Chương 2

ARM STM32F1X

2.1 CÁC DÒNG ARM CỦA HÃNG ST

Tính đến thời điểm 8/2015 thì hãng ST đã cho ra đời những dòng ARM chính sau: STM32L0, STM32L1, STM32L4, STM32F0, STM32F1, STM32F2, STM32F3, STM32F4, STM32F7.

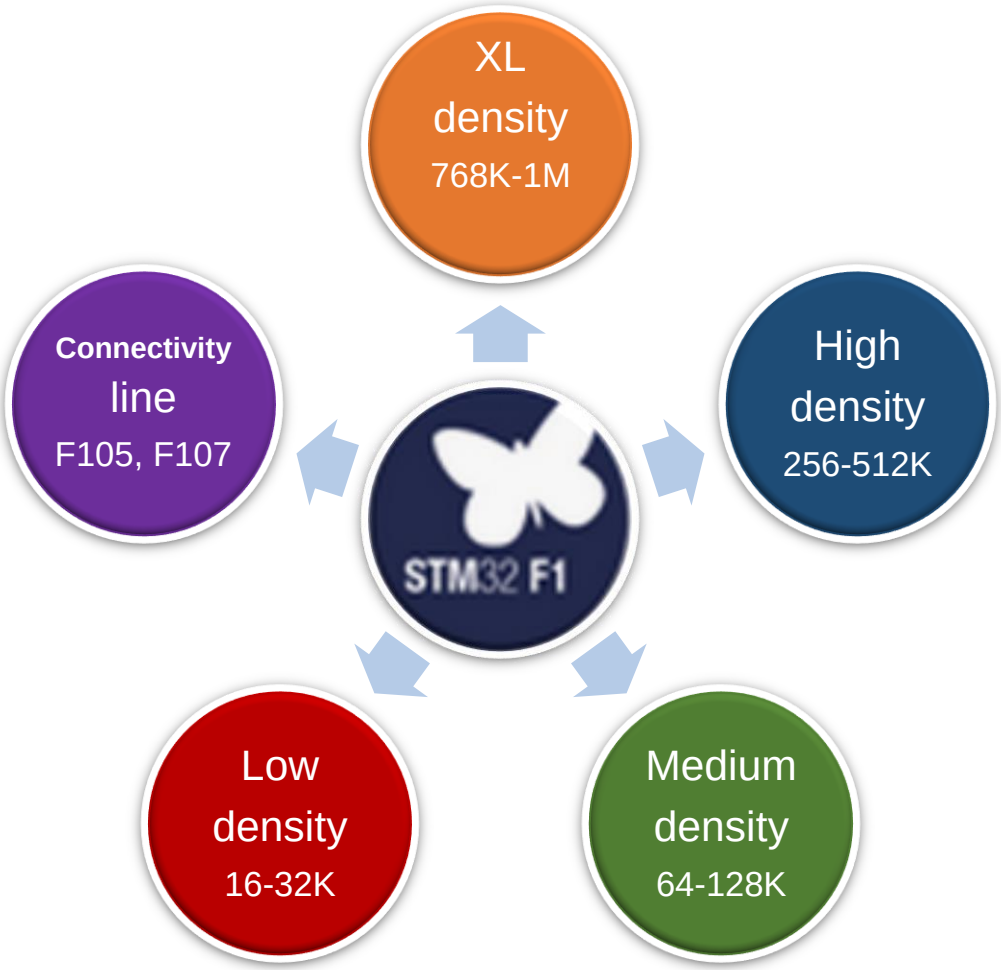
Hình 2.1 Các dòng ARM của hãng ST

2.2 GIỚI THIỆU DÒNG ARM STM32F1

Dòng ARM STM32F1 được chia ra làm 5 nhóm nhỏ, mỗi nhóm sẽ có số dung lương bộ nhớ Flash, SRAM và số lượng ngoại vi khác nhau.

- **Low-density**: Gồm các vi điều khiển STM32F101xx, STM32F102xx và STM32F103xx có bộ nhớ Flash từ 16 đến 32 Kbytes.
- **Medium-density**: Gồm các vi điều khiển STM32F101xx, STM32F102xx và STM32F103xx có bộ nhớ Flash từ 64 đến 128 Kbytes.
- **High-density**: Gồm các vi điều khiển STM32F101xx và STM32F103xx có bộ nhớ Flash từ 256 đến 512 Kbytes.

- **XL-density**: Gồm các vi điều khiển STM32F101xx và STM32F103xx có bộ nhớ Flash từ 768 đến 1 Mbyte.
- **Connectivity line**: Gồm các vi điều khiển STM32F105xx và STM32F107xx.

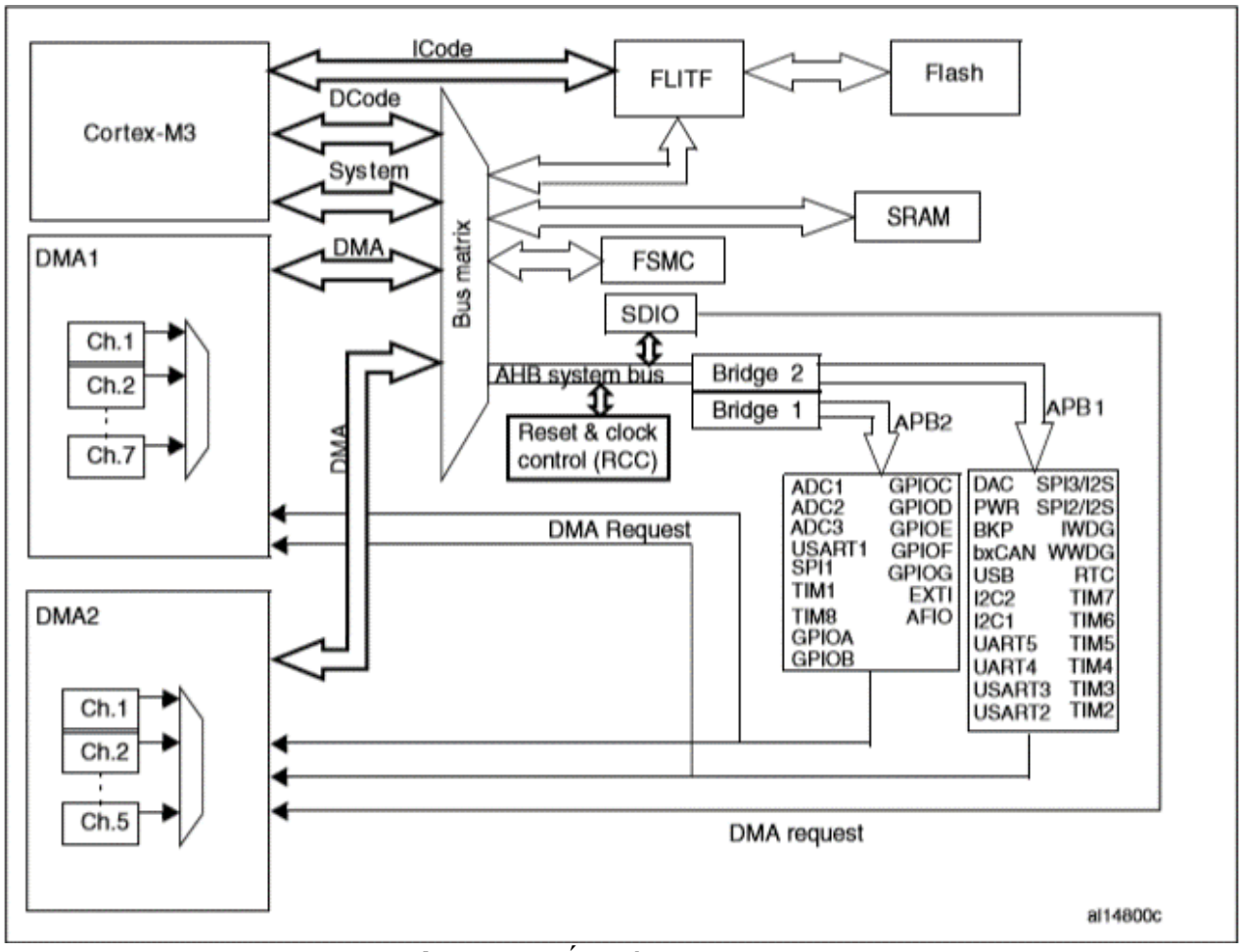


Hình 2.2 Phân loại vi điều khiển thuộc dòng STM32F1

2.3 KIẾN TRÚC CỦA ARM STM32F1

- **Icode bus:** Kết nối lõi Cortex™-M3 với bộ nhớ Flash để truyền **mã lệnh**.
- **Dcode bus:** Kết nối lõi Cortex™-M3 với bộ nhớ Flash để truyền **dữ liệu**.
- **System bus:** Kết nối lõi Cortex™-M3 với BusMatrix và BusMatrix sẽ phân quyền sử dụng bus giữa lõi ARM và khối DMA.
- **DMA bus:** Kết nối DMA với BusMatrix và BusMatrix sẽ quản lý việc truy xuất dữ liệu của CPU, DMA tới SRAM, Flash và các ngoại vi.
- **BusMatrix:** Phân quyền sử dụng bus giữa lõi ARM và khối DMA. Việc phân quyền này dựa trên thuật toán Round Robin (Các khối sẽ thay phiên nhau truy cập bus trong 1 đơn vị thời gian định sẵn).
- **Các cầu AHB/APB :** 2 cầu AHB/APB giúp đồng bộ kết nối giữa AHB với 2 bus APB. APB1 có tốc độ tối đa là 36 MHz và APB2 đạt tốc độ tối đa 72 MHz .

Sau mỗi lần CPU bị reset thì tất cả các nguồn xung clock cấp cho ngoại vi đều bị tắt hết chỉ trừ clock cấp cho SRAM và FLITF vì vậy trước khi sử dụng ngoại vi SV cần cấu hình cấp xung clock cho chúng thông qua 3 thanh ghi RCC_AHBENR, RCC_APB1ENR, RCC_APB2ENR.



Hình 2.3 Kiến trúc của ARM STM32F1

2.4 TỔ CHỨC VÙNG NHỚ CỦA ARM STM32F1

Bộ nhớ chương trình, bộ nhớ dữ liệu và các thanh ghi và các port I/O được bố trí trong không gian địa chỉ tuyến tính 4Gbyte.

2.4.1 Địa chỉ của ngoại vi (địa chỉ nền 0x4000 0000)

Phạm vi địa chỉ của tất cả các ngoại vi được thuộc dòng STM32F1 được liệt kê trong **bảng 2.1**. Bảng liệt kê này giúp ta tính toán được địa chỉ của từng thanh ghi thuộc 1 ngoại vi nào đó thông qua địa chỉ offset của nó.

Ví dụ để tìm địa chỉ thanh ghi GPIOB_ODR, đọc datasheet ta thấy địa chỉ offset của thanh ghi ODR là 0x0C

⇨ Vậy địa chỉ thanh ghi GPIOB_ODR = địa chỉ nền PORTB + địa chỉ offset của ODR

= 0x4001 0C00 + 0x0C = 0x4001 0C0C

Việc tính được địa chỉ của các thanh ghi giúp ta có thể truy xuất chúng thông qua địa chỉ đó, nhưng trên thực tế khi lập trình các địa chỉ này đều đã được định nghĩa sẵn trong thư viện "**stm32f10x.h**" SV không cần phải tính toán lại.

Bảng 2.1 Địa chỉ của các ngoại vi trong họ ARM STM32F1

Phạm vi địa chỉ	Ngoại vi	Bus
0x5000 0000 - 0x5000 03FF	USB OTG FS	
0x4003 0000 - 0x4FFF FFFF	Reserved	
0x4002 8000 - 0x4002 9FFF	Ethernet	
0x4002 3400 - 0x4002 7FFF	Reserved	
0x4002 3000 - 0x4002 33FF	CRC	
0x4002 2000 - 0x4002 23FF	Flash memory interface	

0x4002 1400 - 0x4002 1FFF	Reserved	**AHB**
0x4002 1000 - 0x4002 13FF	Reset and clock control RCC	
0x4002 0800 - 0x4002 0FFF	Reserved	
0x4002 0400 - 0x4002 07FF	DMA2	
0x4002 0000 - 0x4002 03FF	DMA1	
0x4001 8400 - 0x4001 7FFF	Reserved	
0x4001 8000 - 0x4001 83FF	SDIO	
0x4001 4000 - 0x4001 7FFF	Reserved	**APB2**
0x4001 3C00 - 0x4001 3FFF	ADC3	
0x4001 3800 - 0x4001 3BFF	USART1	
0x4001 3400 - 0x4001 37FF	TIM8 timer	
0x4001 3000 - 0x4001 33FF	SPI1	
0x4001 2C00 - 0x4001 2FFF	TIM1 timer	
0x4001 2800 - 0x4001 2BFF	ADC2	
0x4001 2400 - 0x4001 27FF	ADC1	
0x4001 2000 - 0x4001 23FF	GPIO Port G	
0x4001 1C00 - 0x4001 1FFF	GPIO Port F	
0x4001 1800 - 0x4001 1BFF	GPIO Port E	
0x4001 1400 - 0x4001 17FF	GPIO PortD	
0x4001 1000 - 0x4001 13FF	GPIO Port C	
0x4001 0C00 - 0x4001 0FFF	GPIO Port B	
0x4001 0800 - 0x4001 0BFF	GPIO Port A	
0x4001 0400 - 0x4001 07FF	EXTI	
0x4001 0000 - 0x4001 03FF	AFIO	
0x4000 7800 - 0x4000 FFFF	Reserved	**APB1**
0x4000 7400 - 0x4000 77FF	DAC	
0x4000 7000 - 0x4000 73FF	Power control PWR	
0x4000 6C00 - 0x4000 6FFF	Backup registers (BKP)	
0x4000 6800 - 0x4000 6BFF	Reserved	
0x4000 6400 - 0x4000 67FF	bxCAN1	
0x4000 6800 - 0x4000 6BFF	bxCAN2	
0x4000 6000 - 0x4000 63FF	Shared USB/CAN SRAM 512 bytes	
0x4000 5C00 - 0x4000 5FFF	USB device FS registers	
0x4000 5800 - 0x4000 5BFF	I2C2	
0x4000 5400 - 0x4000 57FF	I2C1	
0x4000 5000 - 0x4000 53FF	UART5	
0x4000 4C00 - 0x4000 4FFF	UART4	
0x4000 4800 - 0x4000 4BFF	USART3	
0x4000 4400 - 0x4000 47FF	USART2	
0x4000 4000 - 0x4000 3FFF	Reserved	
0x4000 3C00 - 0x4000 3FFF	SPI3/I2S	
0x4000 3800 - 0x4000 3BFF	SPI2/I2S	
0x4000 3400 - 0x4000 37FF	Reserved	
0x4000 3000 - 0x4000 33FF	Independent watchdog (IWDG)	
0x4000 2C00 - 0x4000 2FFF	Window watchdog (WWDG)	
0x4000 2800 - 0x4000 2BFF	RTC	
0x4000 1800 - 0x4000 27FF	Reserved	
0x4000 1400 - 0x4000 17FF	TIM7 timer	

0x4000 1000 - 0x4000 13FF	TIM6 timer	
0x4000 0C00 - 0x4000 0FFF	TIM5 timer	
0x4000 0800 - 0x4000 0BFF	TIM4 timer	
0x4000 0400 - 0x4000 07FF	TIM3 timer	
0x4000 0000 - 0x4000 03FF	TIM2 timer	

2.4.2 Địa chỉ của SRAM (địa chỉ nền 0x20000000)

Các vi điều khiển thuộc dòng STM32F1có thể có tối đa 64 Kbyte bộ nhớ RAM. Bộ nhớ này có thể được truy xuất theo byte, nửa word (16 bit) hoặc là cả word (32 bit).

2.4.3 Địa chỉ của FLASH (địa chỉ nền 0x0800 0000)

Bộ nhớ Flash có những đặc điểm sau:

- Đối với dòng XL-density: bộ nhớ Flash có kích thước là 1Mbyte và được chia ra làm 2 bank mỗi bank 512 Kbytes.
- Đối với các dòng khác thì bộ nhớ Flash có kích thước tối đa là 512 Kbytes.
- Bộ nhớ Flash được chia ra làm 2 khối và khối chính (main block) và khối thông tin (information block).
- Ngoài việc lưu chương trình Flash còn được sử dụng để lưu các thông tin người dùng như vai trò của 1 EPROM.
- Cho phép bảo vệ đọc và ghi.

Bảng 2.2 Phạm vi main block của các dòng ARM

Dòng ARM	Dung lương Flash tối đa	Phạm vi main block
Low-density	32 Kbytes	0x0800 0000 - 0x0800 7FFF
Medium-density	128 Kbytes	0x0800 0000 - 0x0801 FFFF
High-density	512 Kbytes	0x0800 0000 - 0x0807 FFFF
XL-density	1 Mbyte	0x0800 0000 - 0x080F FFFF
Connectivity line	256 Kbytes	0x0800 0000 - 0x0803 FFFF

Chú ý: Khi muốn sử dụng bộ nhớ Flash để lưu trữ dữ liệu như vai trò của một EPROM thì ta nên dựa vào **bảng 2.2** để biết được phạm vi của bộ nhớ Flash ứng với dòng ta đang sử dụng để từ đó có thể chọn vùng địa chỉ cuối bộ nhớ làm nơi lưu trữ(vì nếu chọn vùng địa chỉ đầu có thể dữ liệu sẽ đè lên chương trình).

Địa chỉ bắt đầu của bộ nhớ Flash là 0x08000000 và được chia ra làm nhiều page. Kích thước của một page phụ thuộc vào phiên bản ARM. Đối với bản low và medium density thì mỗi page là 1 Kbytes, bản high density thì mỗi page có 2 Kbytes. Mỗi khi muốn ghi dữ liệu vào Flash ta cần phải xóa hết dữ liệu trong page muốn ghi ngay cả khi chỉ muốn ghi 1 byte.

a. Các lệnh thông dụng liên quan đến FLASH

Bảng 2.3 Các lệnh cơ bản liên quan đến FLASH

SỬ DỤNG THƯ VIỆN"stm32f10x_flash"	
Lệnh	
Thông số hay dùng	**Giải thích**
FLASH_Unlock(); (Lệnh mở khóa để cho phép xóa Flash)	

FLASH_ClearFlag(A); (Lệnh xóa các cờ báo trạng thái của Flash)	
A: FLASH_FLAG_PGERR FLASH_FLAG_WRPRTERR FLASH_FLAG_EOP	**A:** Cờ cần xóa Lỗi nạp chương trình Lỗi bảo vệ ghi Xử lý xong yêu cầu
FLASH_ErasePage(A); (Lệnh xóa 1 Page)	
A: Địa chỉ Page cần xóa	**A:** Địa chỉ 32 bit
FLASH_ProgramHalfWord(A, B); (Lệnh ghi nửa word-16 bit vào Flash)	
A: Địa chỉ của nửa word cần ghi **B:** Giá trị cần ghi	**A:** Địa chỉ 32 bit **B:** Giá trị 16 bit
FLASH_ProgramWord(A,B); (Lệnh ghi 1 word-32 bit vào Flash)	
A: Địa chỉ của word cần ghi **B:** Giá trị cần ghi	**A:** Địa chỉ 32 bit **B:** Giá trị 32 bit
FLASH_Lock(); (Lệnh khóa Flash để chống xóa dữ liệu)	

b. Ví dụ về ghi và đọc dữ liệu từ bộ nhớ FLASH

Ví dụ 2.1: Viết chương trình con lưu 100 số 16 bit của mảng "**unsigned short datasave[100]**" vào page cuối bộ nhớ Flash của ARM STM32F103VET6 và đồng thời viết chương trình con đọc về các giá trị đã lưu để lưu lại vào mảng "**unsigned short dataread[100]**"

a. Tính toán địa chỉ

ARM STM32F103VET6 là dòng high density có Flash 512 Kbytes, mỗi page 2 Kbytes

⇨ Page cuối cùng là page số: (512:2) -1=255

⇨ Địa chỉ page cuối cùng = 255*2048+0x08000000 = 0x0807F800

b. **Chương trình**

```
void WriteFlash()
{    char i;
     FLASH_Unlock();
     // Mở khóa Flash cho phép xóa dữ liệu
     FLASH_ClearFlag(FLASH_FLAG_EOP
                    |FLASH_FLAG_PGERR
                    |FLASH_FLAG_WRPRTERR);
     // Xóa các cờ báo trạng thái của Flash
     FLASH_ErasePage(0x0807F800);
     // Xóa page cần ghi
     for(i =0; i<100; i++)
       FLASH_ProgramHalfWord(0x0807F800 + i*2,datasave[i]);
     // Ghi 100 số 16 bit vào Flash
     FLASH_Lock();
     // Khóa Flash để cấm xóa
}
void ReadFlash()
{
     char i;
```

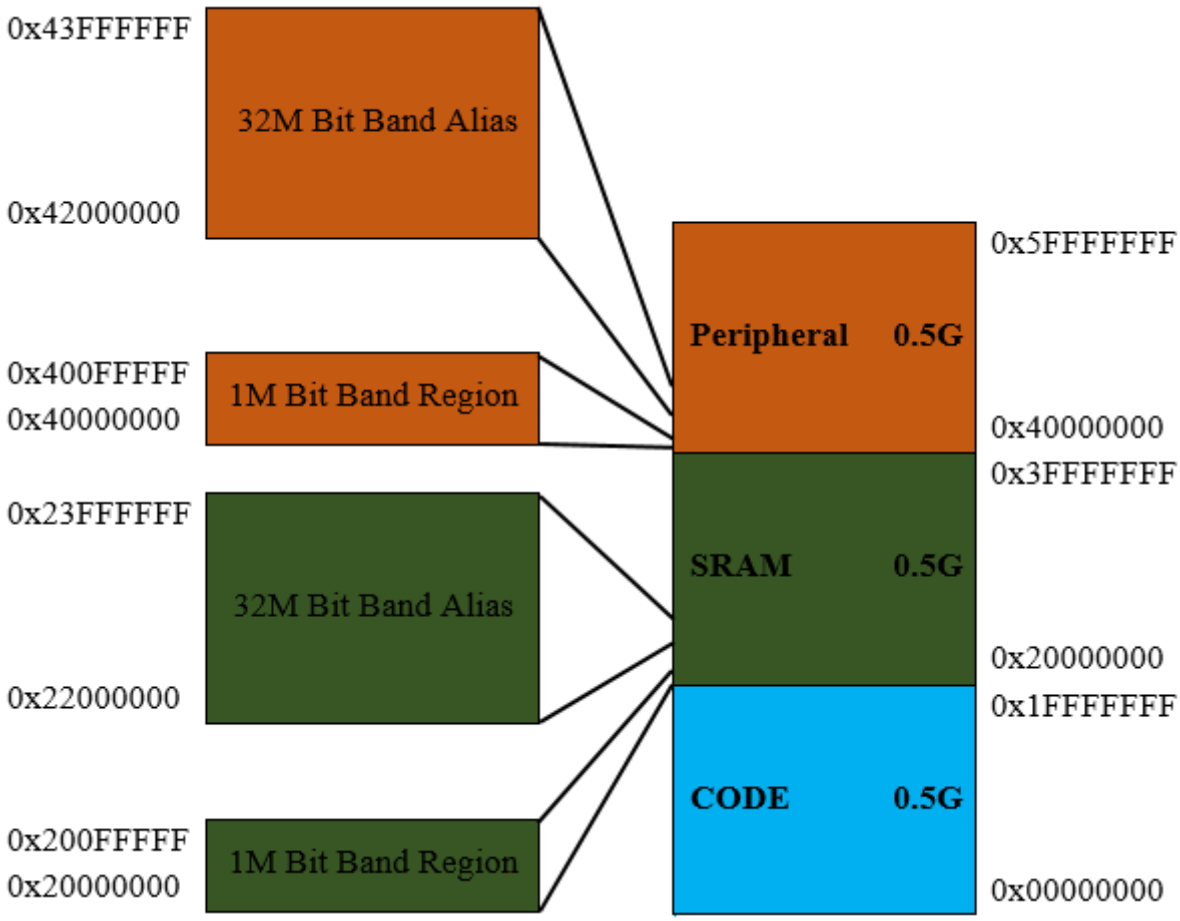
```
    for(i=0; i<100; i++)
       dataread[i] = *(unsigned int *)(0x0807F800 + i*2);
    // Đọc 100 số 16 bit từ Flash về
}
```

2.5 BIT BAND CỦA ARM STM32F1X

Các phiên bản ARM trước chỉ có thể thực hiện thao tác bit trên vùng nhớ SRAM và vùng nhớ ngoại vi bằng các lệnh AND và OR. Điều này đòi hỏi CPU phải tốn nhiều thời gian cho việc đọc → sửa đổi → ghi lại dữ liệu vào bộ nhớ.

Kỹ thuật bit band cho phép bộ xử lý ARM STM32F1 thao tác bit trực tiếp mà không cần dùng phép AND hay OR do đó giúp giảm thiểu thời gian thực hiện.

Bit band hoạt động bằng cách ánh xạ mỗi bit trong "**bit band region**" tới một địa chỉ word trong vùng "**bit band alias**" và bằng cách đặt (set) hay xóa (clear) word này ta có thể đặt hoặc xóa bit tương ứng trong "**bit band region**".



Hình 2.4 Bit band của vùng nhớ SRAM và ngoại vi

Công thức tính địa chỉ của 1 bit thuộc "**Bit band region**" được ánh xạ trong "**Bit band alias**" :

Địa chỉ trong Bit Band Alias = Bit Band Alias Base Address + Bit Word Offset
Bit Word Offset = Byte offset from Bit Band Region Base * 32 + bit number * 4

2.5.1 Ví dụ về sử dụng bit band

Yêu cầu: Sử dụng kỹ thuật Bit Band để điều khiển sáng hoặc tắt LED được kết nối với chân B8. Biết thanh ghi ngõ ra của PORTB (GPIOB_ODR) có địa chỉ offset là 0x0C và địa chỉ nền (Base Address) của PORTB là 0x40010C00.

(Các thông số trên có thể xem trong datasheet hoặc xem trong file "**stm32f10x.h**")

a. Tính toán địa chỉ

- Tìm "**Bit Band Alias Base Address**" : Do PORTB là ngoại vi nên

⇨ Bit Band Alias Base Address = 0x42000000

- Tìm "**Bit Word Offset**" :

Byte offset from Bit Band Region Base = 0x40010C0C - 0x40000000 = 0x10C0C

Với : 0x40000000 là Peripheral Bit Band Region Base Address

0x40010C0C là địa chỉ vật lý của GPIOB_ODR = 0x40010C00 + 0x0C

⇨ Bit Word Offset = 0x10C0C *32 + 8*4 = 0x2181A0

Vậy ta có :

Địa chỉ trong vùng Bit Band Alias của Bit 8 GPIOB_ODR (GPIOB_ODR_Bit8)

GPIOB_ODR_Bit8 = 0x42000000 + 0x2181A0 = 0x422181A0

b. Lập trình

Khi lập trình ta có thể tạo 1 con trỏ đến địa chỉ này bằng định nghĩa sau:

```
#define PB8    (*(unsigned long *)0x422181A0)
```

Chú ý: để khỏi tốn công tính toán như trên ta có thể định viết như sau:

```
#define      Bitband(ad,bn)   *(unsigned long*)\
((ad&0xF0000000)+0x2000000+((ad&0xFFFFF)*32)+ bn*4)
#define      PB8        Bitband((unsigned long)&GPIOB->ODR,8)
```

Lúc này ta có thể điều khiển tắt mở LED ở chân B8 một cách dễ dàng bằng 2 lệnh sau:

```
PB8 =1;    // Chân B8 lên mức 1
PB8 =0;    // Chân B8 xuống mức 0
```

2.6 CẤU HÌNH BOOT CHO ARM STM32 F1

Dòng STM32F1 có 3 chế độ boot được chọn bởi 2 chân BOOT[1:0] theo **bảng 2.4**.

Bảng 2.4 Các chế độ boot của ARM STM32F1

Trạng thái chân boot		Chế độ boot	Giải thích
BOOT1	BOOT1		
x	0	Bộ nhớ Flash chính	Chọn boot từ bộ nhớ Flash chính
0	1	Bộ nhớ hệ thống	Chọn boot từ bộ nhớ hệ thống
1	1	SRAM	Chọn boot từ bộ nhớ SRAM

Trạng thái của các chân BOOT được cập nhật vào thời điểm có cạnh lên thứ 4 của xung SYSCLK sau khi Reset. Việc chọn chế độ boot phụ thuộc vào việc cài đặt của người dùng đối với 2 chân BOOT1 và BOOT0 và trạng thái các chân boot này sẽ được cập nhật lại sau mỗi lần thoát khỏi chế độ Standby.

Boot loader được lập trình bởi hãng ST và mặc định lưu trong bộ nhớ hệ thống. Boot loader được sử dụng để nạp chương trình vào bộ nhớ Flash bằng một số chuẩn truyền nối tiếp:

- Đối với các dòng low, medium, high density thì boot loader sử dụng USART1 để nạp chương trình vào Flash
- Đối với dòng conectivity line thì boot loader sử dụng USART1, USART2 (remap), CAN2 (remap) hoặc USB OTG FS hoạt động ở chế độ DFU-Device Firmware Upgrade

Trong chế độ này USART sẽ hoạt động nhờ giao động nội 8 Mhz (HSI) còn CAN và USB chỉ có thể hoạt động nhờ giao động ngoại HSE khi kết nối với thạch anh 8 Mhz, 14.7456 MHz hoặc 25 MHz.

2.7 CẤP NGUỒN CHO ARM STM32F1

Nguồn cung cấp cho ARM (VDD) phải nằm trong phạm vi từ 2 đến 3.6 V(thường cấp 3.3V) . Một bộ điều chỉnh điện áp bên trong được sử dụng để cung cấp nguồn 1.8V cho lõi điều khiển, SRAM và ngoại vi số.

Đồng hồ thời gian thực(RTC) và các thanh ghi backup được cấp nguồn bằng Pin 3V vào chân VBAT để hệ thống vẫn định thời đúng khi tắt nguồn VDD. Nếu không dùng nguồn Pin thì chân VBAT phải được nối lên nguồn VDD và cần được lọc nhiễu bằng tụ 100 nF.

Các chân VDDA và VSSA phải được kết nối tương ứng với VDD và VSS xem **hình 2.5**

Chân V_{REF-} phải được nối với V_{SSA} nếu vi điều khiển được dùng có chân này.

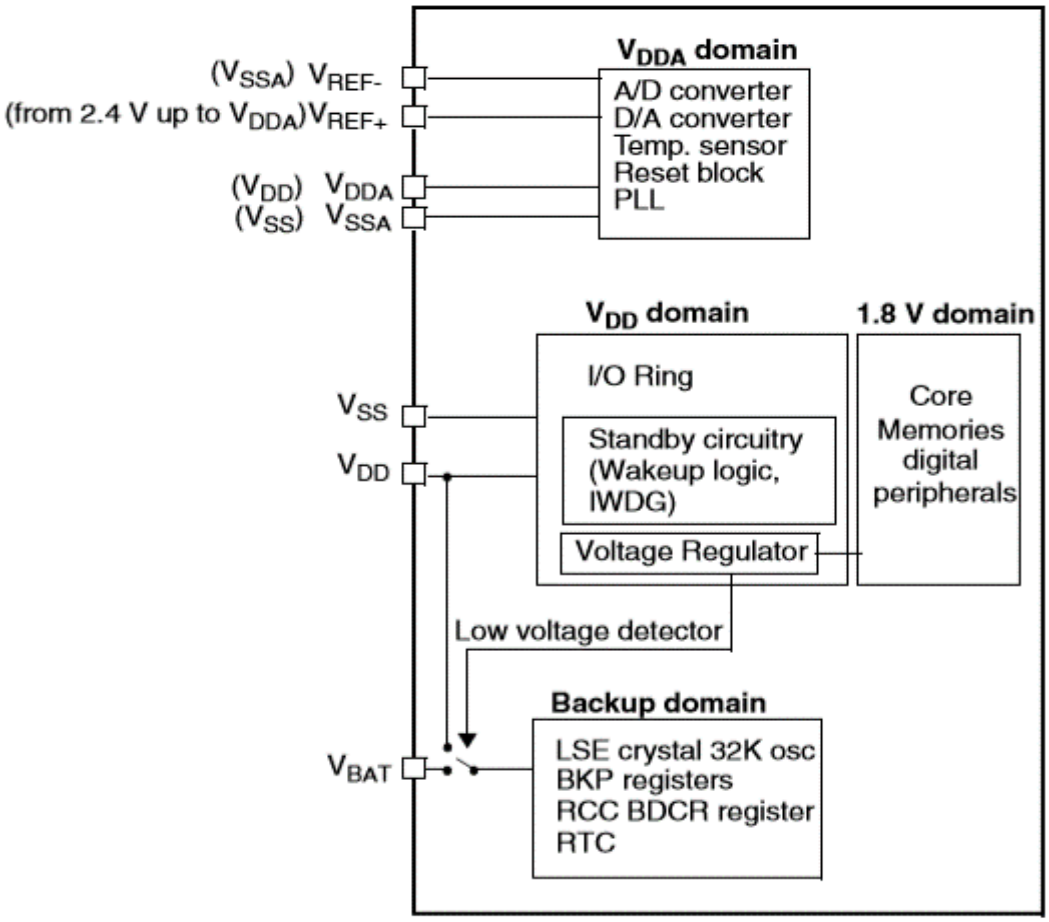
Đối với các phiên bản ARM 100 chân và 144 chân chân V_{REF+} của ADC và DAC được thiết kế riêng, để tăng độ chính xác của kết quả chuyển đổi người dùng nên nối chân này với một nguồn cung cấp độc lập với VDD. Giá trị điện áp cung cấp cho V_{REF+} nằm trong phạm vi từ 2.4V tới VDD.

Đối với các phiên bản ARM 64 chân trở xuống thì chân V_{REF+} và V_{REF-} không tồn tại vì chúng đã được kết nối bên trong với nguồn cấp cho ADC (VDDA và VSSA).

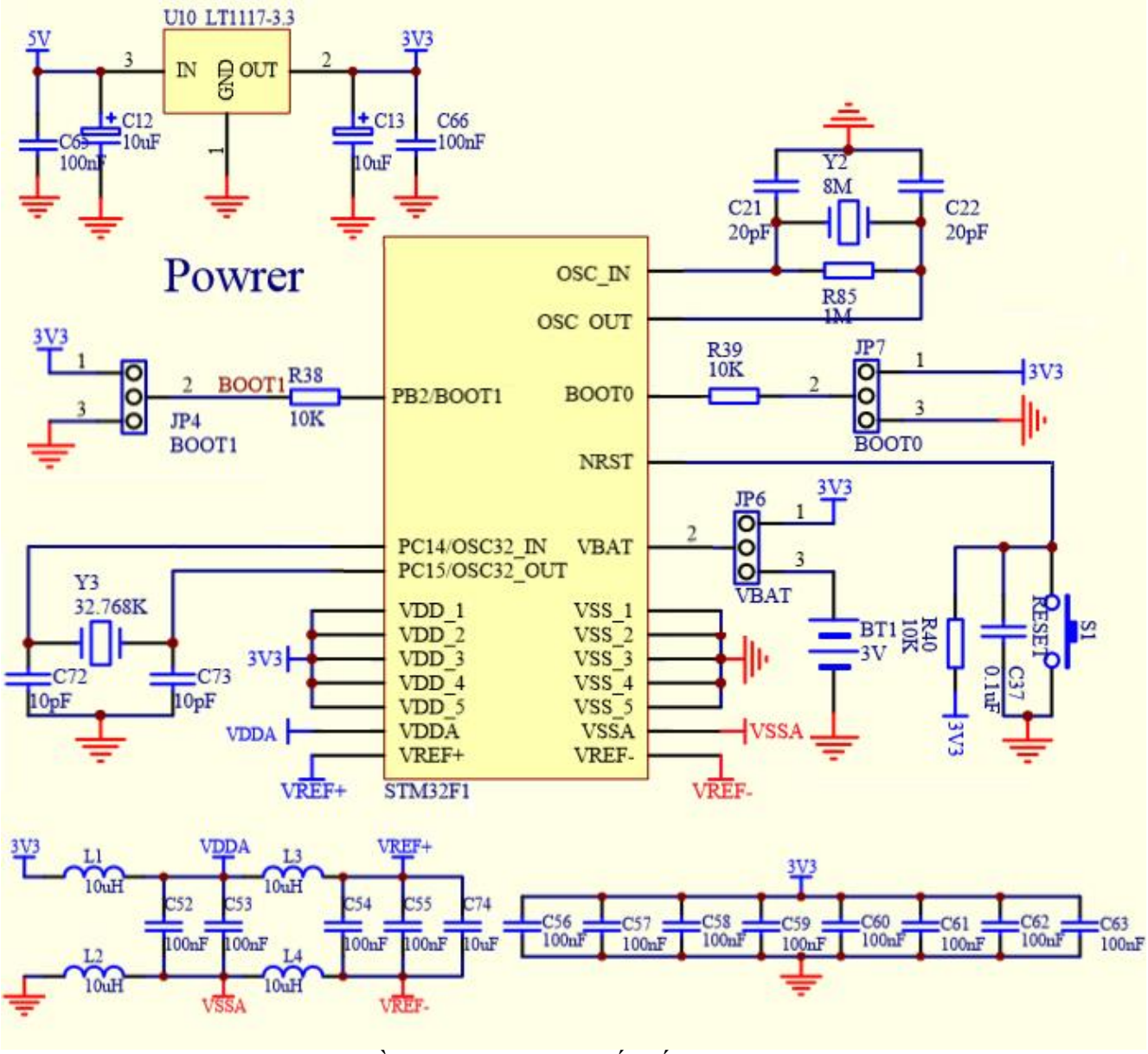
2.7.1 Khối điều chỉnh điện áp(Voltage regulator)

Khối điều chỉnh điện áp luôn được mở ngay sau khi Reset. Khối này có thể làm việc ở 3 chế độ khác nhau:

- Trong chế độ Run khối điều chỉnh điệp áp cung cấp nguồn 1.8V cho tất cả các khối sử dụng nguồn này như là lõi xử lý, bộ nhớ và các ngoại vi số.
- Trong chế độ Stop khối điều chỉnh điện áp chỉ cung cấp nguồn 1.8V cho SRAM để bảo vệ dữ liệu của các thanh ghi trong SRAM.
- Trong chế độ Standby khối điều chỉnh điện áp bị tắt đi do đó dữ liệu SRAM bị mất đi chỉ trừ các thanh ghi của Standby circuitry và Backup Domain.



Hình 2.5 Tổng quan về các lại nguồn cấp cho STM32F1x



Hình 2.6 Sơ đồ nguyên lý mạch kết nối cơ bản cho ARM STM32F1

2.8 CÁC CHẾ ĐỘ TIẾT KIỆM NĂNG LƯỢNG

Mặc định sau khi Power Reset thì vi điều khiển hoạt động ở chế độ Run. Bên cạnh chế độ Run hãng ST cũng đã thiết kế một vài chế độ tiết kiệm năng lượng nhằm giảm công suất tiêu thụ của CPU khi CPU không cần phải hoạt động, ví dụ như khi CPU đang chờ sự kiện tác động từ bên ngoài. Các chế độ tiết kiệm năng lượng này có thể được lập trình bằng phần mềm.

Dòng ARM STM32F1 có 3 chế độ tiết kiệm năng lượng sau:

- **Chế độ Sleep** (ngủ): Tắt nguồn xung clock của CPU, nhưng tất cả các ngoại vi kể cả các ngoại vi của lõi Cortex-M3 như là NVIC, SysTick,… vẫn hoạt động.
- **Chế độ Stop** (dừng) : Tắt hết tất cả các nguồn xung clock.
- **Chế độ Standby**(chờ): Tắt nguồn 1.8V, làm chậm lại SYSCLK, chặn nguồn xung clock cấp cho các ngoại vi thuộc APB và AHB khi không dùng đến.

2.8.1 Chế độ Sleep

a. Cách đưa CPU vào chế độ Sleep

Đưa CPU vào chế độ Sleep bằng cách thực thi các lệnh WFI(Wait For Interrup- Chờ xảy ra ngắt) hoặc WFE(Wait For Event- Chờ xảy ra sự kiện) – **Chương 5** sẽ trình bày rõ hơn về ngắt và sự kiện.

Tùy thuộc vào trạng thái bit SLEEPONEXIT trong thanh ghi "Cortex-M3 System Control" mà ta có các chế độ:

- **Sleep now**: Khi bit SLEEPONEXIT = 0, CPU sẽ rơi vào chế độ Sleep ngay sau khi lệnh WFI hay WFE được thực hiện xong.
- **Sleep on exit**: Khi bit SLEEPONEXIT = 1, CPU sẽ rơi vào trạng thái Sleep ngay sau khi thoát khỏi ngắt có độ ưu tiên thấp nhất.

Trong chế độ Sleep tất cả các chân I/O vẫn giữ giá trị cũ như khi ở chê độ Run.

b. Cách đưa CPU vào chế độ Sleep

Nếu trước đó sử dụng lệnh WFI để đưa CPU vào chế độ Sleep thì để thoát khỏi chế độ này cần phải xảy ra môt ngắt ngoại vi được cho phép bởi NVIC(xem chương 5).

Nếu trước đó sử dụng lệnh WFE để đưa CPU vào chế độ Sleep thì CPU sẽ thoát khỏi chế độ này ngay khi xảy ra 1 sự kiện. Sự kiện này có thể được tạo ra bằng cách:

- Cho phép ngắt ở ngoại vi nhưng không cho phép ngắt ở NVIC và cho phép bit SEVONPEND trong thanh ghi "Cortex-M3 System Control". Khi CPU thoát khỏi chế độ tiết kiệm năng lượng từ WFE thì cần phải xóa các bit báo ngắt ở ngoại vi và NVIC.
- Cấu hình kênh ngắt (EXTI line) ở chế độ sự kiện(event). Khi CPU thoát khỏi chế độ tiết kiệm năng lượng từ WFE thì không cần phải xóa các cờ ngắt.

2.8.2 Chế độ Stop

Chế độ Stop là sự kết hợp giữa chế độ DeepSleep (ngủ sâu) của lõi Cortex-M3 với chế độ kiểm soát nguồn xung clock ngoại vi. Bộ điều chỉnh nguồn có thể được cấu hình theo chế độ bình thường hoặt chế độ tiết kiệm năng lượng.

Trong chế độ này tất cả các nguồn xung clock trong vùng 1.8V đều tắt, các bộ giao động PLL, HSI, HSE cũng được tắt hết. Dữ liệu trong SRAM được bảo vệ và tất trạng thái tất cả các I/O không thay đổi.

a. Cách đưa CPU vào chế độ Stop

Để vào chế độ Stop ta dùng lệnh chờ ngắt WFI hoặc chờ sự kiện WFE kèm theo:

- Đặt bit SLEEPDEEP của thanh ghi "Cortex™-M3 System Control" lên mức '1'.
- Xóa bit PDDS của thanh thi "Power Control register (PWR_CR)" về '0' .
- Chọn chế độ hoạt động cho bộ điều chỉnh điện áp bằng cách cấu hình bit LPDS trong thanh ghi " Power control register (PWR_CR)".

Chú ý: để có thể vào được chế độ Stop thì tất cả các bit cờ báo ngắt ngoài trong thanh ghi EXTI_PR và cờ báo động của đồng hồ thời gian thực(RTC Alarm flag) phải được xóa trước đó nếu không chế độ Stop sẽ bị bỏ qua và chương trình tiếp tục hoạt động bình thường.

b. Cách đưa CPU thoát khỏi chế độ Stop

Nếu trước đó sử dụng lệnh WFI để đưa CPU vào chế độ Stop thì để thoát khỏi chế độ này cần phải xảy ra môt ngắt ngoại vi được cho phép bởi NVIC.

Nếu trước đó sử dụng lệnh WFE để đưa CPU vào chế độ Stop thì CPU sẽ thoát khỏi chế độ này ngay khi xảy ra 1 sự kiện.

2.8.3 Chế độ Standby

Chế độ Standby là chế độ tiết kiệm năng lượng nhất, Standby hoạt động dựa trên việc đưa CPU vào chế độ ngủ sâu đồng thời tắt bộ điều chỉnh điện áp 1.8V. Các khối giao động PLL, HSI, và HSE cũng bị tắt hết. Dữ liệu trong SRAM và các thanh ghi bị mất chỉ trừ các thanh ghi của Backup domain và Standby circuitry là không mất.

Trong chế độ Standby trạng thái tất cả các I/O đều ở tổng trở cao ngoại trừ :

- Chân Reset vẫn còn tác động được.
- Chân TAMPER và chân WKUP nếu được cho phép trươc đó thì vẫn hoạt động được.

a. Cách đưa CPU vào chế độ Standby

Để vào chế độ Standby ta dùng lệnh chờ ngắt WFI hoặc chờ sự kiện WFE kèm theo:

- Đặt bit SLEEPDEEP của thanh ghi "Cortex™-M3 System Control" lên mức '1'.
- Đặt bit PDDS của thanh thi "Power Control register (PWR_CR)" lên mức '1'.
- Xóa bit WUF trong thanh ghi "Power Control/Status register (PWR_CSR)".

b. Cách đưa CPU thoát khỏi chế độ Standby

CPU sẽ thoát khỏi chế độ Standby nếu một trong các điều kiện sau xảy ra:

- Có tín hiệu cạnh lên tác động vào chân WKUP
- Có tín hiệu báo động của đồng hồ thời gian thực(RTC alarm).
- Reset bằng chân NRST.
- IWDG Reset.

2.8.4 Tự động thức dậy (Auto Wakeup-AWU) từ các chế độ tiết kiệm năng lượng

Đồng hồ thời gian thực có thể được sử dụng để đánh thức CPU khỏi chế độ tiết kiệm năng lượng mà không cần phải chờ xảy ra ngắt ngoài. Người lập trình có thể cài đặt một lượng thời gian nào đó vào RTC để khi hết thời gian này thì RTC sẽ tự động đánh thức CPU khỏi chế độ Stop hoặc Standby.

Để đánh thức CPU khỏi chế độ Stop bằng sự kiện RTC alarm thì cần phải:

- Cấu hình kênh ngắt ngoài số 17 (EXTI Line 17) tích cực cạnh lên.
- Cấu hình RTC để cho phép tạo ra tín hiệu RTC alarm.

Để đánh thức CPU khỏi chế độ Standby, thì không cần cấu hình kênh ngắt ngoài số 17.

2.8.5 Các lệnh thông dụng liên quan đến chế độ tiết kiệm năng lượng

Bảng 2.5 Các lệnh thông dụng liên quan đến chế độ tiết kiệm năng lượng

SỬ DỤNG THƯ VIỆN "stm32f10x_pwr"	
Lệnh	
Thông số hay dùng	**Giải thích**
PWR_DeInit(); (Lệnh reset các thanh ghi liên quan đến chế độ tiết kiệm năng lượng về mặc định)	
PWR_WakeUpPinCmd(A); (Lệnh cho phép hoặc cấm chân WakeUp đánh thức CPU)	
A: ENABLE DISABLE	**A:** Cho phép hoặc cấm Cho phép Cấm
PWR_EnterSTOPMode(A, B) ; (Lệnh điều khiển CPU vào chế độ Stop)	
A: PWR_Regulator_ON PWR_Regulator_LowPower **B:** PWR_STOPEntry_WFI PWR_STOPEntry_WFE	**A:** Chọn chế độ hoạt động cho bộ điều chỉnh điện áp khi CPU vào chế độ Stop Chế độ hoạt động bình thường Chế độ tiết kiệm năng lượng **B:** Chọn cách đưa CPU vào chế độ Stop Chờ xảy ra ngắt Chờ xảy ra sự kiện
PWR_EnterSTANDBYMode(); (Lệnh điều khiển CPU vào chế độ Standby)	

Các ví dụ về các chế độ tiết kiệm năng lượng được trình bày ở **chương 5** sau khi SV đã tìm hiểu về ngắt và sự kiện.