МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **КНІТ**
Кафедра **ПЗ**

Лабораторна робота № 2

На тему: *“Дослідження впливу середовища програмування на розмір та швидкість виконання програмного коду”*

З дисципліни: *“Основи інтернету речей”*

Львів – 2021

Лабораторна робота № 1

Тема. Ознайомитися з особливостями роботи середовищ програмування **Keil uVision** та **STM32CubeIDE**.

Мета. Встановити взаємозв'язок між середовищем програмування та розміром і швидкістю виконання програмного кодую

Теоретичні відомості

Середовище програмування Keil uVision

Keil uVision – спеціалізоване середовище для програмування мікроконтролерів архітектури ARM, є одним з найпопулярніших. Демо-версію можна скачати з сайту https://www.keil.com/download/product/.

Демо-версія Keil uVision має істотне обмеження – обсяг коду не повинен перевищувати 32 КБ.

На платі STM32F4DISCOVERY встановлений мікроконтролер STM32F407VG. Документацію на контролер, приклади і ряд додаткових корисних ресурсів можна взяти з сайту компанії STMicroelectronics (www.st.com).

Спробуємо створити новий проект 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 проект, його слід закрити перед створенням нового проекту (Project => Close Project). Слід виконати Project => New µVision Project. Появиться діалогове вікно, в якому вводимо ім'я проекту і місце його розташування на диску. Проект 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. 1). 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 проект зберегти не вдасться, адже вибір конкретного мікроконтролера для проекту 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.

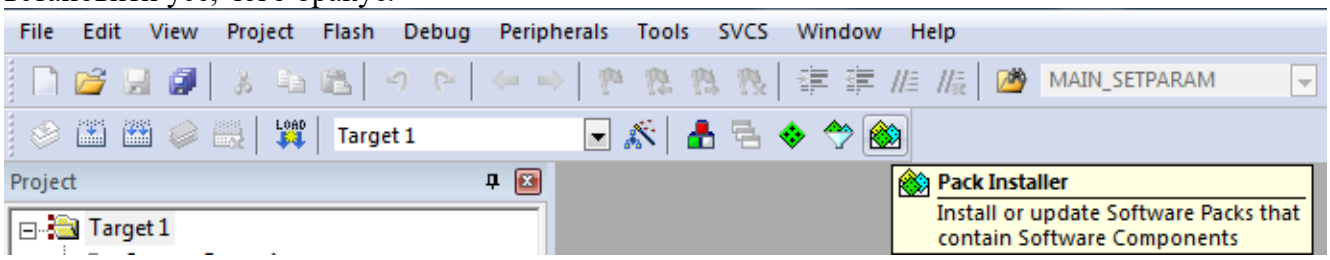


Рис. 1. Виклик вікна для встановлення додаткових компонент

Якщо ж мікроконтролер (target device) успішно вибраний, побачимо вікно, зображене на рис. 2.

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

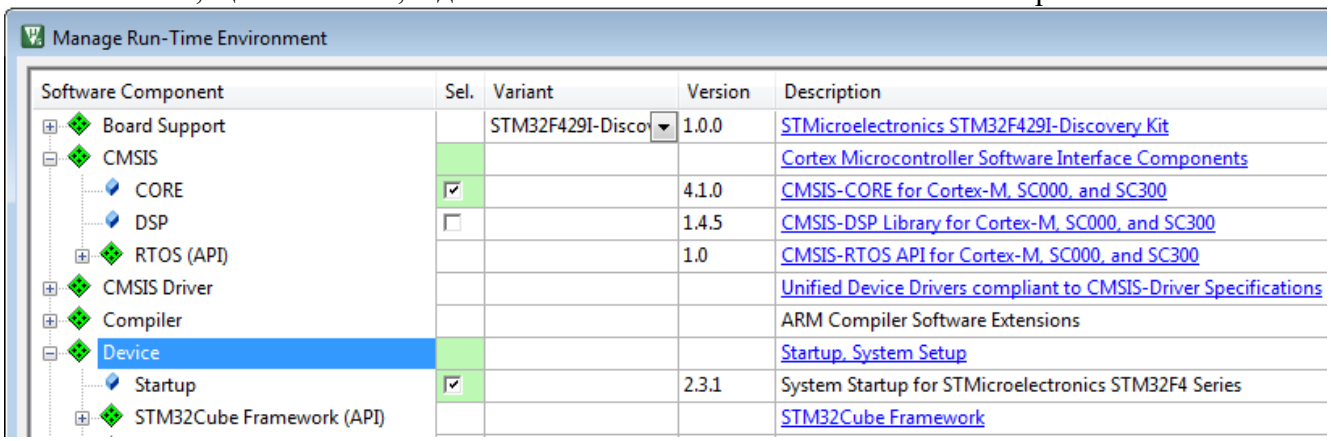


Рис. 2. Структура середовища Keil uVision

У проекті 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 проект 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.

Щойно створений проект міститиме певні бібліотечні файли, але цього недостатньо. Компіляція такого проекту дасть результати на зразок тих, що показані на рис. 3.

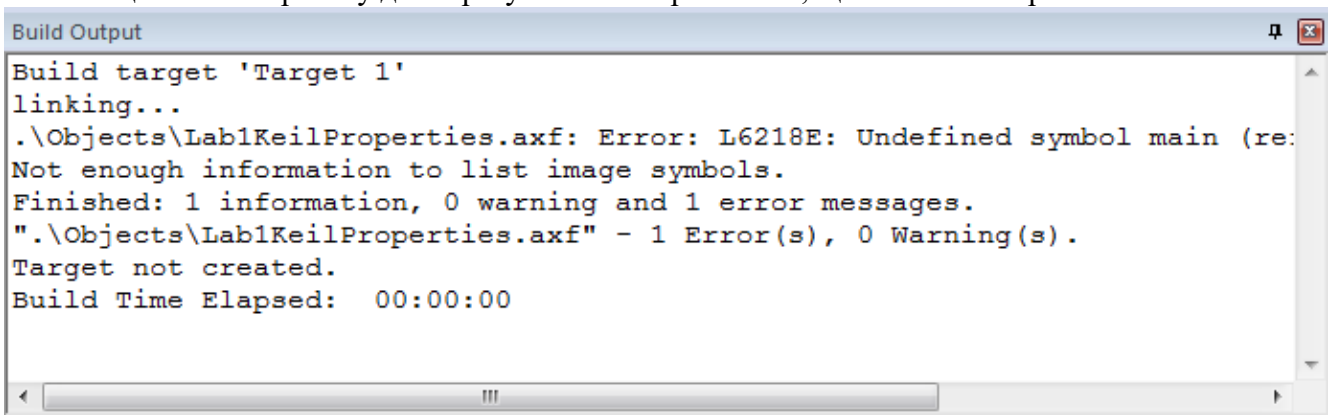
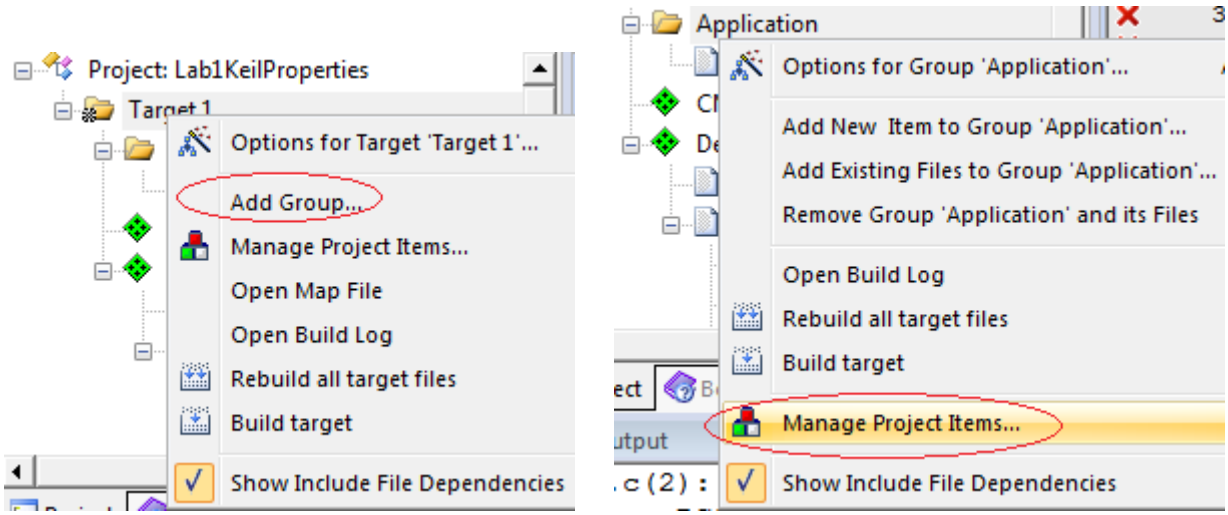


Рис. 3. Результати компіляції проекту без функції main

"Target not created" вказує на те, що код, готовий для запису у пам'ять мікроконтролера, не був створений. Як видно з вікна Build Output знайдена помилка є помилкою лінкування, на що вказує, по-перше, звіт про помилку, розміщений після рядка linking…, а по-друге – код помилки (L6218E). Знати код помилки корисно для пошуку її пояснення та запиту технічної допомоги.

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 проекті

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

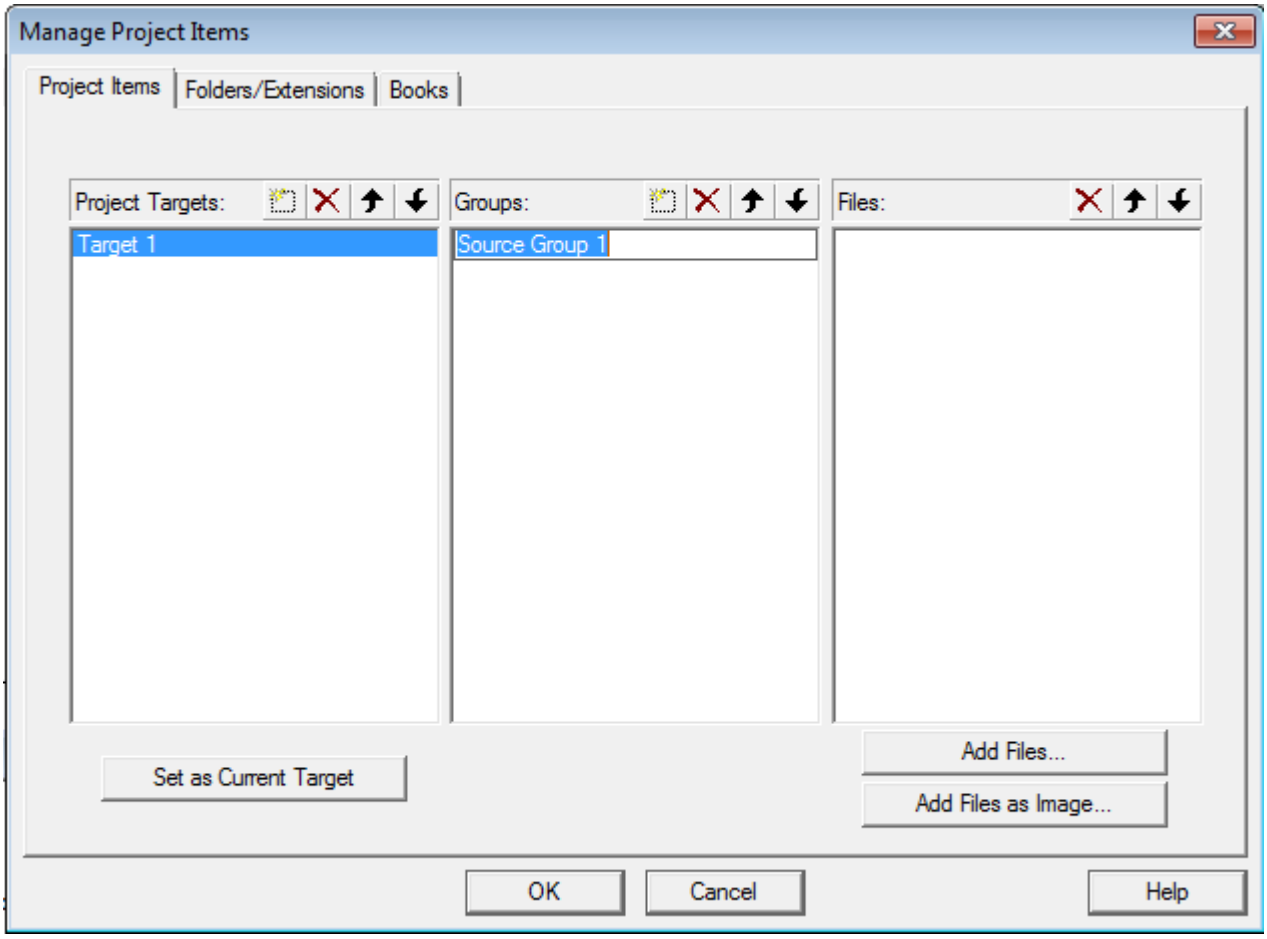


Рис. 5. Діалог “Manage Project Items”

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).

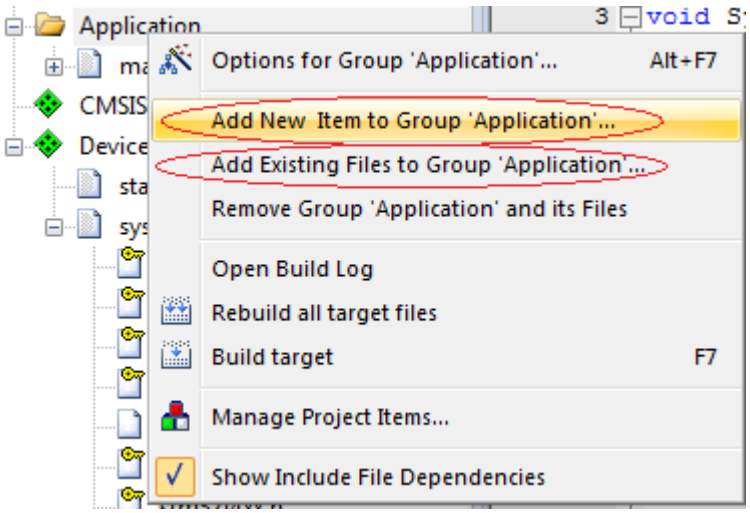


Рис. 6. Додавання нового або існуючого файлу

Створимо новий файл main.c і запишемо у ньому:

```
int main (void){
	while(1)
	{

	}
}
```

Результати компіляції показані на рис. 7.

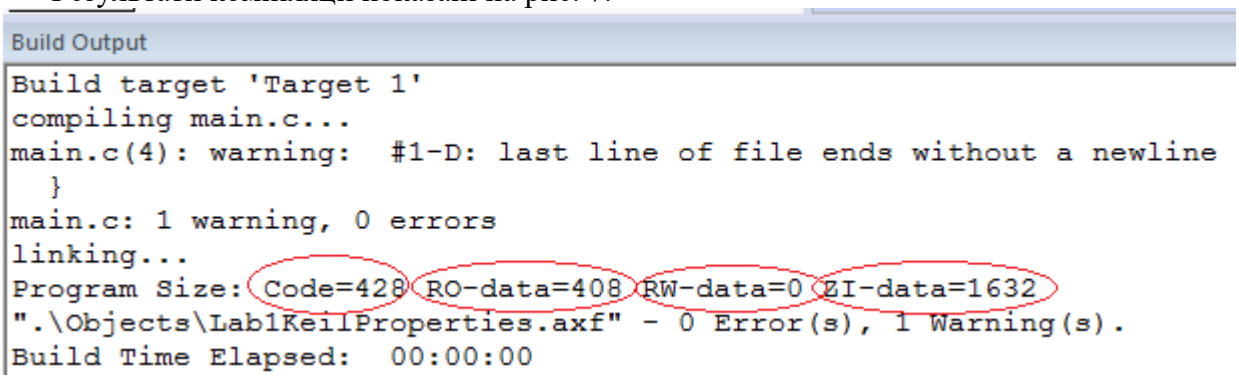


Рис. 7. Результати компіляції проекту

З рис. 7 бачимо, що будь-який файл повинен закінчуватися пустим рядком. Процес компіляції був успішним і у папці .\Objects\ знаходиться файл з кодом для мікроконтролера, Lab1KeilProperties.axf. Code означає розмір вихідного коду у байтах, RO-data – (від read only) – дані, доступні лише для читання, RW-data – дані для запису, ZI-data (Zero Initialized) – дані, ініціалізовані нулями.

Код функції main, наведений вище, є синтаксично правильним, але не несе жодної користі. Однак, у ньому видно одну з рис, специфічних для програмування мікроконтролерів, – безмежний цикл. Оскільки мікроконтролер працює від моменту подачі живлення до вимкнення, він весь час повинен щось робити, звідси і безмежний цикл. Утім, безмежний цикл може бути не лише у функції main! Навпаки – цю функцію застосовують лише для старту, далі управління передається іншим функціям.

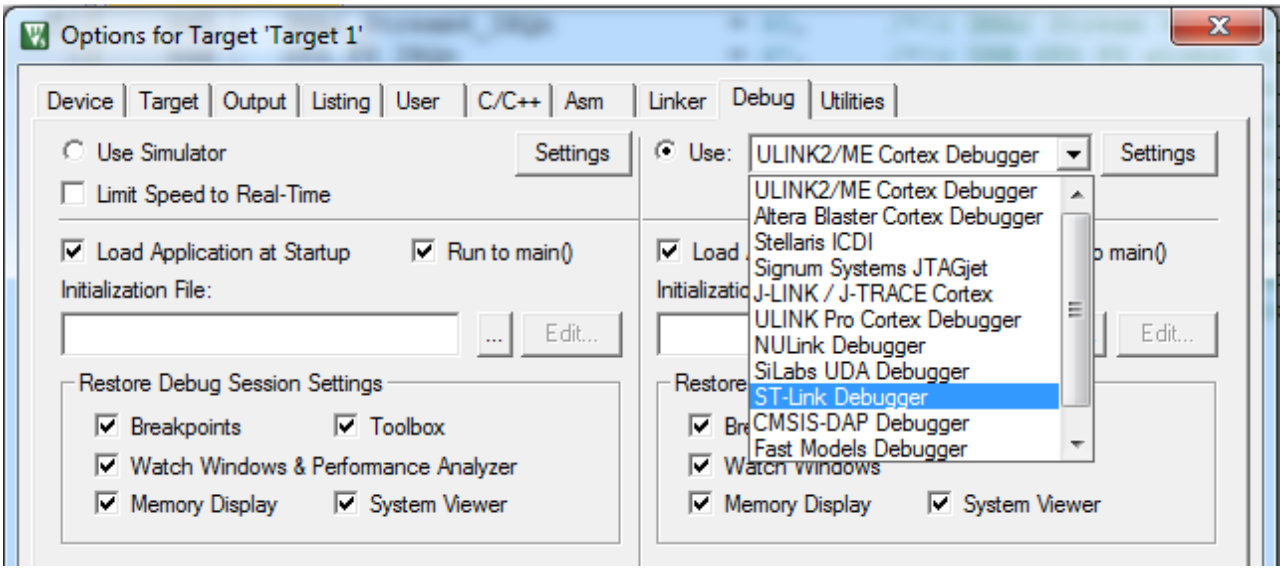
Для того, щоб побачити якісь зміні на платі, замість поданого вище коду функції main запишемо наступний код:

Лістинг 1

```
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){
	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);
	}
}
```

В результаті компіляції будуть виявлені помилки – усі однотипні, «невідомий ідентифікатор». Слід підключити файл заголовний файл "stm32f4xx.h". Цього разу проект скомпілюється успішно.

Після компіляції можна завантажувати отриманий файл у пам’ять мікроконтролера. Для цього спочатку слід налаштувати проект відповідним чином. Спочатку потрібно викликати



діалог “Options for Target...” (перший пункт у контекстному меню проекту), рис. 8.

Рис. 8. Вибір debugger'a

Після цього слід натиснути кнопку “Settings” і переконатися, що ядро знайдене (рис. 9).

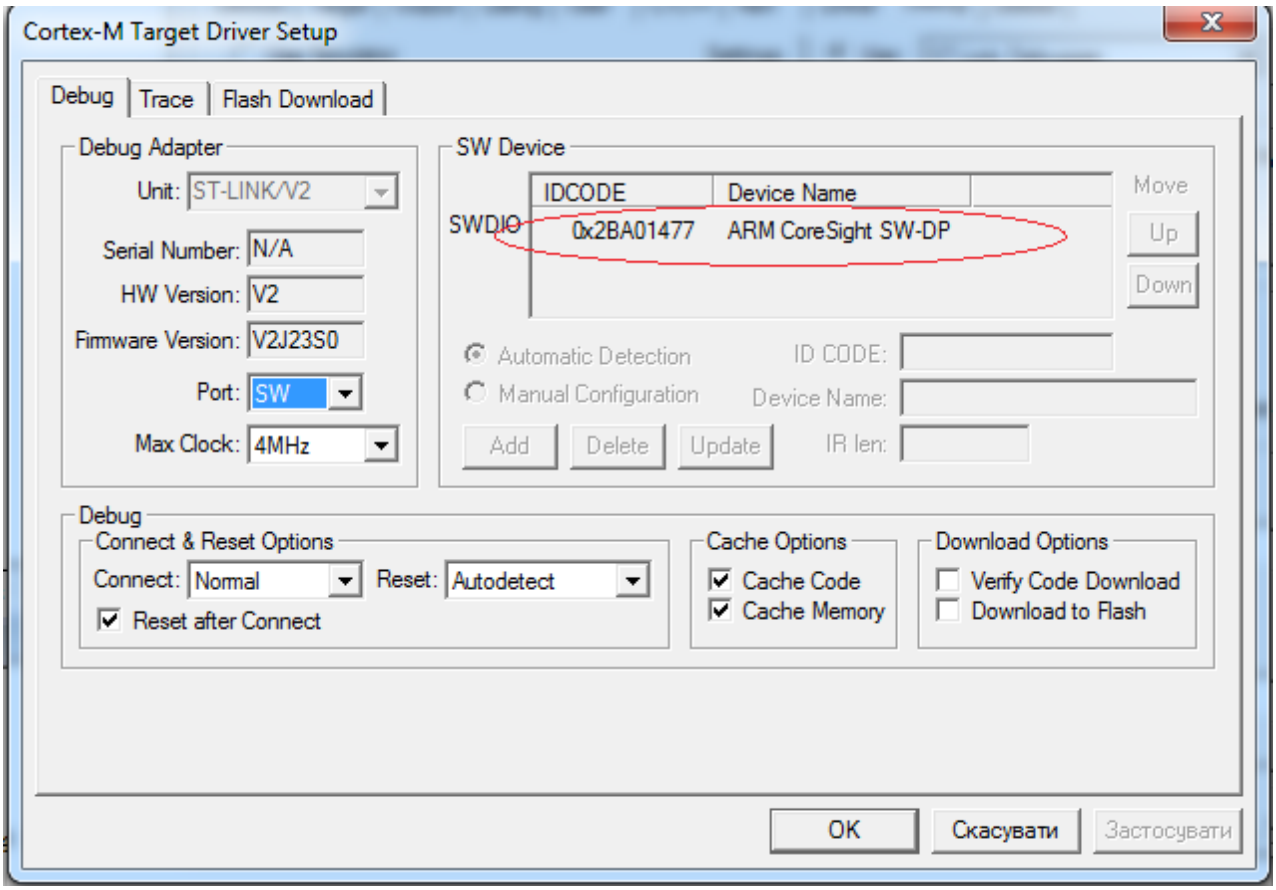


Рис. 9. Налаштування debugger'a

“Останній штрих” - у діалозі “Cortex-M Target Driver Setup” вибираємо вкладку Flash Download і вибираємо алгоритм запису (рис. 10).

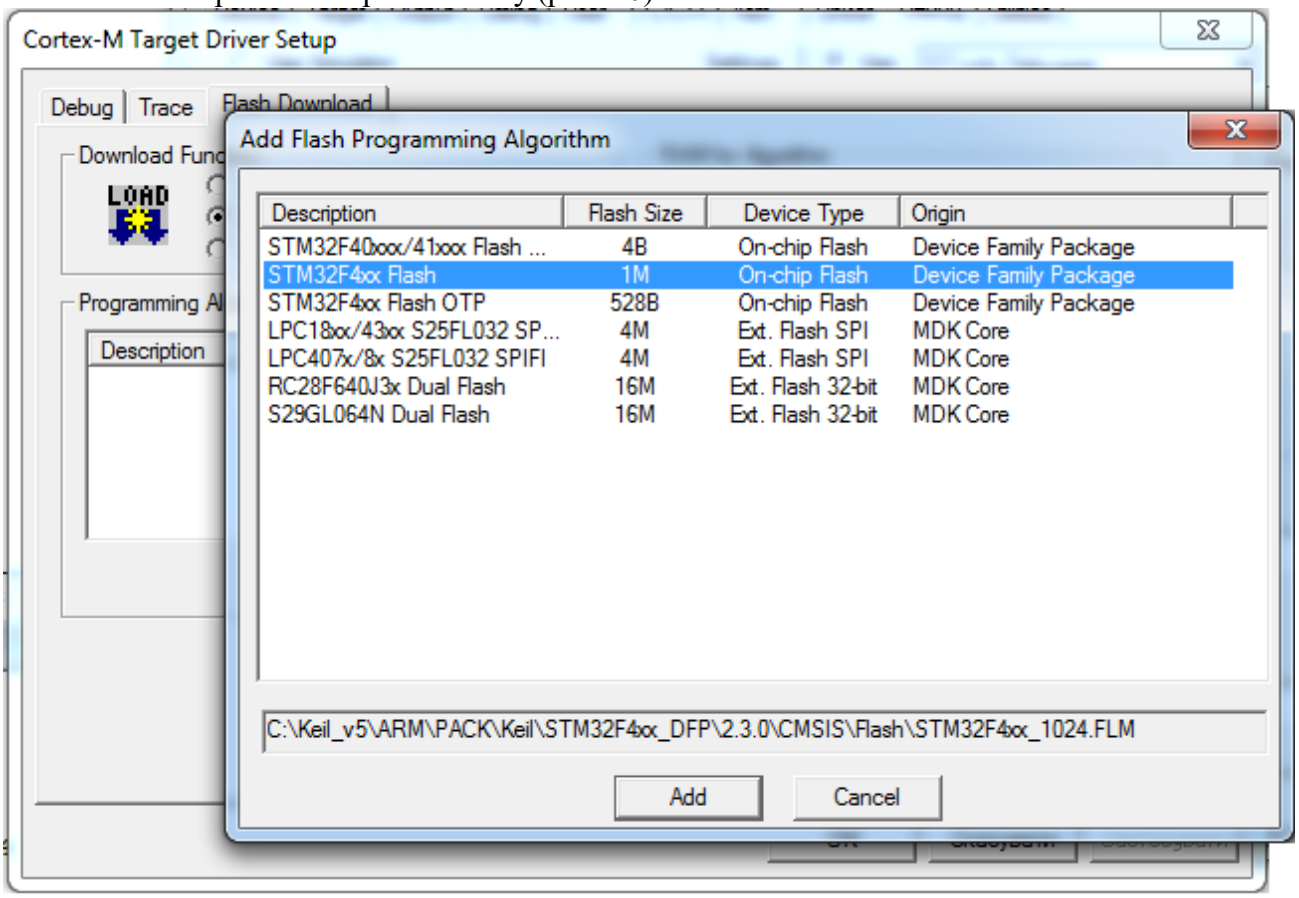


Рис. 10. Вибір алгоритму запису мікропрограми у пам'ять мікроконтролера

Тепер можна скористатися однією з піктограм, виділених на рис. 11.

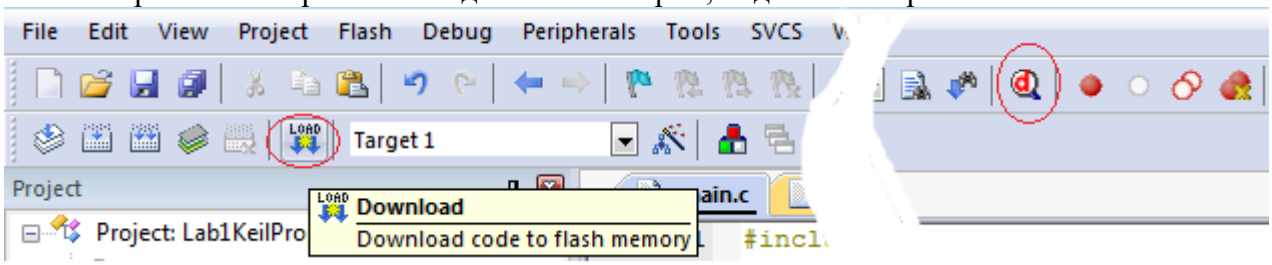


Рис. 11. Засоби запису мікропрограми у пам’ять мікроконтролера

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. 12).

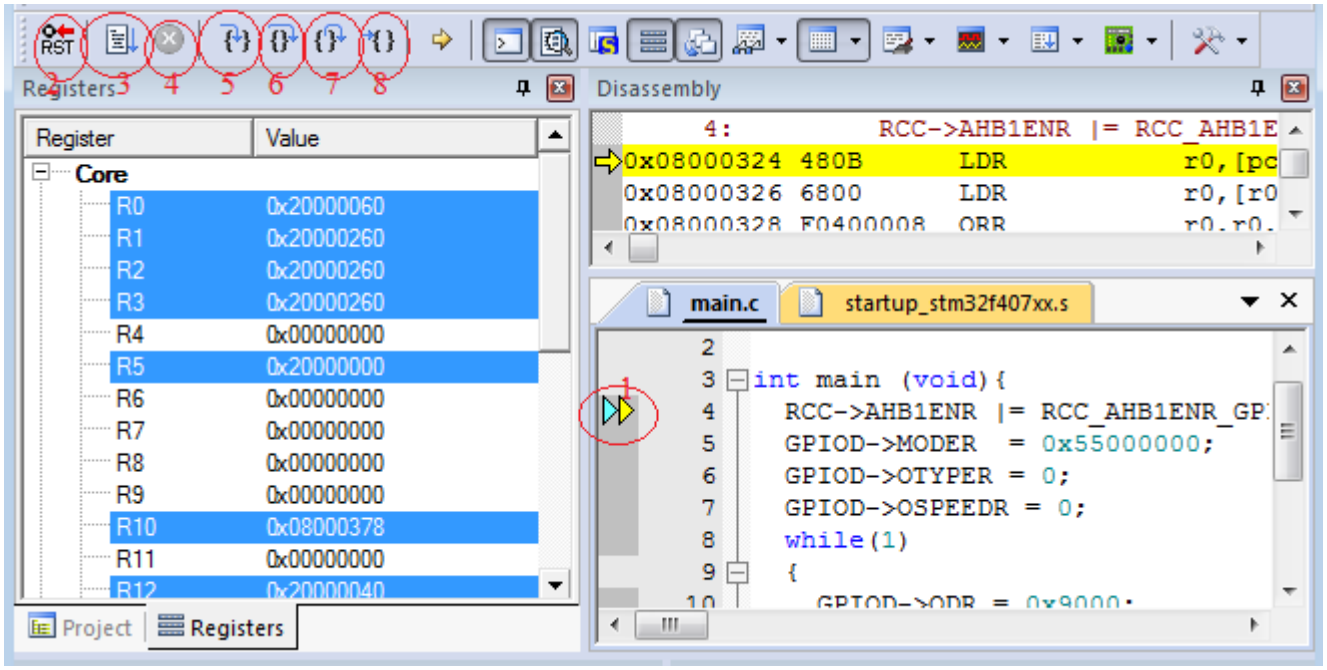


Рис. 12. Keil у режимі виконання

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:

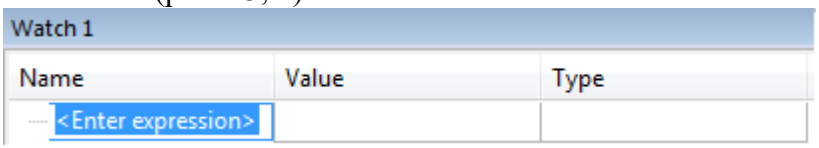
1) запустити повне виконання, натиснувши піктограму 3 (Run, F5);
2) встановити точки призупинення (breakpoints) і скористатися піктограмою 3;
3) виконувати код покроково за допомогою ряду піктограм 5 – 8.

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

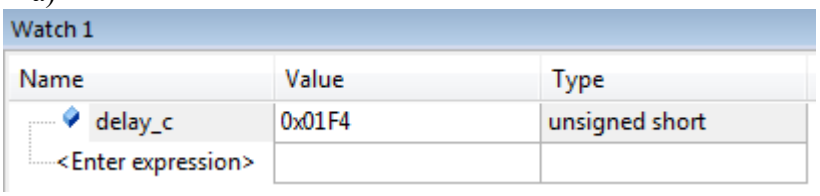
Натиснувши 4, зупиняємо виконання коду, при цьому можна побачити, що саме зараз виконується. RST дозволяє повернутися на початок мікропрограмного забезпечення.

Натиснемо F5 і побачимо результат…

Нехай потрібно перевірити значення тих чи інших змінних (в нашому поточному проекті 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. 13, а) ввести ім’я змінної – буде показано її поточне значення (рис. 13, б).



а)



б)

Рис. 13. Вікна для відстеження поточних значень змінних/виразів

Пояснення змісту коду

Час зрозуміти код, поданий і лістингу 1.

Це ніщо інше, як “Hello, world” для мікроконтролера – блимання світлодіодами, оскільки поки що до плати STM32F4DISCOVERY не підключено нічого іншого, щоб побачити реакцію з боку мікроконтролера (окрім трасування коду і використання вікон спостереження).

Мікроконтролер «спілкується» із зовнішнім світом за допомогою портів вводу/виводу. Щоб дізнатися, до яких портів підключені світлодіоди на платі STM32F4DISCOVERY, доведеться звернутися до документації, потрібний фрагмент схеми показаний на рис. 14.

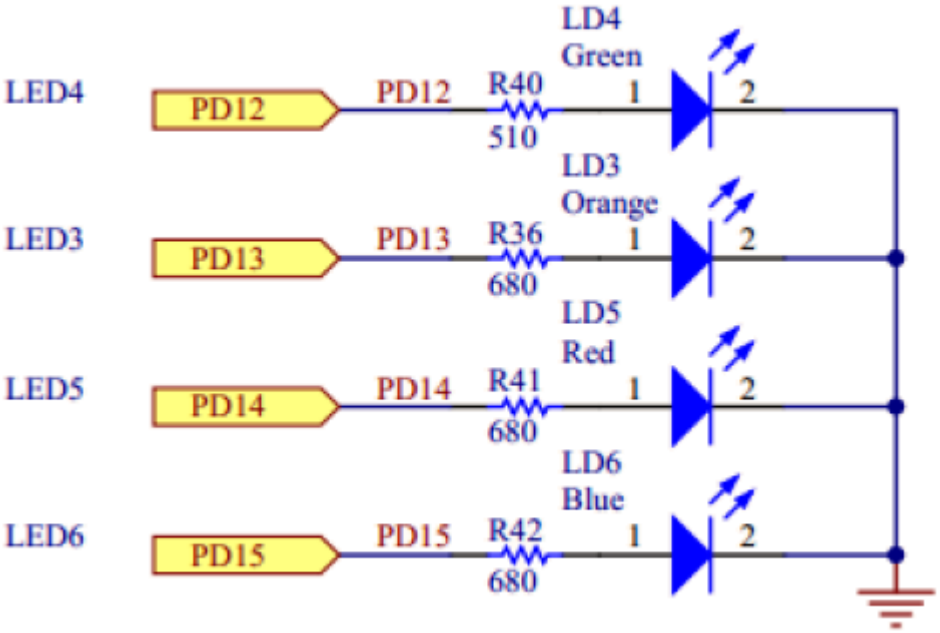


Рис. 14. Підключення світлодіодів до STM32F4DISCOVERY

PD – це порт D, числа 12, 13, 14, 15 – це номери виводів (ніжок, пінів) цього порту, вся робота зі світлодіодами має вестися через порт D.

Враховуючи, що до сучасних вбудованих систем ставиться вимога 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.

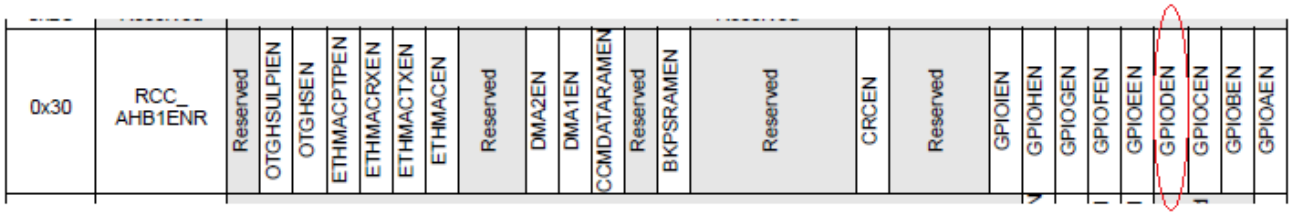
Власне, це і робиться наступний рядком (першим у функції main):

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

RCC – Reset & Clock Control – блок регістрів для управління 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 і т.д.). 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. 15), 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. 15. Фрагмент документації для визначення регістра тактування певного порту

Портів вводу/виводу загального призначення (GPIO – General Purpose Input Output) може бути різна кількість, у нашому випадку (для плати STM32F4DISCOVERY) є 5 портів GPIO: A, B, C, D і E. Порти, як бачимо, іменують великими латинськими літерами.

Кожен порт є 16-бітним і зв'язаний з 8 32-бітними регістрами:

4 регістрами конфігурації GPIOx_MODER, GPIOx_TYPER, GPIOx_ORD (тут *x* заміняє ім'я порту);

2 регістрами даних GPIOx_ODR, GPIOx_IDR;

2 регістрами вибору додаткових функцій GPIOx_AFRH, GPIOx_AFRL.

Оскільки 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. 16, 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. 16. Можливі значення регістра GPIOD->MODER

Для вибору правильного значення регістру GPIOD->OTYPER потрібне глибше розуміння принципу роботи порту, тому приймемо наразі, 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, а оскільки швидкість не є принциповою для прикладу блимання світлодіодами, достатньо задати значення 0 для всіх виводів. Тому GPIOD->SPEEDR = 0.

Оскільки напрям передачі порту – на вихід, то з регістрів даних потрібно налаштовувати GPIOD->ODR (Output Data Register), не IDR (Input Data Register).

У 32-бітному регістрі GPIOD->ODR старші 16 бітів не використовуються (зарезервовані), а молодші 16 відповідають якраз виводам 16-розрядного порту (рис. 17).

Щоб засвітити світло діод, треба подати 1 на відповідний пін. Відповідно якщо GPIOD->ODR = 0x9000, це відповідає числу 1001000000000000 у двійковій системі, тобто, згідно зі схемою на рис. 14, таким чином засвічуємо синій і зелений світлодіоди.

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. 17. Регістр GPIOx->ODR

Функція delay_ms забезпечує затримку у задану кількість мілісекунд. Відповідно, зміст лістингу 1 – вмикання і вимикання блакитного і зеленого світлодіодів.

До всіх регістрів у лістингу 1 ми зверталися за допомогою бібліотеки 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.

Передумова її виникнення – розробники вбудованих систем різних компаній вирішували ряд типових задач заново, кожен по-своєму, тож настав момент, коли у ARM Ltd здогадалися ввести єдиний стандарт.

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

Інакшим помічним засобом є бібліотека ***Standard Peripheral Library*** (SPL), яка є продуктом компанії STMicroelectronics і призначена для полегшення програмування мікроконтролерів виробництва цієї компанії.

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 STM32F407VG у складі STM32F4DISCOVERY має архітектуру Cortex M-4, то для даної оціночної плати 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 проект, він, у свою чергу, включає core_cmN.h і system_<конкретний_МК>.h.

Для плати STM32F4DISCOVERY матимемо такий набір файлів:

system_stm32f4xx.c

system_stm32f4xx.h

stm32f4xx.h

Бібліотека SPL поширюється у вигляді zip-архіву з підкаталогами CMSIS і STM32Fyxx_StdPeriph_Driver. 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.

Наприклад, для роботи з портами вводу/виводу 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 вводу/виводу є структура GPIO_InitTypeDef, наведена на рис. 18:

```
//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. 18. Структура GPIO_InitTypeDef і можливі значення її полів

Слід створити екземпляр цієї структури і задати значення усім її полям. Значення – це елементи перелічень (рис. 18). Наприклад:

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 у 1*/
GPIO_ResetBits(GPIOx, GPIO_Pin_y); /* встановлення біта GPIO_Pin_y порту GPIOx у 0*/

Завдання на лабораторну

1. Встановити середовища **STM32CubeIDE (повинно бути вже встановлено для виконання ЛР№1) та Keil uVision.**
2. Трансформувати індивідуальне завдання для забезпечення можливості вимірювання часу виконання коду.
3. Створити проект згідно індивідуального завдання для борду STM32F4 Discovery з застоуванням середовища програмування **STM32CubeIDE**.
4. Створити проект згідно індивідуального завдання для мікроконтролера STM32F407VG з застоуванням середовища програмування **STM32CubeIDE**.
5. Створити проект згідно індивідуального завдання для мікроконтролера STM32F407VG з застоуванням середовища програмування **Keil uVision**.
6. Порівняти розмір отриманого коду та час його виконання у пунктах 3-5.
7. Оптимізувати розмір коду отриманий в п.3 шляхом вилучення зайвих автоматично згенерованих ділянок .

Індивідуальні завдання

Індивідуальне завдання обирається згідно порядкового номеру в списку підгрупи. У випадку коли порядковий номер у списку перевищує 12 студент обирає варіант за формулою **«порядковий номер» -12**

1.Засвічення червоного та синього світлодіодів, після цього – зеленого та жовтого з інтервалом у 1 с, циклічно

2. Вмикання/вимикання червоного та синього світлодіодів з інтервалом у 0.5 с циклічно, але не більше 20 ітерацій.

3. Вмикання по черзі червоного та синього світлодіодів, через 1 с – зеленого та жовтого світлодіоду одночасно.

4. Вмикання/вимикання червоного світлодіоду з інтервалом, що з кожною ітерацією збільшується на 0.5 с.

5. Вмикання/вимикання червоного та жовтого світлодіодів з інтервалом, що з кожною ітерацією збільшується на 0.5 с.

6. Почергове циклічне увімкнення жовтого, зеленого, синього та червоного світлодіодів.

7. Почергове циклічне увімкнення зеленого, жовтого, синього та червоного світлодіодів.

8. Почергове циклічне увімкнення синього, жовтого, зеленого, та червоного світлодіодів.

9. Почергове циклічне увімкнення червоного, жовтого, зеленого та синього світлодіодів.

10. Почергове увімкнення/вимкнення жовтого, зеленого, синього та червоного світлодіодів. Перед засвіченням кожного наступного світлодіоду, попередній слід вимкнути.

11. Почергове циклічне увімкнення зеленого, жовтого, синього та червоного світлодіодів. Перед засвіченням кожного наступного світлодіоду, попередній слід вимкнути.

12. Почергове циклічне увімкнення червоного, жовтого, зеленого та синього світлодіодів. Перед засвіченням кожного наступного світлодіоду, попередній слід вимкнути.

Контрольні запитання

1. Для яких мікроконтролерів призначена бібліотека CMSIS?
2. Для яких мікроконтролерів призначена бібліотека Standard Peripherals Library?
3. Який файл утворюється у результаті компіляції у Keil uVision?
4. З яких файлів складається бібліотека CMSIS?
5. Яку структуру має бібліотека Standard Peripherals Library?
6. Як відстежити значення змінної у середовищі Keil?
7. Які інструменти для трасування коду є у середовищі Keil?
8. Назвіть типову структуру проекту у Keil uVision, відповідь обґрунтуйте.
9. Як задати тактування периферійного модуля із застосуванням CMSIS?
10. Як задати тактування периферійного модуля із застосуванням Standard Peripherals Library?
11. Чому робота з периферійними модулями повинна починатися з увімкнення тактування?
12. Як сконфігурувати порт у CMSIS?
13. Як сконфігурувати порт у SPL?
14. Які є порти вводу/виводу загального призначення на платі STM32F$DISCOVERY?
15. Якими регістрами конфігурується кожен порт?
16. Для чого призначений регістр GPIOx->MODER?
17. Для чого призначений регістр GPIOx->SPEEDR?
18. Яку структуру має регістр GPIOx->ODR?
19. Опишіть поля структури GPIO_InitTypeDef.
20. Що означає перше поле у структурі GPIO_InitTypeDef?