Програмування мікроконтролерів

ЛЕКЦІЯ 1

Базові поняття програмування мікроконтролерів. Бібліотеки CMSIS і SPL

Організація курсу

ВНС: пароль - Ghiraldi

Лабораторних — 8 , у сумі — 40 балів
Іспит — 60 балів, три рівні

1 — тести:
вибрати правильну відповідь із запропонованих варіантів або вписати слово (ним може бути і ім'я бібліотечного файлу, структури, функції тощо), словосполучення, числове значення
15 питань

2 — написати два-три речення відповіді (лише конкретика) на задане питання, наприклад, дати визначення або ж пояснити різницю між якимись двома поняттями

3 — розгорнута відповідь, наприклад, пояснити принцип роботи того чи іншого інтерфейсу, порівняти два інтерфейси, написати фрагмент коду для реалізації взаємодії мікроконтролера з якоюсь периферією тощо

Dinamica Generale S.p.A (1)

www.dinamicagenerale.com

- AGRI Solutions
  - GRAIN CART
  - SPREADER
  - TRAILER
  - SEEDER
  - HARVESTING
  - BALER
  - BIOGAS
  - FIELDtrace
- PRECISION feeding
  - FEED MIXER
  - DTM suite
  - dg precisionFEEDING
  - AgriNIR ANALYZER
  - LIVESTOCK
  - FARM of the FUTURE

- ON-BOARD solutions
  - WHEEL/FRONT LOADER
  - TELESCOPIC HANDLER
  - FORK LIFT
  - EXCAVATOR
  - CONCRETE MIXER
  - DUMPER
- INDUSTRIAL solutions
  - VEHICLE WEIGHING
  - LOGISTIC WEIGHING
  - PRESSURE TRANSDUCERS
  - INDICATORS
  - LOAD CELLS
  - WEIGHING UNIT
- MEDICAL solutions
- NIR solutions
- SOFTWARE solutions
- CUSTOM solutions
- REMOTE CONTROL solutions

Dinamica Generale S.p.A (2)

Dinamica Generale S.p.A (3)

Dinamica Generale US, Inc.
29 января ·

Take the Laboratory to the Farm with our hand held forage and grain analyzer!

Показать перевод

https://youtu.be/3VZ0OxEiQD0

Dinamica Generale S.p.A (4)

Dinamica Generale S.p.A (3)

DG500

PRODUCT

Universal and programmable weighing indicator having Recipes, Distribution programs and data storage.

DG600

PRODUCT

Universal and programmable weighing indicator with high number of Recipes, Distribution programs and data storage.

Segger Emwin

Області застосування мікроконтролерів (1)

1. Побутова техніка з автоматичним керуванням, телекомунікаційні пристрої, офісна техніка:

- мобільні телефони
- фотоапарати
- телевізори
- GPS-навігатори
- мікрокалькулятор
- пральні машини
- мікрохвильові печі
- домофони
- іграшки
- “розумний будинок”

Області застосування мікроконтролерів (2)

2. Спеціалізована техніка в ряді галузей

медичні прилади (електрокардіографи, тонометри, електронні мікроскопи,…)

авіоніка

сільське господарство

автомобілебудування

банкомати, платіжні термінали, “чіпові” кредитні картки

рОботи у різних областях застосування

. . .

Характеристики вбудованих систем

Відмінності є умовними, межі між вбудованими системами і комп'ютерами з часом стираються

Конкретне призначення, на противагу до універсальності

Апаратні обмеження: обмежені обчислювальні можливості, енергоспоживання, обсяг пам'яті

Програмні обмеження: менші обсяги коду, відсутність ОС або «урізані» ОС

Підвищені вимоги до якості і надійності (MISRA!)

Мікроконтролери та мікропроцесори

Мікропроцесор – Процесор “в мініатюрі”, основна діяльність – читання/запис даних з/у регістри та виконання арифметичних і логічних дій над цими даними

Пам’ять? Пристрої вводу/виводу?

Мікроконтролер – Мікропроцесор + інтерфейсні схеми для взаємодії з пристроями вводу/виводу; “однокристальний комп’ютер”; “незалежний” чіп, підключаючи периферійні пристрої до якого і завантажуючи мікропрограму, можемо одержати логічно завершений пристрій

Основна задача мікроконтролера, як і випливає з його назви, - управляти різноманітними електронними пристроями

Мікропрограмне забезпечення

Мікропрограма (“прошивка”, мікропрограмне забезпечення, firmware) – 1) Вміст пам’яті програм мікроконтролера, його програмне забезпечення.
“Прошивкою” називають також:
процес запису firmware у пам’ять програм мікроконтролера;
файл з програмою, готовий для запису у пам’ять мікроконтролера

“Перепрошити” контролер можна різними способами, як замінивши контролер фізично, так і переписавши його пам’ять (за допомогою програматора або спеціальної програми – bootloader’а)

Bootloader – Логічно завершена мікропрограма, яка міститься у пам’яті програм мікроконтролера і слугує для завантаження його “прошивки” в інші ділянки пам’яті програм (типово для поновлення версії прошивки у “польових умовах”). “Прошивка” типово переписується із зовнішньої пам’яті, зокрема, USB флеш-пам’яті

Мікроконтролер

Класична аналогія: мікроконтролер – “мозок” пристрою, firmware – його “свідомість”.

В залежності від задач до контролера під’єднують ті чи інші периферійні пристрої.

ПЗП МК

Є *масив комірок пам'яті*. Кожна комірка має свою адресу. У цих комірках "живуть" числа, що є *кодами команд*, які *здатний виконувати даний МК*. Усе, що напише програміст мовою С (чи ще якоюсь мовою), перетворюється у *послідовність кодів команд МК*. Ця послідовність, власне, і міститься у *firmware*.

Кожній команді МК відповідає своє число.

МК при подачі на нього живлення послідовно зчитує коди команд, дешифрує їх (дізнається, яка команда ховається за цим кодом) і виконує команди почергово

В кожного МК – свій набір команд!

Будова мікроконтролера

Типовий контролер містить такий мінімум елементів:

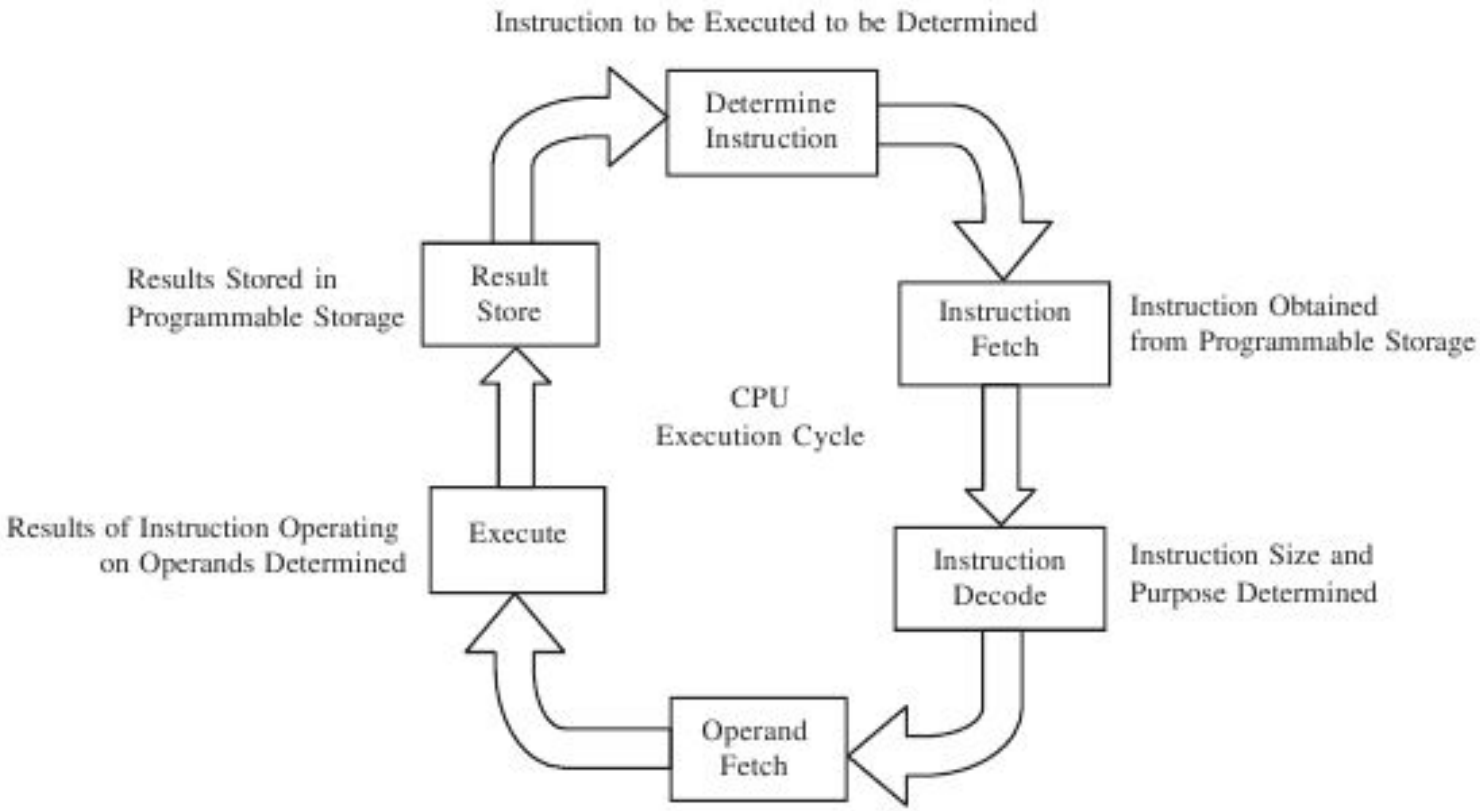
- CPU (Арифметико-логічний пристрій (АЛП, ALU), акумулятор, дешифратор команд);
- ОЗП (оперативна пам'ять);
- ПЗП (постійна пам'ять);
- Генератор тактової частоти;
- Порти вводу/виводу;
- Шини;
- Таймери;
- АЦП, ЦАП, аналоговий компаратор

АЛП

Арифметико-логічний пристрій (АЛП) – “центральний пристрій” будь-якого мікроконтролера, здійснює арифметичні (додавання, віднімання, порівняння...) та логічні (“І”, “АБО”, “НЕ”, “виключне АБО”, зсув вправо, зсув вліво,...) операції над бінарними даними.

АЛП виконує послідовність команд почергово.
Команди зчитуються з ПЗП.
Кожна зчитана команда розміщується в регістрі команд.
АЛП постійно перевіряє регістр команд. Коли цей регістр непустий, АЛП одразу починає виконувати команду.
Вміст ПЗП – послідовність команд – це мікропрограма, записана туди програмістом.

Цикл виконання команд



Лічильник команд

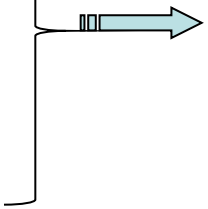
Для того, щоб “витягти” наступну команду в послідовності, застосовується лічильник команд.

Лічильник команд – регістр спеціального призначення, що містить адресу наступної команди для виконання. Лічильник команд може читати чи записувати лише сам МК, не програміст.

Команда переходу?

Переривання?

Виклик функції?



Адреса наступної команди поміщається у *стек*
Після обробки переривання чи виходу з функції ця адреса читається зі стека і основна програма продовжується

Якщо програма виконується лінійно, лічильник вказуватиме на кожну наступну команду.

Розрядність АЛП

Розрядність АЛП – визначає, якими числами оперує АЛП. Розрядність 16 означає, що АЛП бере два 16-розрядні операнди і повертає 16-розрядний результат.
І операнди, і результат обчислення АЛП розміщуються у *регістрі загального призначення*

І операнди, і результат обчислення АЛП розміщуються у *регістрі загального призначення*.

Регістр загального призначення – оперативна (тимчасова) пам'ять, у якій АЛП зберігає *те, з чим працює зараз*, усі інші дані запам'ятовуються в ОЗП.

Пам'ять поділена на комірки. Розмір комірки відповідає розрядності. Якщо розрядність 16, розмір комірки — 16 біт (2 байти). Кожна комірка має адресу.

Регістри

Регістри – Комірки оперативної пам’яті, до яких МК звертається більш короткими і швидкими командами. В усьому іншому регістри аналогічні іншим коміркам оперативної пам’яті

У часи становлення мікроконтролерів, регістрів було мало, а внутрішньої оперативної пам’яті не існувало, працювати з регістрами було набагато швидше, ніж із зовнішньою ОП, тому програми старалися писати так, щоб обійтися лише регістрами (для цього мінімізували кількість змінних і констант).

Банки регістрів – група регістрів, з якими можна працювати одночасно.
Можна “перемикатися” між банками, за допомогою регістра PSW (Program Status Word)

Порти вводу/виводу

Порт вводу/виводу МК – іменована сукупність N виводів (синоніми виводу — пін, pin, “ніжка”), через кожен з яких мікроконтролер може приймати або передавати сигнали. Число N може бути різним, наприклад, 8 чи 16. Імена портів — великі латинські літери — A, B, C, D, E.

Звернення до портів вводу/виводу завжди робиться через регістри портів вводу/виводу.

Виводи можуть бути налаштовані як на вихід, так і на вхід. Якщо потрібно зчитати дані (наприклад, від кнопок), порт конфігурують на вхід, якщо передати дані периферійним пристроям — на вихід.
Одні виводи одного і того ж порту можуть бути налаштовані на вихід, інші — на вхід.

Таймери/лічильники

Таймер – пристрій для формування часових інтервалів; цифровий лічильник, який рахує імпульси від внутрішнього генератора частоти або від зовнішнього джерела сигналу.

Watchdog – “Вартовий таймер”, призначений для перезапуску мікропрограми через заданий проміжок часу. Використовується для виводу МК з ненормальних умов роботи. Принцип використання такий: мікропрограма в ході свого нормального виконання занулює (“скидає”) таймер, тому заданий проміжок часу не буде досягнутий. Якщо ж програма “зависла”, час відраховується і відбудеться перезапуск.

Додаткові функціональні блоки МК

Аналоговий компаратор – пристрій для контролю напруги. Порівнює виміряну напругу з опорною і:
формує логічну 1, якщо виміряне значення більше за опорне;
формує логічний 0 у протилежному випадку. Приклад застосування контроль часу завершення зарядки акумулятора

АЦП – Аналого-цифровий перетворювач, пристрій для перетворення аналогового сигналу у цифрову форму

ЦАП – Цифро-аналоговий перетворювач, пристрій для перетворення цифрового сигналу у аналогову форму

Шини мікроконтролера (1)

Функціональні блоки мікроконтролера обмінюються даними через шини.
Зокрема, дані з пам'яті в АЛП передаються через шину.

Шина – сукупність провідників, по яким передаються цифрові сигнали, середовище обміну даними між різними частинами МК

Розрядність шини – кількість бітів, яку можна передати по шині одночасно

Мікроконтролер має:

шину даних
шину адреси
шину управління

Шини мікроконтролера (2)

Функціональні блоки МК обмінюються даними через шини. Зокрема, дані з пам'яті в АЛП передаються через шину.

Шина даних – шина, призначена для передачі даних у двох напрямках. Мінімальна розрядність шини даних – 8. Що?

Шина адреси – шина, на якій встановлюється адреса комірки пам'яті, до якої слід звернутися (щоб вичитати команду, наступну для виконання, або записати дані). Звідки/куди?

Шина управління – шина, по якій передаються керуючі сигнали. Керуючі сигнали, які і випливає з їхньої назви, управляють процесом роботи з пам'яттю. Як?

Архітектури мікроконтролера (1)

Є сотні мікроконтролерів, але всі їх можна поділити на групи.

Мікроконтролери відрізняються за *архітектурою*.

Архітектуру визначає набір команд, які здатний виконувати мікроконтролер.

Два мікроконтролери є однакової архітектури, якщо вони здатні виконувати один і той самий набір команд

Архітектури мікроконтролера (2)

Архітектура	Процесор	Виробник
AMD	Au1xxx	Advanced Micro Devices, …
ARM	ARM7, ARM9, …	ARM, …
C16X	C167CS, C165H, C164CI, …	Infineon, …
ColdFire	5282, 5272, 5307, 5407, …	Motorola/Freescale, …
I960	I960	Vmetro, …
M32/R	32170, 32180, 32182, 32192, …	Renesas/Mitsubishi, …
M Core	MMC2113, MMC2114, …	Motorola/Freescale
MIPS32	R3K, R4K, 5K, 16, …	MTI4kx, IDT, MIPS Technologies, …
NEC	Vr55xx, Vr54xx, Vr41xx	NEC Corporation, …
PowerPC	82xx, 74xx,8xx,7xx,6xx,5xx,4xx	IBM, Motorola/Freescale, …
68k	680x0 (68K, 68030, 68040, 68060, …), 683xx	Motorola/Freescale, …
SuperH (SH)	SH3 (7702,7707, 7708,7709), SH4 (7750)	Hitachi, …
SHARC	SHARC	Analog Devices, Transtech DSP, Radstone, …
strongARM	strongARM	Intel, …
SPARC	UltraSPARC II	Sun Microsystems, …
TMS320C6xxx	TMS320C6xxx	Texas Instruments, …
x86	X86 [386,486,Pentium (II, III, IV)…]	Intel, Transmeta, National Semiconductor, Atlas, …
TriCore	TriCore1, TriCore2, …	Infineon, …

Організація пам’яті (1)

2 архітектури:

1. *прінстонська* архітектура (архітектура фон Неймана)
застосовує одну спільну шину для доступу до програм і даних (доступ почерговий!)

2. *гарвардська* архітектура
програми та дані розділені, шини різні (доступ може бути одночасний)

Причина застосовності гарвардської архітектури у МК – це те, що дані не вимагають стільки пам’яті, скільки програми.

Організація пам’яті (2)

Переваги гарвардської архітектури:

1. *швидкий пошук операндів (даних).*
Оскільки дані є в окремому просторі пам’яті, а не “змішані” з програмою, до них простіше звернутися.

2. *менша довжина команди.*
Команди і дані “змішані”? =>
розмір команд зростає за рахунок збільшення розрядів для адресації операндів.
Розділення команд і даних => зменшення довжини команд.

3. *більша швидкість.*
Вибір наступної команди є одночасним з виконанням поточної команди.

Система команд МК (1)

RISC & CISC:

1. *CISC-архітектура (Complicated Instruction Set Computer)* — архітектура зі складною системою команд.

CISC-архітектура заснована IBM, відомі приклади її застосування — X86 і Pentium (продукти Intel).

2. RISC-архітектура (Reduced Instruction Set Computer) — архітектура з обмеженим набором команд.
Аналіз програм для МК => Принцип Парето (80% програм використовували 20% команд) => ідея RISC-архітектури
Перший RISC-процесор був створений в університеті Берклі, він містив 31 команду.

Система команд МК (2)

RISC:

1. однаковий час виконання команд (вибірка і виконання за один такт)

2. типово кожна команда займає 1 комірку пам’яті

3. набір команд зведений до мінімуму, тож складну команду виконуємо декількома простими.

CISC:

1. різні команди мають різний час виконання (1..12 тактів)

2. в різних команд різний розмір, 1, 2, 3, рідше 4 байти

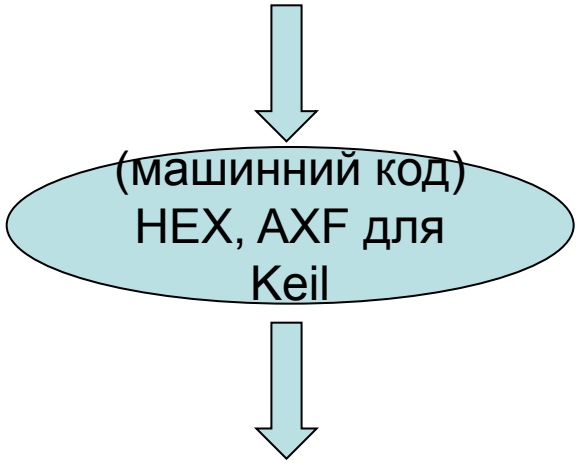
3. набір команд розвинений

Кроки створення мікропрограми

1. Написання програми однією з мов

2. Компіляція (може бути у декілька етапів)

3. Збірка (linking)



Програма специфічна для мікроконтролера певного типу!

Кожен мікроконтролер “розуміє” певний набір команд

4. Запис машинного коду у ПЗП мікроконтролера

Програматори

Тактова частота мікроконтролера

Важливий параметр, для систем реального часу – критичний.

Впливає на:
швидкість роботи МК (кожна команда виконується певну кількість тактів, швидший такт – швидше виконається)

роботу з периферією (USART, SPI,…), критично для USB

Робота блоків залежить від того, чи вони отримують тактові сигнали. *Нема тактування – нема роботи.*

Архітектура ARM

Продукт Acorn Computers, перший випуск — у 1982 р.

Інтерпретація: Acorn RISC Machine, Advanced RISC Machine

Мета та ідея створення:
низьке енергоспоживання,
простота, мінімалізм

Базується на RISC

Компанія ARM Limited займається розробленням архітектур ARM та ліцензуванням, вироблення чіпів і масове виробництво — справа ліцензіатів

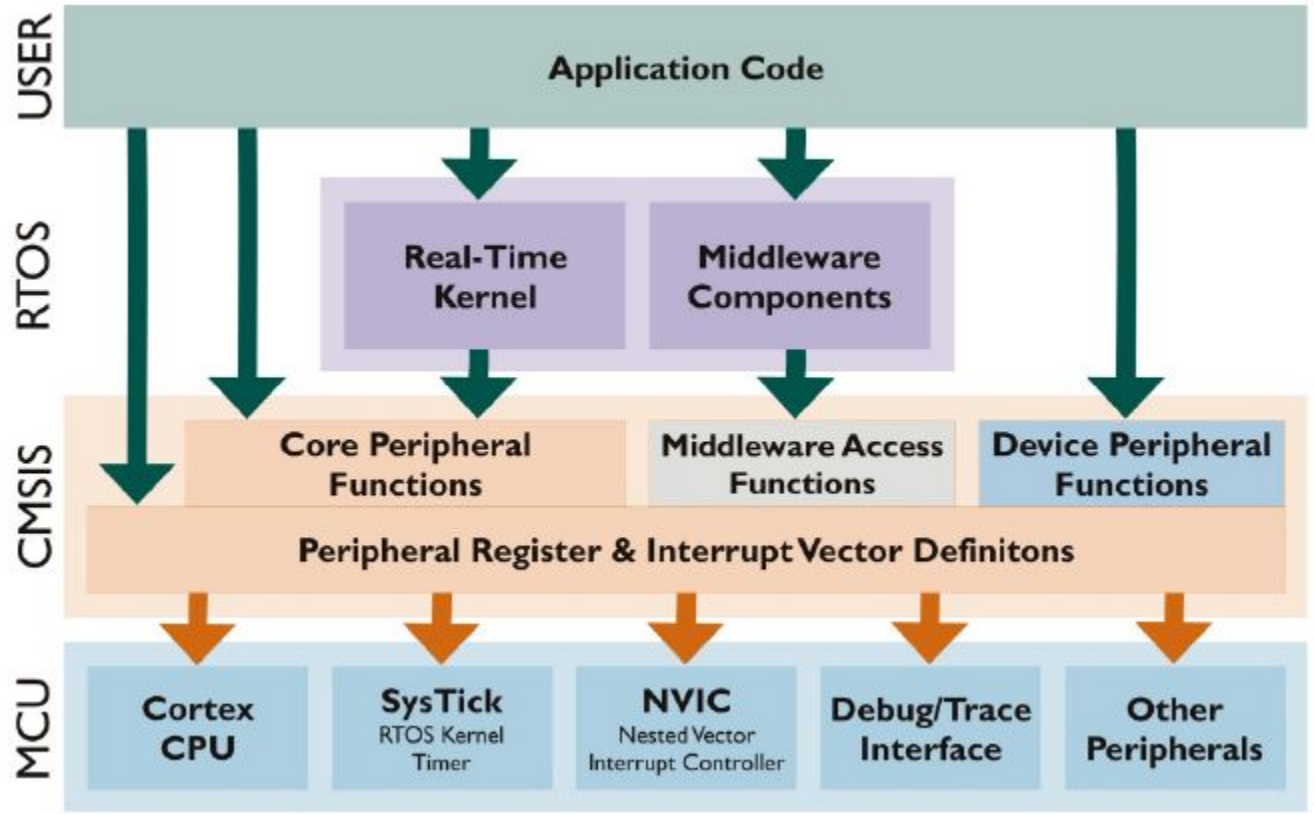
CMSIS

CMSIS – Бібліотека, стандартна для всіх МК з ядром *ARM Cortex*. Стандартизується ARM Limited.
Різні виробники МК з цим ядром *доповнюють* CMSIS файлами з описом периферійних модулів, специфічних для МК, які вони випускають
Бібліотека надає доступ до *периферійних модулів* за допомогою елементів *структур* мовою С

Передумови виникнення CMSIS:
Багато спільних апаратних ресурсів =>
різні реалізації взаємодії з ними, що роблять те саме, але по-різному, через відсутність стандартів =>
значні кошти на розробку firmware, бо потрібно організовувати зв'язок з hardware =>

компанія ARM зрозуміла, що пора виробити стандарт!

Структура CMSIS (концепція)



Функціональні рівні CMSIS

3 Рівні:

1. Core Peripheral Access Layer (CPAL) – адреси і методи доступу до компонентів для всіх МК на основі Cortex-M (регістри, NVIC,…)

2. Middleware Access Layer (MWAL) – API для периферії, специфічний для виробників

3. Device Peripheral Access Layer (DPAL)

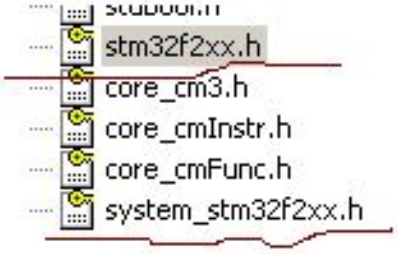
Файли у складі CMSIS

2 частини:

1. спільна для всіх МК з однаковим ядром (Cortex):
core_cm3.c + core_cm3.h, містить глобальні оголошення та визначення

2. специфічна для виробника:
system_<конкретний_МК>.c – визначення (definitions)
+ system_<конкретний_МК>.h – оголошення (declarations)
+ <конкретний_МК>.h – цей файл підключають у проект, він, у свою чергу, включає core_cm3.h і system_<конкретний_МК>.h , досить одного заголовного файлу верхнього рівня, решта “підтягуються”

для stm:
system_stm32fxxx.c +
system_stm32fxxx.h +
stm32fxxx.h



Файли у складі CMSIS (2)

system_<конкретний_МК>.h + system_<конкретний_МК>.c
конфігурація System Clock + настройки Flash + функція SystemInit, яка викликається одразу після старту/перезапуску перед початком основної програми, виклик є у startup_stm32f2xx.s

startup_stm32f2xx.s містить весь код, необхідний для запуску Cortex-M3 + вектори переривань

Startup

Містить:

первинну ініціалізацію МК
налаштування стека
занулення секції BSS
виклик функції *main*

Оформлення коду у CMSIS

CMSIS дотримується Doxygen.
Doxygen – система документування коду, написаного на C.

Зокрема, для функцій:
@brief – стислий опис функції
@param – детальний опис параметрів
@return – детальний опис результатів, які повертає функція,
після чого – детальний опис функції

```
/**
 * @brief  Enable Interrupt in NVIC Interrupt Controller
 *
 * @param  IRQn_Type IRQn specifies the interrupt number
 * @return none
 *
 * Enable a device specific interupt in the NVIC interrupt controller.
 * The interrupt number cannot be a negative value.
 */
static __INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)
{
  …
}
```

CMSIS: stm32fxxx.h

Вміст файлу: структури з описом периферійних модулів + макроси

```
402
403 typedef struct
404 {
405   __IO uint32_t DR;          /*!< CRC Data register,              A
406   __IO uint8_t  IDR;         /*!< CRC Independent data register, A
407   uint8_t       RESERVED0;   /*!< Reserved, 0x05
408   uint16_t      RESERVED1;   /*!< Reserved, 0x06
409   __IO uint32_t CR;          /*!< CRC Control register,          A
410 } CRC_TypeDef;
411
412 /**
413   * @brief Digital to Analog Converter
414   */
415
416 typedef struct
417 {
418   __IO uint32_t CR;        /*!< DAC control register,
419   __IO uint32_t SWTRIGR;   /*!< DAC software trigger register,
420   __IO uint32_t DHR12R1;   /*!< DAC channel1 12-bit right-aligned
421   __IO uint32_t DHR12L1;   /*!< DAC channel1 12-bit left aligned d
422   __IO uint32_t DHR8R1;    /*!< DAC channel1 8-bit right aligned d
423   __IO uint32_t DHR12R2;   /*!< DAC channel2 12-bit right aligned
424   __IO uint32_t DHR12L2;   /*!< DAC channel2 12-bit left aligned d
425   __IO uint32_t DHR8R2;    /*!< DAC channel2 8-bit right-aligned d
426   __IO uint32_t DHR12RD;   /*!< Dual DAC 12-bit right-aligned data
427   __IO uint32_t DHR12LD;   /*!< DUAL DAC 12-bit left aligned data
428   __IO uint32_t DHR8RD;    /*!< DUAL DAC 8-bit right aligned data
429   __IO uint32_t DOR1;      /*!< DAC channel1 data output register,
430   __IO uint32_t DOR2;      /*!< DAC channel2 data output register,
431   __IO uint32_t SR;        /*!< DAC status register,
432 } DAC_TypeDef;
433
```

Standard Peripherals Library (SPL)

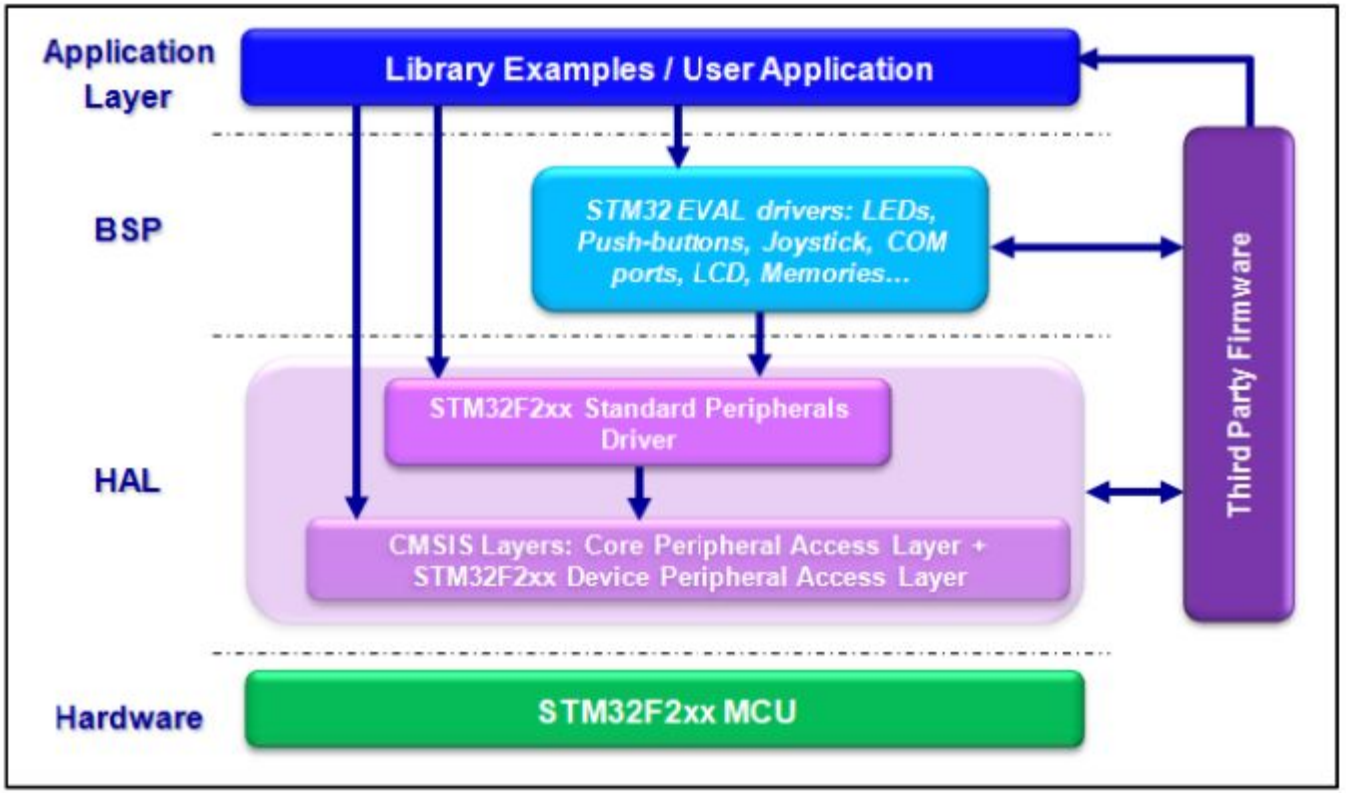
SPL – Бібліотека, створена STMicroelectronics для полегшення доступу до периферії МК stmXX.

Містить:
1. засоби звертання до всіх регістрів, разом з їхніми виводами, мовою С

2. драйвери, написані у відповідності до *MISRA C 2004* на *Strict ANSI C* (для забезпечення незалежності від засобів розробки), повністю *документовані*,
кожен драйвер = (структури + набір функцій, створені строго за визначеними правилами)

Крім того, STMicroelectronics пропонує ряд навчальний проектів, що показують застосування бібліотеки для вирішення типових задач

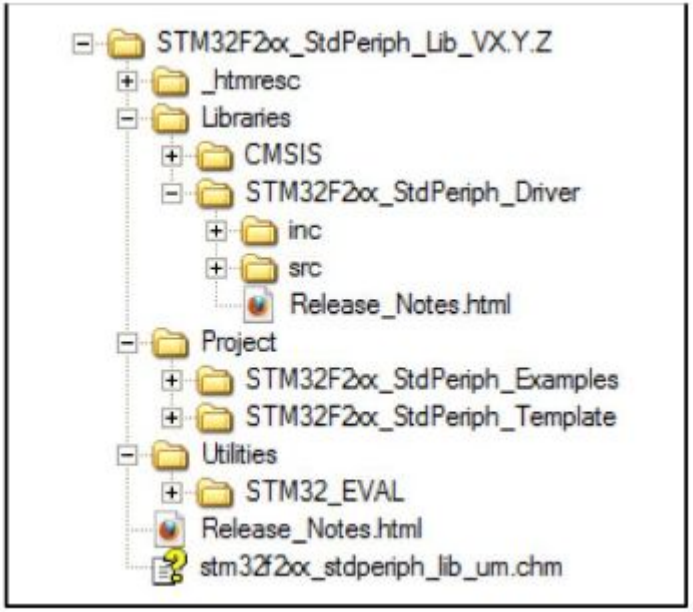
Архітектура SPL



Структура бібліотеки SPL

Бібліотека SPL постачається у zip-архіві

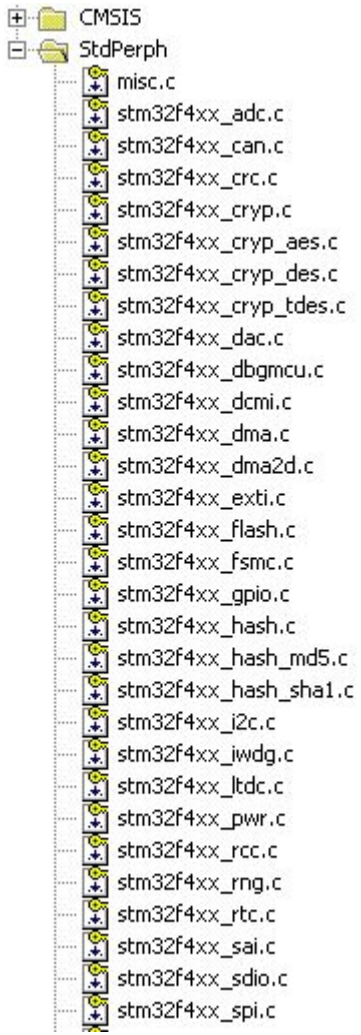
підкаталоги:
CMSIS
STM32Fxxx_StdPeriph_Driver



SPL: STM32Fxxx_StdPeriph_Driver

підкаталоги:
inc – заголовні файли драйверів
src – c-файли драйверів

Для будь-якого периферійного модуля є пара файлів (.h у inc + .c у src),
ім’я цих файлів складається з префіксу stm32fxxx, знаку “_” та імені периферії, наприклад:
stm32fxxx_gpio.h, stm32fxxx_gpio.c – для роботи з портами вводу/виводу



Конфігурація/ініціалізація периферії (1)

Вся периферія описана структурами. Потрібні кроки:

1. Підключити потрібний .h-файл

2. Створити екземпляр структури вигляду PPP_InitTypeDef (PPP заміняє ім'я периферійного модуля):
PPP-InitTypeDef PPPInitStructure;

3. Увімкнути тактування! Нема тактування – нема роботи. Усі модулі спочатку неактивні, енергії не споживають

Тактування периферійного модуля вмикається однією з цих функцій:

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

Конфігурація/ініціалізація периферії (2)

Щоб знати, яку з 3 функцій для тактування слід викликати, *потрібно знати, до якої шини підключений периферійний модуль* (інф-ія з документації або з файлу stm32f10x_rcc.c/h) із документації на функцію у стилі Doxygen видно, що може бути її параметром

В ARM Cortex-M3 3 шини, шини даних і управління об'єднані у AHB (ARM Hi-Speed Bus).

Пристрої вводу/виводу з'єднуються з AHB через проміжні шини (ARM Peripheral Bus) APB1 (швидкість нижча) и APB2 (швидкість вища).

Типово пристрої, що працюють на меншій швидкості, під'єднані до APB1

Конфігурація/ініціалізація периферії (3)

4. Задати значення полям екземпляра структури

5. Викликати функцію PPP_Init(PPP, &PPP_InitStructure);

На цьому ініціалізація завершена! Для зміни конфігурації слід переприсвоїти значення полям екземпляра структури та повторно викликати функцію PPP_Init

Приклад конфігурації GPIO (1)

1. Функції і дані для роботи з портами вводу/виводу загального призначення описані у stm32fxxx_gpio.h/c, структура GPIO_InitTypedef міститься у stm32fxxx_gpio.h

2. Створення екземпляра структури: GPIO_InitTypeDef GPIO_InitStructure

```
GPIO_InitTypeDef GPIO_InitStructure;

#ifdef USE_STM32H_103

        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
        GPIO_Init(GPIOC, &amp;GPIO_InitStructure);
```

```
//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;
```

Приклад конфігурації GPIO (2)

3. Після увімкнення тактування однією з функцій RCC_APB2PeriphClockCmd, RCC_APB1PeriphClockCmd, RCC_APHPeriphClockCmd слід

4. задати значення екземпляру структури, наприклад:

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

5. і викликати

```
GPIO_Init( GPIOC, &GPIO_InitStructure);
```

Одним викликом RCC_APB2PeriphClockCmd можна ввімкнути тактування декількох модулів (звісно, підключених до тієї ж шини), через “або”:

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA |
RCC_APB2Periph_GPIOC, ENABLE);
```

Ввімкнення периферійного модуля

Після конфігурації та ініціалізації для ряду пристроїв викликають

PPP_Cmd(<ім’я_модуля>)

Для портів вводу/виводу натомість застосовуються функції:

GPIO_SetBits(GPIOx,GPIO_Pin_y);
GPIO_ResetBits(GPIOx,GPIO_Pin_y);
Одним викликом цих двох функцій можна встановлювати в 1 або в 0 декілька виводів одночасно, скориставшись “або”