

Лабораторна робота №3. Пряме програмування портів GPIO.

Проект на мові асемблера і C.

Вступ

З попередньої лабораторної роботи ми з'ясували, що кожна програма, написана для ARM архітектури, повинна мати декілька основних компонентів: стек, таблиця векторів переривань, ну і звісно тіло самої програми. У стеку будуть зберігатися локальні змінні, які розташовані у регістрах загального призначення, а також в деяких випадках значення адреси повернення з підпрограм, тобто вміст регістру R14(LR), і навіть вміст програмного лічильника R15(PC). У таблиці векторів має бути мінімум дві адреси: перша – адреса з якої починається стек, друга – адреса початку програми,. Приклад найпростішої програми на мові асемблеру можна побачити на рис. 1.

```
1  ; Initialize Stack Size
2  ; Equ 400 hex (1024) bytes
3  Stack_Size      EQU      0x00000400
4  ; Area STACK (NOINIT and READ/WRITE)
5  |               AREA     STACK, NOINIT, READWRITE, ALIGN=4
6  ; Initialize memory equals Stack_Size
7  Stack_Mem       SPACE    Stack_Size
8  __initial_sp
9  ; Vector Table Mapped to Address 0 at Reset
10 |               AREA     RESET, DATA, READONLY
11 |               EXPORT    __Vectors
12 |               EXPORT    Reset_Handler                [WEAK]
13 |__Vectors
14 |               DCD       __initial_sp                  ; Top of Stack
15 |               DCD       Reset_Handler                ; Reset Handler
16 |__Vectors_End
17 |               AREA     |.text| , CODE, READONLY
18 |
19 |Reset_Handler   PROC
20 |               LDR       R0, =__main
21 |               BX        R0
22 |               ENDP
23 |__main          PROC
24 |               ; place you initialization code here
25 |__mainloop
26 |               ; place you application code here
27 |               B __mainloop
28 |               ENDP
29 |               ALIGN
30 |               END
```

Рис. 1. - Лістинг коду найпростішої програми

1. Підготовка до створення програми. Огляд необхідної документації

Однак ця програма нічого не робить. Тож давайте допишемо її так, щоб вона увімкнула світлодіод, який розташований на платі STM32F401 NUCLEO, або STM32F407 Discovery (далі просто NUCLEO і Discovery)¹. Для цього відкриємо інструкцію користувача (user manual UM1724) [1] і знайдемо розділ «LEDs» (ст. 23). Тут написано, що користувачу доступний світлодіод LD2 (рис. 2). Керування ним відбувається за допомогою периферійних блоків GPIO (*General Purpose Input/Outputs*), які прийнято називати *портами вводу/виводу*. Вони позначаються англійськими буквами А,В,С, ...,J,K . Кожен порт містить 16 виводів (pins - пінів). Отже світлодіодом на платі NUCLEO керує пін «5» порту «А» («PA5»), і щоб його увімкнути необхідно подати на його анод високий рівень. На платі STM32F4 Discovery, згідно документу user manual UM1472 [3], програмам користувача доступні 4 світлодіоди LD3 – LD6, підключені до виводів PD12 – PD15. На рис. 3 надано схему підключення світлодіодів на платах NUCLEO і Discovery.

6.4 LEDs

The tricolor LED (green, orange, red) LD1 (COM) provides information about ST-LINK communication status. LD1 default color is red. LD1 turns to green to indicate that communication is in progress between the PC and the ST-LINK/V2-1, with the following setup:

- Slow blinking Red/Off: at power-on before USB initialization
- Fast blinking Red/Off: after the first correct communication between the PC and ST-LINK/V2-1 (enumeration)
- Red LED On: when the initialization between the PC and ST-LINK/V2-1 is complete
- Green LED On: after a successful target communication initialization
- Blinking Red/Green: during communication with target
- Green On: communication finished and successful
- Orange On: Communication failure

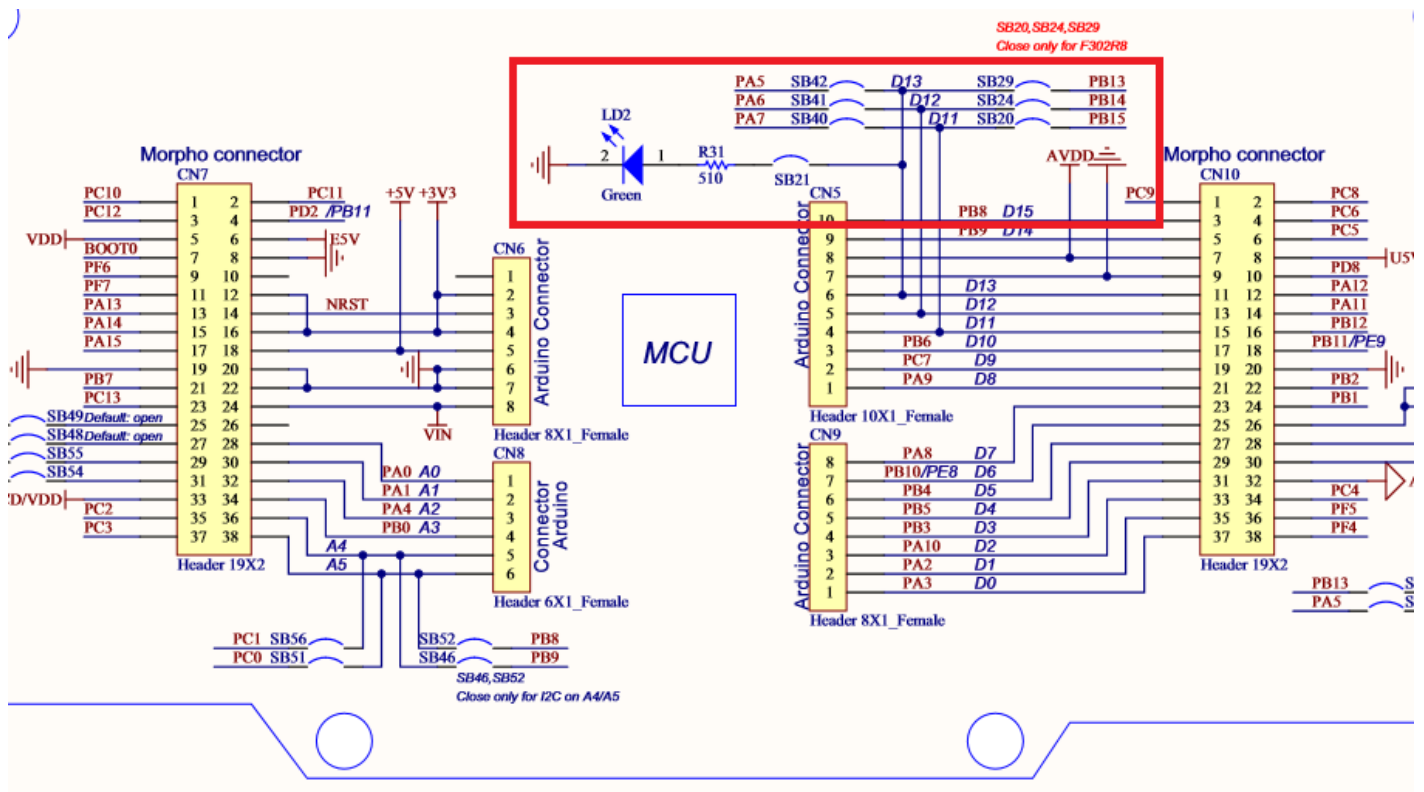
User LD2: the green LED is a user LED connected to Arduino signal D13 corresponding to STM32 I/O PA5 (pin 21) or PB13 (pin 34) depending on the STM32 target. Refer to [Table 11](#) to [Table 23](#) when:

- the I/O is HIGH value, the LED is on
- the I/O is LOW, the LED is off

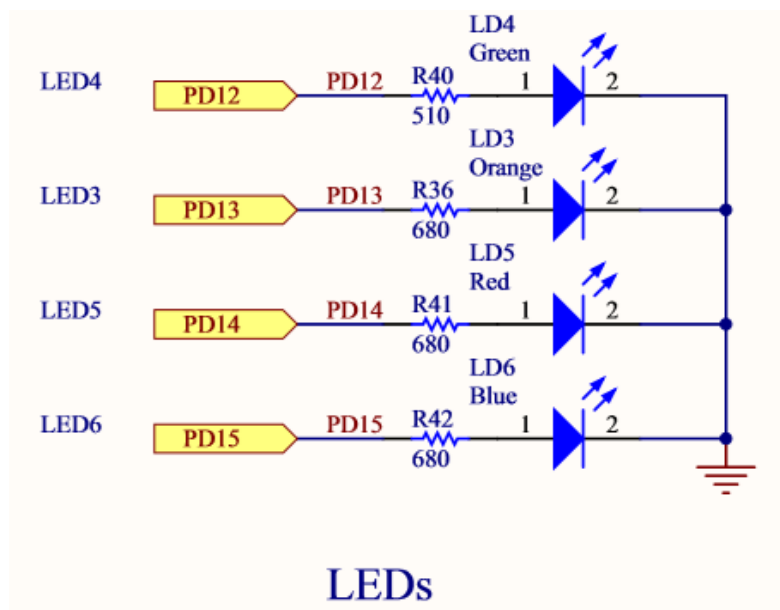
LD3 PWR: the red LED indicates that the STM32 part is powered and +5V power is available.

Рис. 2. - Опис світлодіодів, розташованих на платі NUCLEO

¹ Далі буде йти про обидві плати NUCLEO і Discovery. Аналогічні дані будемо подавати у тексті через «/».



a) NUCLEO



6) Discovery

Рис. 3. - Схема підключення світлодіодів на платах

a) NUCLEO, 6) Dsccovery

Для зниження електроспоживання мікроконтролера (MCU), після його скидання (RESET), практично всі блоки периферійних пристроїв і у тому числі всі порти вимкнуті. Увімкнення або вимкнення їх відбувається подачею або припиненням подачі на них такуючого сигналу. Керування такуючим сигналом відбувається за допомогою регістрів «RCC XXX peripheral clock enable» з загальної групи керування скиданням та синхронізацією RCC (*Reset and clock control*) у якій знаходяться всі регістри керування налаштуванням генераторів, блоків ФАПЧ та шин. Замість «XXX» в позначення регістру має бути назва шини «AHB1», «AHB2» або «APB1», яка забезпечує доступ до певного ресурсу MCU. Тому відкриваємо «reference manual RM0368/RM0090» [2]/[4] і в розділі «6.3. RCC registers»/«7.3 RCC registers» шукаємо регістр, який відповідає за дозвіл (*enable*) тактування портів «A» і «D». Ця інформація є на ст. 117/178 (рис. 4).

6.3.9 RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

7.3.10 Address offset: 0x30

Reset value: 0x0010 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	OTGHS ULPIEN	OTGHS SEN	ETHMACPTP EN	ETHMACRX EN	ETHMACTX EN	ETHMACEN	Reserved			DMA2EN	DMA1EN	CCMDAT ARAMEN	Res.	BKPSR AMEN	Reserved
	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
Reserved			CRCE N	Reserved			GPIOEN	GPIOH EN	GPIOG EN	GPIOF EN	GPIOEEN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
			rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 31 Reserved, must be kept at reset value.

Bit 3 GPIOEN: IO port D clock enable
This bit is set and cleared by software.
0: IO port D clock disabled
1: IO port D clock enabled

Bit 0 GPIOAEN: IO port A clock enable
This bit is set and cleared by software.
0: IO port A clock disabled
1: IO port A clock enabled

Рис. 4. - Біти дозволу/заборони тактування портів GPIOA та GPIOD

Бачимо, що треба в біт 0 / 3 регістру «RCC_AHB1ENR» записати 1. Також тут написано, що зміщення цього регістру відносно базової адреси становить «0x30».

Осталось визначити базові адреси «RCC» і портів «GPIOA» та «GPIOD». Оскільки карта пам'яті для однакових ресурсів всіх MCU STM32F4xx співпадає, то відкриємо розділ «2.3 Memory map» (ст.38 [2] або ст.65 [4]) і побачимо, що значення цих адрес становить: «0x40023800» - для RCC, «0x40020000» - для порту «GPIOA», та «0x40020C00»- для порту «GPIOD» (рис. 5).

2.3 Memory map

See the datasheet corresponding to your device for a comprehensive diagram of the memory map. [Table 1](#) gives the boundary addresses of the peripherals available in STM32F401xB/C and STM32F401xD/E devices.

Table 1. STM32F401xB/C and STM32F401xD/E register boundary addresses

Boundary address	Peripheral	Bus	Register map
0x5000 0000 - 0x5003 FFFF	USB OTG FS	AHB2	Section 22.16.6: OTG_FS register map on page 746
0x4002 6400 - 0x4002 67FF	DMA2	AHB1	Section 9.5.11: DMA register map on page 197
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 136
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 162
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. 5. – Фрагмент опису «memory map»

Більш стисло цю інформацію подано у розділі 6.3.22 [2] /7.3.25 [4] (рис.6):

6.3.22 RCC register map

7.3.25 [Table 22](#) gives the register map and reset values
[Table 34](#) gives the register map and reset values.

Table 34. RCC register map and reset values

Addr. offset	Register name	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			
0x00	RCC_CR	Reserved				PLL12SRDY	PLL12SON	PLL12RDY	PLL12ON	Reserved				CSSON	HSEBYP	HSERDY	HSEON	HSICAL7	HSICAL6	HSICAL5	HSICAL4	HSICAL3	HSICAL2	HSICAL1	HSICAL0	HSITRIM4	HSITRIM3	HSITRIM2	HSITRIM1	HSITRIM0	Reserved	HSIRDY	HSION			
0x1C	Reserved	Reserved																																		
0x2C	Reserved	Reserved																																		
0x30	RCC_AHB1ENR	Reserved	OTGHSULPIEN	OTGHSN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Reserved	DMA2EN	DMA1EN	CCMDATARAMEN	Reserved	BKPSRAMEN	Reserved				Reserved				CRCEN	Reserved				GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN

Рис.6 – Карта регістрів RCC групи

Тепер, коли порт «А» / «D» тактується, нам необхідно задати режим його роботи. Це робиться за допомогою регістру «GPIO port mode register» (**GPIOx_MODER**), що описаний на ст. 157 (рис. 7). Його зміщення становить «0x00», а щоб настроїти пін 5 / 12 на «вивід» даних («output mode») з відповідного порту **GPIOx** (**x=A,B,C,D,...,H/...,K**) необхідно записати в біти «11,10» / «25,24» значення «01».

Для того ж, щоб встановити на виході високий або низький рівень, треба записати значення «1» відповідно в біти «5»/«12» або «21»/«28» регістру «GPIO port bit set/reset register» (**GPIOx_BSRR**). Інформацію про це можна знайти на ст. 160 (рис. 8).

Зверніть увагу на те, що запис логічної одиниці «1» у молодші біти 0 ÷ 15, позначені як «BSy, y=0÷15», призводить до встановлення («Set») високого рівня на відповідному виводі («піні») порту **GPIOx.y** (y=0÷15), а запис «1» у біти 16 ÷ 31, позначених «BRy, y=0÷15» – до скидання («Reset») у низький рівень тих самих виводів **GPIOx.y** (y=0÷15).

8.4 GPIO registers

This section gives a detailed description of the GPIO registers.

For a summary of register bits, register address offsets and reset values, refer to [Table 27](#).

The GPIO registers can be accessed by byte (8 bits), half-words (16 bits) or words (32 bits).

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. 7. - Опис регістру «GPIO port mode register»

8.4.7 GPIO port bit set/reset register (GPIOx_BSRR) (x = A..E and H)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bits 31:16 **BRy**: Port x reset bit y (y = 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Resets the corresponding ODRx bit

Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy**: Port x set bit y (y = 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Sets the corresponding ODRx bit

Рис. 8. - Опис регістру «GPIO port bit set/reset register»

2. Програма на мові ASSEMBLER

Отже, у нас є вся необхідна інформація і ми можемо починати писати програму. Додаймо декілька констант (рис. 9), щоб код легше було читати, а «магічних» чисел в ньому було менше.

```
1 ; constants
2 Stack_Size          EQU 0x00000400
3
4 RCC_AHB1_BASE       EQU 0x40023800
5 GPIOA_BASE          EQU 0x40020000
6
7 RCC_AHB1_ENR_OFFSET  EQU 0x30
8 GPIOA_MODER_OFFSET  EQU 0x00
9 GPIOA_BSRR_OFFSET   EQU 0x18
10
11 RCC_AHB1_GPIOA_CLOCK_BIT EQU 0
12 GPIOA_LED_PIN       EQU 5
13
```

Рис. 9 - Константи програми

Також додаймо код, що вмикає світлодіод перед входом до «вічного» циклу (рис. 10).

Якщо ви хочете випробувати роботу отриманої програми вже зараз, тобто у «чисто асемблерному варіанті», то закоментуйте рядки 37 і 57, які відносяться до матеріалу який буде розглянуто в наступному пункті². Далі побудуйте проект так, як ви це вже робили у лабораторній роботі №1, і відслідкуйте його виконання у Debugger. А щоб перевірити працездатність програми на налагоджувальній платі Nucleo, завантажте в неї отриманий код (цю дію ви також робили в першій роботі), і запустіть програму на виконання. Як що ви будете досліджувати програму у debugger (крок за кроком), то світлодіод повинен загорітися після виконання останньої команди з блоку встановлення високого рівню на виводі PA5 (див. Рис. 10):

```
str r1, [ro, #GPIOA_BSRR_OFFSET]
```

² Ця дія не є обов'язковою, оскільки ім'я підпрограми `main_loop`, що викликається у рядку 57 визначено директивою `EXTERN` у рядку 37 як зовнішнє, тобто таке, що завдане у іншому файлі. Тому асемблер не виявить синтаксичної помилки типу «не описане ім'я»


```

14      ; stack area
15      AREA STACK, NOINIT, READWRITE, ALIGN = 3
16
17      Stack_Mem SPACE Stack_Size ; reserve stack memory
18      __initial_sp ; initial stack pointer value
19
20      ; interrupt vector area
21      AREA RESET, DATA, READONLY
22
23      EXPORT __Vectors
24      __Vectors ; table of vectors
25      DCD __initial_sp
26      DCD Reset_Handler
27      __Vectors_End
28
29      ; code area
30      AREA |.text|, CODE, READONLY
31
32      EXPORT Reset_Handler
33      Reset_Handler PROC
34      B main
35      ENDP
36
37      EXTERN main_loop
38      main PROC ; main program entry point
39      ; enable GPIO_A clock
40      LDR R0, = RCC_AHB1_BASE
41      LDR R1, [R0, #RCC_AHB1_ENR_OFFSET]
42      ORR R1, R1, #(1 << RCC_AHB1_GPIOA_CLOCK_BIT)
43      STR R1, [R0, #RCC_AHB1_ENR_OFFSET]
44
45      ; set output mode for PA5
46      LDR R0, = GPIOA_BASE
47      LDR R1, [R0, #GPIOA_MODER_OFFSET]
48      ORR R1, R1, #(1 << (GPIOA_LED_PIN * 2))
49      BIC R1, R1, #(1 << (GPIOA_LED_PIN * 2 + 1))
50      STR R1, [R0, #GPIOA_MODER_OFFSET]
51
52      ; set high value on PA5
53      LDR R1, [R0, #GPIOA_BSRR_OFFSET]
54      ORR R1, R1, #(1 << GPIOA_LED_PIN)
55      STR R1, [R0, #GPIOA_BSRR_OFFSET]
56
57      BL main_loop
58      B main ; endless loop
59      ENDP
60      ALIGN
61
62      END

```

Закоментуйте ці рядки при
випробуванні асемблерного
варіанту програми

Вилучите ці рядки
при випробуванні
ASM+C проекту

Рис. 10 - Код для увімкнення світлодіоду

3.Увімкнення світлодіоду з програми на мові C

Давайте спробуємо також написати увімкнення світлодіоду на мові «C», щоб потім порівняти з уже написаним на мові асемблеру в дебагері. Саме для цього у 37 рядку ми кажемо компілятору, що десь за межами даного файлу є **EXTERN**-функція, яка називається «**main_loop**», а в рядку 57 – переходимо на цю функцію. Тепер створимо файл з розширенням «.c» і напишемо там код, зображений на рис. 11. Тут видно, що ми створили функцію «**main_loop**» і саме сюди перейде програма, після виклику функції з рядку 57 асемблерного файлу..

```
1  typedef unsigned long      uint32;
2  typedef unsigned long long uint64;
3
4  //  AHB1 bus enable register
5  #define RCC_AHB1_ENABLE_REG (uint32*)0x40023830
6
7  //  mode register
8  #define GPIOA_MODER        (uint32*)0x40020000
9
10 //  bit set/reset register (atomic)
11 #define GPIOA_BSRR         (uint32*)0x40020018
12
13 #define RCC_AHB1_GPIOA_CLOCK 0
14 #define GPIOA_LED_PIN        5
15
16 void main_loop()
17 {
18     *RCC_AHB1_ENABLE_REG |= (1 << RCC_AHB1_GPIOA_CLOCK);
19     *GPIOA_MODER &= ~(1 << (GPIOA_LED_PIN * 2 + 1));
20     *GPIOA_MODER |= (1 << (GPIOA_LED_PIN * 2));
21     *GPIOA_BSRR |= (1 << GPIOA_LED_PIN);
22
23     while (1) {
24     }
25 }
```

Рис. 11. Код програми на мові «C»

Взагалі-то тепер («для чистоти експерименту») у асемблерному файлі main.s рядки з 39 по 56 (див. рис. 10) бажано вилучити, або закоментувати. Однак ми залишимо їх,

щоб за один раз провести порівняння інструкцій вихідного асемблерного коду із інструкціями, які створюються при компіляції відповідних операцій на мові C.

Додайте до проекту створений файл **main_loop.c**³. Структура нашого проекту з двома файлами зображена на рис. 12.

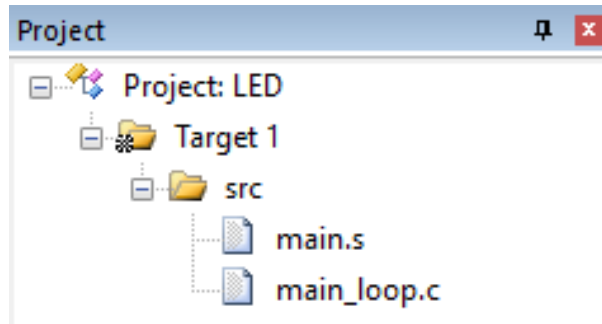


Рис. 12. Структура проекту

Щоб скомпілювати проект можна натиснути клавішу «F7» або на відповідну піктограму. Однак, як що проект вже було компільовано з іншими файлами у складі, то краще виконати команду <Rebuild>, натиснувши іншу кнопку (рис. 13).

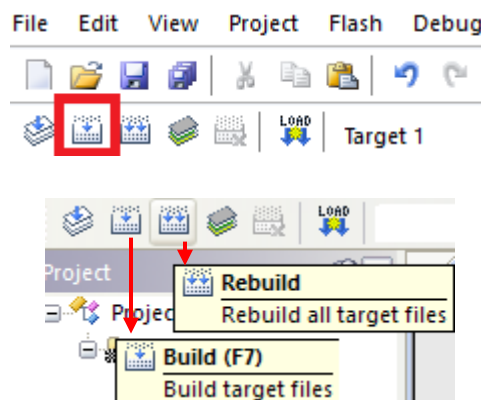


Рис. 13. Піктограма для побудови проекту

Якщо все зробити правильно, то має бути 0 помилок та 0 попереджень.

³ Назва файлу не обов'язково має збігатися з ім'ям функції, що описано у ньому.

Відлагодження може відбуватися як на платі, так і у симуляторі. Це залежить від того, що обрано у налаштуваннях проекту «Project -> Options for Target...» у вкладці «Debug» (рис. 14).

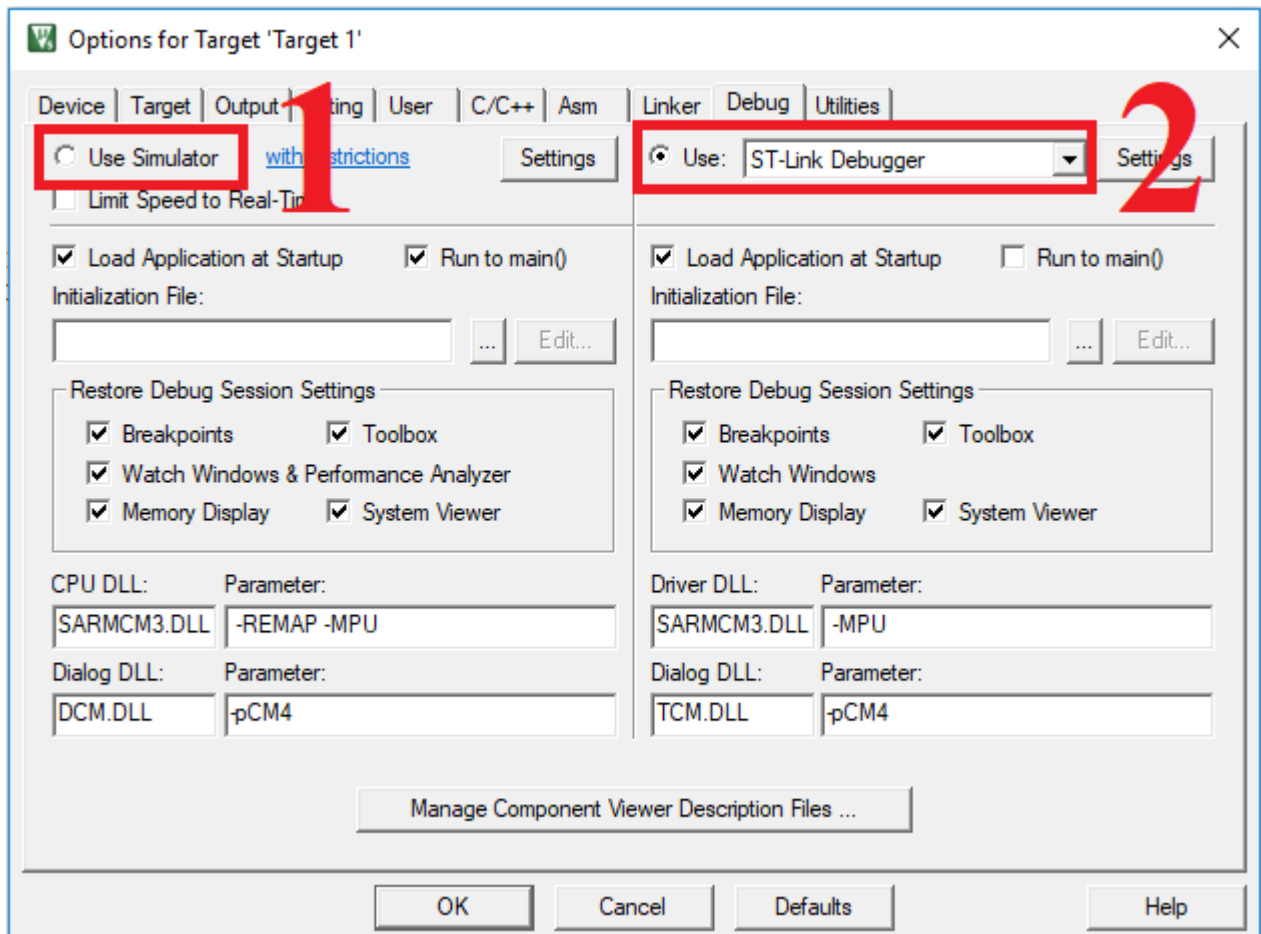



Рис. 14. Вибір способу відлагодження

Виберемо варіант 2, завантажимо програму ( або «Ctrl-F5») і почнемо покрокове відлагодження програми. При цьому відкриваються нові вікна (рис. 15).

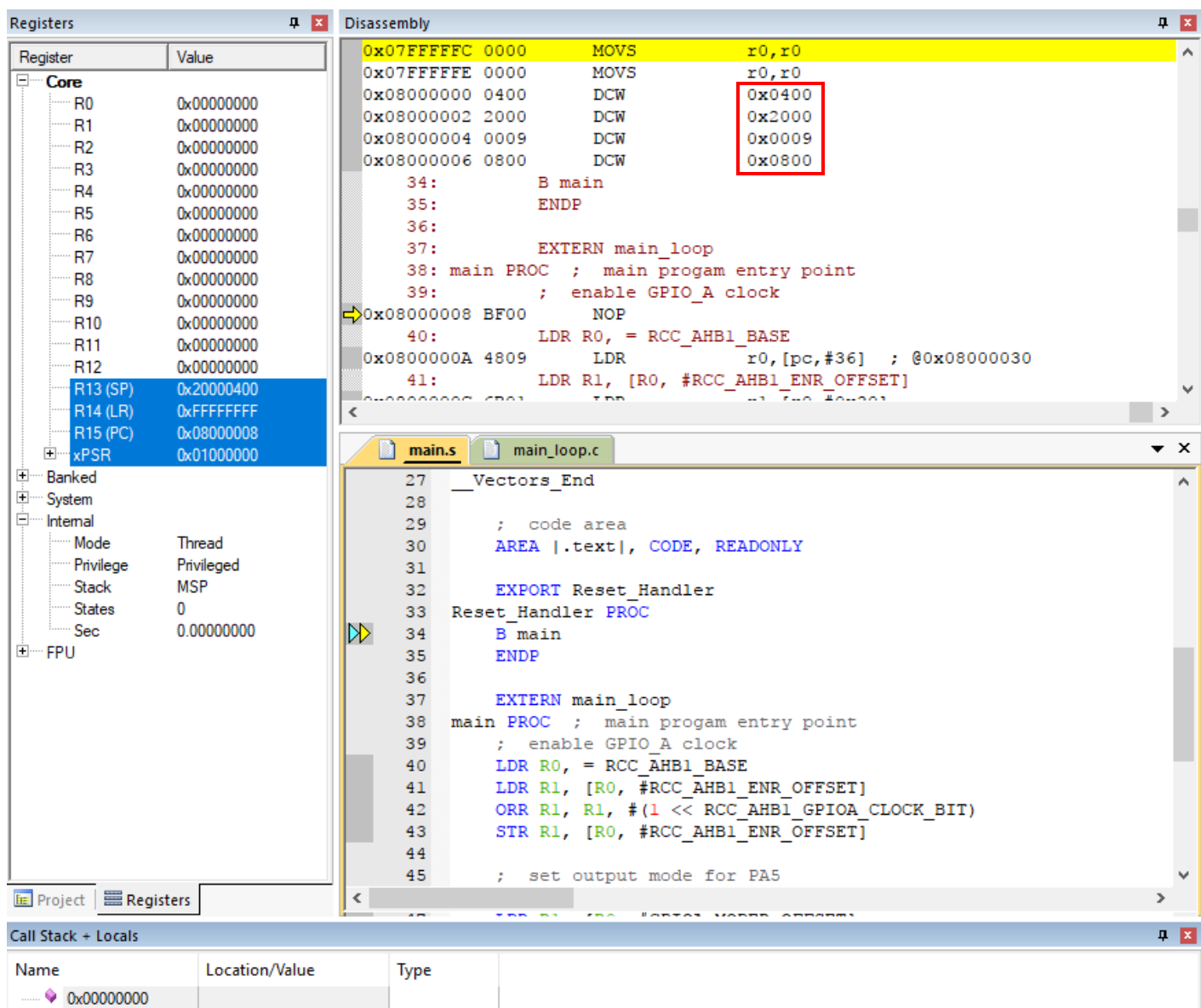


Рис. 15. Вікна з інформацією про відлагодження

Як і у роботі №1 праворуч зверху можна бачити вікно із результатом дизасемблювання скомпільованої програми, тобто саме ті інструкції, які і буде виконувати мікроконтролер. Жовта стрілка вказує на поточну інструкцію. Тут можна побачити, що починаючи з адреси «0x08000000», тобто у сегменті коду, записані 2 подвійних слова (у вигляді 4 слів, завданих командами DCW – Define Code Word). Це, як ви вже знаєте з першої роботи, – перші два елементи таблиці векторів переривань. Нагадаємо, що перше подвійне слово (перші два слова), це адреса вершини стеку, а друге подвійне слово (два наступних слова) – адреса, з якою починається виконання програми.

Зліва вікно, в якому можна подивитися поточний вміст кожного з регістрів. І як можна бачити в регістрах «SP» та «PC» («R13» та «R15» відповідно) вже знаходяться значення з таблиці векторів переривань.

У нижньої частині вікна Debugger, на відміну від роботи №1, розташовано дві вкладки з вихідними текстами файлів **main.s** та **main_loop.c**.

Поглянувши на дизасембльований код рядків 48-49 файлу «main.s» (рис. 16), можна побачити, що замість бітових зсувів, множень і додавань, завданих у так званому **flexible operand 2**:

#(1 << (GPIOA_LED_PIN*2)) та **#(1 << (GPIOA_LED_PIN*2+1))** компілятор вставив реальні числа «0x400» та «0x800», з якими безпосередньо і виконуються прості операції «порозрядного або» - **ORR** (рядок 48):

```
ORR r1,r1,#0x400 ;r1:=r1ORR #0x400
```

та «очищення біту» **Bit Clear – BIC** (рядок 49):

```
BIC r1,r1,#0x800 ;r1:=r1 AND NOT #0x800
```

Таким чином код залишається зрозумілим, а мікроконтролеру не доводиться робити зайві операції.

```
47:      LDR R1, [R0, #GPIOA_MODER_OFFSET]
0x08000016 6801      LDR      r1, [r0, #0x00]
48:      ORR R1, R1, #(1 << (GPIOA_LED_PIN * 2))
0x08000018 F4416180 ORR      r1, r1, #0x400
49:      BIC R1, R1, #(1 << (GPIOA_LED_PIN * 2 + 1))
0x0800001C F4216100 BIC      r1, r1, #0x800
```

Рис. 16. Дизасембльований код

. Натискаючи клавішу «F11» будемо виконувати інструкції по черзі («крок за кроком»).

На рис. 17 у вікні Disassembly показано результат виконання програми після встановлення тактування порту GPIOA, тобто після встановлення в одиницю $RCC_AHB1ENR[0] = GPIOAEN$ командою STR за адресою 0x8000012.

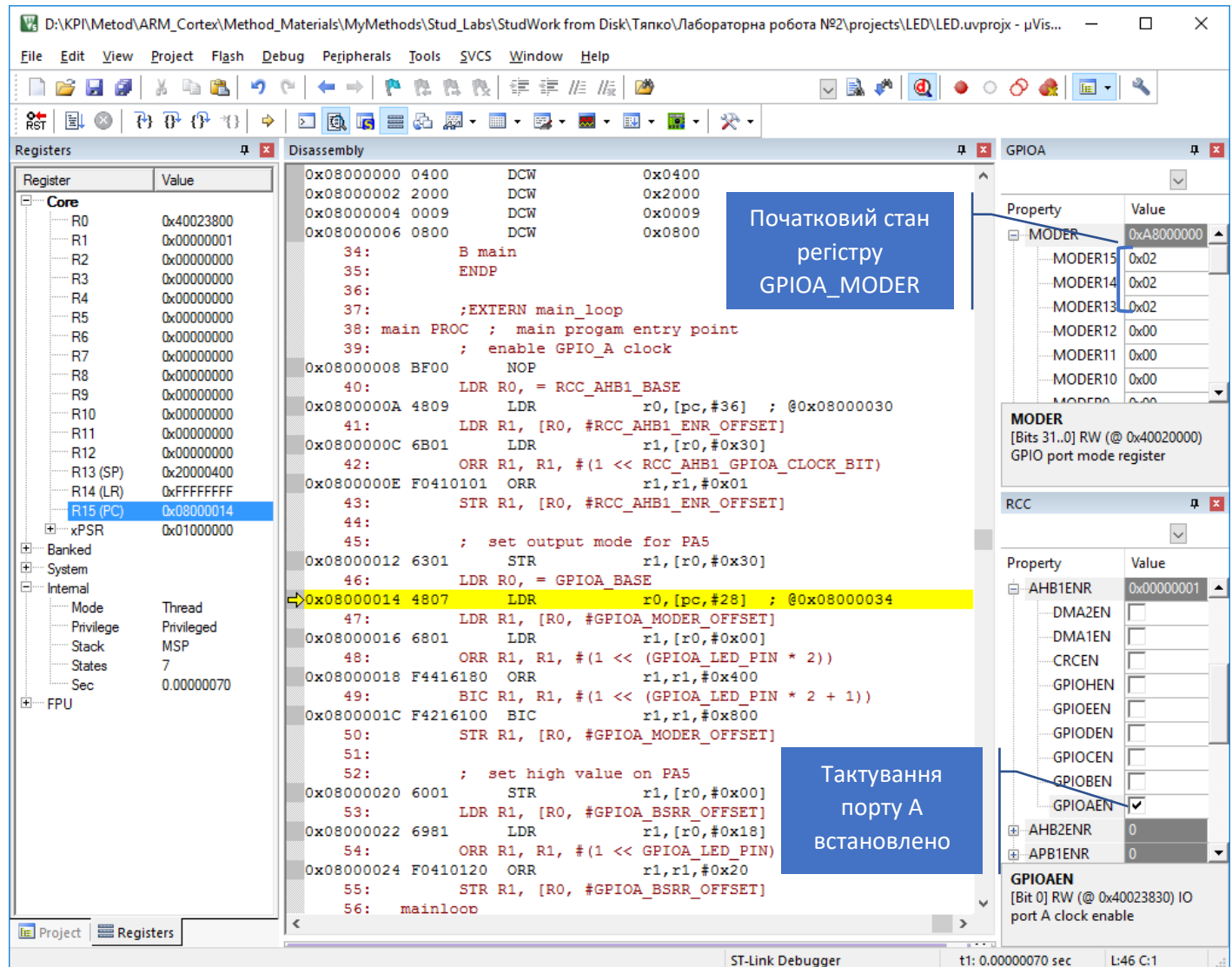
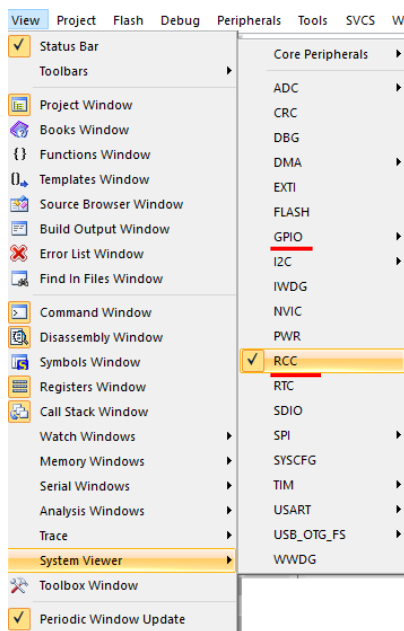
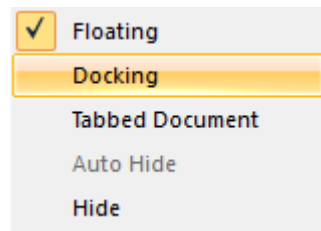


Рис. 17. Відлагодження асемблерної частини програми.
Тактування порту GPIOA включено.



Для нагляду за станом регістрів `RCC_AHB1_ENR` і `GPIO_MODER` командою `View→System Viewer` можна додати відповідні вікна. А маніпулюючи інструментами розміщення вікон, можна встановити їх у зручному місці вікна Debugger. На рис.17 вони встановлені так званим способом Docking, який задається з контекстного меню, що відкривається «правим кліком» мишки и по заголовку вікна.



Праворуч зверху (див. рис. 17) бачимо початкове налаштування пінів `PA15`, `PA14`, `PA13` регістру `GPIOA_MODER`. Поясніть стан цих виводів.

Виконайте фрагмент програми з налаштуванням виводу `PA5` на «вивід» `GPIOA_MODER[11:10] = MODER5=01`, тобто зупиніть виконання програми після виконання команди `STR` – на адресі `0x8000022` (рис. 18).

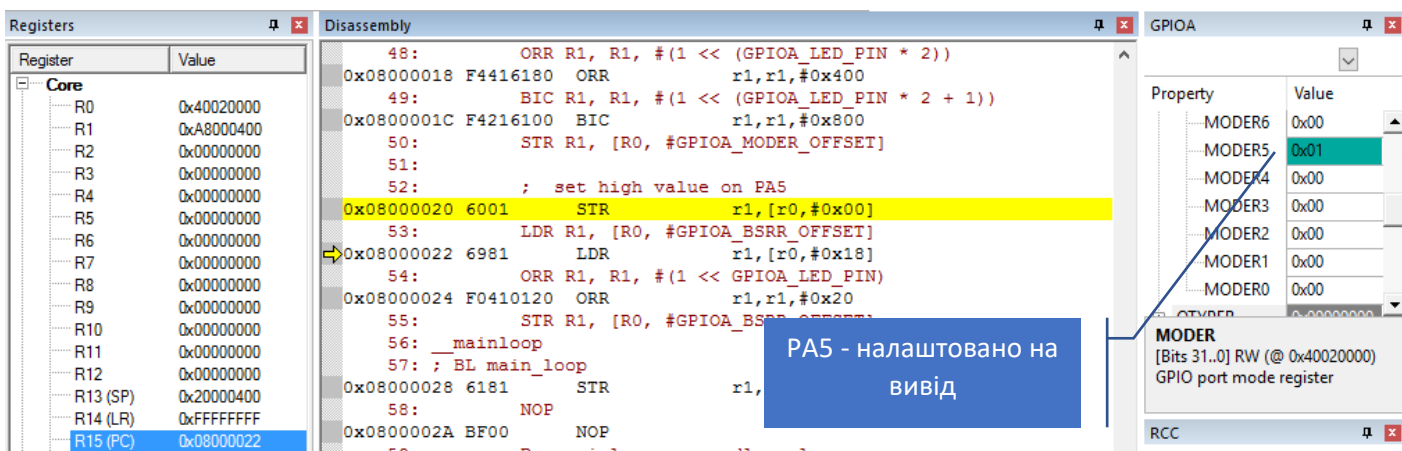


Рис. 18. Порт `PA5` налаштовано на вивід

На рис. 19 показано вікно дізасемблера при завершенні всього фрагменту включення світлодіоду (встановлення високого рівню «1» на виході порту `PA5`).

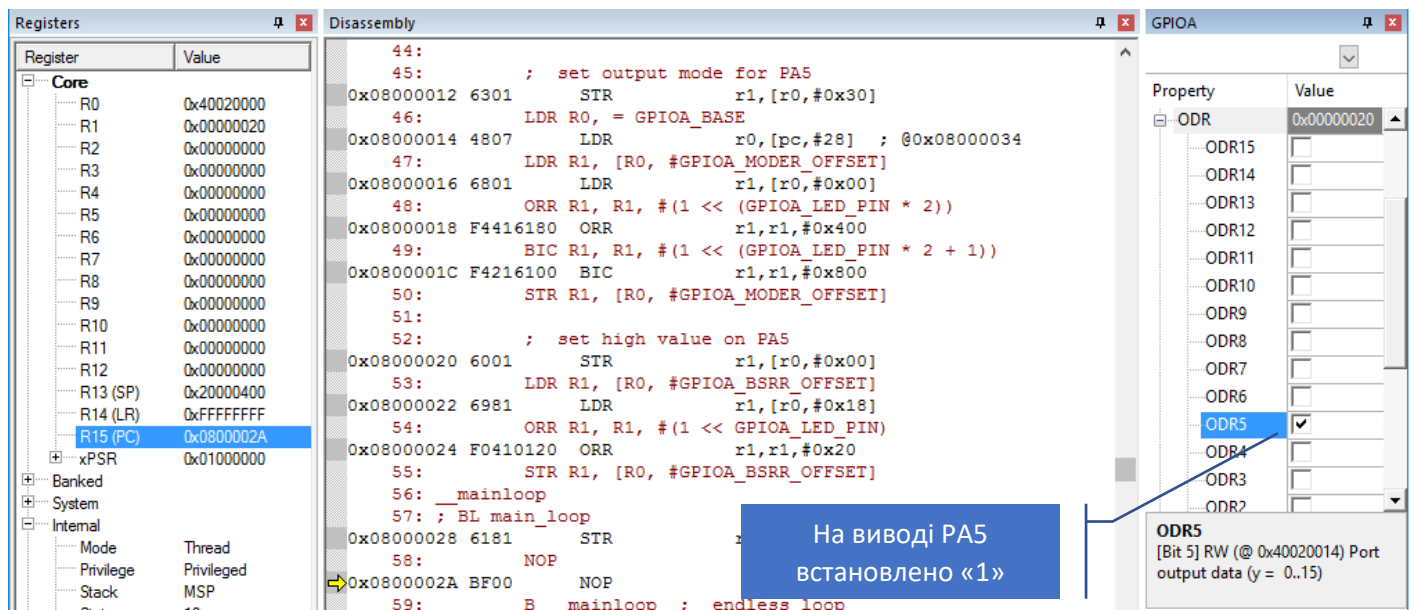


Рис. 19 - Завершення асемблерної частини програми

При цьому світлодіод LED2 засвітиться зеленим світлом.

Зверніть увагу на те, що, хоча ми маніпулювали битому регістру BSRR, стан відповідного біта регістра ODR також змінилося.

Якщо, виконуючи команди, дійти до частини коду, написаної на C, то у вікні Disassembly можна побачити багато «магічних» чисел не дуже зрозумілих людині, але зручних для виконання безпосередньо для мікроконтролера. Однак загалом цей код схожий на наш. Просто свої числові константи компілятор записав як частину програми, в такій ділянці пам'яті, до якої, за нормальних обставин, мікроконтролер не дійде, і не почне інтерпретувати їх як коди команд. Звертається компілятор до них завдяки зміщенню відносно регістру «PC» записаному у квадратних дужках в операції завантаження в регістр («LDR»).

Поглянемо на рис. 20 на ньому зображено першу інструкцію з функції «**main_loop**» та результат її дизасемблювання. Видно, щось завантажується у регістр «R0» з пам'яті за адресою на 44 байти більшою ніж поточне значення регістру «PC». Також, як коментар, правіше, після знаку «собачка», написана вже порахована та адреса

«0x08000068». Якщо в цьому ж вікні дизасемблеру перейти до цієї адреси (рис. 21), то можна побачити два подвійних слова, це наша константа «RCC_AHB1_ENABLE_REG», яка мала значення адреси однойменного регістру в пам'яті мікроконтролера, отже в регістрі «R0» тепер знаходиться ця адреса. Далі йде вже зчитування значення за адресою, що зберігається в регістрі «R0», тобто значення самого регістру «RCC_AHB1_ENABLE_REG» і записування прочитаних даних у регістр «R0». Потім відбувається операція «логічного побітового І» для регістру «R0» з константою «1». Значення цієї константи отримане шляхом зсуву «1» на кількість бітів, записану в константу «RCC_AHB1_GPIOA_CLOCK», тобто на «0» бітів. Тепер адреса регістру «RCC_AHB1_ENABLE_REG» вже записується в регістр «R1», і останньою операцією значення з регістру «R0» записується за адресою, яка зберігається в регістрі «R1».

```

18:          *RCC_AHB1_ENABLE_REG |= (1 << RCC_AHB1_GPIOA_CLOCK);
0x08000038 480B      LDR          r0, [pc, #44] ; @0x08000068
0x0800003A 6800      LDR          r0, [r0, #0x00]
0x0800003C F0400001  ORR          r0, r0, #0x01
0x08000040 4909      LDR          r1, [pc, #36] ; @0x08000068
0x08000042 6008      STR          r0, [r1, #0x00]

```

Рис. 20 – Дизасемблювання інструкції написаної на мові «C»

```

0x08000068 3830      DCW          0x3830
0x0800006A 4002      DCW          0x4002

```

Рис. 21 - Числові константи, записані як частина програми

На рис. 22 надано результат дизасемблювання всієї процедури **main_loop**.

По завершенні виконання всіх команд програми, побачимо, що стан регістрів GPIOA той же, що і у асемблерному варіанті програми. До речі, як що ви у вікні GPIOA відкриєте регістр BSRR, то виявите, що стан його бітів (у тому числі і BS5) не змінилося! Але ж біт ODR5 (PA5) очікувано встановлено у «1» і зелений світлодіод LED2 знову засвітився.

The screenshot displays the ST-Link Debugger interface with the following components:

- Registers:** Shows the current state of registers. R15 (PC) is highlighted at 0x0800003C.
- Disassembly:** Shows the assembly code for the `main_loop` procedure. Key instructions include:
 - `*RCC_AHB1_ENABLE_REG |= (1 << RCC_AHB1_GPIOA_CLOCK);`
 - `*GPIOA_MODE &= ~(1 << (GPIOA_LED_PIN * 2 + 1));`
 - `*GPIOA_MODE |= (1 << (GPIOA_LED_PIN * 2));`
 - `*GPIOA_BSRR |= (1 << GPIOA_LED_PIN);`
- GPIOA:** Shows the configuration of the GPIOA output register (ODR5) set to 1.
- Code Editor:** Shows the corresponding C code for the `main_loop` procedure.

Рис. 22 - Результат дизасемблювання процедури `main_loop`
Результат роботи програми зображений на рис. 23.

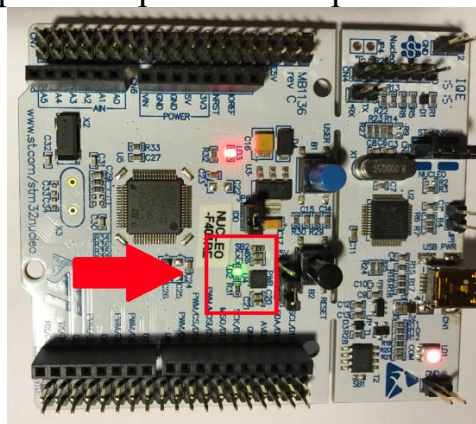


Рис. 23 - Результат роботи програми у NUCLEO

Сподіваюсь, що з набутими знаннями та за допомогою інструментів створення і відлагодження програм, які надає IDE «Keil μVision5» з MDK-ARM Keil, ви зможете самостійно створити подібні проекти та здійснити аналіз роботи відповідних програм, що складає суть аудиторного і індивідуального завдань.

4.Завдання для виконання в аудиторії

1. Створить вихідні файли `main.s` і `main_loop.c` проекту для плати NUCLEO відповідно до цієї інструкції.
2. Виконайте всі етапи створення і відлагодження проекту.

Індивідуальні завдання

3. Переробіть вихідні файли `main.s` і `main_loop.c` для проекту із платою Discovery.
Варіанти:

- 3.1. Увімкніть та здійсніть помітне для очей мерехтіння світлодіоду LED 3.
- 3.2. Увімкніть та здійсніть помітне для очей мерехтіння світлодіоду LED 4
- 3.3. Увімкніть та здійсніть помітне для очей мерехтіння світлодіоду діод LED 5
- 3.4. Увімкніть та здійсніть помітне для очей мерехтіння світлодіоду LED 6
- 3.5. Засвічуйте світлодіоди по чергово із деякою помітною для очей затримкою.

Примітка. Затримку завдайте програмно.

4. Варіанти завдань для лабораторних макетів пристроїв на основі плати **NUCLEO/Discovery та стенду №1** (кнопки та світлодіоди).
 - 4.1. Здійсніть по чергове увімкнення світлодіодів LED 1, Led 2, Led 3
 - 4.2. Увімкнення світлодіоду (LED №) відповідно до натиснутої кнопки
 - 4.3. Зміна послідовності увімкнення світлодіодів натисненням кнопки
 - 4.4. Зменшення/збільшення затримки горіння світлодіоду натисненням кнопок SW1 ("L") та SW3 ("R").
Кнопка SW2 ("OK") підтверджує вибір значення величини затримки.

Примітка. Послідовність зміни тривалості затримки
зменшення "OK" → "L" {"L" ... "L"} → "OK" ;
збільшення "OK" → "R" {"R" ... "R"} → "OK";
повернення значення затримки до початкового стану

5. Варіанти завдань для лабораторних макетів пристроїв на основі плати **NUCLEO/Discovery та стенду №2** (кнопки та семісегментні індикатори).

Більш детально варіанти завдань див. у файлі «Теми і варіанти робіт.docx»

Сворють звіт з виконання «аудиторного» і індивідуального завдань.

6. Додайте у звіт з виконання роботи «скріншоти» з основних етапів відлагодження проектів і ваші коментарі.

5.Список літератури

1. Лекція 7. GPIO/- презентація на Google Disk: \\Lessons\ Less 7 GPIO
2. User_manual_Nucleo(UM1724).pdf – Руководство пользователя по плате Nucleo /–документ на Google Disk: \\Matherials\Documentations\STM32 NUCLEO
3. Reference_manual_STMF401(RM0368).pdf – Руководство по STM32F401 /– документ на Google Disk: \\Matherials\Documentations\STM32 NUCLEO [Електронний ресурс] Режим доступа: http://www.st.com/web/en/resource/technical/document/reference_manual/DM00096844.pdf
4. Discovery_User_manual(UM1472).pdf – Руководство пользователя по плате Discovery /– документ на Google Disk:\\Matherials\Documentations\STM32 Discovery
5. Reference Manual _STM405-07-43xxx (RM0090) – Руководство по STM32F407 /–документ на Google Disk:\\Matherials\Documentations\STM32 Discovery [Електронний ресурс] Режим доступа: www.st.com/resource/en/reference_manual/dm00031020.pdf
6. Programming_manual_STM32F3, STM32F4 and STM32L4 (PM0214).pdf – Руководство по программированию МК STM32xx /–документ на Google Disk:\\Matherials\Documentations\Cortex-M4 [Електронний ресурс] Режим доступа: www.st.com/resource/en/programming_manual/dm00046982.pdf
7. ARMv7-M_ARM Architecture (RM 0403).pdf – Описание архитектуры ARMv7-M /–документ на Google Disk:\\Matherials\Documentations\Cortex-M4
8. Лабораторна робота №1. Ваш перший проект для STM32. /– Google Disk:\\ 1 семестр 2017\Labs\Lab1
9. Начинаем изучать Cortex-M на примере STM32 (Часть 1) /[Електронний ресурс] Режим доступа: <https://habrahabr.ru/post/216843/>
- 10.Олег Вальпа «Современные 32-разрядные ARM-микроконтроллеры серии STM32: порты общего назначения GPIO» / CHIP NEWS Украина, #9 (129), ноябрь, 2013, с. 72 – 75.