Chương 6

TRUY XUẤT BỘ NHỚ TRỰC TIẾP

DIRECT MEMORY ACCESS - DMA

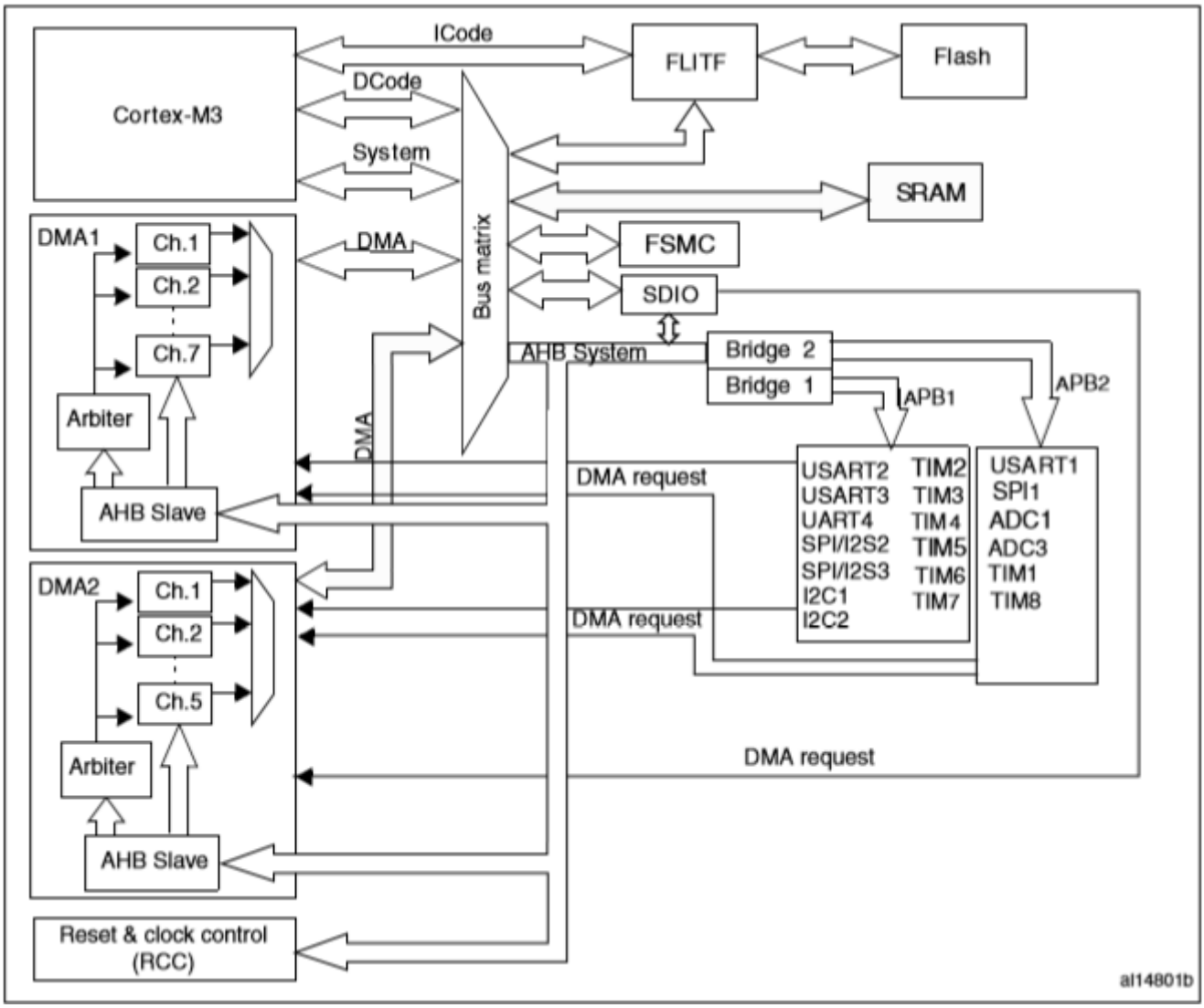
6.1 GIỚI THIỆU DMA

DMA được sử dụng để truyền dữ liệu tốc độ cao giữa :

- Ngoại vi → bộ nhớ
- Bộ nhớ → ngoại vi
- Bộ nhớ → bộ nhớ

Thông qua DMA dữ liệu được truyền đi với tốc độ cao mà không cần có sự can thiệp của CPU, điều này giúp giải phóng tài nguyên CPU để phục vụ cho những tác vụ khác.

Dòng ARM STM32F10X có 2 khối DMA và 2 khối này quản lý tổng cộng 12 kênh (DMA1 có 7 kênh, DMA2 có 5 kênh). 12 kênh DMA này có thể được cấu hình hoạt động độc lập với nhau và được phân chia thứ tự ưu tiên bằng phần mềm.



Hình 6.1 Sơ đồ khối DMA của dòng low, medium, high density

6.2 CHỨC NĂNG CHÍNH CỦA DMA

Có tổng cộng 12 kênh DMA trong đó 7 kênh của DMA1 và 5 kênh của DMA2.

Mỗi kênh được kết nối phần cứng sẵn do đó muốn sử dụng DMA cho ngoại vi nào thì ta phải xem xét xem việc truyền dữ liệu của ngoại vi đó thuộc kênh DMA nào quản lý.

Hỗ trợ chế độ quản lý bộ đệm xoay vòng.

Có 4 cấp độ ưu tiên cho các kênh trong cùng một khối DMA:

- Rất cao
- Cao
- Trung bình
- Thấp

Có thể lập trình cấu hình độc lập chiều rộng dữ liệu cho cả nguồn lẫn đích:

- Byte – 8 bit
- Haft Word – 16 bit
- Word – 32 bit

Có 3 cờ báo trạng thái truyền:

- Truyền một nửa
- Truyền hoàn tất
- Truyền bị lỗi

Có thể tạo ra yêu cầu ngắt ứng với các sự kiện:

- Truyền một nửa
- Truyền hoàn tất
- Truyền bị lỗi

DMA có khả năng truy xuất bộ nhớ Flash, SRAM, các ngoại vi thuộc APB1, APB2, AHB… và sử dụng chúng như là các nguồn và đích để truyền dữ liệu.

Lập trình được số lượng gói dữ liệu cần truyền (tối đa 65536) cho từng kênh.

Chú ý: Chỉ có dòng ARM high density trở lên mới có DMA2. Và do khối DMA2 này quản lý ADC3, SPI/I2S3, UART4, SDIO, TIM5, TIM6, TIM7, TIM8 nên ta không thể sử dụng DMA cho các ngoại vi này trên các dòng ARM thấp hơn.

6.3 DMA VÀ CÁC NGOẠI VI LIÊN QUAN

6.3.1 Các ngoại vi liên quan đến DMA1

Bảng 6.1 Các ngoại vi liên quan đến DMA1

Ngoại vi	Kênh 1	Kênh 2	Kênh 3	Kênh 4	Kênh 5	Kênh 6	Kênh 7
ADC1	ADC1						
SPI/I²S		SPI1_RX	SPI1_TX	SPI/I2S2_RX	SPI/I2S2_TX		
USART		USART3_TX	USART3_RX	USART2_TX	USART2_RX	USART1_TX	USART1_RX
I²C				I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	TIM1_CH2	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
TIM2	TIM2_CH3	TIM2_UP			TIM2_CH1		TIM2_CH2 TIM2_CH4
TIM3		TIM3_CH3	TIM3_CH4 TIM3_UP			TIM3_CH1 TIM3_TRIG	
TIM4	TIM4_CH1			TIM4_CH2	TIM4_CH3		TIM4_UP

6.3.2 Các ngoại vi liên quan đến DMA2

Bảng 6.2 Các ngoại vi liên quan đến DMA2

Ngoại vi	Kênh 1	Kênh 2	Kênh 3	Kênh 4	Kênh 5
ADC3					ADC3
SPI/I2S3	SPI/I2S3_RX	SPI/I2S3_TX			
UART4			UART4_RX		UART4_TX
SDIO				SDIO	
TIM5	TIM5_CH4 TIM5_TRIG	TIM5_CH3 TIM5_UP		TIM5_CH2	TIM5_CH1
TIM6/ DAC_Channel1			TIM6_UP/ DAC_Channel1		
TIM7/ DAC_Channel2				TIM7_UP/ DAC_Channel2	
TIM8	TIM8_CH3 TIM8_UP	TIM8_CH4 TIM8_TRIG TIM8_COM	TIM8_CH1		TIM8_CH2

Chú ý: Khi cần sử dụng DMA cho 1 ngoại vi nào đó ta phải tra trong **bảng 6.1** và **bảng 6.2** để tìm xem ngoại vi đó thuộc kênh DMA nào quản lý nhằm cấu hình cho đúng.

6.4 CÁC LỆNH THÔNG DỤNG LIÊN QUAN ĐẾN DMA

Bảng 6.3 Các lệnh thông dụng để cấu hình DMA

SỬ DỤNG THƯ VIỆN "stm32f10x_dma"	
Lệnh	
Thông số hay dùng	**Giải thích**
DMA_InitTypeDef A; (Khai báo biến **A** thuộc kiểu DMA_InitTypeDef)	
DMA_DeInit(B); (Lệnh hủy các cài đặt trước đó của DMA)	
B: DMA1_Channel1 … DMA1_Channel7 DMA2_Channel1 … DMA2_Channel5	**B:** Kênh DMA cần hủy cài đặt DMA1_Channel1 … DMA1_Channel7 DMA2_Channel1 … DMA2_Channel5
A.DMA_PeripheralBaseAddr= C; (Lệnh cài đặt địa chỉ của vùng nhớ thuộc ngoại vi cần truy xuất)	
C: Địa chỉ 32 bit	**C:** Địa chỉ vùng nhớ thuộc ngoại vi **Chú ý**: ép kiểu 32 bit nếu giá trị B chưa phải là số 32 bit.
A.DMA_MemoryBaseAddr = D; (Lệnh cài đặt địa chỉ vùng nhớ cần truy xuất)	
D: Địa chỉ 32 bit	**D:** Địa chỉ vùng nhớ cần truy xuất **Chú ý**: ép kiểu 32 bit nếu giá trị D chưa phải là số 32 bit.

A.DMA_DIR = E; (Lệnh cấu hình hướng truyền dữ liệu)	
E: DMA_DIR_PeripheralDST DMA_DIR_PeripheralSRC	**E:** Hướng truyền Bộ nhớ → ngoại vi Ngoại vi → bộ nhớ
A.DMA_BufferSize = F; (Lệnh cấu hình số gói dữ liệu cần truyền)	
F: 0 - 65535	**F:** Số gói dữ liệu cần truyền Tối đa 65535 gói
A.DMA_PeripheralInc = G; (Lệnh cấu hình cho phép hoặc cấm tăng địa chỉ truy xuất thuộc ngoại vi)	
G: DMA_PeripheralInc_Enable DMA_PeripheralInc_Disable	**G:** Cho phép hoặc cấm tăng địa chỉ ngoại vi Cho phép tăng Cấm tăng
A.DMA_MemoryInc = H; (Lệnh cấu hình cho phép hoặc cấm tăng địa chỉ truy xuất thuộc bộ nhớ)	
H: DMA_MemoryInc_Enable DMA_MemoryInc_Disable	**H:** Cho phép hoặc cấm tăng địa chỉ Cho phép tăng Cấm tăng
A.DMA_PeripheralDataSize = I; (Lệnh cấu hình bề rộng dữ liệu thuộc ngoại vi)	
I: DMA_PeripheralDataSize_Byte DMA_PeripheralDataSize_HalfWord DMA_PeripheralDataSize_Word	**I:** Bề rộng dữ liệu thuộc ngoại vi 8 bit 16 bit 32 bit
A.DMA_MemoryDataSize = J; (Lệnh cấu hình bề rộng dữ liệu thuộc bộ nhớ)	
J: DMA_MemoryDataSize_Byte DMA_MemoryDataSize_HalfWord DMA_MemoryDataSize_Word	**J:** Bề rộng dữ liệu thuộc bộ nhớ 8 bit 16 bit 32 bit
A.DMA_Mode = K; (Lệnh cấu hình chế độ hoạt động cho DMA)	
K: DMA_Mode_Normal DMA_Mode_Circular	**K:** Chế độ hoạt động Bình thường Xoay vòng
A.DMA_Priority = L; (Lệnh cấu hình cấp độ ưu tiên cho kênh DMA đang dùng)	
L: DMA_Priority_Low DMA_Priority_Medium DMA_Priority_High DMA_Priority_VeryHigh	**L:** Cấp độ ưu tiên Thấp Trung bình Cao Rất cao

A.DMA_M2M = **M;** (Lệnh cho phép hoặc cấm truyền từ bộ nhớ đến bộ nhớ)	
M: DMA_M2M_Enable DMA_M2M_Disable	**M:** Cho phép hoặc cấm Cho phép Cấm
DMA_Init(N,&A); (Lệnh cấu hình các thông số cài đặt lưu trong biến A cho kênh DMA **N**)	
N: DMA1_Channel1 … DMA1_Channel7 DMA2_Channel1 … DMA2_Channel5	**N:** Kênh DMA cần cấu hình DMA1 kênh 1 … DMA1 kênh 7 DMA2 kênh 1 … DMA2 kênh 5
DMA_Cmd(N, O); (Lệnh cho phép hoặc cấm kênh DMA **N** hoạt động)	
N: DMA1_Channel1 … DMA1_Channel7 DMA2_Channel1 … DMA2_Channel5 **O:** ENABLE DISABLE	**N:** Kênh DMA cần cấu hình DMA1 kênh 1 … DMA1 kênh 7 DMA2 kênh 1 … DMA2 kênh 5 **O:** Cho phép hoặc cấm Cho phép Cấm

Bảng 6.4 Các lệnh thông dụng để kiểm tra việc truyền của DMA

SỬ DỤNG THƯ VIỆN "stm32f10x_dma"	
Lệnh	
Thông số hay dùng	**Giải thích**
DMA_GetFlagStatus(A); (Lệnh đọc cờ trạng thái của DMA)	
A: DMA1_FLAG_GL1 DMA1_FLAG_TC1 DMA1_FLAG_HT1 DMA1_FLAG_TE1 … DMA2_FLAG_GL5 DMA2_FLAG_TC5 DMA2_FLAG_HT5 DMA2_FLAG_TE5	**A:** Cờ cần đọc Toàn cục DMA1 kênh 1 Truyền xong DMA1 kênh 1 Truyền ½ DMA1 kênh 1 Truyền bị lỗi DMA1 kênh 1 … Toàn cục DAM2 kênh 5 Truyền xong DMA2 kênh 5 Truyền ½ DMA2 kênh 5 Truyền bị lỗi DMA2 kênh 5

DMA_ClearFlag (A); (Lệnh xóa cờ báo báo trạng thái truyền của DMA)	
A:	**A:** Cờ cần xóa
DMA1_FLAG_GL1	Toàn cục DMA1 kênh 1
DMA1_FLAG_TC1	Truyền xong DMA1 kênh 1
DMA1_FLAG_HT1	Truyền ½ DMA1 kênh 1
DMA1_FLAG_TE1	Truyền bị lỗi DMA1 kênh 1
…	…
DMA2_FLAG_GL5	Toàn cục DAM2 kênh 5
DMA2_FLAG_TC5	Truyền xong DMA2 kênh 5
DMA2_FLAG_HT5	Truyền ½ DMA2 kênh 5
DMA2_FLAG_TE5	Truyền bị lỗi DMA2 kênh 5

Bảng 6.5 Các lệnh thông dụng khi sử dụng ngắt DMA

SỬ DỤNG THƯ VIỆN "stm32f10x_dma"	
Lệnh	
Thông số hay dùng	**Giải thích**
DMA_ITConfig (A, B,C); (Lệnh cho phép hoặc cấm kênh DMA **A** ngắt)	
A:	**A:** Kênh cần cho phép hoặc cấm ngắt
DMA1_Channel1	DMA1 kênh 1
DMA1_Channel2	DMA1 kênh 2
…	…
DMA1_Channel7	DMA1 kênh 7
DMA2_Channel1	DMA2 kênh 1
DMA2_Channel2	DMA2 kênh 2
…	…
DMA2_Channel5	DMA2 kênh 5
B:	**B:** Sự kiện xảy ra ngắt
DMA_IT_TC	Ngắt khi truyền hoàn tất
DMA_IT_HT	Ngắt khi truyền một nửa
DMA_IT_TE	Ngắt khi truyền bị lỗi
C:	**C**: Cho phép hoặc cấm ngắt
ENABLE	Cho phép
DISABLE	Cấm

DMA_GetITStatus (D); (Lệnh đọc cờ báo ngắt của DMA)	
D:	**D:** Cờ ngắt cần đọc
DMA1_IT_GL1	Toàn cục DMA1 kênh 1
DMA1_IT_TC1	Truyền xong DMA1 kênh 1
DMA1_IT_HT1	Truyền ½ DMA1 kênh 1
DMA1_IT_TE1	Truyền bị lỗi DMA1 kênh 1
…	…
DMA2_IT_GL5	Toàn cục DAM2 kênh 5
DMA2_IT_TC5	Truyền xong DMA2 kênh 5
DMA2_IT_HT5	Truyền ½ DMA2 kênh 5
DMA2_IT_TE5	Truyền bị lỗi DMA2 kênh 5
DMA_ClearITPendingBit(D); (Lệnh xóa cờ báo ngắt của DMA)	
D:	**D:** Cờ cần xóa
DMA1_IT_GL1	Toàn cục DMA1 kênh 1
DMA1_IT_TC1	Truyền xong DMA1 kênh 1
DMA1_IT_HT1	Truyền ½ DMA1 kênh 1
DMA1_IT_TE1	Truyền bị lỗi DMA1 kênh 1
…	…
DMA2_IT_GL5	Toàn cục DAM2 kênh 5
DMA2_IT_TC5	Truyền xong DMA2 kênh 5
DMA2_IT_HT5	Truyền ½ DMA2 kênh 5
DMA2_IT_TE5	Truyền bị lỗi DMA2 kênh 5

Chú ý: Để DMA có thể ngắt được thì cần phải cấu hình cho phép ngắt DMA ở NVIC

Bảng 6.6 Các bước để cấu hình DMA

1	Cài đặt địa chỉ	Địa chỉ thanh ghi cần truy xuất ở ngoại vi
		Địa chỉ vùng nhớ cần truy xuất
2	Cài đặt hướng truyền dữ liệu	
3	Cài đặt số gói dữ liệu cần truyền	
4	Cài đặt chế độ tăng địa chỉ	Cho phép hoặc cấm ngoại vi tăng địa chỉ
		Cho phép hoặc cấm bộ nhớ tăng địa chỉ
5	Cài đặt bề rộng dữ liệu	Bề rộng dữ liệu của ngoại vi
		Bề rộng dữ liệu ở ram
6	Cài đặt chế độ hoạt động của kênh DMA đang cấu hình	
7	Cài đặt thứ tự ưu tiên của kênh DMA đang cấu hình	
8	Cho phép hoặc cấm truyền từ bộ nhớ tới bộ nhớ	
9	Cho phép kênh DMA đang cấu hình hoạt động	

6.5 CÁC VÍ DỤ LIÊN QUAN ĐẾN DMA

6.5.1 Ví dụ về cấu hình truyền dữ liệu từ ngoại vi đến bộ nhớ

Ví dụ 6.1: Cấu hình sử DMA để truyền kết quả chuyển đổi của ADC1(giá trị 16 bit) vào một mảng 20 phần tử "**unsigned short KQADC[20]**". Với yêu cầu dữ liệu cần được cập nhật tự động và liên tục. Biết thanh ghi lưu kết quả chuyển đổi của ADC1 là ADC1->DR.

a. Phân tích yêu cầu

Dò trong **bảng 6.1 và 6.2** ta thấy việc truyền dữ liệu của ADC1 do DMA1 kênh 1 quản lý

⇨ Cần phải cấu hình và sử dụng **DMA1 kênh 1**

Đề bài yêu cầu lưu vào mảng 20 phần tử và cập nhật tự động liên tục

⇨ Chọn số gói dữ liệu là 20 và chế độ hoạt động của DMA là **DMA_Mode_Circular**

Do giá kết quả chuyển đổi của ADC là số 16 bit

⇨ Chọn chiều rộng dữ liệu là **HalfWord**

Yêu cầu truyền dữ liệu từ ADC về một mảng thuộc bộ nhớ

⇨ Chọn hướng chuyền là **DMA_DIR_PeripheralSRC.**

b. Chương trình

```
void CauhinhDMA()
{
    DMA_InitTypeDef DMA_ADC;
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
    // Cấp xung clock cho DMA1
    DMA_DeInit(DMA1_Channel1);
    // Xóa cấu hình trước đó của DMA1 kênh 1
    DMA_ADC.DMA_PeripheralBaseAddr = (uint32_t)&ADC1->DR;
    // Chọn địa chỉ ngoại vi là địa chỉ của thanh ghi ADC1_DR
    DMA_ADC.DMA_MemoryBaseAddr = (uint32_t)&KQADC;
    // Chọn địa chỉ bộ nhớ là địa chỉ của biến KQADC
    DMA_ADC.DMA_DIR = DMA_DIR_PeripheralSRC;
    // Hướng truyền từ ngoại vi đến bộ nhớ
    DMA_ADC.DMA_BufferSize = 20;
    // Số lượng gói dữ liệu cần truyền là 20
    DMA_ADC.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    // Không tăng địa chỉ ngoại vi do kết quả chuyển đổi
    // ADC được lưu cố định tại ADC1_DR
    DMA_ADC.DMA_MemoryInc = DMA_MemoryInc_Enable;
    // Tăng địa chỉ bộ nhớ do phải lưu 20
    //phần tử của mảng KQADC
    DMA_ADC.DMA_PeripheralDataSize = \
                            DMA_PeripheralDataSize_HalfWord;
    // Kích thước dữ liệu cần truyền của ngoại vi là 16 bit
    DMA_ADC.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
    // Kích thước dữ liệu cần nhận của bộ nhớ là 16 bit
    DMA_ADC.DMA_Mode = DMA_Mode_Circular;
    // Chọn chế độ xoay vòng
    DMA_ADC.DMA_Priority = DMA_Priority_High;
    // Chọn độ ưu tiên cao
    DMA_ADC.DMA_M2M = DMA_M2M_Disable;
    // Tắt chế độ truyền từ bộ nhớ đến bộ nhớ
    DMA_Init(DMA1_Channel1, &DMA_ADC);
```

```
    // Cấu hình các thông số đã chọn cho DMA1 kênh 1
    DMA_Cmd(DMA1_Channel1, ENABLE);
    // Cho phép DMA1 kênh 1 hoạt động
}
```

6.5.2 Ví dụ về truyền dữ liệu từ bộ nhớ đến bộ nhớ, kiểm tra cờ để biết truyền xong

Ví dụ 6.2: Viết chương trình sử dụng DMA để chuyển mảng dữ liệu lưu trong Flash "**const unsigned short DLN[5]={0x0800,0x0400,0x0200,0x0100,0x0000};**" sang mảng dữ liệu lưu trong SRAM "**unsigned short DLD[5];**" với các yêu cầu sau:
1. Dùng phương pháp kiểm tra cờ truyền hoàn tất để chắc là DMA đã truyền xong rồi mới xử lý tiếp.
2. Sau khi chắc chắn là DMA đã truyền xong thì xuất mảng DLD ra 4 led đơn được nối với các chân từ D8 đến D11 để quan sát việc DMA chuyển dữ liệu từ bộ nhớ tới bộ nhớ.

a. Phân tích yêu cầu

Đề bài yêu cầu lưu vào mảng 5 phần tử và chỉ cần cập nhật 1 lần
⇨ Chọn số gói dữ liệu là 5 và chế độ hoạt động của DMA là **DMA_Mode_Normal**
Do dữ liệu vận chuyển là số 16 bit
⇨ Chọn chiều rộng dữ liệu là **HalfWord**

b. Chương trình

```
#include<stm32f10x.h>
const unsigned short DLN[5]={0x0800,0x0400,0x0200,0x0100,0};
unsigned short DLD[5];
void delay(unsigned long t){while(t--);}
void cauhinhLED()
{
    GPIO_InitTypeDef    GPIO_InitStructure;
    RCC_APB2PeriphClockCmd (RCC_APB2Periph_GPIOD,ENABLE );
    GPIO_InitStructure.GPIO_Pin =
                GPIO_Pin_8|GPIO_Pin_9|GPIO_Pin_10|GPIO_Pin_11;
    GPIO_InitStructure .GPIO_Mode  = GPIO_Mode_Out_PP  ;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz ;
    GPIO_Init(GPIOD,&GPIO_InitStructure);
}
void cauhinhDMA()
{
    DMA_InitTypeDef DMA_InitStructure;
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
    DMA_DeInit(DMA1_Channel1);
    DMA_InitStructure.DMA_PeripheralBaseAddr = (u32)&DLN;
    DMA_InitStructure.DMA_MemoryBaseAddr = (u32)&DLD;
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
    DMA_InitStructure.DMA_BufferSize = 5;
    DMA_InitStructure.DMA_PeripheralInc =
                                    DMA_PeripheralInc_Enable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
```

```
    DMA_InitStructure.DMA_PeripheralDataSize =
                                    DMA_PeripheralDataSize_HalfWord;
    DMA_InitStructure.DMA_MemoryDataSize =
                                   DMA_MemoryDataSize_HalfWord;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
    // Cho phép chế độ truyền từ bộ nhớ đến bộ nhớ
    DMA_Init(DMA1_Channel1, &DMA_InitStructure);
    DMA_Cmd(DMA1_Channel1, ENABLE);
}
int main()
{
    char i;
    SystemInit();
    cauhinhDMA();
    cauhinhLED();
    while(!DMA_GetFlagStatus(DMA1_FLAG_TC1));
    // Chờ truyền xong
    DMA_ClearFlag (DMA1_FLAG_TC1);
    // Xóa cờ truyền hoàn tất
    while(1)
     {
            for(i=0;i<5;i++)
            {
                    GPIOD->ODR = DLD[i];
                    delay(500000);
            }
     }
 }
```

6.5.3 Ví dụ về truyền dữ liệu từ bộ nhớ đến bộ nhớ sử dụng ngắt khi truyền xong

Ví dụ 6.3: Yêu cầu giống **ví dụ 6.2** chỉ khác ở chỗ là sử dụng ngắt để biết truyền hoàn tất thay vì kiểm tra cờ báo.

a. **Phân tích yêu cầu**
   (giống bài **ví dụ 6.2**)

b. **Chương trình**

```
#include<stm32f10x.h>
const unsigned short DLN[5]={0x0800,0x0400,0x0200,0x0100,0};
unsigned short DLD[5];
char i;
void delay(unsigned long t){while(t--);}
void cauhinhLED()
{
    GPIO_InitTypeDef    GPIO_InitStructure;
    RCC_APB2PeriphClockCmd (RCC_APB2Periph_GPIOD,ENABLE );
    GPIO_InitStructure.GPIO_Pin =
```

```
                    GPIO_Pin_8|GPIO_Pin_9|GPIO_Pin_10|GPIO_Pin_11;
    GPIO_InitStructure .GPIO_Mode  = GPIO_Mode_Out_PP  ;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz ;
    GPIO_Init(GPIOD,&GPIO_InitStructure);
}
void cauhinhDMA()
{
    DMA_InitTypeDef DMA_InitStructure;
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
    DMA_DeInit(DMA1_Channel1);
    DMA_InitStructure.DMA_PeripheralBaseAddr = (u32)&DLN;
    DMA_InitStructure.DMA_MemoryBaseAddr = (u32)&DLD;
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
    DMA_InitStructure.DMA_BufferSize = 5;
    DMA_InitStructure.DMA_PeripheralInc =
                                    DMA_PeripheralInc_Enable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_PeripheralDataSize =
                             DMA_PeripheralDataSize_HalfWord;
    DMA_InitStructure.DMA_MemoryDataSize =
                                  DMA_MemoryDataSize_HalfWord;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
    DMA_Init(DMA1_Channel1, &DMA_InitStructure);
    DMA_ITConfig (DMA1_Channel1 ,DMA_IT_TC ,ENABLE );
    // Cho phép ngắt truyền hoàn tất DMA1 kênh 1
    DMA_Cmd(DMA1_Channel1, ENABLE);
}
void cauhinhNVIC()
{
    NVIC_InitTypeDef        NVIC_InitStructure;
    #ifdef  VECT_TAB_RAM
    NVIC_SetVectorTable(NVIC_VectTab_RAM, 0x0);
    #else
      NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
    #endif
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0 );
    NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel1_IRQn;
    // Cho phép ngắt DMA1 kênh 1
    NVIC_InitStructure.NVIC_IRQChannelSubPriority =0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
void DMA1_Channel1_IRQHandler()
{
    if(DMA_GetITStatus(DMA1_IT_TC1))
    // Kiểm tra cờ ngắt truyền hoàn tất DMA1 kênh 1
    {
         DMA_ClearITPendingBit (DMA1_IT_TC1);
```

```
        // Xóa cờ ngắt truyền hoàn tất DMA1 kênh 1

        while(1)
         {
            for(i=0;i<5;i++)
            {
                    GPIOD->ODR = DLD[i];
                    delay(5000000);
            }
         }
    }
}
int main()
{
    SystemInit();
    cauhinhLED();
    cauhinhDMA();
    cauhinhNVIC();
    while(1){}
 }
```