HỆ THỐNG NHÚNG

THS. LƯU HOÀNG

TRƯỜNG ĐẠI HỌC BÀ RỊA – VŨNG TÀU
VIỆN CNNT - ĐIỆN - ĐIỆN TỬ

1

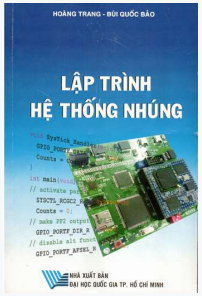
Nội dung môn học

1. Khái niệm Hệ thống nhúng
2. Ôn tập Lập trình C
3. Giới thiệu ARM
4. KIT ARM STM32F407 Discovery
5. Các phần mềm lập trình ARM
6. Các bài tập trên KIT ARM STM32F407 Discovery
7. Tiểu luận

2

Tài liệu tham khảo

- *Bài giảng Hệ thống nhúng* – ĐH BRVT.
- *Lập trình hệ thống nhúng* – Hoàng Trang, Bùi Quốc Bảo – NXB ĐHQG TPHCM.
- *Lập trình nhúng căn bản* – Vũ Đức Lung, Trần Ngọc Đức – NXB ĐHQG TPHCM.
- *Hệ thống điều khiển nhúng* – Lưu Hồng Việt.
- Web: hocarm.org



3

Mục đích của môn học

- Nắm được khái niệm chung về hệ thống nhúng
- Có kiến thức về ARM
- Có khả năng lập trình bằng C cho ARM
- Có khả năng tự xây dựng ứng dụng nhúng trên ARM và các loại VĐK khác

4

Đánh giá kết quả học tập

- **Bài kiểm tra (20% điểm)**
  - Kiểm tra 45' vào giữa học kỳ.
- **Điểm chuyên cần (20% điểm)**
  - Đi học đầy đủ và nghiêm túc.
  - Làm đủ bài kiểm tra và đạt trên trung bình.
- **Điểm cuối kỳ (60% điểm)**
  - Tiểu luận nhóm: Thiết kế hệ thống nhúng dùng ARM STM32F407

5

KHÁI NIỆM VỀ HỆ THỐNG NHÚNG

1. KHÁI NIỆM HỆ THỐNG NHÚNG
2. ỨNG DỤNG CỦA HỆ THỐNG NHÚNG

6

HỆ THỐNG NHÚNG LÀ GÌ?

- **Hệ thống nhúng** (*embedded system*) là một thuật ngữ để chỉ một hệ thống có khả năng tự trị được nhúng vào trong một môi trường hay một hệ thống mẹ.
- Là các hệ thống tích hợp cả phần cứng và phần mềm phục vụ các bài toán chuyên dụng trong nhiều lĩnh vực công nghiệp, tự động hoá điều khiển, quan trắc và truyền tin. Đặc điểm của các hệ thống nhúng là hoạt động ổn định và có tính năng tự động hoá cao.

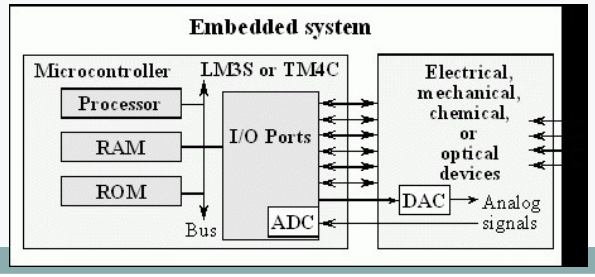
7

HỆ THỐNG NHÚNG LÀ GÌ?

8

HỆ THỐNG NHÚNG LÀ GÌ?

- Một hệ thống nhúng sẽ có một hoặc nhiều microcomputer bên trong. Một vi điều khiển (microcontroller) là một microcomputer kết hợp với bộ xử lý(Processor), RAM, ROM, và các cổng I/O thành một khối duy nhất. Đây là loại thường được dùng cho các hệ thống nhúng rất nhiều bởi vì nó rẻ, kích thước nhỏ và đáp ứng được các yêu cầu về tiêu thụ năng lượng thấp.



9

HỆ THỐNG NHÚNG LÀ GÌ?

- Hệ thống nhúng thường được thiết kế để thực hiện một chức năng chuyên biệt nào đó.
- Một hệ thống nhúng chỉ thực hiện một hoặc một vài chức năng nhất định, thường đi kèm với những yêu cầu cụ thể và bao gồm một số thiết bị máy móc và phần cứng chuyên dụng.
- Hệ thống nhúng có thể tối ưu hóa nó nhằm giảm thiểu kích thước và chi phí sản xuất.
- Các hệ thống nhúng thường được sản xuất hàng loạt với số lượng lớn.

10

HỆ THỐNG NHÚNG VỚI MÁY TÍNH

11

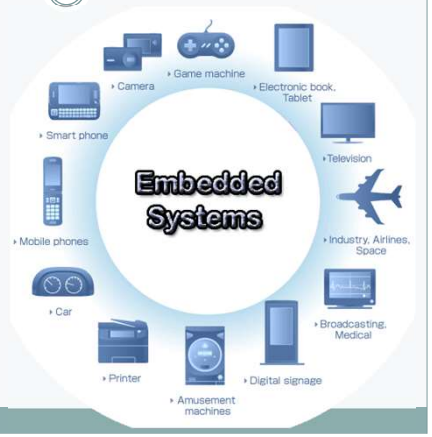
HỆ THỐNG NHÚNG VỚI MÁY TÍNH

COMPUTER	WASHING MACHINE
Sử dụng phần cứng và phần mềm	Sử dụng phần cứng và phần mềm
Khả năng xử lý linh hoạt nhiều tác vụ với tốc độ rất cao	Chỉ có khả năng thực một công việc chuyên biệt trên một hệ thống được thiết kế riêng
Khả năng xử lý một lượng dữ liệu rất lớn	Khả năng xử lý một lượng dữ liệu giới hạn

12

ỨNG DỤNG CỦA HỆ THỐNG NHÚNG

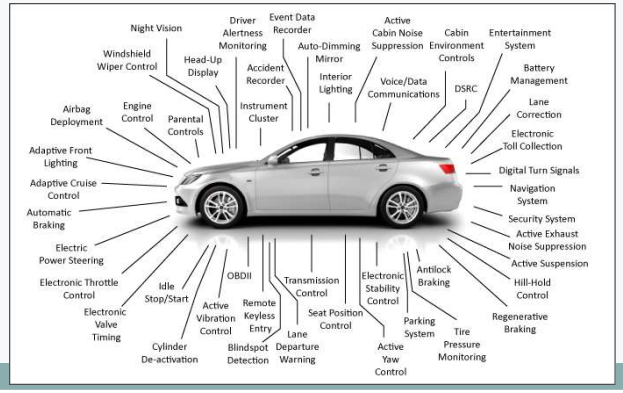
- Các thiết bị điều khiển
- Ôtô, tàu điện
- Truyền thông
- Thiết bị y tế
- Hệ thống đo lường
- Toà nhà thông minh
- Thiết bị trong các dây chuyền sản xuất
- Robot
- Thiết bị gia dụng
- ...



13

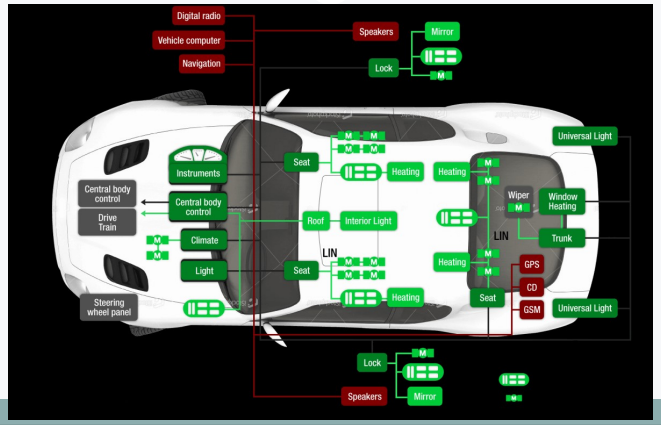
ỨNG DỤNG CỦA HỆ THỐNG NHÚNG

- VD: Xe ôtô ngày nay có trên 50 hệ thống máy tính.



14

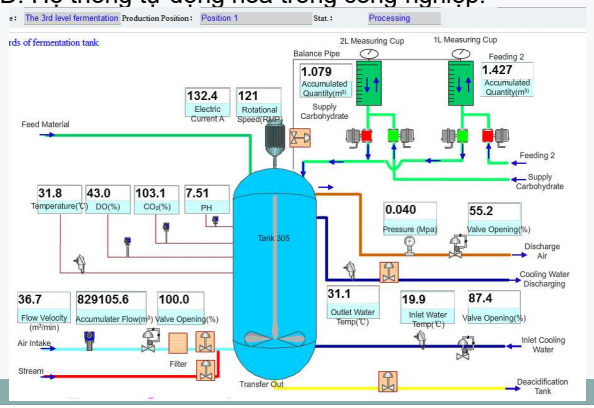
ỨNG DỤNG CỦA HỆ THỐNG NHÚNG



15

ỨNG DỤNG CỦA HỆ THỐNG NHÚNG

- VD: Hệ thống tự động hóa trong công nghiệp.



16

VI XỬ LÝ TRONG HỆ THỐNG NHÚNG

Tùy thuộc vào ứng dụng và giá thành, người thiết kế quyết định loại vi xử lý dùng trong hệ thống nhúng.

- Họ 8086
- PowerPC
- Họ 8051
- PIC
- AVR
- ARM
- …

17

LẬP TRÌNH C CHO VI ĐIỀU KHIỂN

MỘT SỐ KHÁI NIỆM NGÔN NGỮ LẬP TRÌNH C.
CẤU TRÚC ĐIỀU KHIỂN VÀ HÀM.

18

Một số khái niệm C cho Vi điều khiển

- Một chương trình C thường bao gồm các thành phần như:
  - Các kiểu dữ liệu
  - chú thích (comments)
  - biểu thức (expressions)
  - câu lệnh (statements)
  - khối (blocks)
  - toán tử (operator)
  - cấu trúc điều khiển (Flow controls)
  - hàm (functions)

19

Các kiểu dữ liệu

Tên kiểu dữ liệu (Data type)	Số byte	Khoảng dữ liệu (Range)
char	1	–128 to 127 or 0 to 255
unsigned char	1	0 to 255
int	2	–32,768 to 32,767
unsigned int	2	0 to 65,535
long	4	–2,147,483,648 to 2,147,483,647
unsigned long	4	0 to 4,294,967,295
float	4	6 digits of precision

20

Chú thích (comments)

- Có 2 cách để tạo phần chú thích trong C là:
  - chú thích từng dòng bằng 2 dấu "//"

  ví dụ: //day la chu thich
  - chú thích block bằng cách kẹp block cần chú thích vào giữa /* ….*/

  ví dụ:

```
/*
Ban co the type bat ky chu thich nao trong block nay
Ngay ca khi ban xuong dong
Phan chu thich thuong co mau chu la green
*/
```

21

Tiền xử lý (preprocessor)

- Tiền xử lý là một tiện ích của ngôn ngữ C, các preprocessor được trình biên dịch xử lý trước tất cả các phần khác.
- Preprocessor được bắt đầu bằng dấu "#".
- Trong ngôn ngữ C có hai preprocessors được sử dụng phổ biến nhất là #include và #define.
- #include chỉ định 1 file được đính kèm trong quá trình biên dịch.
- #define để định nghĩa 1 chuỗi thay thế hoặc 1 macro.

22

Tiền xử lý (preprocessor)

- #include delay.h

/* đính kèm nội dung file delay.h trong lúc biên dịch */

- #define Lamp1 PORTA.5

/* định nghĩa Lamp1 thay thể cho PORTA.5 */

- #define max (a,b) ((a)>(b)? (a): (b))

/* định nghĩa một macro tìm số lớn nhất trong 2 số a và b, trong chương trình nếu bạn gọi x=max(2,3) thì kết quả thu được x=3 */

23

Biểu thức (Expressions)

- Biểu thức là 1 phần của các câu lệnh.
- Biểu thức có thể bao gồm biến, toán tử, gọi hàm…, biểu thức trả về 1 giá trị đơn. Biểu thức không phải là 1 câu lệnh hoàn chỉnh.
- Ví dụ: A>B.

24

Câu lệnh (Statement)

- Câu lệnh là 1 dòng lệnh hoàn chỉnh, có thể bao gồm các keywords, biểu thức và các câu lệnh khác.
- Câu lệnh được kết thúc bằng dấu ";".
- Ví dụ: if(A>B) A=A-B; … là một các câu lệnh

25

Khối (Blocks)

- Khối là sự kết hợp của nhiều câu lệnh để thực hiện chung 1 nhiệm vụ nào đó.
- Khối được bao bởi 2 dấu mở khối "{" và đóng khối "}".
- Ví dụ 1 khối:

```
while(1){
    PORTB=val;
    _delay_loop_2(65000);
    val*=2;
    if (!val) val=1;
}
```

26

Toán tử (Operators)

- Toán tử là những ký hiệu báo cho trình biên dịch các nhiệm vụ cần thực hiện.
- Các bảng dưới đây tóm tắt các toán tử C dùng cho lập trình AVR.

27

Bảng 1: các toán tử đại số

- Các toán tử đại số dùng thực hiện các phép toán đại số quen thuộc.
- Chú ý phân biệt y=x++ và y=++x

Operator	Name	Example	Defined
*	Multiplication	x*y	Multiply x times y
/	Division	x/y	Divide x by y
%	Modulo	x%y	Provide the remainder of x divided by y
+	Addition	x+y	Add x and y
-	Subtraction	x-y	Subtract y from x
++	Increment	x++	Increment x after using it
--	Decrement	--x	Decrement x before using it
-	Negation	-x	Multiply x by –1
+	Unary Plus	+x	Show x is positive (not really needed)

28

Bảng 2: Toán tử truy cập và kích thước

- Toán tử [] thường được sử dụng khi dùng mảng, phần tử thứ i của mảng sẽ được truy xuất thông qua [i].
- Chú ý mảng trong C bắt đầu từ 0.

Operator	Name	Example	Defined
[]	Array element	x[6]	Seventh element of array x
.	Member selection	PORTD.2	Bit 2 of Port D
->	Member selection	pStruct->x	Member x of the structure pointed to by pStruct
*	Indirection	*p	Contents of memory located at address p
&	Address of	&x	Address of the variable x

29

Mảng và chuỗi (Array anh String)

- Mảng là tập hợp các biến giống nhau có cùng tên gọi.
- Các phần tử của mảng được xác định bằng chỉ số (bắt đầu từ 0).
- Kích thước của mảng là cố định.
- VD: `int a[4] = {1,2,3,4};`
  `int b = a[0];`

30

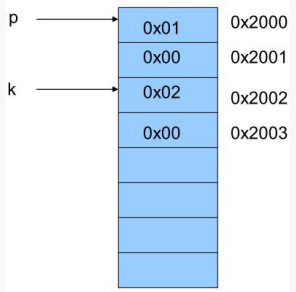
Mảng và chuỗi (Array anh String)

- Chuỗi tương tự mảng, ngoại trừ Chuỗi có thể có số lượng phần tử thay đổi.
- Các phần tử của chuỗi là ký tự ASCII kết thúc bằng giá trị 0.
- Chuỗi được khai báo như 1 con trỏ kiểu char.
- VD: `char *  mystring = “embedded”;`

31

Con trỏ (Pointer)

- Con trỏ là 1 biến chứa một địa chỉ.
- Giá trị của con trỏ có thể thay đổi được.
- VD:
```
char *p;
char *k;
p = 0x2000;
*p = 1;
k = p+2;
*k = *p+1;
```



32

Phép toán với Con trỏ

- **Phép cộng:** Con trỏ luôn chỉ vào địa chỉ đầu của một đối tượng (object). Cộng 1 vào con trỏ làm nó chỉ đến đối tượng tiếp theo.
- **Phép so sánh:** Khi so sánh 2 con trỏ, giá trị chúng đang mang được coi như số không dấu.

33

Bảng 3: Toán tử Logic và quan hệ

- Toán tử Logic và quan hệ: thực hiện các phép so sánh và logic, thường được dùng làm điều kiện trong các cấu trúc điều khiển.

Operator	Name	Example	Defined
>	Greater than	x>y	1 if x is greater than y, otherwise 0
>=	Greater than or equal to	x>=y	1 if x is greater than or equal to y, otherwise 0
<	Less than	x<y	1 if x is less than y, otherwise 0
<=	Less than or equal to	x<=y	1 if x is less than or equal to y, otherwise 0
==	Equal to	x==y	1 if x equals y, otherwise 0
!=	Not equal to	x!=y	1 if x is not equal to y, otherwise 0
!	Logical NOT	!x	1 if x is 0, otherwise 0
&&	Logical AND	x&&y	0 if either x or y is 0, otherwise 1
||	Logical OR	x||y	0 if both x and y are 0, otherwise 1

34

Bảng 4: Toán tử thao tác Bit

- Toán tử thao tác Bit (Bitwise operator): là các toán tử thực hiện trên từng bit nhị phân của các con số.

Operator	Name	Example	Defined
~	Bitwise complement NOT	~x	Changes 1 bits to 0 and 0 bits to 1
&	Bitwise AND	x&y	Bitwise AND of x and y
|	Bitwise OR	x|y	Bitwise OR of x and y
>>	Right shift	x>>3	Bits in x shifted right 3 bit positions
<<	Left shift	x<<2	Bits in x shifted left 2 bit positions

35

Bảng 5: Các toán tử khác

Operator	Name	Example	Defined
()	Funtion	Delay(10)	Call Delay with an argument of 10
(type)	Type cast	(int)x	X converted to a int
?:	Conditional	x?y:z	If x is not đ evalute y, otherwise evalute z
,	Sequential evaluation	x++,y++	Increment x first, then increment y

36

Cấu trúc điều khiển

- Cấu trúc IF
- Cấu trúc IF … ELSE …
- Cấu trúc SWITCH
- Cấu trúc WHILE
- Cấu trúc DO … WHILE
- Cấu trúc FOR

37

Cấu trúc IF

- Cú pháp: ***If (điều kiện) statement;***
- Giải thích: nếu điều kiện là đúng thì thực hiện statement theo sau.
- statement có thể được trình bày cùng dòng hoặc dòng sau điều khiển If.
- Điều kiện có thể là một biểu thức bất kỳ, có thể là sự kết hợp của nhiều điều kiện bằng các toán tử quan hệ AND (&&), OR (||)…Điều kiện được cho là đúng khi nó khác 0.
- Ví dụ: **If (x==1 && y==2) result='A';**

38

Cấu trúc IF .. ELSE

- Cú pháp: ***If (điều kiện) statement1; else statement2;***
- Giải thích: nếu điều kiện đúng thì thực hiện statement1, ngược lại thực thi statement2.
- Việc đặt các statement và else..trên cùng 1 dòng hay trên những dòng khác nhau đều không ảnh hưởng đến kết quả.
- Nếu có nhiều statements thì cần đặt chúng trong 1 khối.
- Ví dụ: **If (x>y) result='A' else result='B';**

39

Cấu trúc SWITCH

- Cú pháp:

```
switch (biểu thức) {
    case hằng_số_1:
        các statement1;
    break;
    case hằng_số_2:
        các statement2;
    break;
    default:
        các statement khác;
    break;
}
```

- Giải thích: trong trường hợp có nhiều khả năng có thể xảy ra cho 1 biểu thức (hay 1 biến), ứng với mỗi khả năng chương trình cần thực hiện một việc nào đó, khi này nên sử dụng cấu trúc Switch.

40

Cấu trúc SWITCH

- Ví dụ:
```
switch (Command) {
    case 1:
        PWM=255;
        ON_Motor();
        break;
    case 2:
        PWM=0;
        OFF_Motor();;
        break;
    default:
        Get_Cmd();
         break;
    }
```

41

Cấu trúc WHILE

- Cú pháp: *while (điều kiện) statement1;*
- Giải thích: while là một cấu trúc lặp (Loop), ý nghĩa của cấu trúc while là khi điều kiện còn đúng thì sẽ thực hiện statement1 (hoặc các statements nếu chúng được đặt trong 1 khối {} như trong trường hợp của if được giới thiệu ở trên).
- Ví dụ:
```
x=10;y=3;
while(x>y) x=x-y;
```

42

Cấu trúc DO ... WHILE

- Cú pháp:
```
Do { statement1;
     statement2;
   ...}
While (điều kiện );
```
- Giải thích: tương tự cấu trúc while nhưng cấu trúc do… while sẽ thực thi các câu lệnh trước 1 lần mới kiểm tra điều kiện.
- Ví dụ:
```
x=3;y=3;
do x=x-y;
while(x>y);
```

43

Cấu trúc FOR

- Cú pháp:
```
for (biểu_thức_1 ; biểu_thức_2 ; biểu_thức_3) {
     các statement;
}
```
- Giải thích: là một cấu trúc lặp khác, trong cấu trúc for, biểu_thức_1 thường được hiểu là khởi tạo, biểu_thức_2 là điều kiện và biểu_thức_3 là biểu thức được thực hiện sau mỗi vòng lặp.
- Ví dụ:
```
for (i=0 ; i<200 ; i++){
     PORTB=i;
     delay_ms(1000);
};
```

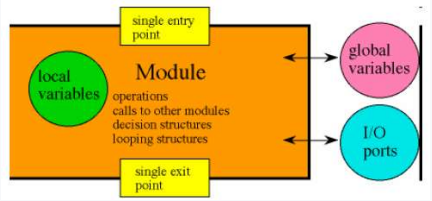
44

FUNTION
(HÀM, CHƯƠNG TRÌNH CON)

- Một chương trình lớn thường được chia thành nhiều khối (module)
- Module là 1 tác vụ nhận dữ liệu vào (input), xử lý và xuất ra kết quả (output).
- Các module được tạo ra như là các hàm (function).

45

FUNTION
(HÀM, CHƯƠNG TRÌNH CON)



- Ví dụ: hàm chuyển đổi độ F sang độ C

```
int FtoC(int TempF){
    int TempC;
    TempC=(5*(TempF-32))/9; // conversion
    return TempC;}
```

46

ĐỊNH NGHĨA HÀM
(FUNTION DEFINATION)

- Một định nghĩa của hàm là cách mà hàm đó thực thi.

```
type Name(parameter list){
    Statement
};
```

- Ví dụ: hàm chuyển đổi độ F sang độ C

```
int FtoC(int TempF){
    int TempC;
    TempC=(5*(TempF-32))/9; // conversion
    return TempC;}
```

47

MÔ TẢ HÀM
(FUNTION DECLARATION)

- Mô tả hàm cho ta biết tên hàm, kiểu của các tham số và kiểu kết quả trả về.

// declaration	input	output
void Ritual(void);	*// none*	*none*
char InChar(void);	*// none*	*8-bit*
void OutChar(char letter);	*// 8-bit*	*none*
short InSDec(void);	*// none*	*16-bit*
void OutSDec(short i);	*// 16-bit*	*none*
char Max(char in1,char in2);	*// two 8-bit*	*8-bit*
int EMax(int in1,int in2);	*// two 16-bit*	*16-bit*
void OutString(char mystring);*	*// pointer to 8-bit*	*none*
*char *alloc(int size);*	*// 16-bit*	*pointer to 8-bit*
*int Exec(void(*fnctPt)(void));*	*// function pointer*	*16-bit*

48

Cấu trúc Chương trình C

- một chương trình C cho AVR phải bao gồm 1 chương trình chính main, tất cả các nội dung chính sẽ được đặt bên trong chương trình chính. Cấu trúc chương trình chính có thể như sau:

```
void main(void){
    // Declare your local variables here
    while (1){
    // Place your code here

    }
    // Place your funtions here
}
```

49

VI ĐIỀU KHIỂN ARM

1. GIỚI THIỆU
2. MỘT SỐ LOẠI ARM THÔNG DỤNG
3. KIT STM32F407 DISCOVERY

50

GIỚI THIỆU ARM

- ARM (**Advanced RISC Machine**) là một loại cấu trúc vi xử lý 32 bit và 64 bit kiểu RISC được sử dụng rộng rãi trong các thiết kế nhúng.
- Do có đặc điểm tiết kiệm năng lượng, các bộ CPU ARM chiếm ưu thế trong các sản phẩm điện tử di động, mà với các sản phẩm này việc tiêu tán công suất thấp là một mục tiêu thiết kế quan trọng hàng đầu.

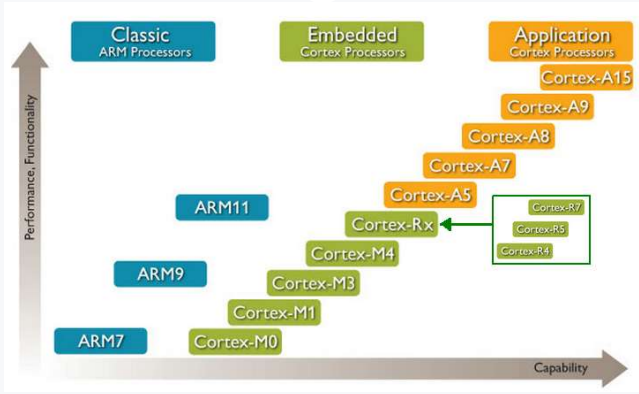
51

GIỚI THIỆU ARM

- Ngày nay, hơn 75% CPU nhúng 32-bit là thuộc họ ARM, điều này khiến ARM trở thành cấu trúc 32-bit được sản xuất nhiều nhất trên thế giới.
- Các nhà sản xuất IC đưa ra thị trường hon 240 dòng vi điều khiển sử dụng lõi ARM .
- CPU ARM được tìm thấy khắp nơi trong các sản phẩm thương mại điện tử, từ thiết bị cầm tay (PDA, điện thoại di động, máy đa phương tiện, máy trò chơi cầm tay, và máy tính cầm tay) cho đến các thiết bị ngoại vi máy tính (ổ đĩa cứng, bộ định tuyến để bàn).

52

GIỚI THIỆU ARM



53

Các dòng ARM

- Các chip ARM7 và ARM9 được các nhà sản xuất bán dẫn thiết kế với giải pháp riêng của mình, đặc biệt là phần xử lý các các ngắt đặc biệt (exception) và các ngắt thông thường (interrupt).
- Dòng ARM Cortex là một bộ xử lý thế hệ mới đưa ra một kiến trúc chuẩn cho nhu cầu đa dạng về công nghệ.
- Không giống như các chip ARM khác, dòng Cortex là một lõi xử lý hoàn thiện, đưa ra một chuẩn CPU và kiến trúc hệ thống chung.

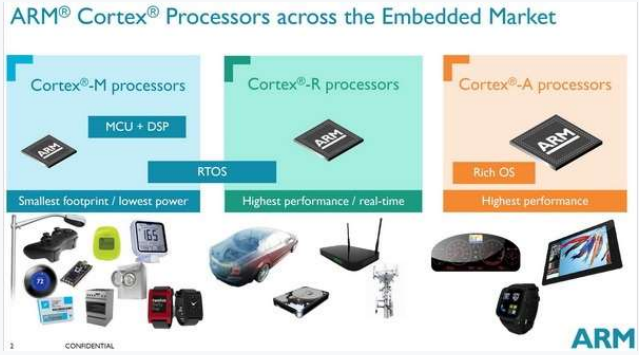
54

Các dòng ARM

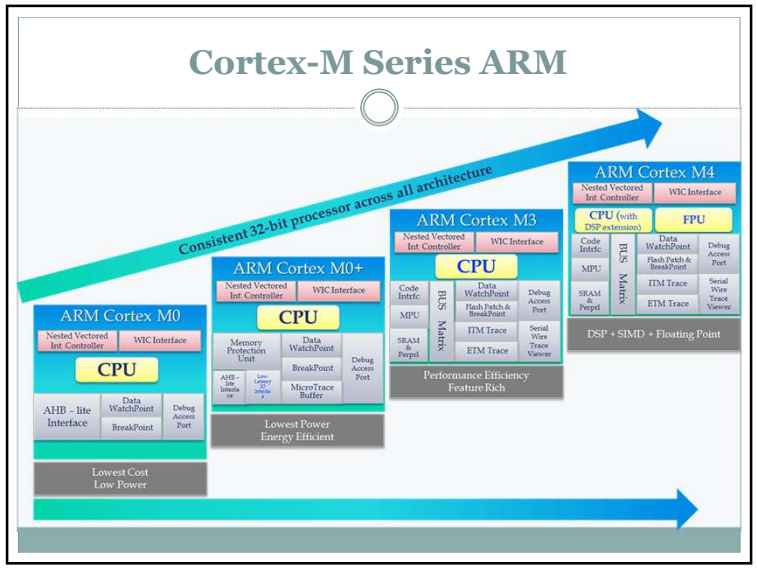
- Dòng Cortex gồm có 3 phân nhánh chính:
  - dòng A dành cho các ứng dụng cao cấp.
  - dòng R dành cho các ứng dụng thời gian thực.
  - dòng M dành cho các ứng dụng vi điều khiển và chi phí thấp.
- STM32 được thiết kế dựa trên dòng Cortex-M3, dòng Cortex-M3 được thiết kế đặc biệt để nâng cao hiệu suất hệ thống, kết hợp với tiêu thụ năng lượng thấp
- Cortex-M3 được thiết kế trên nền kiến trúc mới, do đó chi phí sản xuất đủ thấp để cạnh tranh với các dòng vi điều khiển 8 và 16-bit truyền thống.

55

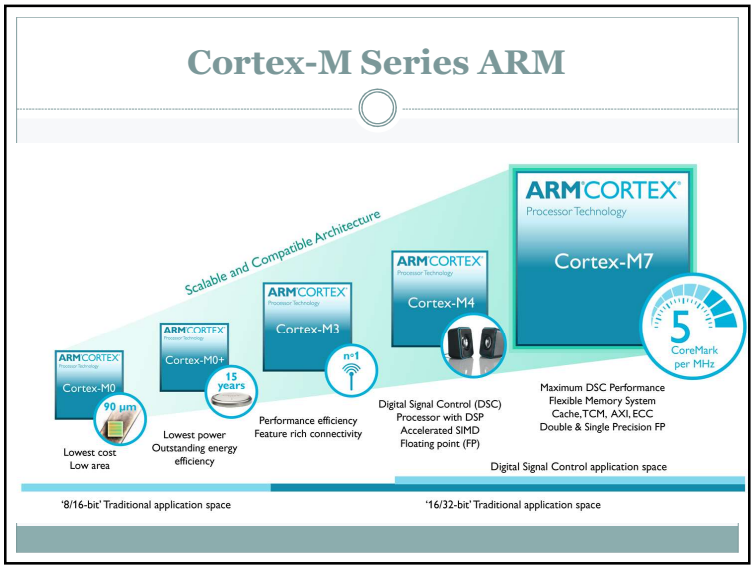
Các dòng ARM



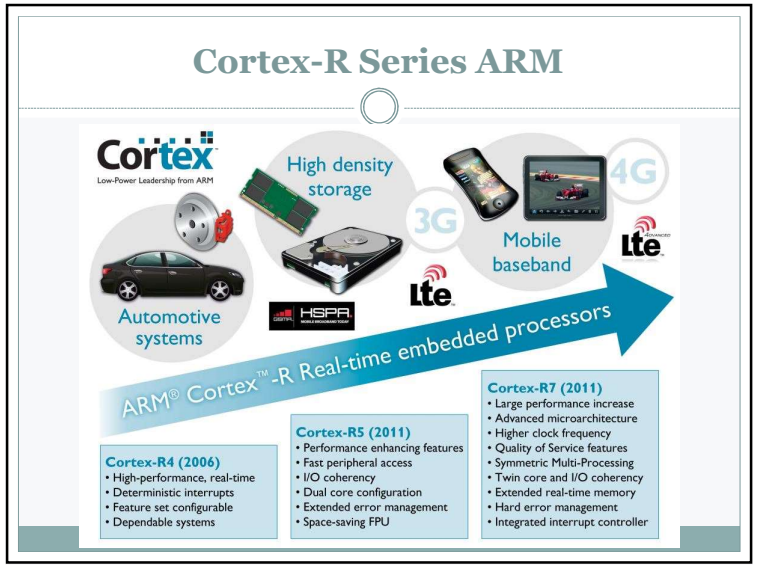
56


Cortex-M Series ARM
Consistent 32-bit processor across all architecture
ARM Cortex M0
Nested Vectored Int Controller
WIC Interface
CPU
AHB – lite Interface
Data WatchPoint
BreakPoint
Debug Access Port
Lowest Cost Low Power
ARM Cortex M0+
Nested Vectored Int Controller
WIC Interface
CPU
Memory Protection Unit
Data WatchPoint
BreakPoint
MicroTrace Buffer
Debug Access Port
Lowest Power Energy Efficient
ARM Cortex M3
Nested Vectored Int Controller
WIC Interface
CPU
Code Intrfc
MPU
SRAM & Perpri
BUS Matrix
Data WatchPoint
Flash Patch & BreakPoint
ITM Trace
ETM Trace
Serial Wire Trace Viewer
Debug Access Port
Performance Efficiency Feature Rich
ARM Cortex M4
Nested Vectored Int Controller
WIC Interface
CPU (with DSP extension)
FPU
DSP + SIMD + Floating Point

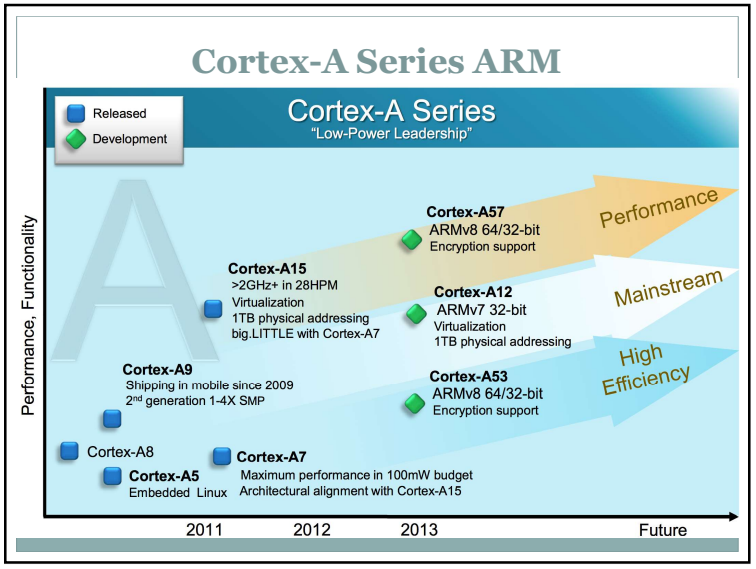
57


Cortex-M Series ARM
Scalable and Compatible Architecture
Cortex-M0
90 µm
Lowest cost
Low area
Cortex-M0+
15 years
Lowest power
Outstanding energy efficiency
Cortex-M3
n-1
Performance efficiency
Feature rich connectivity
Cortex-M4
Digital Signal Control (DSC)
Processor with DSP
Accelerated SIMD
Floating point (FP)
ARM CORTEX
Processor Technology
Cortex-M7
5 CoreMark per MHz
Maximum DSC Performance
Flexible Memory System
Cache,TCM, AXI, ECC
Double & Single Precision FP
Digital Signal Control application space
'8/16-bit' Traditional application space
'16/32-bit' Traditional application space

58

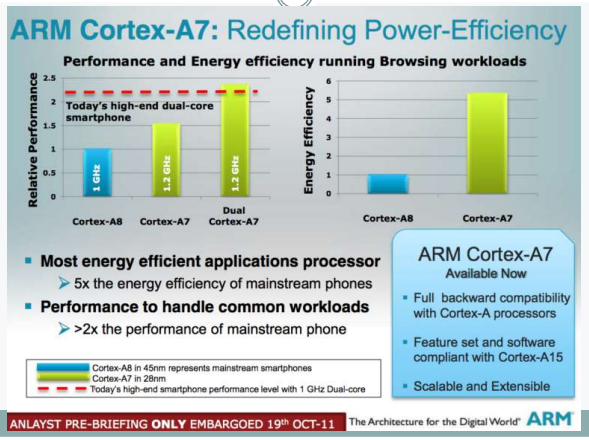

Cortex-R Series ARM
Cortex
Low-Power Leadership from ARM
High density storage
3G
4G
Mobile baseband
Lte
Automotive systems
HSPA
ARM® Cortex™ -R Real-time embedded processors
Cortex-R4 (2006)
• High-performance, real-time
• Deterministic interrupts
• Feature set configurable
• Dependable systems
Cortex-R5 (2011)
• Performance enhancing features
• Fast peripheral access
• I/O coherency
• Dual core configuration
• Extended error management
• Space-saving FPU
Cortex-R7 (2011)
• Large performance increase
• Advanced microarchitecture
• Higher clock frequency
• Quality of Service features
• Symmetric Multi-Processing
• Twin core and I/O coherency
• Extended real-time memory
• Hard error management
• Integrated interrupt controller

59


Cortex-A Series ARM
Released
Development
Cortex-A Series
"Low-Power Leadership"
Performance, Functionality
Performance
Mainstream
High Efficiency
Cortex-A57
ARMv8 64/32-bit
Encryption support
Cortex-A15
>2GHz+ in 28HPM
Virtualization
1TB physical addressing
big.LITTLE with Cortex-A7
Cortex-A12
ARMv7 32-bit
Virtualization
1TB physical addressing
Cortex-A9
Shipping in mobile since 2009
2nd generation 1-4X SMP
Cortex-A53
ARMv8 64/32-bit
Encryption support
Cortex-A8
Cortex-A5
Embedded Linux
Cortex-A7
Maximum performance in 100mW budget
Architectural alignment with Cortex-A15
2011
2012
2013
Future

60

Cortex-A Series ARM



61

Ưu điểm của ARM

- Được xây dựng theo cấu trúc mở, quy trình xử lý các thuật toán hiệu quả hơn để bảo vệ CPU không bị quá tải, tiết kiệm bộ nhớ và năng lượng.
- Giá thành ngày càng rẻ.
- Khả năng hỗ trợ của hãng sx, nhiều công cụ phát triển.

62

Ưu điểm của ARM

- Bên cạnh lập trình bằng ASM, cấu trúc ARM được thiết kế tương thích C.
- Nguồn tài nguyên về source code, tài liệu, application note…rất lớn trên internet.
- So sánh khả năng xử lý lệnh nhân 2 số 16 bit của ARM với các dòng vi điều khiển 8 bit và 16 bit:

63

Ưu điểm của ARM

8-bit example		16-bit example	ARM Cortex-M

MOV A, XL ; 2 bytes
MOV B, YL ; 3 bytes
MUL AB; 1 byte
MOV R0, A; 1 byte
MOV R1, B; 3 bytes
MOV A, XL ; 2 bytes
MOV B, YH ; 3 bytes
MUL AB; 1 byte
ADD A, R1; 1 byte
MOV R1, A; 1 byte
MOV A, B ; 2 bytes
ADDC A, #0 ; 2 bytes
MOV R2, A; 1 byte
MOV A, XH ; 2 bytes
MOV B, YL ; 3 bytes
	MUL AB; 1 byte
	ADD A, R1; 1 byte
	MOV R1, A; 1 byte
	MOV A, B ; 2 bytes
	ADDC A, R2 ; 1 bytes
	MOV R2, A; 1 byte
	MOV A, XH ; 2 bytes
	MOV B, YH ; 3 bytes
	MUL AB; 1 byte
	ADD A, R2; 1 byte
	MOV R2, A; 1 byte
	MOV A, B ; 2 bytes
	ADDC A, #0 ; 2 bytes
	MOV R3, A; 1 byte
		MOV R4,&0130h
		MOV R5,&0138h
		MOV SumLo,R6
		MOV SumHi,R7
		(Operands are moved to and from a memory mapped hardware multiply unit)
			MULS r0,r1,r0

64

Giới thiệu STM32

- Tập đoàn ST Microelectronic cho ra mắt dòng STM32, vi điều khiển đầu tiên dựa trên nền lõi ARM Cortex-M3 thế hệ mới do hãng ARM thiết kế, lõi ARM Cortex-M3 là sự cải tiến của lõi ARM7 truyền thống.
-

65

Giới thiệu STM32

- Dòng ARM Cortex™-M là thế hệ mới, thiết lập các tiêu chuẩn mới về hiệu suất, chi phí, ứng dụng cho các thiết bị cần tiêu thụ năng lượng thấp, và đáp ứng yêu cầu thời gian thực khắc khe.

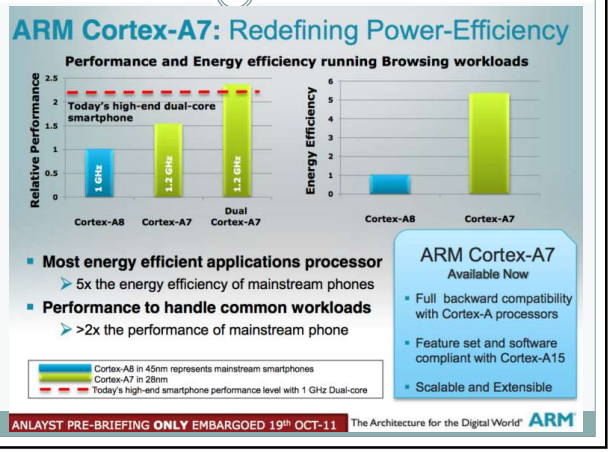
66

Ứng dụng STM32

67

Tính năng nổi bật

Tiêu thụ năng lượng cực thấp với hiệu suất cực cao



68

Tính năng nổi bật

- **Coding cực dễ:** Với sự đồ sộ về ngoại vi (GPIO, I2C, SPI, ADC, USB, Ethernet, CAN….), ST cung cấp cho chúng ta các thư viện trực tiếp cho mỗi dòng ARM (gọi là CMSIS – Cortex Microcontroller Software Interface Standard), nhiệm vụ của chúng ta không thể dễ dàng hơn: khai báo và sử dụng mà thôi.
- **Giá tiền cực rẻ:** một chip STM32F100x giá khoảng 29K (tương đương 1 chip ATMega8) mà STM32F100x chạy tốc độ 24Mhz.

69

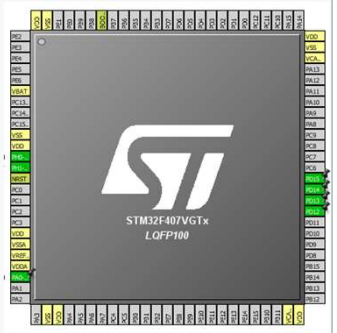
KIT STM32F4 DISCOVERY

- Ưu điểm: nhiều cảm biến, nhiều I/O và tốc độ cao. Được sử dụng rất nhiều ở Việt Nam, sự hỗ trợ từ cộng đồng VN cũng khá nhiều.

70

KIT STM32F4 DISCOVERY

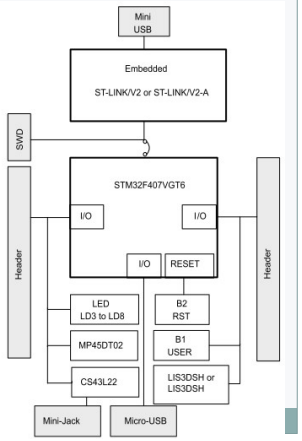
- Sử dụng MCU STM32F407VGT6 32-bit lõi ARM Cortex-M4F.
- 1 MB Flash, 192 KB RAM và có 100 chân
- Tích hợp sẵn mạch nạp ST-LINK/V2 trên board
- Nguồn sử dụng từ USB hoặc nguồn ngoài 5V
- 8 LED: 2 led báo nguồn và kết nối USB, 4 LED có thể lập trình, 2 LED cho USB OTG
- 2 Nút nhấn, 1 nút RESET và 1 nút người dùng.



71

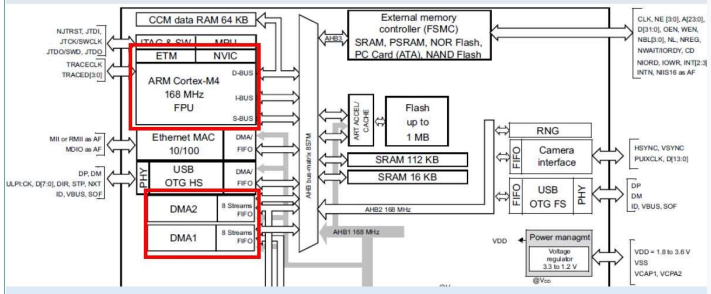
KIT STM32F4 DISCOVERY

- Hỗ trợ USB OTG
- Có các header mở rộng để kết nối với các cảm biến khác
- Cảm biến chuyển động và gia tốc 3 trục LIS302DL, ST MEMS
- Cảm biến cho các ứng dụng audio MP45DT02, ST MEMS và xuất âm thanh với CS43L22



72

KIT STM32F4 DISCOVERY



73

KIT STM32F4 DISCOVERY

Product	F_{CPU} (MHz)	Flash (bytes)	RAM (KB)	Ethernet I/F IEEE 1588	Camera I/F	SDRAM I/F	SAI I/F	Chrom-ART Graphic Accelerator™	TFT LCD controller	MIPI DSI
				2x CAN		Dual Quad SPI	SPDIF RX			
STM32F469[3]	180	512 K to 2 M	384	• •	•	• •	•	•	•	•
STM32F429[3]	180	512 K to 2 M	256	• •	•	•	•	•	•	
STM32F427[4]	180	1 to 2 M	256	• •	•	•	•	•		
STM32F446	180	256 K to 512 K	128	•	•	• •	• •			
STM32F407[3]	168	512 K to 1 M	192	• •	•					
STM32F405[3]	168	512 K to 1 M	192							

Product	F_{CPU} (MHz)	Flash (KB)	RAM (KB)	Dynamic Efficiency™	RUN current (µA/MHz)	STOP current (µA)	Small package (mm)	DMA Batch Acquisition Mode
STM32F411	100	256 to 512	128	•	Down to 100	Down to 12	Down to 3.034x3.22	•
STM32F401	84	128 to 512	96	•	Down to 128	Down to 10	Down to 3x3	

74

CÀI ĐẶT PHẦN MỀM

1. PHẦN MỀM KHỞI TẠO CODE STM32CUBE MX
2. PHẦN MỀM LẬP TRÌNH ARM KEIL UVISION5
3. LÀM BÀI TẬP

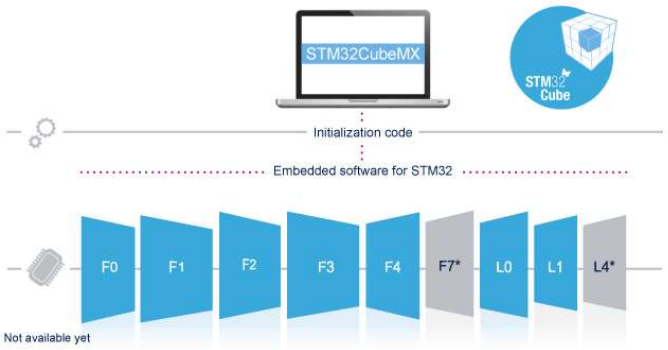
75

PHẦN MỀM KHỞI TẠO CODE STM32CubeMX

- STM32CubeMX là công cụ giúp khởi tạo phần cứng, ngoại vi, xung nhịp… cho vi điều khiển STM32.
- ***Ưu điểm của STM32CubeMX***:
  - **Giúp cấu hình ngoại vi dễ dàng. Chọn các pin trên chip và chọn các tính năng mong muốn gắn với nó.**
  - **Cấu hình Middlewares (FATS, FREERTOS), các ngoại vi như CRC, IWDG, TIMERS…**
  - **Cấu hình Clock**
  - **Tính toán mức độ tiêu hao năng lượng.**

76

PHẦN MỀM KHỞI TẠO CODE STM32CubeMX



77

PHẦN MỀM KHỞI TẠO CODE STM32CubeMX

Offer	Available for STM32								
STM32Snippets	Now	N.A.	N.A.	N.A.	N.A.	N.A.	Now	N.A.	N.A.
Standard Peripheral Library	Now	Now	Now	Now	Now	N.A.	N.A.	Now	N.A.
STM32Cube HAL	Now	Now	Now	Now	Now	Now	Now	Now	Now
STM32Cube LL NEW	Now	Q1 2017	Now	Q1 2017	Q1 2017	Q4 2016	Now	Now	Now

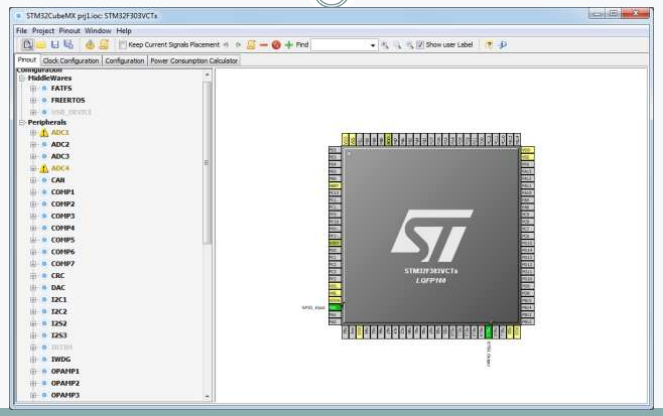
78

PHẦN MỀM KHỞI TẠO CODE STM32CubeMX

- STM32CubeMX tự động download các driver mới nhất của ST dành cho các dòng chip của mình.
- ST đã không còn phát triển Standard Peripheral Libraries nữa, thay vào đó họ phát triển cấu trúc firmware mới bao gồm lớp cách ly phần cứng (HAL) bao gồm:
- Các driver cho ngoại vi, lớp Middleware bao gồm hỗ trợ TCP/IP, USB, Graphics, FAT file system, Touch library, và hệ điều hành mã nguồn mở RTOS.
- Cấu trúc firmware mới này có mức độ trừu tượng cao hơn, tập trung vào các tính năng phần cứng chung thay vì tập trung thuần túy vào phần cứng. Mức độ trừu tượng cao hơn giúp phát triển các API thân thiện và có thể dễ dàng chuyển từ phần cứng này sang phần cứng khác.

79

PHẦN MỀM KHỞI TẠO CODE STM32CubeMX



80

Cấu trúc Firmware của STM32

– ***Level 0:***

- Board Support Package (BSP): cung cấp các API liên quan đến các thành phần phần cứng trên các board(ví dụ driver LCD, MicroSD). Khi sử dụng các board của ST ví dụ như STM32F3 Discovery, các API này giúp chúng ta nhanh chóng cấu hình/sử dụng các phần cứng có sẵn trên đó ví dụ LED, Buttons, Gyroscope…
- Hardware Abstraction Layer (HAL): Cung cấp các driver ở mức thấp và các phương thức giao diện phần cứng để giao tiếp với các mức trên (application, libraries và stacks). HAL API chia làm 2 nhóm: nhóm thứ nhất cung cấp các API chung đối với tất cả các serie STM32, và nhóm API mở rộng riêng cho từng dòng chip.

81

Cấu trúc Firmware của STM32

– ***Level 1:***

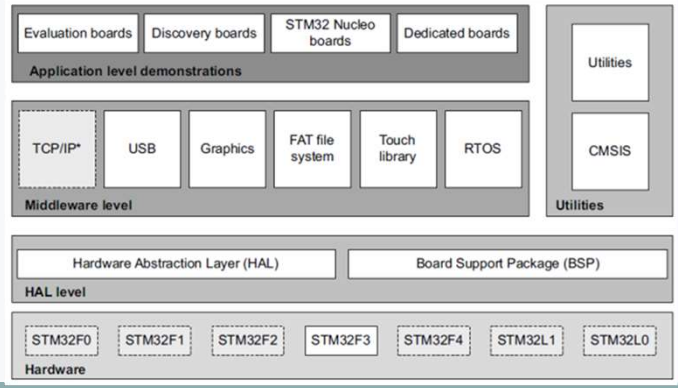
- Các thành phần Middlewares: Bao gồm các thư viện USB Device, STMTouch, thư viện đồ họa STemWin, hệ điều hành FreeRTOS, FatFS.

– ***Level 2:***

- Dựa trên lớp dịch vụ Middleware, lớp trừu tượng mức độ thấp và sử dụng các ngoại vi.

82

Cấu trúc Firmware của STM32



83

Tạo Project với STM32CubeMX

- Mở phần mềm STM32CubeMX lên, nhấn vào ***New Project*** để bắt đầu tạo project mới.
- Cửa sổ hiện ra với các thiết lập:
  - ***Series***: Chọn họ MCU bạn sử dụng.
  - ***Lines***: Chọn dòng MCU bạn sử dụng.
  - ***Package***: Chọn kiểu đóng gói của MCU.
  - Chọn loại MCU chính xác trong phần ***MCUs List***
  - Nhấn **OK**.

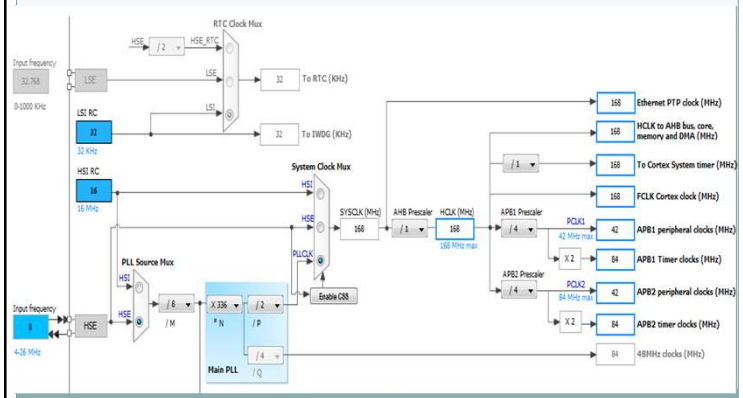
84

Cấu hình xung nhịp

- Sau khi đã lựa chọn ngoại vi cần thiết, tiến hành cấu hình xung đồng hồ cho ngoại vi tại thẻ ***Clock Configuration***.

85

Cấu hình xung nhịp



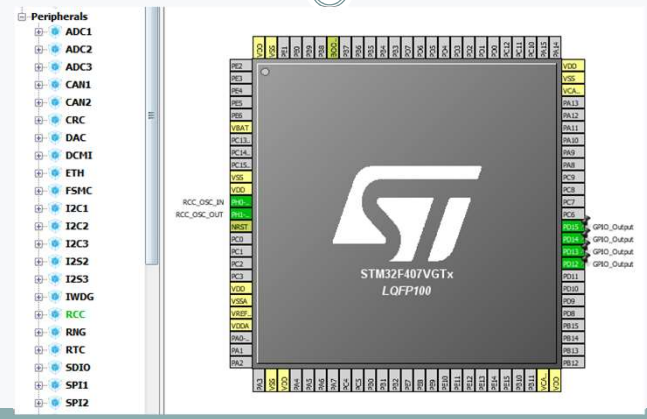
86

Chọn ngoại vi

- Sau khi đã lựa chọn xong MCU, tiến hành chọn ngoại vi cần dùng tại thẻ ***PinOut***:
- Trong danh sách ***Peripheral*** được liệt kê bên trái có cách ngoại vi mà MCU hỗ trợ, sử dụng ngoại vi nào thì Enable ngoại vi đó lên.
- Tại hình MCU trong khung bên phải, bạn có thể trực tiếp cấu hình trực quan từng chân của MCU theo các tính năng GPIO mà MCU hỗ trợ bằng cách click vào chân MCU và chọn chức năng cần thiết.

87

Chọn ngoại vi



88

Cấu hình ngoại vi

- Tiến hành cấu hình đặc tính của ngoại vi đã chọn tại thẻ ***Configuration***.
- Tại đây phần mềm có biểu diễn theo cấu trúc firmware, tùy vào loại ngoại vi đã chọn mà phần mềm xếp vào nhóm chức năng riêng.

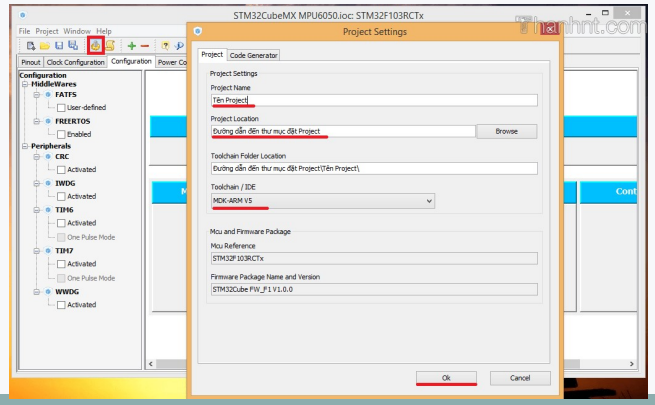
89

Xuất mã nguồn đã cấu hình

- Sau khi đã điều chỉnh, cấu hình những ngoại vi cần thiết, chúng ta tiến hành xuất mã nguồn để import vào các trình biên dịch như IAR, KeilC…
- Nhấn nút ***Generate source code*** hoặc chọn ***Project >> Generate code*** để mở cửa sổ cấu hình đường dẫn Project.
  - ***Project name***: Gõ tên Project.
  - ***Project Location***: chọn đường dẫn lưa thư mục Project.
  - ***Toolchain/IDE***: Chọn *MDK-ARM V5* hoặc *MDK-ARM V4* nếu dùng KeilC, chọn *EWARM* dùng IAR.
- Nhấn ***OK*** để phần mềm bắt đầu quá trình xuất mã nguồn. Phần mềm STM32CubeMX sẽ tự động tải về hoặc tự động cập nhật bộ API mới từ nhà sản xuất ST.

90

Xuất mã nguồn đã cấu hình



91

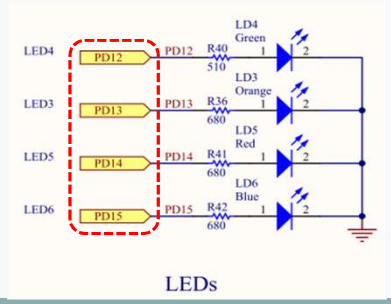
ĐIỀU KHIỂN LED ĐƠN

1. KHỞI TẠO CODE BẰNG STM32CUBEMX
2. VIẾT CODE BẰNG KEIL V5
3. BIÊN DỊCH VÀ NẠP XUỐNG KIT
4. LÀM BÀI TẬP

92

Bài tập mẫu

- Điều khiển 4 led trên KIT STM32F4discovery chớp tắt với tần số 1 Hz.

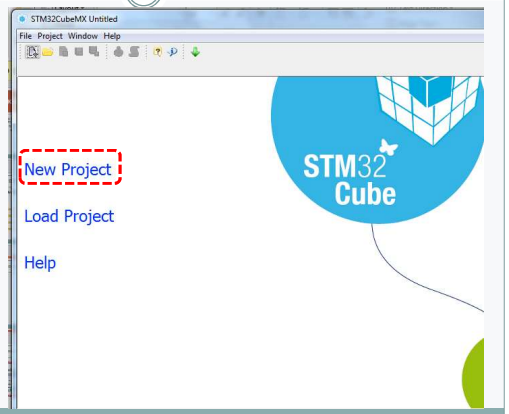


93

Khởi tạo code với CubeMX

- Mở CubeMX

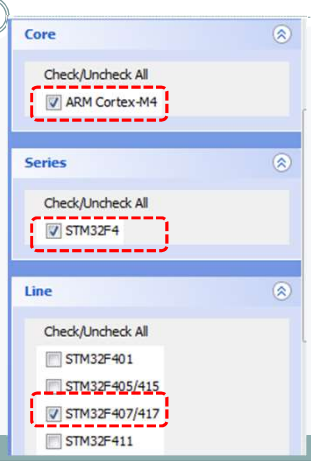
- Chọn New Project.



94

Khởi tạo code với CubeMX

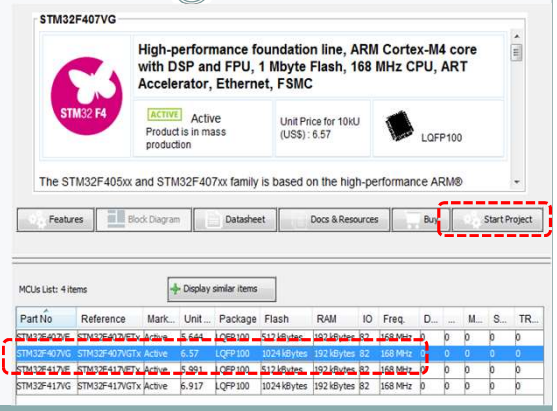
- Mục Core chọn ARM cortex-M4
- Mục Series chọn STM32F4
- Mục Line chọn STM32F407/417
- Mục Package chọn LQFP100



95

Khởi tạo code với CubeMX

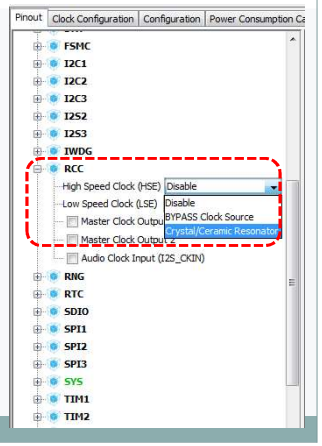
- Ở cửa sổ bên phải, chọn loại 1024KB flash.
- Bấm Start Project để bắt đầu



96

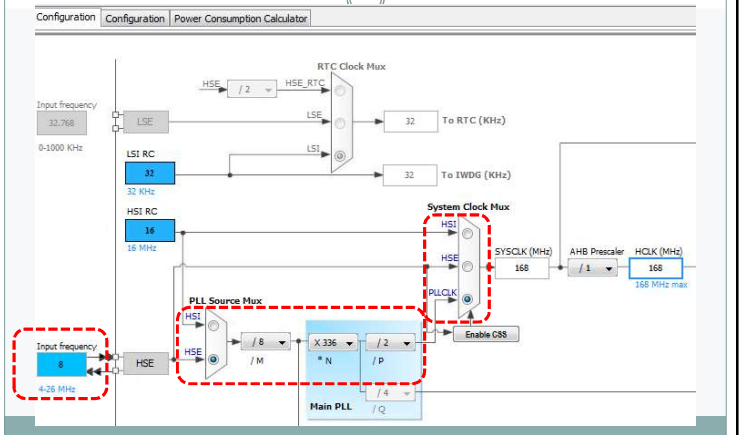
Cài đặt Clock

- Mục **RCC (Reset & Clock Control)** chọn như hình.
- Nếu dung bộ clock nội thì chọn mục **Disable**
- Qua thẻ **Clock Cofiguration** để cấu hình cho Clock



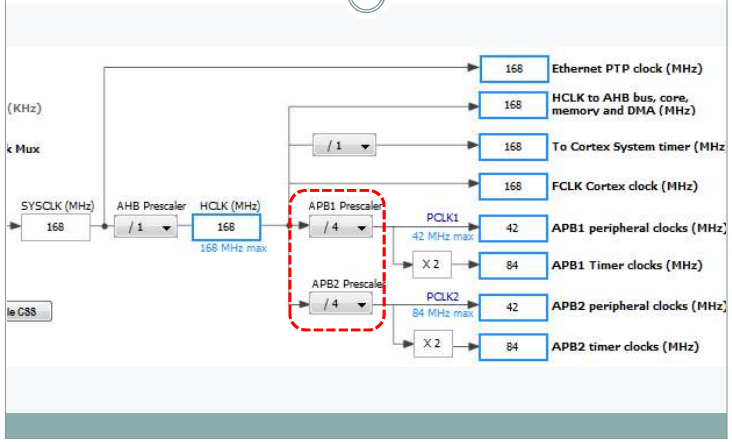
97

Clock Configuration



98

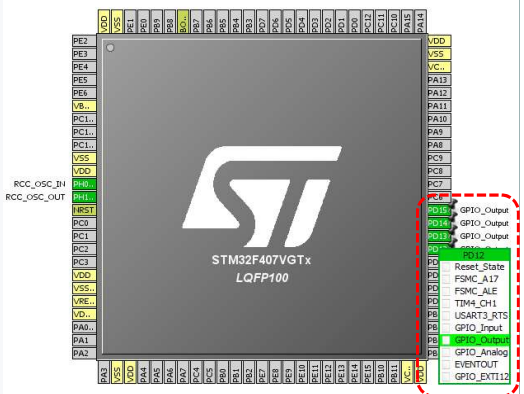
Clock Configuration



99

Cài đặt I/O

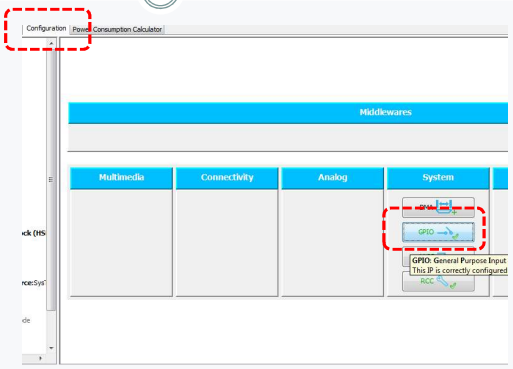
- Quay lại thẻ **Pinout** chọn 4 pin **PD12** đến **PD15** ở chế độ **GPIO_Output**. (General Purpose Input Output)



100

Cấu hình I/O

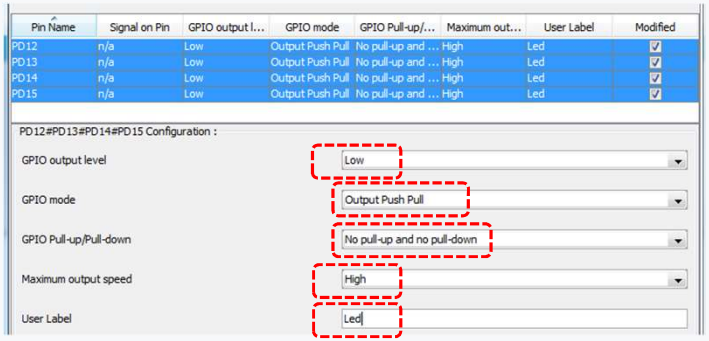
- Qua thẻ **Configuraton** chọn nút **GPIO**. (General Purpose Input Output)



101

Cấu hình I/O

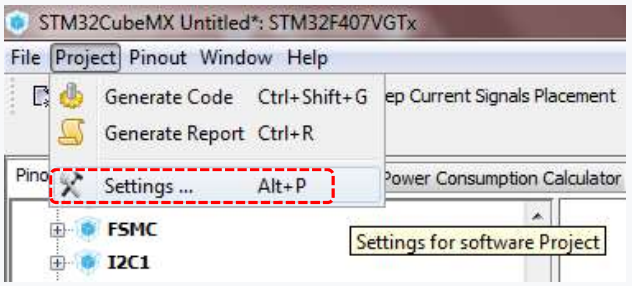
- Chọn cả 4 dòng và cấu hình như sau:



102

Cài đặt Project

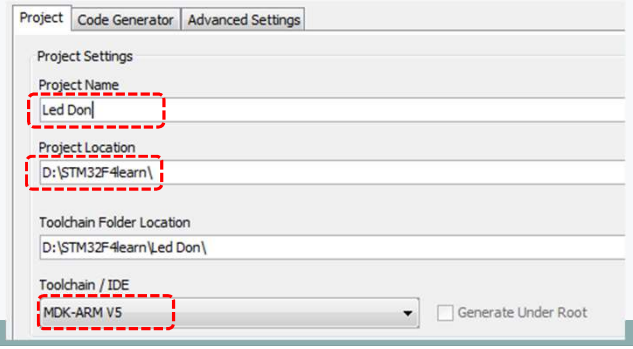
- Trước khi xuất code sang Keil V5, vào mục Project > Settings để cài đặt các yêu cầu cho trình sinh code CubeMX:



103

Cài đặt Project

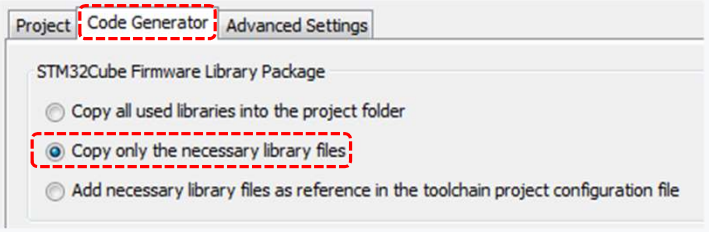
- Ở cửa sổ mới, đặt tên cho Project, thư mục lưu và chọn công cụ lập trình như hình:



104

Cài đặt Project

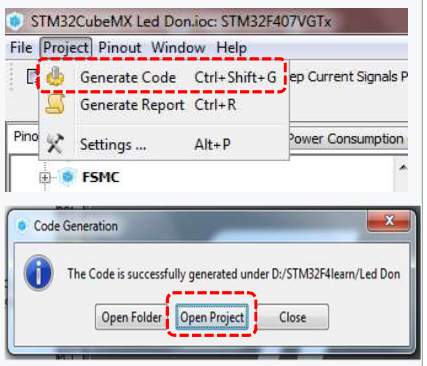
- Qua thẻ **Code Generator**, Chọn như hình:
- Sau đó nhấn OK để hoàn thành.



105

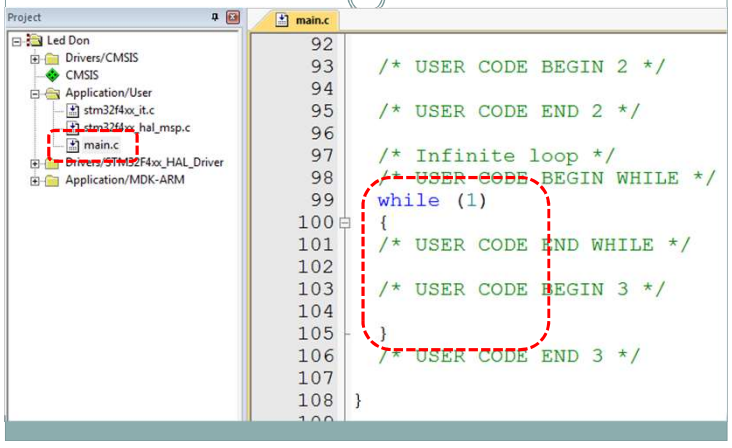
Khởi tạo code

- Vào mục **Project > Generate Code** hoặc nhấn lên nút trên thanh công cụ như hình.
- Quá trình khởi tạo code được thực hiện. Chọn **Open Project** để chuyển qua phần mềm lập trình **Keil V5**.



106

Lập trình với Keil V5



107

Lập trình với Keil V5

** Các hàm dùng trong bài*

- **Hàm đổi trạng thái output:**

HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);

- **Hàm set/ reset output:**

HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);

HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);

- **Hàm Delay ms:**

HAL_Delay(500);

108

Lập trình với Keil V5

- Code chớp tắt 4 led trên PORTD.

```
while (1)
{
  HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);
  HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_13);
  HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);
  HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
  HAL_Delay(500);
}
while (1)
{
  HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12
  |GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15);
  HAL_Delay(500);
}
```

109

Bài tập

** SV dựa vào bài tập mẫu để viết các chương trình sau:*

- **Điều khiển 16 chân của PORTD chớp tắt.**
- **Điều khiển 16 chân của PORTD sáng dần tắt dần.**
- **Viết hàm choptat, sangdan, tatdan.**

** Kiểm tra trạng thái các chân qua chức năng Debug.*

110

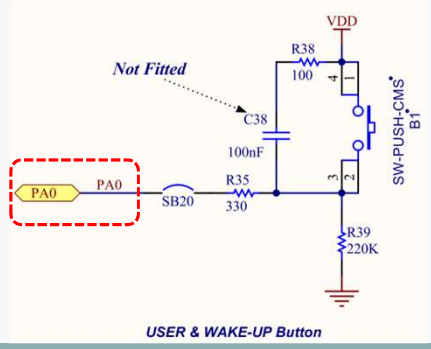
NGẮT NGOÀI

1. KHỞI TẠO CODE: ĐẾM SỰ KIỆN NHẤN NÚT BẰNG NGẮT NGOÀI
2. VIẾT CODE BẰNG KEIL V5
3. BIÊN DỊCH VÀ DEBUG
4. LÀM BÀI TẬP

111

Bài tập mẫu

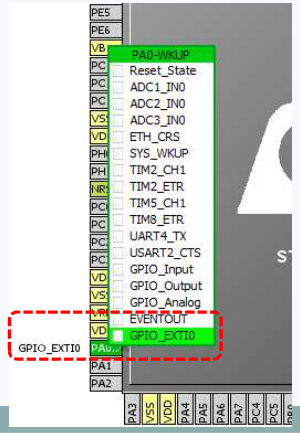
- Đếm số lần nhấn nút B1 trên board dùng ngắt ngoài.



112

Khởi tạo Code

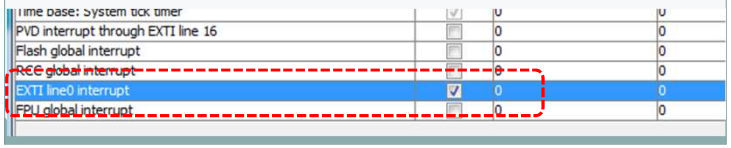
- Mở CubeMX.
- Tạo project mới.
- Cấu hình clock.
- Chọn chân PA0 với chức năng GPIO_EXTI0.
- Chọn chân PD12-15 với chức năng GPIO_Output.



113

Khởi tạo Code

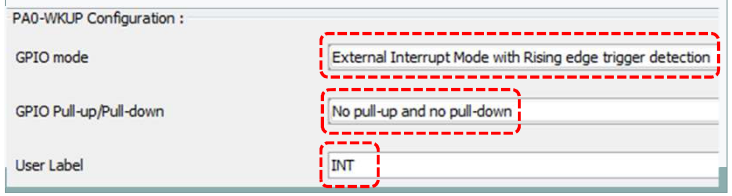
- Qua thẻ Configuration, chọn nút NVIC (Nested Vectored Interrupt Controller).
- Check vào mục EXTI line0 interrupt để cho phép ngắt ngoài.



114

Khởi tạo Code

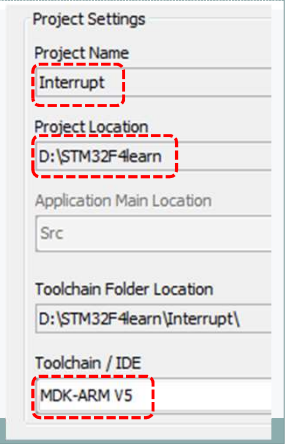
- Chọn nút GPIO, chọn PA0.
- Chọn chức năng như hình.



115

Khởi tạo Code

- Lưu Project với tên Interrupt.
- Chuyển qua phần mềm soạn thảo Keil V5



116

Viết Code bằng Keil V5

- Biên dịch chương trình.
- Khai báo biến **count** ở khu vực **Private variables** dùng để đếm số lần ngắt do ấn nút B1

```
#include "main.h"
#include "stm32f4xx_hal.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private variables ---------
  unsigned char count = 0;
/* USER CODE BEGIN PV */
/* Private variables ---------

/* USER CODE END PV */
```

117

Viết Code bằng Keil V5

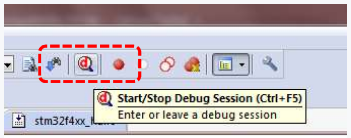
- Viết hàm ngắt ngoài ở khu vực **USER CODE BEGIN 4**.
- Hàm **HAL_GPIO_EXTI_Callback** sẽ được thực thi khi có tín hiệu ngắt ngoài.
- Biến **count** sẽ được tăng lên mỗi khi có ngắt xảy ra.

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
    if(GPIO_Pin == GPIO_PIN_0){
        count++;
    }
}
/* USER CODE END 4 */
```

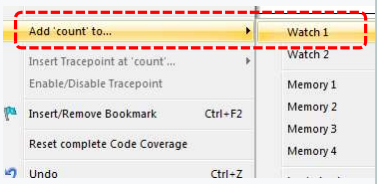
118

Chạy chương trình ở chế độ Debug

- Sau khi Biên dịch chương trình và không phát sinh lỗi, vào chế độ debug để giám sát biến **count**.



- Click phải vào biến **count** chọn như hình.



119

Bài tập

** SV dựa vào bài tập mẫu để viết các chương trình sau:*

- **Dùng nút nhấn ở chế độ ngắt để chuyển chế độ sáng của 4 đèn led.**
- **Viết các hàm choptat, xence, sangdan, tatdan cho 4 led.**

** Kiểm tra trạng thái các chân qua chức năng Debug.*

120

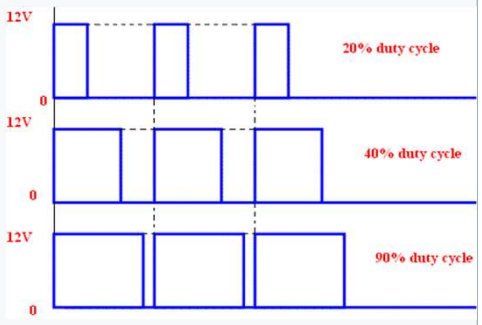
PWM
Pulse Width Modulation

1. PWM LÀ GÌ
2. BÀI TẬP MẪU
3. KHỞI TẠO CODE
4. VIẾT CODE BẰNG KEIL V5
5. BIÊN DỊCH VÀ DEBUG
6. BÀI TẬP NÂNG CAO

121

PWM là gì ?

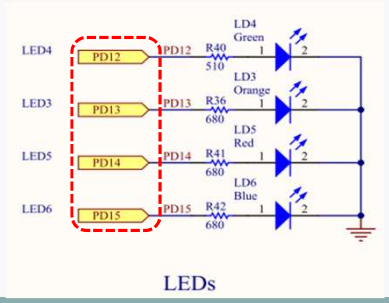
- PWM là kỹ thuật thay đổi độ rộng của xung để thay đổi điện áp trung bình ở ngõ ra.
- PWM thường dùng để điều khiển độ sáng của đèn, điều khiển tốc độ động cơ DC.



122

Bài tập mẫu

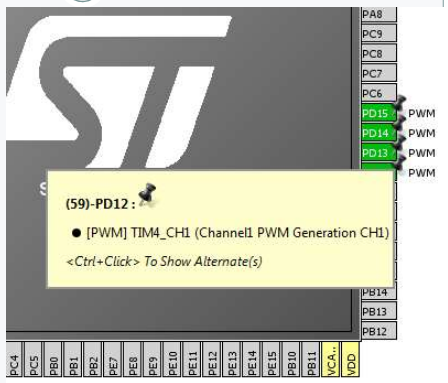
- Điều khiển độ sáng của 4 led trên PORT D dùng PWM có tần số 10KHz với độ phân giải là 400.



123

Khởi tạo Code

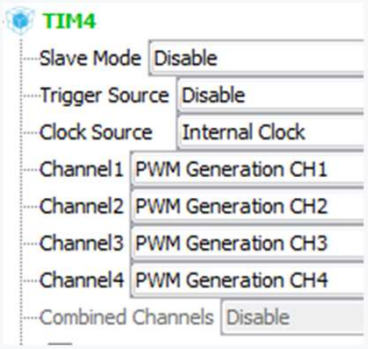
- Mở **CubeMX**.
- Tạo project mới.
- Cấu hình clock.
- Chọn chân **PD12-15** với chức năng **TIM4_CH1 – CH4**.



124

Khởi tạo Code

- Chọn các chức năng của timer 4 như hình.



125

Khởi tạo Code

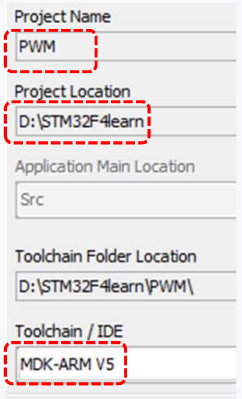
- Qua thẻ **Configuration** để cầu hình cho timer4 như sau:
- **Step = 400**
- **Ftimer = 84MHz**
- **Fpwm = 10KHz**
- **Prescale = Ftimer/(Step*Fpwm) = 21**

Counter Settings	
Prescaler (PSC - 16 bits value)	21
Counter Mode	Up
Counter Period (AutoReload Re...	400
Internal Clock Division (CKD)	No Division
Trigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigge
Trigger Event Selection	Reset (UG bit
PWM Generation Channel 1	
Mode	PWM mode 1
Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High

126

Khởi tạo Code

- **Lưu Project với tên PWM.**
- Chuyển qua phần mềm soạn thảo **Keil V5**



127

Viết Code bằng Keil V5

- Biên dịch chương trình.
- Khai báo biến **duty** và **step** ở khu vực **Private variables** dùng để thay đổi độ rộng của xung

```
/* Private variables ---------
TIM_HandleTypeDef htim4;

/* USER CODE BEGIN PV */
/* Private variables ---------
  short int duty = 0, step = 4;
/* USER CODE END PV */
```

128

Viết Code bằng Keil V5

- Khởi tạo timer 4 chế độ PWM ở khu vực **USER CODE BEGIN 2**.
- Chú ý khởi tạo cho cả 4 kênh.

```
/* USER CODE BEGIN 2 */
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_3);
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);
/* USER CODE END 2 */
```

129

Viết Code bằng Keil V5

- Viết code ở khu vực **USER CODE BEGIN 3**.
- Macro **__HAL_TIM_SET_COMPARE** dùng để set giá trị so sánh cho timer.

```
while (1){
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
    __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_1,duty);
    __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_2,duty);
    __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_3,duty);
    __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_4,duty);
    HAL_Delay(50);
    duty += step;
    if((duty==400)||(duty==0)) step = -step;
}
/* USER CODE END 3 */
```

130

Viết Code bằng Keil V5

- Chạy ở chế độ **Debug** để quan sát 2 biến **duty** và **step**.
- Quan sát độ sáng của led.

Watch 1

Name	Value	Type
duty	0	short
step	4	short
<Enter expression>		

131

Bài tập

1. Viết chương trình tạo hiệu ứng đèn sáng xoay trên 4 led.
2. Kết hợp với nút nhấn để đảo chiều xoay.

132

ADC
Analog to Digital Converter

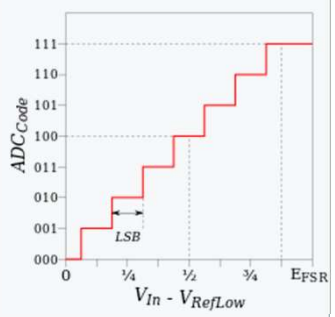
1. ADC LÀ GÌ
2. BÀI TẬP MẪU
3. KHỞI TẠO CODE
4. VIẾT CODE BẰNG KEIL V5
5. BIÊN DỊCH VÀ DEBUG
6. BÀI TẬP NÂNG CAO

133

ADC là gì ?

- ADC là kỹ thuật chuyển đổi một tín hiệu tương tự dạng điện áp thành dữ liệu số.
- Giá trị ADC nhận được tỷ lệ với giá trị tín hiệu vào Vin so với điện áp tham chiếu Vref và độ phân giải của ADC.

ADCvalue = (Vin/Vref)*MaxValue



134

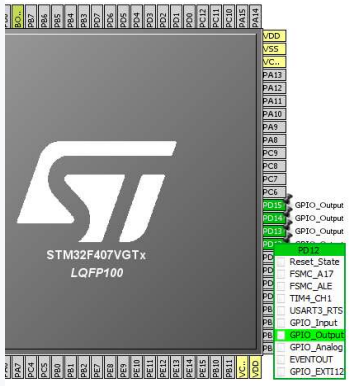
Bài tập mẫu

- Đo nhiệt độ và điện áp tham chiếu bên trong chip STM32 dùng ADC và DMA.

135

Khởi tạo Code

- Mở **CubeMX**.
- Tạo project mới.
- Cấu hình clock.
- Cấu hình output trên PD12 đến PD15.



136

Khởi tạo Code

- Chọn **ADC1**.
- Check các mục:
  - IN0
  - Temperature Sensor
  - Vrefint Channel
- IN0 được nối đến nút nhấn B1

137

Khởi tạo Code

- Qua thẻ **Configuration**.
- Chọn ADC1.
- Cấu hình như hình bên.

ADCs_Common_Settings	
Mode	Independent mode
ADC_Settings	
Clock Prescaler	PCLK2 divided by 2
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Enabled
Continuous Conversion Mode	Enabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Enabled
End Of Conversion Selection	EOC flag at the end of single ch
ADC_Regular_ConversionMode	
Number Of Conversion	3
External Trigger Conversion Source	Regular Conversion launched by
External Trigger Conversion Edge	None

138

Khởi tạo Code

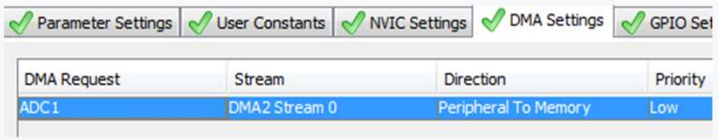
- Cấu hình **Rank**.
- Mỗi Rank ứng với 1 kênh đã chọn.
- **Sampling Time**: thời gian lấy mẫu càng lớn thì kết quả càng chính xác.

Rank	1
Channel	Channel 0
Sampling Time	480 Cycles
Rank	2
Channel	Channel Temperature Sensor
Sampling Time	480 Cycles
Rank	3
Channel	Channel Vrefint
Sampling Time	480 Cycles

139

Khởi tạo Code

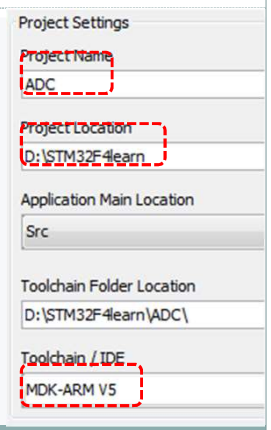
- Qua thẻ **DMA**.
- Add một DMA request là ADC1.
- **Mode: Circular**.



140

Khởi tạo Code

- **Lưu Project** với tên **ADC**.
- Chuyển qua phần mềm soạn thảo **Keil V5**



141

Viết Code bằng Keil V5

- Biên dịch chương trình.
- Khai báo mảng **adcData**, biến **vin0, temp** và **vref** ở khu vực **Private variables** dùng để lưu giá trị của ADC trước và sau khi quy đổi.

```
/* Private variables ---------
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;
/* USER CODE BEGIN PV */
/* Private variables ---------
short int adcData[3];
float vin0,temp,vref;
/* USER CODE END PV */
```

142

Viết Code bằng Keil V5

- Khởi tạo ADC1 chế độ DMA ở khu vực **USER CODE BEGIN 2.**

```
/* USER CODE BEGIN 2 */
HAL_ADC_Start_DMA(&hadc1,(unsigned int*)adcData,3);
/* USER CODE END 2 */
```

143

Viết Code bằng Keil V5

- Viết code ở khu vực **USER CODE BEGIN 3.**
- Tính giá trị của vin0, vref và temp.

```
while (1){
/* USER CODE END WHILE */
  vin0 = ((float)adcData[0]/4095)*3;
  vref = ((float)adcData[2]/4095)*3;
  temp = ((float)adcData[1]-760)/25 + 25;
  HAL_Delay(500);
/* USER CODE BEGIN 3 */
```

144

Viết Code bằng Keil V5

- Chạy ở chế độ **Debug** để quan sát các biến **adcData, vin0, temp** và **vref**.
- Nhấn nút B1 để xem sự thay đổi của vin0.

Watch 1

Name	Value	Type
adcData	0x2000001C adcData	unsigned short[3]
[0]	9	unsigned short
[1]	1104	unsigned short
[2]	1691	unsigned short
vin0	0.00586080644	float
temp	39.2000008	float
vref	1.25201464	float

145

Bài tập kiểm tra

- Đọc nhiệt độ và ADC kênh 0.
- Nếu nhiệt độ lớn hơn 37oC thì Led1 sáng.
- Nếu ADC kênh 0 lớn hơn 1.5V thì Led2 sáng.

146

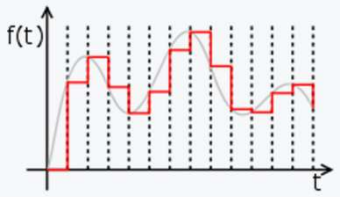
DAC
Digital to Analog Converter

1. DAC LÀ GÌ
2. BÀI TẬP MẪU
3. KHỞI TẠO CODE
4. VIẾT CODE BẰNG KEIL V5
5. BIÊN DỊCH VÀ DEBUG
6. BÀI TẬP NÂNG CAO
7. STM32F411 KHÔNG CÓ DAC

147

DAC là gì ?

- ADC là kỹ thuật chuyển đổi một tín hiệu số thành dữ liệu tương tự dạng điện áp.
- Giá trị DAC tỷ lệ với giá trị dữ liệu số so với độ phân giải của DAC và điện áp tham chiếu Vref.



Vdac = (DACvalue/ MaxValue) * Vref

148

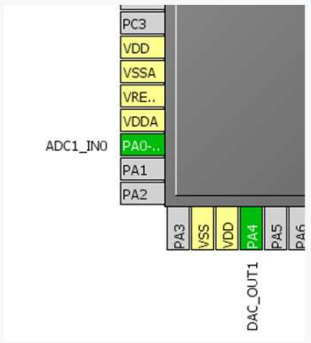
Bài tập mẫu

- Tạo điện áp ngõ ra thay đổi dùng DAC kênh 1. Đo giá trị điện áp ngõ ra bộ DAC bằng ADC1 kênh 0.

149

Khởi tạo Code

- Mở **CubeMX**.
- Tạo project mới.
- Cấu hình clock.
- Chọn bộ **ADC1** kênh **IN0**.
- Chọn bộ **DAC** kênh **OUT1**.



150

Khởi tạo Code

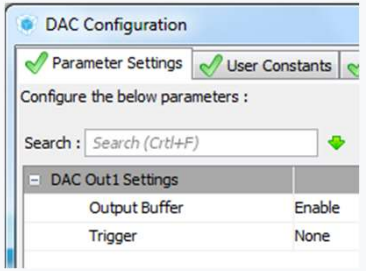
- Chọn thẻ Configuration.
- Cấu hình **ADC1** như hình.

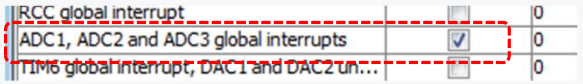
ADCs_Common_Settings	
Mode	Independent mode
ADC_Settings	
Clock Prescaler	PCLK2 divided by 2
Resolution	12 bits (15 ADC Cloc
Data Alignment	Right alignment
Scan Conversion Mode	Disabled
Continuous Conversion Mode	Enabled
Discontinuous Conversion M...	Disabled
DMA Continuous Requests	Disabled
End Of Conversion Selection	EOC flag at the end
ADC_Regular_ConversionMode	
Number Of Conversion	1
External Trigger Conversion ...	Regular Conversion l
External Trigger Conversion ...	None
Rank	1
Channel	Channel 0
Sampling Time	480 Cycles

151

Khởi tạo Code

- Cấu hình **DAC** như hình.
- Qua thẻ **NVIC** cho phép **ngắt ADC**.

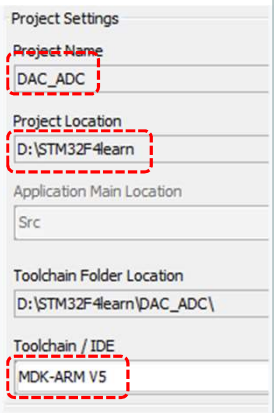




152

Khởi tạo Code

- **Lưu Project** với tên **DAC_ADC**.
- Chuyển qua phần mềm soạn thảo **Keil V5**



153

Viết Code bằng Keil V5

- Biên dịch chương trình.
- Khai báo mảng **adcValue** và **dacValue** ở khu vực **Private variables** dùng để lưu giá trị của ADC và DAC.

```
/* Includes ------------------
#include "main.h"
#include "stm32f4xx_hal.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private variables ---------
ADC_HandleTypeDef hadc1;
DAC_HandleTypeDef hdac;
/* USER CODE BEGIN PV */
/* Private variables ---------
float dacValue=0, adcValue;
/* USER CODE END PV */
```

154

Viết Code bằng Keil V5

- Ở khu vực **USER CODE BEGIN 2**:
- Khởi tạo ADC1 chế độ IT (interrupt),
- Khởi tạo DAC kênh 1.

```
  /* USER CODE BEGIN 2 */
HAL_ADC_Start_IT(&hadc1);
HAL_DAC_Start(&hdac,DAC_CHANNEL_1);
  /* USER CODE END 2 */
```

155

Viết Code bằng Keil V5

- Viết code ở khu vực **USER CODE BEGIN 3**.
- Tính giá trị của DAC trước khi xuất ra kênh 1.
- Tăng dần dacValue đến 3V thì lặp lại.

```
while (1){
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
  HAL_DAC_SetValue(&hdac,DAC_CHANNEL_1,0,
                   (int)((dacValue/3)*4095));
  HAL_Delay(2000);
  dacValue += (float)0.2;
  if(dacValue > (float)3.1) dacValue=0;
}
```

156

Viết Code bằng Keil V5

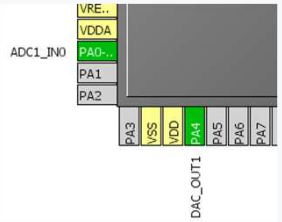
- Viết code ở khu vực **USER CODE BEGIN 4**.
- Viết hàm ngắt ADC và tính adcValue theo điện áp.

```
/* USER CODE BEGIN 4 */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc){
  if(hadc->Instance==hadc1.Instance){
    adcValue=(HAL_ADC_GetValue(&hadc1)/4095.0)*3.0;
  }
}
/* USER CODE END 4 */
```

157

Viết Code bằng Keil V5

- Nối **PA4** với **PA0**.
- Chạy ở chế độ **Debug** để quan sát các biến **adcValue** và **dacValue**.



Watch 1

Name	Value	Type
dacValue	0	float
adcValue	0	float
<Enter expression>		

158

Bài tập

- Làm lại bài tập mẫu nhưng ADC dùng DMA.
- Dùng DAC kênh 1 kết nối với dây dẫn làm đầu dò để đo điện áp các điểm trên board.

159

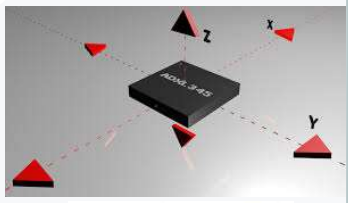
Cảm biến gia tốc Accelerometer

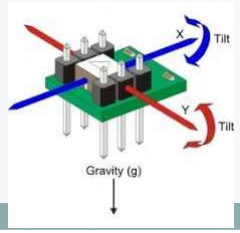
1. ACCELEROMETER LÀ GÌ
2. BÀI TẬP MẪU
3. KHỞI TẠO CODE
4. VIẾT CODE BẰNG KEIL V5
5. BIÊN DỊCH VÀ DEBUG
6. BÀI TẬP NÂNG CAO

160

Accelerometer là gì ?

- **Accelerometer** là cảm biến gia tốc được chế tạo theo công nghệ MEMS (Micro ElectroMechanic System).
- Tuỳ theo số trục của cảm biến (1, 2 hay 3 trục), giá trị đọc về của cảm biến sẽ là "hình chiếu của gia tốc trọng trường" trên từng trục toạ độ tương ứng.

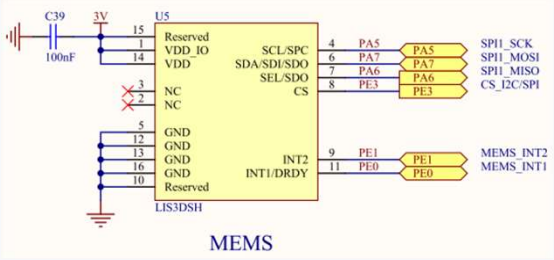




161

Sơ đồ kết nối Accelerometer

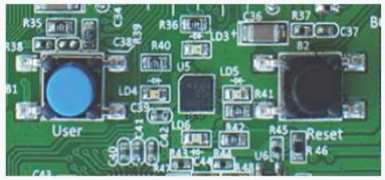
- Trên board, cảm biến **Accelerometer** là chip U5 được kết nối đến SPI1 của STM32F4.

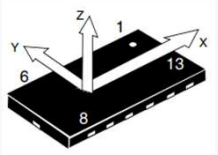


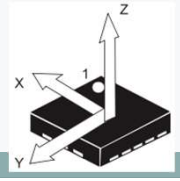
162

Bài tập mẫu

- Đọc tín hiệu theo trục XYZ từ cảm biến LIS3DSH. Bật led sáng tương ứng theo độ nghiêng của board.



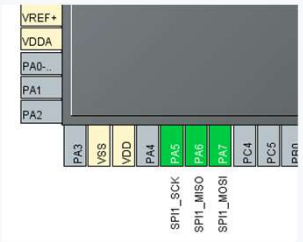




163

Khởi tạo Code

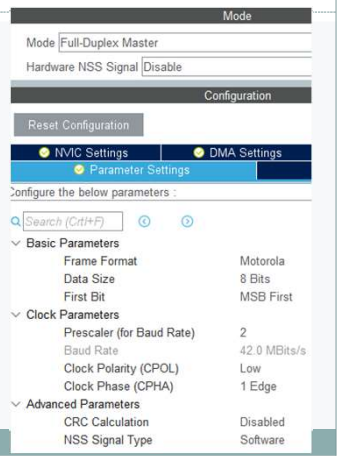
- Mở **CubeMX**.
- Tạo project mới.
- Cấu hình clock.
- Chọn bộ **SPI1** là giao tiếp giữa VXL và cảm biến.



164

Khởi tạo Code

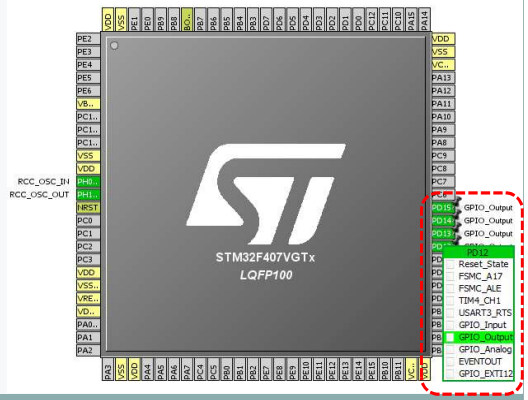
- Chọn thẻ Configuration.
- Cấu hình SPI1 như hình.



165

Khởi tạo Code

- Cấu hình 4 pin PD12 đến PD15 ở chế độ GPIO_Output. (General Purpose Input Output)



166

Viết Code bằng ARM V5

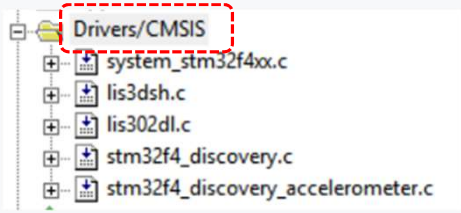
- Biên dịch chương trình.
- Khai báo mảng xyz ở khu vực Private variables dùng để lưu giá trị cảm biến của 3 trục x,y,z.

```
/* Includes ------------------------------
#include "main.h"
#include "stm32f4xx_hal.h"
#include "stm32f4_discovery_accelerometer.h"
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */
/* Private variables ---------------------
SPI_HandleTypeDef hspi1;

/* USER CODE BEGIN PV */
/* Private variables ---------------------
short xyz[3];
/* USER CODE END PV */

/* Private function prototypes -----------
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_SPI1_Init(void);
```

167

Viết Code bằng ARM V5

- Click phải vào mục Drivers/CMSIS để add thêm các thư viện như hình.



168

Viết Code bằng ARM V5

- Ở khu vực **USER CODE BEGIN 2**:
- Khởi tạo cảm biến bằng hàm: **BSP_ACCELERO_Init();**

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_SPI1_Init();
/* USER CODE BEGIN 2 */
BSP_ACCELERO_Init();
/* USER CODE END 2 */
```

169

Viết Code bằng ARM V5

- Viết code ở khu vực **USER CODE BEGIN 3**.
- Đọc giá trị cảm biến bằng hàm: **BSP_ACCELERO_GetXYZ(xyz);** và lưu vào mảng xyz.
- Chạy ở chế độ Debug để quan sát biến này thay đổi khi thay đổi vị trí của board.

```
while (1){
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
  BSP_ACCELERO_GetXYZ(xyz);
```

170

Bài tập

- Bật sáng đèn led tương ứng về phía nghiêng của board. VD nghiêng board sang trái và lên trên thì led trái và led trên sẽ sáng.

```
while (1){
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
  BSP_ACCELERO_GetXYZ(xyz);

  HAL_GPIO_WritePin(GPIOD,LED3|LED4|LED5|LED6,GPIO_PIN_RESET);
  if(xyz[0]>50)HAL_GPIO_WritePin(GPIOD,LED5,GPIO_PIN_SET);
  else if(xyz[0]< -50)HAL_GPIO_WritePin(GPIOD,LED4,GPIO_PIN_SET);
  if(xyz[1]>50)HAL_GPIO_WritePin(GPIOD,LED3,GPIO_PIN_SET);
  else if(xyz[1]< -50)HAL_GPIO_WritePin(GPIOD,LED6,GPIO_PIN_SET);
}
/* USER CODE END 3 */
```

171