка кнопок, формування сигналів на семис. індикатор та перевірка числа.

```
37 while (1){
38     //Обробка кнопок
39     //dec
40     buttonDec = 0;
41     buttonDec = (0 == (*GPIOC_IN_DATA_REG & (1 << 4))); //PC4
42
43     if(0 != ButtonPress(buttonDec,&button_flagDec)){
44     if(0 != ButtonReleased(buttonDec,&button_flagDec)){
45         num --;
46     }
47     fixNumber(&num);
48
49     //inc
50     buttonInc = 0;
51     buttonInc = (0 == (*GPIOC_IN_DATA_REG & (1 << 13))); //PC13 (USER BUTTON 1)
52
53     if(0 != ButtonPress(buttonInc,&button_flagInc)){
54     if(0 != ButtonReleased(buttonInc,&button_flagInc)){
55         num ++;
56     }
57     fixNumber(&num);
58
59     switchNum(num);
60     }
61 }
62
```

Рис.9. Нескінченний цикл для формування сигналів симисигм. інд. і т.п.

На строці 41 та 51 виконується зчитування бітів регістру IDR порту С, за допомогою побітового і, з маскою (1 здвинутої вліво на к-сть біт, номер якого необхідно зчитати) і порівнюючи отримане значення з нулем. Цей алгоритм реалізований під варіант, коли при натисканні кнопки записується в певний біт регістру IDR нуль, а не одиниця. Тобто коли у нас натиснута кнопка, в змінні

записується 1. а коли ні , 0. На строчках 43,44,53,54 виконуються функції обробки натискання кнопки, які борються з дрязькітом контактів. Також у випадку віджимання кнопок які підкл. до пінів 4 та 13 GPIOC виконується декрементування і інкрементування числа, яке виводиться на семисигм. індикатор, відповідно.

На строчка 47 та 57 виконується функція виправлення числа. У випадку якщо число більше за 9 , то воно обнуляється, а якщо менше за 0, воно стає рівне 9.

На строчці виконується функція запису в регістр ODR значень для формування сигналів семисигм. індикатору.

На рис.10. представлена реалізація всіх вище згаданих функцій.

```
1  #include "header.h"
2
3  int ButtonPress (const int but, int *but_flag){
4      int i = 0;
5      if (but == 1 && *but_flag == 0){
6          for(i = 0; i < 1000000; i ++){}
7
8          *but_flag = 1;
9          return 1;
10     }
11     return 0;
12 }
13
14 int ButtonReleased (const int but, int *but_flag){
15     int i = 0;
16     if (but == 0 && *but_flag == 1){
17         for(i = 0; i < 1000000; i ++){}
18
19         *but_flag = 0;
20         return 1;
21     }
22     return 0;
23 }
24
25 void delay (long sec){
26     long i = 0;
27     for ( i = 0; i < sec * 4000000; i ++){}
28 }
29
30 void fixNumber(int *num){
31     if(*num > 9) *num = 0;
32     if(*num < 0) *num = 9;
33 }
34
```

рис.10.а. Тіла функцій обробки натискання кнопок, виправлення числа.

```

35 void switchNum(int num) {
36     switch(num) {
37         case 0: *GPIOB_OUT_DATA_REG ^= *GPIOB_OUT_DATA_REG;
38                 *GPIOB_OUT_DATA_REG |= ZERO;
39                 break;
40         case 1: *GPIOB_OUT_DATA_REG ^= *GPIOB_OUT_DATA_REG;
41                 *GPIOB_OUT_DATA_REG |= ONE;
42                 break;
43         case 2: *GPIOB_OUT_DATA_REG ^= *GPIOB_OUT_DATA_REG;
44                 *GPIOB_OUT_DATA_REG |= TWO;
45                 break;
46         case 3: *GPIOB_OUT_DATA_REG ^= *GPIOB_OUT_DATA_REG;
47                 *GPIOB_OUT_DATA_REG |= THREE;
48                 break;
49         case 4: *GPIOB_OUT_DATA_REG ^= *GPIOB_OUT_DATA_REG;
50                 *GPIOB_OUT_DATA_REG |= FOUR;
51                 break;
52         case 5: *GPIOB_OUT_DATA_REG ^= *GPIOB_OUT_DATA_REG;
53                 *GPIOB_OUT_DATA_REG |= FIVE;
54                 break;
55         case 6: *GPIOB_OUT_DATA_REG ^= *GPIOB_OUT_DATA_REG;
56                 *GPIOB_OUT_DATA_REG |= SIX;
57                 break;
58         case 7: *GPIOB_OUT_DATA_REG ^= *GPIOB_OUT_DATA_REG;
59                 *GPIOB_OUT_DATA_REG |= SEVEN;
60                 break;
61         case 8: *GPIOB_OUT_DATA_REG ^= *GPIOB_OUT_DATA_REG;
62                 *GPIOB_OUT_DATA_REG |= EIGHT;
63                 break;
64         case 9: *GPIOB_OUT_DATA_REG ^= *GPIOB_OUT_DATA_REG;
65                 *GPIOB_OUT_DATA_REG |= NINE;
66                 break;
67     }
68 }
69

```

Рис.10.6. Тіло ф-ції формування сигналів для семисигм. індикатору

Виклик головної ф-ї main_loop виконуємо за допомогою безумовного переходу в стартап файлі.

```

15 ; Area STACK (NOINIT and READ/WRITE)
16         AREA    STACK, NOINIT, READWRITE, ALIGN=4
17 ; Initialize memory equals Stack_Size
18 Stack_Mem    SPACE    Stack_Size
19 __initial_sp
20
21 ; Vector Table Mapped to Address 0 at Reset
22         AREA    RESET, DATA, READONLY
23         EXPORT  __Vectors
24         EXPORT  Reset_Handler            [WEAK]
25 __Vectors
26         DCD     __initial_sp            ; Top of Stack
27         DCD     Reset_Handler          ; Reset Handler
28 __Vectors_End
29
30         ; code area
31         AREA    |.text|, CODE, READONLY
32         EXPORT  Reset_Handler
33 Reset_Handler
34         PROC
35         B main
36         ENDP
37
38         EXTERN  main_loop
39
40 main PROC
41     BL main_loop
42     B main ;end loop
43     ENDP
44     ALIGN
45
46         END

```

Рис.11. Стартап файл