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

КАФЕДРА КОНСТРУЮВАННЯ ЕЛЕКТРОННО-ОБЧИСЛЮВАЛЬНОЇ АПАРАТУРИ

ЗВІТ

з лабораторної робіти № 4
по курсу «Обчислювальні та МП засоби в РЕА-2»

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

Перевірив:
ст. викладач
Бондаренко Н.О.

Київ – 2021

Завдання

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

Теоретична частина

Розглянемо теорію, необхідну для виконання лабораторної роботи.

Щоб працювати з портами вводу/виводу (GPIO) мікроконтроллеру 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-і.

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				GPIOHEN	Reserved		GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
			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 << 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
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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 данних 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 считування, 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.

В бібліотеці CMSIS задекларовані адреси всіх регістрів, згаданих вище, і всіх інших, які будуть необхідні нам в подальшому вивчені роботи з мікропроцесорами Cortex-M. Бібліотека являє собою опис структур, вказівник яких вказують на базові адреси певних підрозділів карти памяті (таких як RCC, GPIOx, CRC, DMA і т.п.). В тілах структур містяться зміщення до певних регістрів відносно базових адрес, таких як GPIOA-> ODR, GPIOB->IDR, RCC->AHB1ENB і т.п. Також бібліотека містить в собі всі необхідні маски для зручної роботи з регістрами.

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 viev → 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 підключимо бібліотеку CMSIS, прописавши наступну строчку в файлы header.h:

#include "stm32f4xx.h"

Тепер ми можемо не оприділять адреси регістрів так як всі вони є в бібліотеці CMSIS

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

```
3  //constants
4  #define ZERO 63
5  #define ONE 6
6  #define TWO 91
7  #define THREE 79
8  #define FOUR 102
9  #define FIVE 109
10 #define SIX 125
11 #define SEVEN 7
12 #define EIGHT 127
13 #define NINE 111
```

рис.5. Визначення констант для форм. сигнл. семисиг. індикатору

Визначимо прототипи всіх необхідних функцій:

```
15  int ButtonPress (const int but, int *but_flag);
16  int ButtonReleased (const int but, int *but_flag);
17  void delay (long sec);
18  void fixNumber(int *num);
19  void switchNum(int num);
20  void debugLed(long sec);
21
```

Рис.6. Визначення прототипи функцій обробки кнопок, і т.п.

Перейдемо до головної функції main.

При налаштуванні проекту, ми додали файл startup.s, який являє собою asm файл, який виконує стартове налаштування периферії, оприділення векторів переривань і т.п. Після налаштувань виконується перехід на мітку головної функції main:

```
171  ; Reset handler
172  Reset_Handler    PROC
173                   EXPORT  Reset_Handler             [WEAK]
174          IMPORT  SystemInit
175          IMPORT  __main
176
177                   LDR     R0, =SystemInit
178                   BLX     R0
179                   LDR     R0, =__main
180                   BX      R0
181                   ENDP
182
```

Рис.7. Виконання переходу на мітку main

Перейдемо до основного файлу main:

```
1  #include "header.h"
2
3  int main(void){
4
5    int buttonDec = 0;
6    int buttonInc = 0;
7    int button_flagDec = 0;
8    int button_flagInc = 0;
9    int num = 0;
10   long sec = 1;
11
12   //SETUP
13   //Clock enable.....................................
14   //*RCC_AHB1_ENABLE_REG |= (1 << 0);
15     RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN; //включили тактування порту B
16     RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN; //включили тактування порту C
17
18   //port mode.....................................
19
20   //GPIOC
21     GPIOC->MODER &= ~GPIO_MODER_MODE4_0;
22     GPIOC->MODER &= ~GPIO_MODER_MODE4_1;
23
24     GPIOC->MODER &= ~GPIO_MODER_MODE13_0;
25     GPIOC->MODER &= ~GPIO_MODER_MODE13_1;
26
27   //GPIOB
28   GPIOB->MODER |= GPIO_MODER_MODE0_0;
29   GPIOB->MODER &= ~GPIO_MODER_MODE0_1;
30
31   GPIOB->MODER |= GPIO_MODER_MODE1_0;
32   GPIOB->MODER &= ~GPIO_MODER_MODE1_1;
33
34   GPIOB->MODER |= GPIO_MODER_MODE2_0;
35   GPIOB->MODER &= ~GPIO_MODER_MODE2_1;
36
37   GPIOB->MODER |= GPIO_MODER_MODE3_0;
38   GPIOB->MODER &= ~GPIO_MODER_MODE3_1;
39
40   GPIOB->MODER |= GPIO_MODER_MODE4_0;
41   GPIOB->MODER &= ~GPIO_MODER_MODE4_1;
42
43   GPIOB->MODER |= GPIO_MODER_MODE5_0;
44   GPIOB->MODER &= ~GPIO_MODER_MODE5_1;
45
46   GPIOB->MODER |= GPIO_MODER_MODE6_0;
47   GPIOB->MODER &= ~GPIO_MODER_MODE6_1;
```

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 B та C. Виконуємо побітове *або* вмісту регістру(RCC->AHB1ENR) з маскою одиниці, здвитуної вліво на певну кількість біт (RCC_AHB1ENR_GPIOBEN).

Починаючи з строки 21 виконується налаштування регістру MODERx.

Аналогічна процедура, що і з регістром RCC->AHB1ENR, але тепер потрібно змінити 2 біти, для пінів порту C на зчитування, тобто *00*, для пінів порту B на запис *01*

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

```
50    //DEBUG SevenSeg. LEDs
51    debugLed(sec);
52
53    while (1){
54        //Обробка кнопок
55      //dec
56      buttonDec = 0;
57      buttonDec = (!(GPIOC->IDR & GPIO_IDR_ID4)); //PC4
58
59      if(0 != ButtonPress(buttonDec,&button_flagDec)){}
60      if(0 != ButtonReleased(buttonDec,&button_flagDec)){
61        num --;
62      }
63      fixNumber(&num);
64
65      //inc
66      buttonInc = 0;
67      buttonInc = (!(GPIOC->IDR & GPIO_IDR_ID13)); //PC13 (USER BUTTON 1)
68
69      if(0 != ButtonPress(buttonInc,&button_flagInc)){}
70      if(0 != ButtonReleased(buttonInc,&button_flagInc)){
71        num ++;
72      }
73      fixNumber(&num);
74
75      switchNum(num);
76      }
77    return 0;
78  }
```

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

На строці 57 та 67 виконується зчитування бітів регістру IDR порту C, 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 59,60,69,70 виконуються функції обробки натискання кнопки, які боряться 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. іникатор, відповідно.

На строчках 63 та 73 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 }
```

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

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

Рис.10.б. Тіло ф-ції формування сигналів для семисигм. індикатору