Embedded System (http://hethongnhung.com/)

Hướng dẫn lập trình DMA với STM32

Download Source code : STM32_DMA (http://hethongnhung.com/wp-content/uploads/2015/01/STM32_DMA.zip)

1. Tìm hiểu khái niệm DMA là gì?

– DMA (Direct memory access): là một cơ chế truyền dữ liệu tốc độ cao từ ngoại vi tới bộ nhớ cũng như từ bộ nhớ tới bộ nhớ. Dữ liệu có thể được di chuyển một cách nhanh chóng mà không cần tới tác vụ từ CPU, tiết kiệm tài nguyên CPU cho các hoạt động khác.

– Trong nhiều project mcu bạn cần đọc và ghi dữ liệu. Chẳng hạn bạn cần đọc dữ liệu từ ngoại vi như ADC và ghi các giá trị đọc được vào RAM. Hoặc trong trường hợp khác bạn cần gửi 1 khối dữ liệu sử dụng SPI. Khi đó bạn cần phải thực hiện đọc dữ liệu từ RAM và ghi nó vào thanh ghi SPI data. Bình thường nếu sử dụng cpu để làm việc này thì nó sẽ bị mất một khoảng thời gian đáng kể để xử lý. Trong những trường hợp này, để tránh việc cpu bận rộn và giành thời gian cho những thao tác khác thì ở những mcu phổ biến đều có hỗ trợ DMA (direct memory access). Nó sẽ thực hiện việc giao tiếp với memory mà không cần dùng đến cpu.

*** DMA của STM32 :**

STM32 có 2 bộ DMA với 12 kênh (7 kênh DMA1 và 5 kênh DMA2), mỗi bộ quản lý việc truy cập bộ nhớ từ một hoặc nhiều ngoại vi. DMA cũng có chức năng phân xử độ ưu tiên giữa các DMA request.

– 12 kênh DMA độc lập, có thể thiết lập được. 7 kênh DMA1 và 5 kênh DMA2

– Software trigger được hỗ trợ cho mỗi kênh, và được lập trình bởi phần mềm.

– Độ ưu tiên giữa các kênh DMA có thể lập trình bởi phần mềm (có 4 cấp ưu tiên là very high, high, medium, low) hoặc phần cứng.

– Phụ thuộc vào kích thước giữa nguồn và đích (byte, half word, word). Địac chỉ nguồn/đích phải phù hợp với kích thước dữ liệu.

– Hỗ trợ truyền tải giữa:

+ Memory to memory

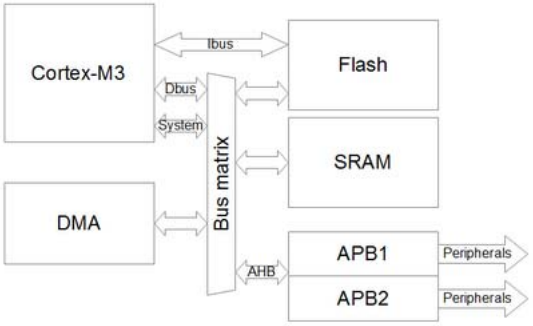
+ Peripheral to memory

+ Memory to peripheral

+ Peripheral to peripheral

– Có thể truy cập vào Flash, Sram, APB1, APB2 và AHB như nguồn và đích.

– Dữ liệu truyền nhận hỗ trợ tới 65536



(http://hethongnhung.com/wp-content/uploads/2015/01/dma1.png)

2. Tìm hiểu cách lập trình DMA trong STM32 thông qua ví dụ cụ thể

Mỗi channel được điều khiển bởi 4 thanh ghi : Memory address, peripheral address, number of data and configuration. Và tất cả các channel đều có 2 thanh ghi được giành riêng là : DMA interrupt status register and interrupt flag clear register. Các channel của DMA có thể tạo ra 3 interupt là : transfer finished, half-finished and transfer error.

Bắt đầu với 1 ví dụ là thực hiện công việc chuyển dữ liệu giữa 2 mảng. Trong đó có 2 trường hợp là có sử dụng DMA và không sử dụng DMA mà để cpu thực hiện bình thường. Sau đó so sánh thời gian trong 2 trường hợp trên.

Trước khi đi vào phân tích code của ví dụ thì mình sẽ tìm hiểu biến cấu trúc của DMA đã được định nghĩa sẵn bao gồm những thanh ghi chức năng như thế nào :

```
typedef struct
{
uint32_t DMA_PeripheralBaseAddr; /*!< Specifies the peripheral base address for DMAy Channelx. */

uint32_t DMA_MemoryBaseAddr; /*!< Specifies the memory base address for DMAy Channelx. */

uint32_t DMA_DIR; /*!< Specifies if the peripheral is the source or destination.
                This parameter can be a value of @ref DMA_data_transfer_direction */

uint32_t DMA_BufferSize; /*!< Specifies the buffer size, in data unit, of the specified Channel.
                   The data unit is equal to the configuration set in DMA_PeripheralDataSize
                   or DMA_MemoryDataSize members depending in the transfer direction. */

uint32_t DMA_PeripheralInc; /*!< Specifies whether the Peripheral address register is incremented or

uint32_t DMA_MemoryInc; /*!< Specifies whether the memory address register is incremented or not.
                        This parameter can be a value of @ref DMA_memory_incremented_mode */

uint32_t DMA_PeripheralDataSize; /*!< Specifies the Peripheral data width.
                                  This parameter can be a value of @ref DMA_peripheral_data_size */

uint32_t DMA_MemoryDataSize; /*!< Specifies the Memory data width.
                             This parameter can be a value of @ref DMA_memory_data_size */

uint32_t DMA_Mode; /*!< Specifies the operation mode of the DMAy Channelx.
                   This parameter can be a value of @ref DMA_circular_normal_mode.
                   @note: The circular buffer mode cannot be used if the memory-to-memory
                   data transfer is configured on the selected Channel */

uint32_t DMA_Priority; /*!< Specifies the software priority for the DMAy Channelx.
                      This parameter can be a value of @ref DMA_priority_level */

uint32_t DMA_M2M; /*!< Specifies if the DMAy Channelx will be used in memory-to-memory transfer.
                   This parameter can be a value of @ref DMA_memory_to_memory */
}DMA_InitTypeDef;
```

Cấu trúc biến init DMA gồm 11 thành phần và ý ngh ĩa của từng thành phần như sau :

– **DMA_PeripheralBaseAddr , DMA_MemoryBaseAddr** : Xác định địa chỉ của ngoại vi và địa chỉ của bộ nhớ cho DMA channel, hay nói cách khác là xác định địa chỉ nguồn và đích trong việc trao đổi dữ liệu.

–**DMA_DIR** : Chọn hướng chuyển dữ liệu từ ngoại vi đến bộ nhớ hay từ bộ nhớ đến ngoại vi.

```
/** @defgroup DMA_data_transfer_direction **/
#define DMA_DIR_PeripheralDST ((uint32_t)0x00000010)
#define DMA_DIR_PeripheralSRC ((uint32_t)0x00000000)
```

– **DMA_BufferSize** : Kích thước của mảng dữ liệu.

–**DMA_PeripheralInc, DMA_MemoryInc** :

+ **DMA_PeripheralInc** : Đối với ngoại vi bạn nên disable mode này do nếu bạn bật mode này thì mỗi lần chuyển dữ liệu thì địa chỉ ngoại vi sẽ tăng dần, điều này là không cần thiết và rất nguy hiểm nếu nh ư bạn không nắm rõ địa chỉ trỏ đến tiếp theo.

+ **DMA_MemoryInc** : Đối với memory bạn cần enable mode này, mỗi khi chuy ển đổi xảy ra bạn cần tăng địa chỉ bộ nhớ của bạn bởi ví dụ biến ADCValue có đến 3 phần tử, nếu không tăng địa chỉ lên thì chỉ duy nhất có biến ADCValue[0] là có dữ liệu.

– **DMA_PeripheralDataSize , DMA_MemoryDataSize** : Chọn kích thước mảng dữ liệu ADCValue gồm : Byte, Haftword và Word.

```
/** @defgroup DMA_peripheral_data_size */
#define DMA_PeripheralDataSize_Byte ((uint32_t)0x00000000)
#define DMA_PeripheralDataSize_HalfWord ((uint32_t)0x00000100)

#define DMA_PeripheralDataSize_Word ((uint32_t)0x00000200)
```

– **DMA_Mode** : Circular mode & NonCircular mode

+ Chọn mode DMA chế độ vòng tròn, có nghĩa là việc chuyển đổi liên tục lặp lại. Khi circular mode được actived thì số dữ liệu được transfer sẽ tự động reload lại với những thiết lập đã được lập trình theo những thông số config cho channel.
+ Nếu channel được config ở chế độ concircular mode thì sẽ không có DMA request được tạo sau mỗi lần transfer.

```
/** @defgroup DMA_circular_normal_mode */
#define DMA_Mode_Circular ((uint32_t)0x00000020)
#define DMA_Mode_Normal ((uint32_t)0x00000000)
```

– **DMA_Priority** : Xác định độ ưu tiên của kênh DMA ,có 4 độ ưu tiên bao gồm :

+ DMA_Priority_High
+ DMA_Priority_Low
+ DMA_Priority_Medium
+ DMA_Priority_VeryHigh

– **DMA_M2M** : Kênh DMA cũng có thể được kích hoạt mà không cần request từ ngoại vi, chế độ này được gọi là memory to memory mode. Nếu bit MEM2MEM trong thanh ghi DMA_CCRx được set thì channel sẽ init transfer ngay sau khi được enable bằng software nghĩa là enable bit EN trong thanh ghi DMA_CCRx. Quá trình transfer sẽ ngừng mỗi khi thanh ghi DMA_CNDTRx zero. Memory to memory mode không được sử dụng đồng thời với Circular mode.

```
/** @defgroup DMA_memory_to_memory **/
#define DMA_M2M_Enable ((uint32_t)0x00004000)
#define DMA_M2M_Disable ((uint32_t)0x00000000)
```

Đây là code DMA chuyển dữ liệu trong trường hợp từ memory đến memory :

```
#include "stm32f10x.h"
#include "leds.h"
#define ARRAYSIZE 800
volatile uint32_t status = 0;
volatile uint32_t i;
int main(void)
{
//initialize source and destination arrays
uint32_t source[ARRAYSIZE];
uint32_t destination[ARRAYSIZE];
//initialize array
for (i=0; i<ARRAYSIZE;i++)
    source[i]=i;
//initialize led
LEDsInit();
//enable DMA1 clock
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
//create DMA structure
DMA_InitTypeDef  DMA_InitStructure;
//reset DMA1 channe1 to default values;
DMA_DeInit(DMA1_Channel1);
//channel will be used for memory to memory transfer
DMA_InitStructure.DMA_M2M = DMA_M2M_Enable;
//setting normal mode (non circular)
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
//medium priority
DMA_InitStructure.DMA_Priority = DMA_Priority_Medium;
//source and destination data size word=32bit
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
//automatic memory increment enable. Destination and source
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
//Location assigned to peripheral register will be source
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
//chunk of data to be transfered
DMA_InitStructure.DMA_BufferSize = ARRAYSIZE;
//source and destination start addresses
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)source;
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)destination;
//send values to DMA registers
DMA_Init(DMA1_Channel1, &DMA_InitStructure);
// Enable DMA1 Channel Transfer Complete interrupt
DMA_ITConfig(DMA1_Channel1, DMA_IT_TC, ENABLE);

NVIC_InitTypeDef NVIC_InitStructure;
//Enable DMA1 channel IRQ Channel */
NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);

//LED on before transfer
LEDToggle(LEDG);
//Enable DMA1 Channel transfer
DMA_Cmd(DMA1_Channel1, ENABLE);
while(status==0) {};
    LEDToggle(LEDB);
    for (i=0; i<ARRAYSIZE;i++)
    {
        destination[i]=source[i];
```

```
    }
    LEDToggle(LEDB);

  while (1)
    {
      //interrupts does the job
    }
  }
```

Bắt đầu phân tích :

- Đầu tiên chúng ta tạo ra 2 mảng dữ liệu là : source và destination. Kích thước của mảng được xác định bởi ARRAYSIZE, trong ví dụ này kích thước là 800.
- Trong ví dụ này,ta sử dụng trạng thái của Led để báo hiệu quá trình transfer dữ liệu start và stop trong cả 2 mode : DMA và CPU.

+ Đầu tiên ta phải cấu hình enable clock cho *DMA1 RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE)*.

+ Sau đó bắt đầu cấu hình thông qua cấu trúc DMA_InitStructure như đã phân tích bên trên. Trong ví dụ này ta chọn DMA1 channel1, gọi hàm *DMA_DeInit(DMA1_Channel1)* để chắc chắn rằng DMA được reset về giá trị mặc định ban đầu.

+ Sau đó chọn DMA mode memory to memory(*DMA_InitStructure.DMA_M2M = DMA_M2M_Enable*)

+ Chọn normal DMA mode hay còn gọi là nonCircular mode(*DMA_InitStructure.DMA_Mode = DMA_Mode_Normal*).

+ Chọn chế độ ưu tiên cho kênh DMA này laafe Medium (*DMA_InitStructure.DMA_Priority = DMA_Priority_Medium*).

+ Chọn kích thước mảng dữ liệu để transfer là 32-bit world *(DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word; DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word*). Tương tự với cả peripheral và memory address.

- Chú ý : Nếu kích thước dữ liệu của 2 thành phần trên không giống nhau,chẳng hạn 32-bit source và 8-bit destination thì DMA sẽ thực hiện thành 4 chu kỳ với mỗi chu kỳ là 8 bit.

+ Sau khi đã cấu hình địa chỉ source và destination, cũng như kích thước dữ liệu để gửi. Ta sử dụng hàm

DMA_Init(DMA_Channel1, &DMA_InitStructure) để init các thông số cấu hình bên trên vào thanh ghi.

+ Bây giờ thì DMA có thể sẵn sàng để transfer dữ liệu, bất cứ khi nào lệnh sau được thực thi DMA_Cmd(*DMA_Channel1, ENABLE*).

+ Để bắt interrupt khi quá trình DMA transfer complete trên channel1. Ta cấu hình interrupt như sau :

```
NVIC_InitTypeDef NVIC_InitStructure;

//Enable DMA1 channel IRQ Channel */

NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel1_IRQn;

NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;

NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;

NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

NVIC_Init(&NVIC_InitStructure);
```

Trước khi bắt đầu chuyển đổi bằng DMA thì ta bật led on để báo trạng thái bắt đầu LEDToggle(LEDG) . Khi quá trình chuyển đổi vừa xong thì nó sẽ tạo ra 1 interrupt complete và thực hiển đảo trạng thái led trong interrupt này để báo hiệu.

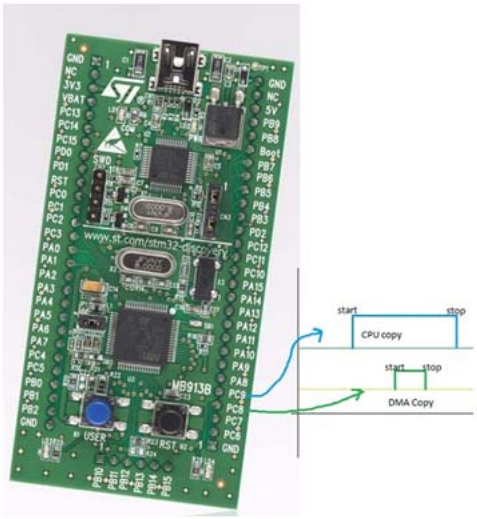
```
voidDMA1_Channel1_IRQHandler(void){

 //Test on DMA1 Channel1 Transfer Complete interrupt

 if(DMA_GetITStatus(DMA1_IT_TC1))

 {

    status=1;
```

```
    LEDToggle(LEDG);
  //Clear DMA1 Channel1 Half Transfer, Transfer Complete and Global interrupt pending bits
   DMA_ClearITPendingBit(DMA1_IT_GL1);
 }
}
```

+ Như vậy đó là code cho phần chuyển dữ liệu bằng DMA ,tiếp theo là phần chuyển dữ liệu bằng CPU bình thường và ta cũng sử dụng trạng thái của LED để quan sát thời gian chuyển đổi.

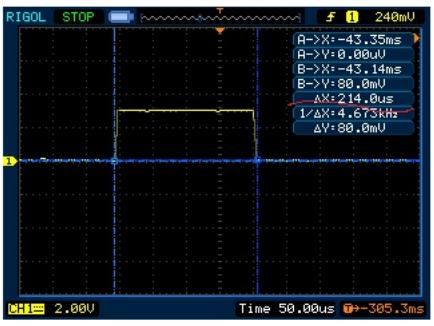
```
//wait for DMA transfer to be finished
while(status==0) {};
   LEDToggle(LEDB);
   for (i=0; i<ARRAYSIZE;i++)
   {
      destination[i]=source[i];
   }
   LEDToggle(LEDB);
```

– Trong ví dụ này, LEDG (DMA) được kết nối đến GPIOC pin 9 và LEDB (CPU) được kết nối đến GPIOC pin 8, Để quan sát rõ hơn quá trình transfer ta kết nối 2 pin này với OSC như sau :



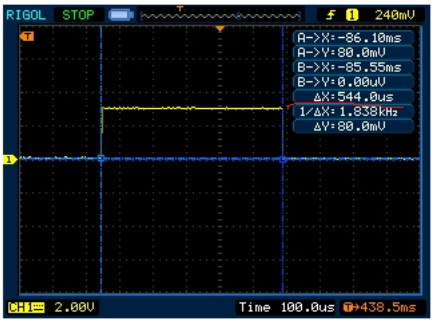
(http://hethongnhung.com/wp-content/uploads/2015/01/dam2.png)

– Transfer sử dụng DMA mất 214µs:



(http://hethongnhung.com/wp-content/uploads/2015/01/dma3.png)

– Trong khi sử dụng CPU để copy memory memory mất 544µs (gần gấp 3 lần so với sử dụng DMA):



(http://hethongnhung.com/wp-content/uploads/2015/01/dma4.png)

– Ví dụ này cho thấy tốc độ truyền dữ liệu nhanh hơn đáng kể so với việc sử dụng CPU bình thường và lợi ích lớn nhất là CPU hoàn toàn rảnh rỗi trong lúc transfer và có thể làm nhiệm vụ khác hoặc chỉ đơn giản là vào chế độ sleep mode.

– Hy vọng ví dụ này giúp bạn có một ý tưởng về tầm quan trọng DMA . Với DMA chúng ta có thể làm vô số công việc ở mức harward .

Leave a comment