Лабораторна робота №º11
з дисципліни "Архітектура комп'ютера"
для студентів напряму 6.050103 "Програмна інженерія"

Використання цифрових портів мікроконтролера STM32F401RB

МЕТА РОБОТИ: опанувати роботу з цифровими портами мікроконтролера STM32F401RB; розвинути навики складання програми мовою С для виведення і введення сигналів через цифрові порти; відтранслювати програму, складену відповідно до свого варіанту в середовищі програмування Keil µVision MDK-ARM; виконати моделювання схеми з мікроконтролером в системі Proteus.

Цифрові порти введення-виведення загального призначення

Кожний мікроконтролер має цифрові лінії введення або виведення. Кожну таку лінію можна програмним шляхом конфігурувати як цифровий вхід, або цифровий вихід, і використовувати для взаємодії із зовнішніми схемами. Для зручності використання лінії введення-виведення об'єднані в порти по 16 ліній. Такі порти називають портами введення-виведення загального призначення. В англомовній літературі лінії введення-виведення прийнято називати терміном GPIO - General-Purpose Input / Output.

До ліній, сконфігурованих як цифрові входи, під'єднують механічні кнопки, вимикачі, контакти реле, давачі тощо. За допомогою таких ліній мікроконтролер отримує інформацію від під'єднаних до нього пристроїв.

Лінії, сконфігуровані як цифрові виходи, дозволяють видавати сигнали керування для під'єднаних до мікроконтролера пристроїв. Таким сигналом можна безпосередньо засвітити світлодіод, а через відповідну схему можна запустити електродвигун, увімкнути електромагнітне реле або лампу розжарювання тощо.

Конфігурування ліній введення-виведення

Для того щоб почати використовувати лінії введення-виведення, потрібно попередньо конфігурувати їх відповідним чином.

На найнижчому рівні робота з портами введення-виведення (та й з усіма іншими периферійними пристроями) здійснюється за допомогою спеціальних регістрів мікроконтролера. Ці регістри доступні як комірки пам'яті, розташовані за певними адресами. Знаючи ці адреси (вони описані в документації на мікроконтролер), можна записувати в регістри певні значення, задаючи необхідну конфігурацію. Через інші регістри можна отримувати дані від периферійних пристроїв.

Портів введення-виведення загального призначення (GPIO – General Purpose Input Output) може бути різна кількість, у нашому випадку є 5 портів GPIO: A, B, C, D і E.

Кожен порт є 16-бітовим (має 16 ліній) і використовує десять 32-бітових регістрів:

чотири регістри конфігурації (GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR);
два регістри даних (GPIOx_IDR, GPIOx_ODR);
регістр встановлення/скидання (GPIOx_BSRR);
регістр блокування (GPIOx_LCKR);
два регістри вибору альтернативної функції (GPIOx_AFRH, GPIOx_AFRL)
(x заміняє ім'я порту).

Якщо світлодіоди керуються лише портом D, надалі замість абстрактної назви GPIOx застосовуємо GPIOD. GPIOD_MODER дозволяє сконфігурувати напрямок даних. Для роботи зі світлодіодом слід сконфігурувати порт на вихід. З документації мікроконтролера (пункт 8.4.1), фрагмент якої показано на рис. 1, видно, що для цього слід задати значення 01 (тут значенням керує комбінація з 2 бітів).

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. 1. Можливі значення регістра GPIOD_MODER

Наразі приймемо, що GPIOD_OTYPER = 0.

Регістр GPIOD_SPEEDR відповідає за швидкість (є 4 рівні швидкості, відповідно, рівень швидкості кодується 2 бітами). Значення 00 відповідає найменшій швидкості, достатньо задати значення 0 для всіх виводів. Тому GPIOD_SPEEDR = 0.

Оскільки напрям передавання порту – на вихід, то з регістрів даних потрібно налаштовувати GPIOD_ODR (Output Data Register), а не GPIOD_IDR (Input Data Register).

У 32-бітовому регістрі GPIOD_ODR старші 16 бітів не використовуються (зарезервовані), а молодші 16 відповідають якраз виводам 16-розрядного порту (рис. 2).

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..I/J/K).

Рис. 2. Регістр GPIOx_ODR

Щоб засвітити світлодіод, треба подати 1 на відповідний пін. Відповідно якщо GPIOD_ODR = 0x9000, це відповідає числу 1001000000000000 у двійковій системі, тобто, згідно зі схемою на рис. 3, таким чином засвічуємо синій і зелений світлодіоди.

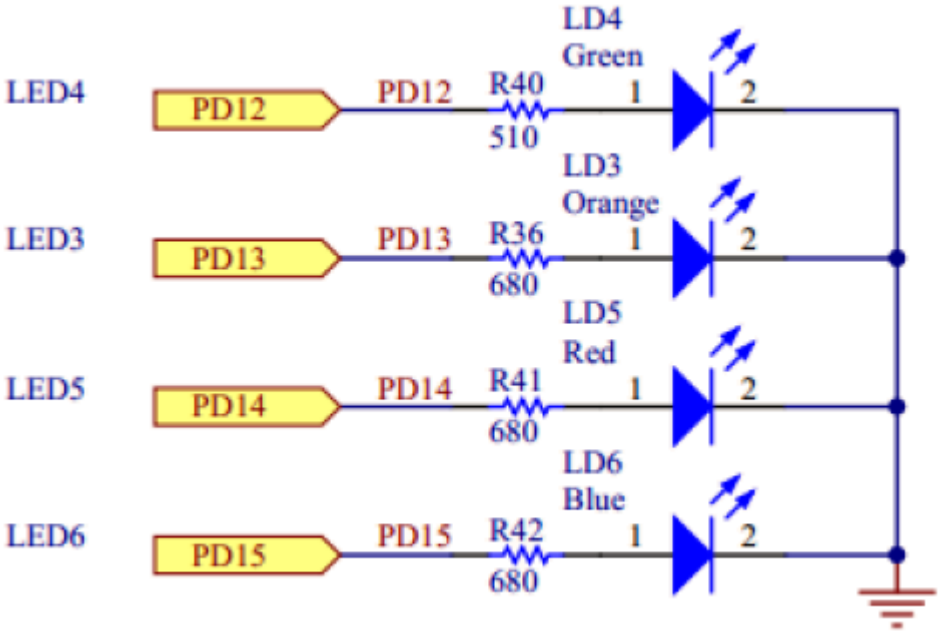


Рис. 3. Підключення світлодіодів до виводів/пінів порту D

PD – це порт D, числа 12, 13, 14, 15 – це номери виводів/пінів/ніжок цього порту.

Для забезпечення мінімального споживання енергії периферія у початковому стані відключена від живлення. Жоден з периферійних модулів мікроконтролера не працюватиме, поки на нього не надходитимуть тактові імпульси. Тому перш ніж щось робити з тим чи іншим периферійним модулем, спочатку слід увімкнути тактування.

Власне, це і робиться наступний рядком (див. лістинг 1):

```
RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN;
```

RCC – Reset & Clock Control – блок регістрів для керування тактуванням. AHB1ENR – один з регістрів у цьому блоці (AHB1 – шина, ENR – Enable Register).

У ARM Cortex є шини AHB (ARM Hi-Speed Bus) та APB (ARM Peripheral Bus). Отже, AHB1 – одна з шин. Молодші 9 бітів регістра AHB1ENR відповідають за тактування 9 портів (наймолодший біт – за порт A, наступний за ним – за порт B тощо). Очевидно, що за порт D відповідає біт на 4-ій позиції справа. Щоб увімкнути тактування для порту D , не порушивши тактування інших портів, слід застосувати побітове «або», що і зроблено у коді-прикладі.

Константа RCC_AHB1ENR_GPIODEN є числом ….000001000. Щоб дізнатися, яким регістром слід скористатися для тактування певного порту, слід заглянути або у файли бібліотеки CMSIS, або у документацію на мікроконтролер (Reference Manual). У документації ці дані містяться у пункті 7.3.25 – RCC Register Map. Зі фрагмента таблиці, поданої у цьому пункті (рис. 4), видно, що тактування порту D вмикається регістром AHB1ENR.

0x30	RCC_ AHB1ENR	Reserved	OTGHSULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Reserved	DMA2EN	DMA1EN	CCMDATARAMEN	Reserved	BKPSRAMEN	Reserved	CRCEN	Reserved	GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN

Рис. 4. Фрагмент документації для визначення регістра тактування певного порту

Для полегшення праці програміста багато розробників мікроконтролерів надають спеціальні бібліотеки функцій для роботи з периферійними пристроями.

Бібліотека CMSIS

CMSIS – бібліотека, стандартна для всіх МК з ядром ARM Cortex. Стандартизується ARM Ltd. Різні виробники МК з цим ядром доповнюють CMSIS файлами з описом периферійних модулів, специфічних для МК, які вони випускають.

Бібліотека надає зручний доступ до периферійних модулів, її застосування спрощує процес розроблення.

Бібліотека CMSIS містить дві частини:

1) спільну для усіх мікроконтролерів з ядром ARM Cortex M, файли:

core_cmN.c

core_cmN.h;

N позначає цифру, що характеризує покоління Cortex. Оскільки мікроконтролер STM32F401RB має архітектуру Cortex M-4, то потрібно застосувати файли core_cm4.c + core_cm4.h;

2) специфічну для виробника – файли:

system_<конкретний_МК>.c

system_<конкретний_МК>.h

<конкретний_МК>.h, саме цей останній файл підключають у проєкт, він, у свою чергу, включає core_cmN.h і system_<конкретний_МК>.h.

Отже матимемо такий набір файлів:

system_stm32f4xx.c

system_stm32f4xx.h

stm32f4xx.h

Програма, що використовує функції бібліотеки CMSIS, наведена в лістингу 1.

Лістинг 1

```
#include <stm32f4xx.h>
uint16_t delay_c = 0;
void SysTick_Handler(void){
        if(delay_c > 0)
                delay_c--;
}
void delay_ms(uint16_t delay_t){
        delay_c = delay_t;
        while(delay_c){};
}
int main (void){
  SysTick_Config(SystemCoreClock/1000);
        RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN;
        GPIOD->MODER= 0x55000000;
        GPIOD->OTYPER = 0;
        GPIOD->OSPEEDR = 0;
        while(1){
```

```
        GPIOD->ODR = 0x9000;
            delay_ms(500);
            GPIOD->ODR = 0x0000;
            delay_ms(500);
        }
    }
```

Слід зазначити, що будь-який файл повинен закінчуватися порожнім рядком.

Щоб відбувалось миготіння світлодіодів, доведеться періодично засвічувати та гасити їх. А щоб між моментами ввімкнення та вимкнення людина встигла щось побачити, потрібно вичекати певну паузу. Для відрахування точних (реальних!) часових інтервалів використовується таймер. У всіх мікроконтролерів ARM Cortex, зокрема і в мікроконтролера STM32F401RB, є 24-бітовий системний таймер, SysTick. Таймер здійснює зворотний відлік від заздалегідь встановленої величини до нуля. Величина задається функцією SysTick_Config, яка має лише один параметр. Для опрацювання ситуації, коли заданий проміжок сплив, застосовується SysTick_Handler. У тілі цього обробника пишемо нашу реакцію на завершення заданого часового проміжку.

Для підрахунку потрібної кількості мілісекунд delay_t напишемо функцію delay_ ms, яка «заблокує» мікроконтролер у циклі, не даючи йому виконувати нічого іншого, поки системний таймер не відрахує потрібну кількість часу. До прикладу, виклик delay_ms(1000) змусить мікроконтролер чекати час удвічі більший, ніж виклик delay_ms(500).

Для того, щоб числа 500 і 1000 стали означати реальну, фізичну кількість мілісекунд, достатньо щоб SysTick_Handler спрацьовував раз на мілісекунду і зменшував змінну, за рахунок якої функція delay_ ms стримує мікроконтролер, не даючи йому виконувати команди, розміщені в коді після виклику delay_ ms. Щоб SysTick_Handler викликався рівно раз на мілісекунду, системний таймер має робити зворотний відлік до нуля, починаючи з такої кількості тактових імпульсів, сумарна тривалість яких становить якраз одну мілісекунду. Як ми знаємо тактова частота – це кількість тактових імпульсів на секунду. Значить, тактова частота, поділена на 1000, є тим числом, яке потрібно передати функції SysTick_Config. Залишається з'ясувати, яка тактова частота в нас є. Вона міститься в системній змінній з іменем SystemCoreClock.

Бібліотека SPL

Standard Peripheral Library (**SPL**) - бібліотека, яка розробляється компанією STMicroelectronics і призначена для полегшення програмування мікроконтролерів виробництва цієї компанії.

Бібліотека SPL поширюється у вигляді zip-архіву з підкаталогами CMSIS і STM32Fyxx_StdPeriph_Driver (її можна завантажити з сайту компанії STMicroelectronics).

У каталозі STM32Fxxx_StdPeriph_Driver є два підкаталоги – inc (із .h-файлами) та src (із .c файлами), імена файлів мають структуру stm32fyxx_<ім’я_модуля>.h/.c. Для роботи з тим чи іншим периферійним модулем досить підключити .h-файл з іменем цього модуля і файл stm32fyxx_rcc.h, зі вже відомою нам метою – для тактування.

До прикладу, для роботи з портами введення/виведення загального призначення слід підключити stm32f4xx_gpio.h.

Для конфігурації будь-якої периферії у SPL визначені структури вигляду <ім'я_периферії>_InitTypeDef (структура визначена у відповідному .h-файлі).

Для портів введення/виведення є структура GPIO_InitTypeDef, наведена на рис. 5.

```
//stm32f10x_gpio.h
typedef struct
{
  uint16_t GPIO_Pin;
  GPIOSpeed_TypeDef GPIO_Speed;
  GPIOMode_TypeDef GPIO_Mode;
}GPIO_InitTypeDef;

typedef enum
{
  GPIO_Speed_10MHz = 1,
  GPIO_Speed_2MHz,
  GPIO_Speed_50MHz
}GPIOSpeed_TypeDef;

typedef enum
{ GPIO_Mode_AIN = 0x0,
  GPIO_Mode_IN_FLOATING = 0x04,
  GPIO_Mode_IPD = 0x28,
  GPIO_Mode_IPU = 0x48,
  GPIO_Mode_Out_OD = 0x14,
  GPIO_Mode_Out_PP = 0x10,
  GPIO_Mode_AF_OD = 0x1C,
  GPIO_Mode_AF_PP = 0x18
}GPIOMode_TypeDef;
```

Рис. 5. Структура GPIO_InitTypeDef і можливі значення її полів

Слід створити екземпляр цієї структури і задати значення усім її полям. Значення – це елементи перелічень (рис. 5). До прикладу:

GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

Після цього слід викликати функцію GPIO_Init, вказавши, для якого саме порту задано конфігурацію в екземплярі структури.

GPIO_Init(GPIOC, &GPIO_InitStructure);

Однак, насамперед потрібно подбати про тактування, яке у SPL здійснюється однією з функцій:

RCC_AHBPeriphClockCmd(RCC_AHBPeriph_PPPx, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_PPPx, ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_PPPx, ENABLE);

PPP – це ім'я периферійного модуля, у даному випадку GPIOD. Знову ж, до якої шини підключено потрібний порт, можна дізнатися з документації або з коментарів у файлах stm32fyxx_rcc.h/.c.

Для роботи з портами у SPL є функції:

GPIO_SetBits(GPIOx, GPIO_Pin_y); /* встановлення біта GPIO_Pin_y порту GPIOx в 1*/

GPIO_ResetBits(GPIOx, GPIO_Pin_y); /* встановлення біта GPIO_Pin_y порту GPIOx в 0*/

Програма, що використовує функції бібліотеки SPL, наведена в лістингу 2.

Лістинг 2

```
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"

void SysTick_Handler(void);
void delay_ms(uint16_t delay_t);
volatile uint16_t delay_c = 0;

void delay_ms(uint16_t delay_t) {
	delay_c = delay_t;
	while (delay_c) { }
}

int main(void) {
	GPIO_InitTypeDef GPIO_InitStructure;
	SysTick_Config(SystemCoreClock/1000);
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
	GPIO_Init(GPIOD, &GPIO_InitStructure);

	while (1) {
		GPIO_SetBits(GPIOD, GPIO_Pin_14);
		delay_ms(500);
		GPIO_ResetBits(GPIOD, GPIO_Pin_14);
		GPIO_SetBits(GPIOD, GPIO_Pin_12);
		delay_ms(500);
		GPIO_ResetBits(GPIOD, GPIO_Pin_12);
		GPIO_SetBits(GPIOD, GPIO_Pin_15);
		delay_ms(500);
		GPIO_ResetBits(GPIOD, GPIO_Pin_15);
		GPIO_SetBits(GPIOD, GPIO_Pin_13);
		delay_ms(500);
		GPIO_ResetBits(GPIOD, GPIO_Pin_13);
	}
}
```

Слід зазначити, що будь-який файл повинен закінчуватися порожнім рядком.

Створення проєкту в Keil uVision.

Спробуємо створити новий проєкт у Keil uVision “з нуля”. Для початку слід запустити Keil uVision (одним із звичних способів). Якщо програма запускається не

вперше і в ній раніше вже був відкритий якийсь проєкт, його слід закрити перед створенням нового проєкту (Project => Close Project). Слід виконати Project => New µVision Project. Появиться діалогове вікно, в якому вводимо ім'я проєкту і місце його розташування на диску. Проєкт потрібно зберігати в окремому каталозі, оскільки він складається з багатьох різних файлів, тож у протилежному випадку не уникнути плутанини. Наступний крок – у діалозі "Select Device for Target <ім'я>" вибрати мікроконтролер, для якого збираємося писати мікропрограмне забезпечення (рис. 2). Може трапитися, що «дерево мікроконтролерів» у діалозі "Select Device for Target <ім'я>" пусте (в залежності від того, як встановлювався Keil uVision). У цьому випадку проєкт зберегти не вдасться, адже вибір конкретного мікроконтролера для проєкту обов'язковий. Це пояснюється тим, що кожен мікроконтролер «розуміє» свій набір команд (якщо невідомий мікроконтролер, невідомо, як компілювати код). У випадку «пустого дерева» слід вибрати "Pack Installer" і встановити усе, чого бракує.

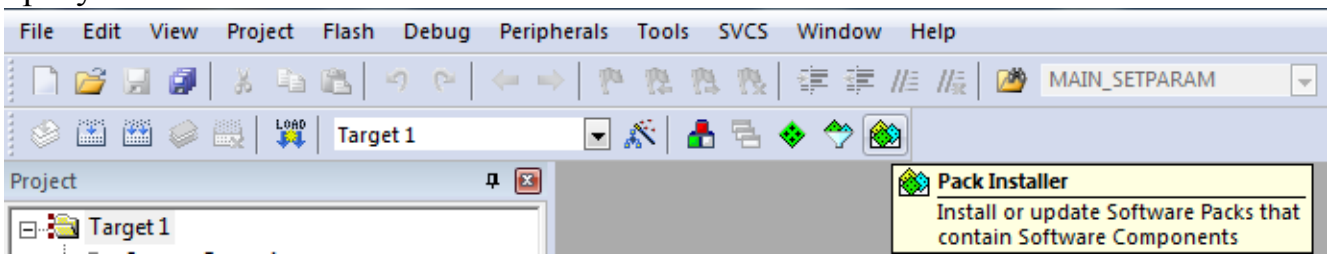


Рис. 2. Виклик вікна для встановлення додаткових компонент

Якщо ж мікроконтролер (target device) успішно вибраний, побачимо вікно, зображене на рис. 3.

У вікні, що появиться, відмітимо CMSIS => CORE і Device => Startup.

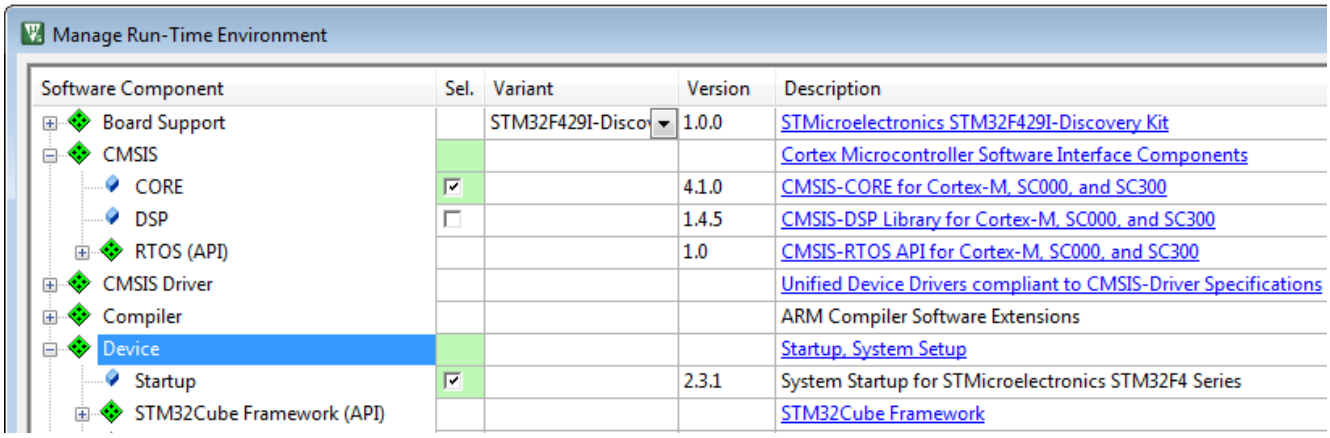
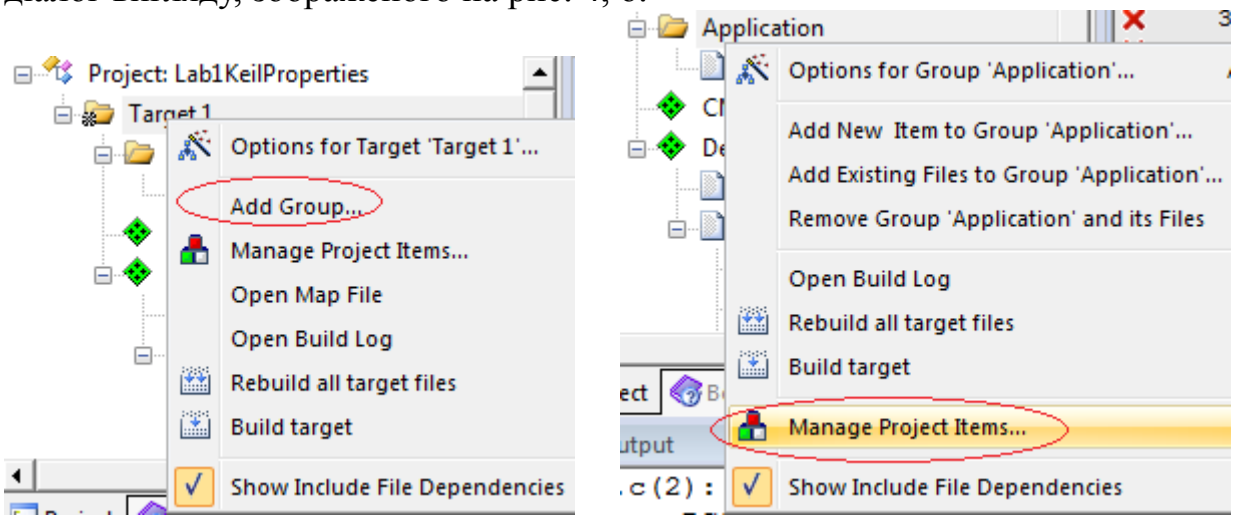


Рис. 3. Структура середовища Keil uVision

У проєкті мають бути включені не лише файли, де описується логіка роботи мікроконтролера, закладена програмістом, але й бібліотечні файли, якими забезпечується «чорнова» робота, виконана фахівцями компаній ARM Limited і STMicroelectronics для того, щоб розробники вбудованих систем могли сконцентрувати свої зусилля на бізнес-логіці. Щоб проєкт був зрозумілим для сприйняття, усі файли групують у папки, даючи їм відповідні назви. Зокрема, прийнято групувати в окремі каталоги файли з кодами, що є тісно пов'язані з апаратною частиною мікроконтролера, файли, що стосуються роботи конкретного периферійного модуля, файли з логікою програми.

Функцію main типово розміщують у файлі main.c, який логічно розмістити у папці з файлами, що реалізують високорівневу логіку роботи мікропрограми. Тому варто створити папку Application (або перейменувати існуючу папку, створену за замовчанням). Для додавання нової групи файлів викликаємо контекстне меню для Target 1 і вибираємо “Add Group” (рис. 4, а). Для зміни існуючої групи у контекстному меню слід обрати “Manage Project Items…”, внаслідок чого появиться діалог вигляду, зображеного на рис. 4, б.



а) б)

Рис. 4. Засоби для групування даних у проєкті

Діалог Manage Project Items (рис. 5) дозволяє створювати, видаляти, перейменовувати групи та задавати їхній порядок (переміщувати у списку). Групування файлів у проєкті не означає існування відповідної ієрархії каталогів. Для зручності структуру каталогів на диску створюють таку ж, як і структуру груп у проєкті.

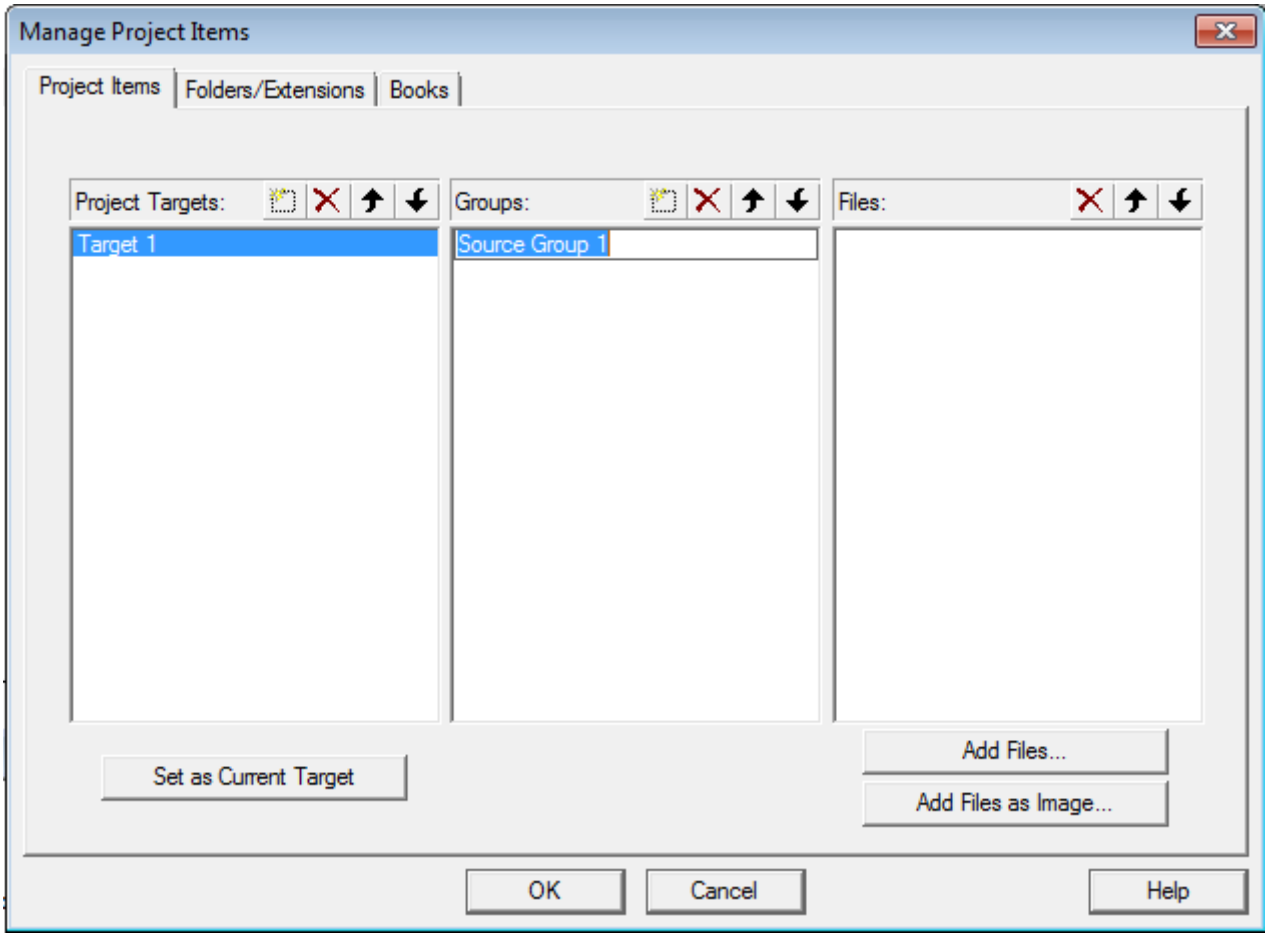


Рис. 5. Діалог “Manage Project Items”

Для додавання нового чи існуючого файлу (.c чи .h) потрібно викликати контекстне меню для потрібної групи файлів і вибрати “Add New Item to Group …” (“Add Existing Files to Group”) відповідно (рис. 6).

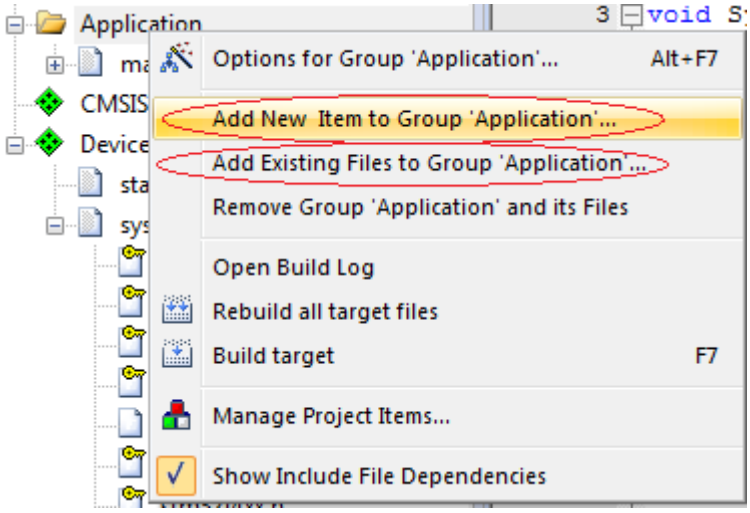


Рис. 6. Додавання нового або існуючого файлу

Створимо новий файл main.c і внесемо в нього програму, наведену в лістингу 1.

Після компіляції можна завантажувати отриманий файл у пам’ять мікроконтролера. Для цього спочатку слід налаштувати проєкт відповідним чином. Спочатку потрібно викликати діалог “Options for Target...” (перший пункт у контекстному меню проєкту), рис. 7.

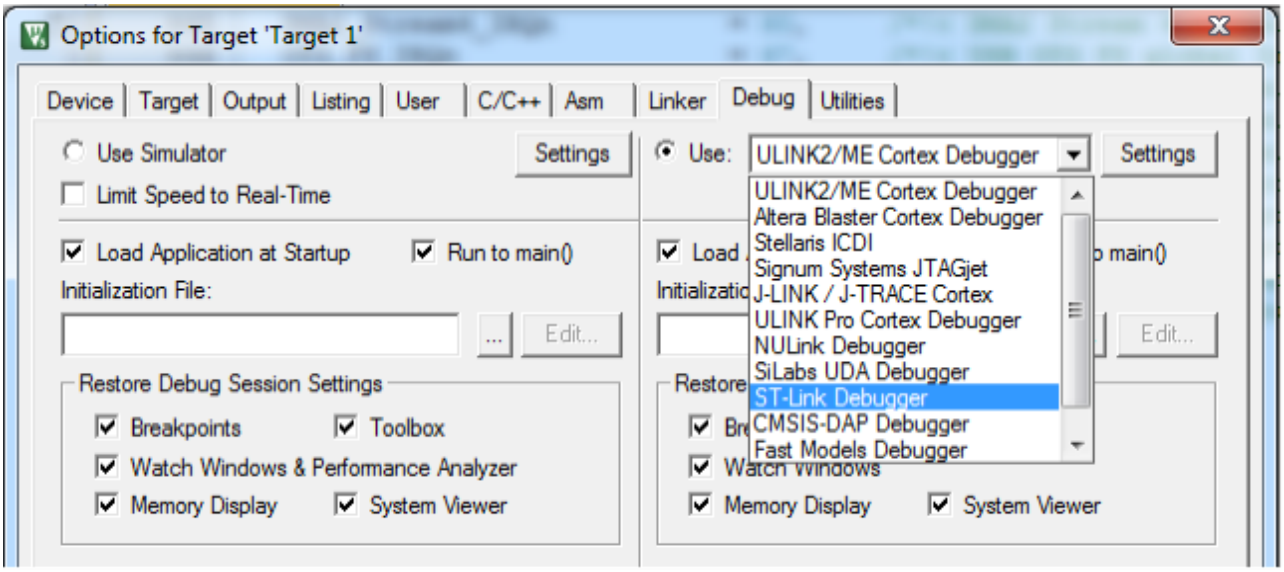


Рис. 7. Вибір debugger'a

Після цього слід натиснути кнопку “Settings” і переконатися, що ядро знайдене (рис. 8).

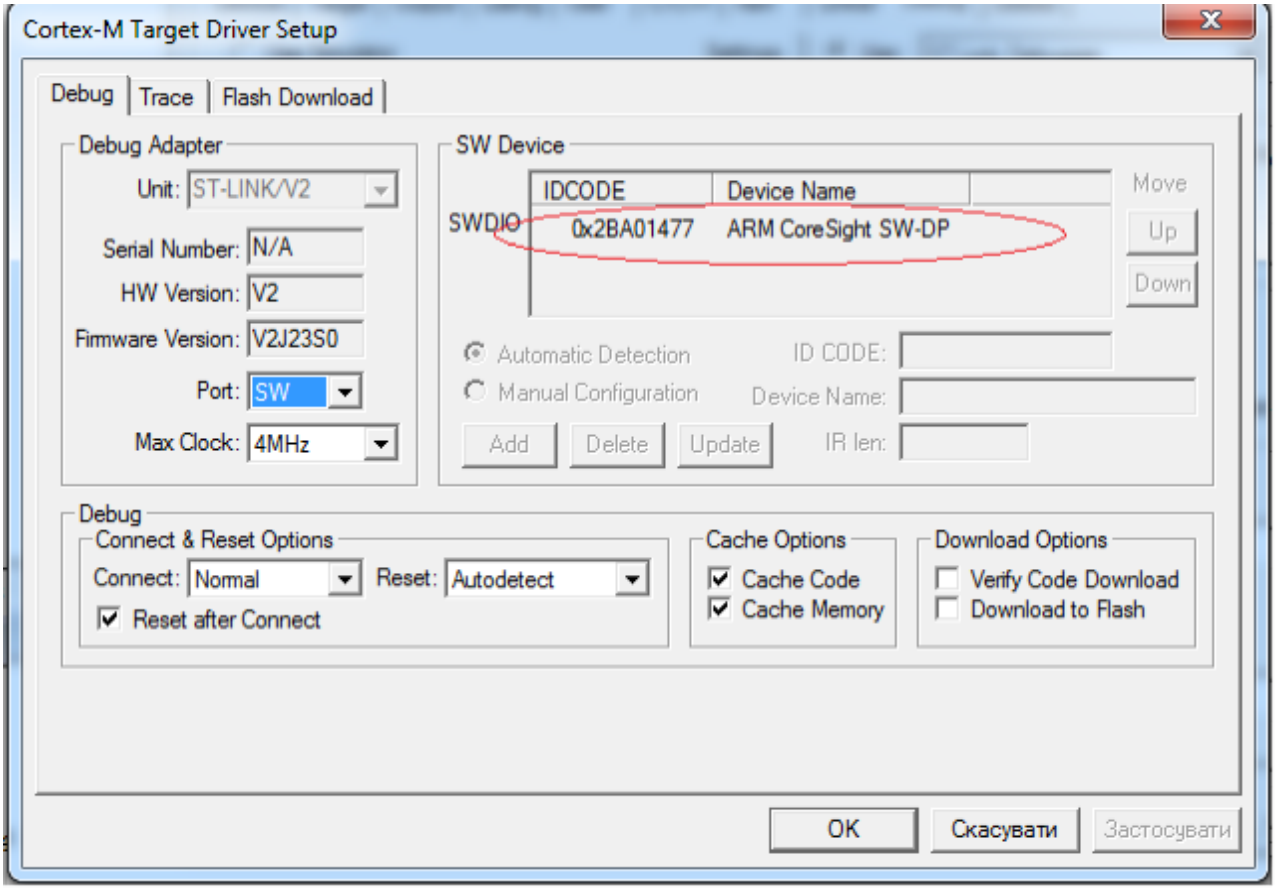


Рис. 8. Налаштування debugger'a

“Останній штрих” - у діалозі “Cortex-M Target Driver Setup” вибираємо вкладку Flash Download і вибираємо алгоритм запису (рис. 9).

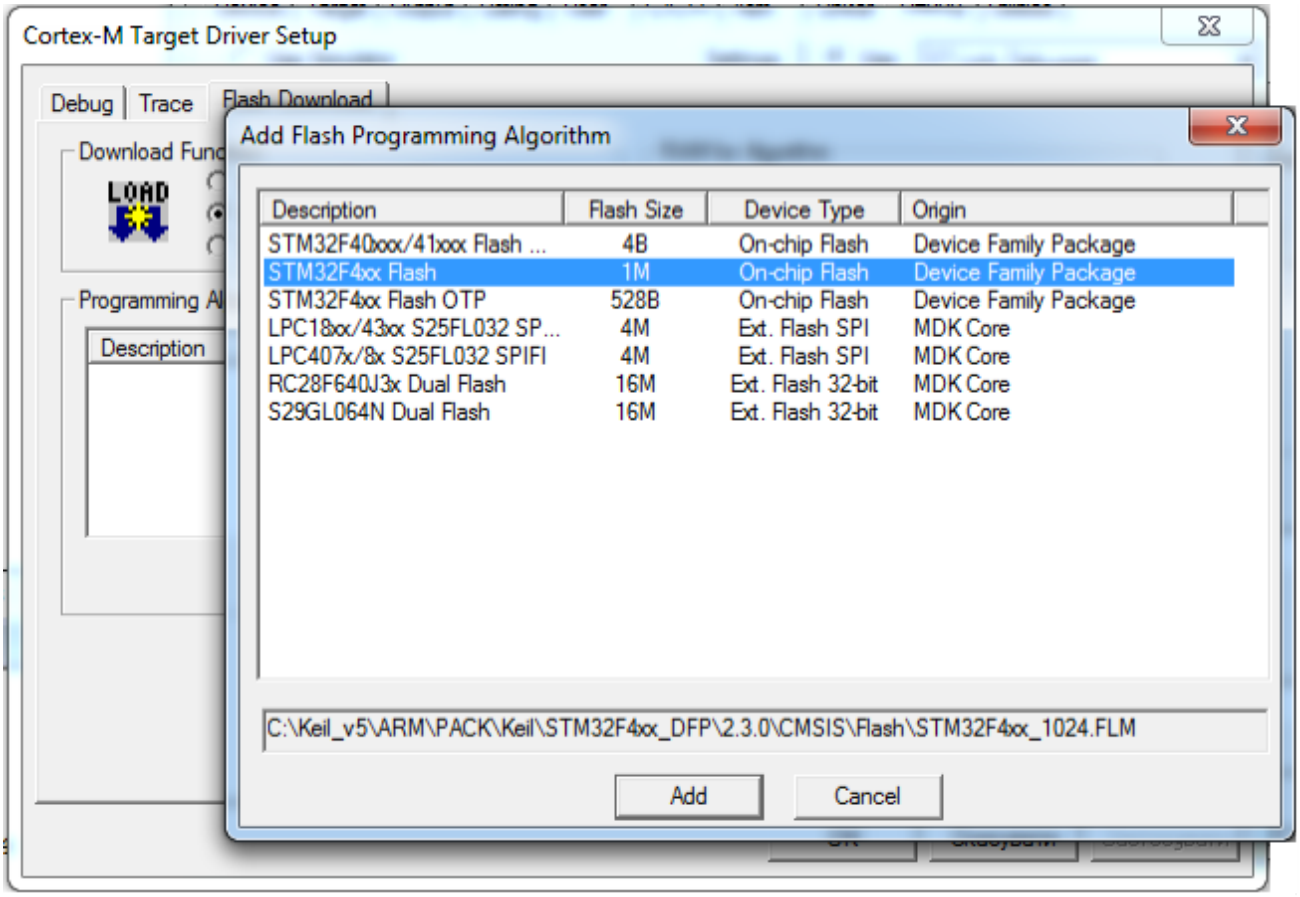


Рис. 9. Вибір алгоритму запису мікропрограми у пам'ять мікроконтролера

Тепер можна скористатися однією з піктограм, виділених на рис. 10.

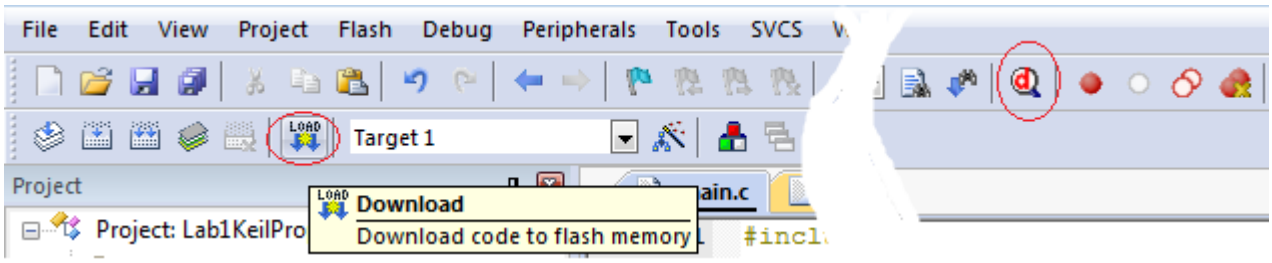


Рис. 10. Засоби запису мікропрограми у пам’ять мікроконтролера

Швидкість завантаження програми в пам’ять мікроконтролера буде більшою, якщо відкрито якомога менше файлів одночасно. Враховуючи, що при програмуванні мікроконтролерів значна частина часу витрачається на компіляцію та завантаження програми, варто спочатку закрити усе зайве.

У нижній частині вікна можна побачити хід процесу запису мікропрограми.

Після завершення завантаження Keil uVision змінює вигляд (рис. 11).

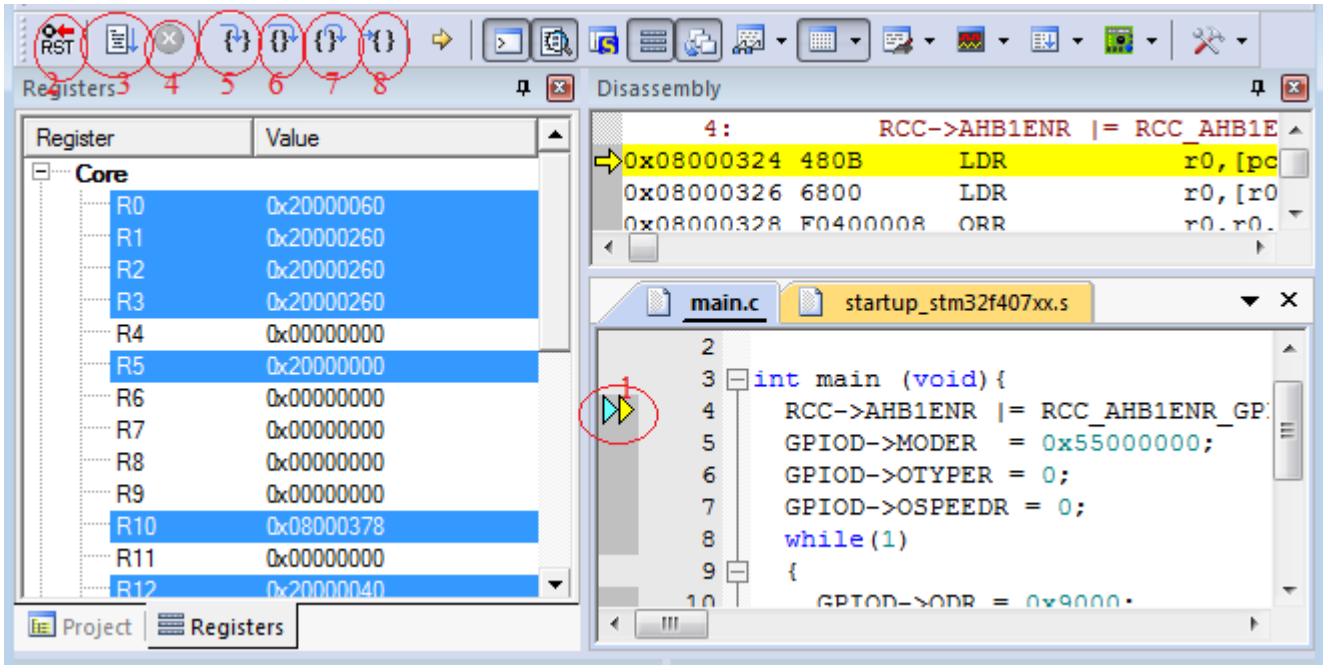


Рис. 11. Keil у режимі виконання

Елемент 1 одразу після запуску вказує на перший рядок функції main. Це добрий знак, хоча не завжди програма стартує з функції main, що буде продемонстровано пізніше, у інших лабораторних. Від програміста чекають подальших дій. Є декілька варіантів:

1) запустити повне виконання, натиснувши піктограму 3 (Run, F5);

2) встановити точки призупинення (breakpoints) і скористатися піктограмою 3;

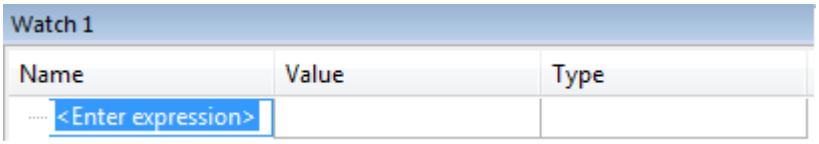
3) виконувати код покроково за допомогою ряду піктограм 5 – 8.

Очевидно, що останній варіант дає шанси «виловити» помилки, оскільки таким чином можна безпосередньо переглянути значення змінних після виконання тих чи інших операторів.

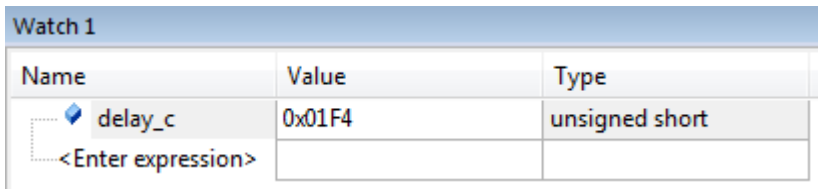
Натиснувши 4, зупиняємо виконання коду, при цьому можна побачити, що саме зараз виконується. RST дозволяє повернутися на початок мікропрограмного забезпечення.

Натиснемо F5 і побачимо результат…

Нехай потрібно перевірити значення тих чи інших змінних (в нашому поточному проєкті наразі вибір невеликий – змінна delay_c). Для цього слід вибрати View = Watch Windows = >Watch 1 (або Watch 2) і у вікні, що при цьому появиться (рис. 12, а) ввести ім’я змінної – буде показано її поточне значення (рис. 12, б).



а)



б)

Рис. 12. Вікна для відстеження поточних значень змінних/виразів

Порядок виконання роботи

1. В середовищі програмування Keil µVision MDK-ARM створіть проєкт, наведений в інструкції, з використанням програми з лістингу 1 або з лістингу 2.
2. Відкомпілюйте програму, створіть проєкт і в режимі відлагоджувача дослідіть його роботу (відслідкуйте стан регістра GPIOD_ODR).
3. Створіть за аналогією проєкт відповідно до свого варіанту (без миготіння) і збережіть його (в тому числі і hex-файл).
4. В системі Proteus введіть схему відповідно до свого варіанту. (для мікроконтролера в полі пошуку введіть stm; для кнопки – but; для світлодіода - led-; для "землі" активізуйте Terminals mode виберіть Ground).
5. Запустіть моделювання і натискайте та відпускайте кнопку.
6. Перевірте роботу програми.
7. У звіті наведіть структуру проєкту, текст програми, копії вікна з схемою.
8. Зробіть висновки про виконану роботу.

Завдання на лабораторну роботу № 11: скласти програму для мікроконтролера STM32F401RB, яка працює таким чином: якщо кнопка, яка під'єднується до k-го біта порту W і до "землі", натиснута, то світиться світлодіод, який під'єднаний до m-го біта порту V, в протилежному випадку – світлодіод гасне.

Завдання для **ПЗ-21**

Варіант	Порт W	Біт k	Порт V	Біт m
1	A	1	B	3
2	A	2	B	4
3	A	3	B	5
4	A	4	B	6
5	A	5	B	7
6	A	6	B	0
7	A	7	B	1
8	A	0	B	2
9	B	1	A	3
10	B	2	A	4

11	B	3	A	5
12	B	4	A	6
13	B	5	A	7
14	B	6	A	0
15	B	7	A	1
16	B	0	A	2
17	C	1	B	3
18	C	2	B	4
19	C	3	B	5
20	C	4	B	6
21	C	5	B	7
22	C	6	B	0
23	C	7	B	1
24	C	0	B	2
25	A	8	C	3
26	A	9	C	4
27	A	10	C	5
28	A	11	C	6
29	A	12	C	7
30	A	13	C	0
31	A	14	C	5
32	A	15	C	6
33	A	9	C	8

Завдання для **ПЗ-22**

Варіант	Порт W	Біт k	Порт V	Біт m
1	D	1	E	6
2	D	2	E	8
3	D	3	E	3
4	D	4	E	4
5	D	5	E	5
6	D	6	E	6
7	D	7	E	7
8	D	0	E	0

9	E	1	D	1
10	E	2	D	2
11	E	3	D	3
12	E	4	D	4
13	E	5	D	5
14	E	6	D	6
15	E	7	D	7
16	E	0	D	0
17	C	1	E	1
18	C	2	E	2
19	C	3	E	3
20	C	4	E	4
21	C	5	E	5
22	C	6	E	6
23	C	7	E	7
24	C	0	E	0
25	D	8	C	1
26	D	9	C	2
27	D	10	C	3
28	D	11	C	4
29	D	12	C	5
30	D	13	C	6
31	D	14	C	7
32	D	15	C	0
33	D	9	C	5

Завдання для **ПЗ-23**

Варіант	Порт W	Біт k	Порт V	Біт m
1	D	15	B	5
2	D	9	B	6
3	D	1	B	8
4	D	2	B	3
5	D	3	B	4
6	D	4	B	5

7	D	5	B	6
8	D	6	B	7
9	B	7	D	0
10	B	0	D	1
11	B	1	D	2
12	B	2	D	3
13	B	3	D	4
14	B	4	D	5
15	B	5	D	6
16	B	6	D	7
17	C	7	B	0
18	C	0	B	1
19	C	1	B	2
20	C	2	B	3
21	C	3	B	4
22	C	4	B	5
23	C	5	B	6
24	C	6	B	7
25	D	7	C	0
26	D	0	C	1
27	D	8	C	2
28	D	9	C	3
29	D	10	C	4
30	D	11	C	5
31	D	12	C	6
32	D	13	C	7
33	D	14	C	0

Завдання для **ПЗ-24**

Варіант	Порт W	Біт k	Порт V	Біт m
1	A	13	D	7
2	A	14	D	0
3	A	15	D	5
4	A	9	D	6

5	A	1	D	8
6	A	2	D	3
7	A	3	D	4
8	A	4	D	5
9	D	5	A	6
10	D	6	A	7
11	D	7	A	0
12	D	0	A	1
13	D	1	A	2
14	D	2	A	3
15	D	3	A	4
16	D	4	A	5
17	C	5	D	6
18	C	6	D	7
19	C	7	D	0
20	C	0	D	1
21	C	1	D	2
22	C	2	D	3
23	C	3	D	4
24	C	4	D	5
25	A	5	C	6
26	A	6	C	7
27	A	7	C	0
28	A	0	C	1
29	A	8	C	2
30	A	9	C	3
31	A	10	C	4
32	A	11	C	5
33	A	12	C	6

Завдання для **ПЗ-25**

Варіант	Порт W	Біт k	Порт V	Біт m
1	A	12	E	6
2	A	13	E	7

3	A	14	E	0
4	A	15	E	5
5	A	9	E	6
6	A	1	E	8
7	A	2	E	3
8	A	3	E	4
9	E	4	A	5
10	E	5	A	6
11	E	6	A	7
12	E	7	A	0
13	E	0	A	1
14	E	1	A	2
15	E	2	A	3
16	E	3	A	4
17	C	4	E	5
18	C	5	E	6
19	C	6	E	7
20	C	7	E	0
21	C	0	E	1
22	C	1	E	2
23	C	2	E	3
24	C	3	E	4
25	A	4	C	5
26	A	5	C	6
27	A	6	C	7
28	A	7	C	0
29	A	0	C	1
30	A	8	C	2
31	A	9	C	3
32	A	10	C	4
33	A	11	C	5

Завдання для **ПЗ-26**

Варіант	Порт W	Біт k	Порт V	Біт m

1	A	11	B	5
2	A	12	B	6
3	A	13	B	7
4	A	14	B	0
5	A	15	B	5
6	A	9	B	6
7	A	1	B	8
8	A	2	B	3
9	B	3	A	4
10	B	4	A	5
11	B	5	A	6
12	B	6	A	7
13	B	7	A	0
14	B	0	A	1
15	B	1	A	2
16	B	2	A	3
17	D	3	B	4
18	D	4	B	5
19	D	5	B	6
20	D	6	B	7
21	D	7	B	0
22	D	0	B	1
23	D	1	B	2
24	D	2	B	3
25	A	3	D	4
26	A	4	D	5
27	A	5	D	6
28	A	6	D	7
29	A	7	D	0
30	A	0	D	1
31	A	8	D	2
32	A	9	D	3
33	A	10	D	4