Chapter 1 Introduction

1.1 Background Describe Direct Memory Access (DMA).

In this chapter, was provided examples and a detailed explanation of the DMA controller used in stm32f429 Discovery board.

DMA controller transfers data from one address to another without CPU involvement.. The most advanced of DMA controller is to avoid occupying CPU. DMA controller transfer data from ADC12 conversion memory to RAM then transfer data from RAM to DAC12 device. The content of DMA controller may have one or more than two DMA channels available using the DMA controller. The number of DMA channel can increase throughput of peripheral module. DMA controller can reduce system power consumption because allow the CPU to remain in a low power mode without having to awaken to move transfer data or from a peripheral or memory.

DMA is a means of taking in a peripheral device control a processor memory bus at once. DMA permits the peripheral, such as a UART, to transfer data directly to or from memory without having each byte (or word) handled by the CPU. This DMA enables more effective use of interrupts, increases data throughput, and potentially reduces hardware costs by doing away with the need for peripheral-specific FIFO buffers.

The DMA controller asserts a DMA request signal to the CPU. The DMA controller requests the permit to use the AHB bus. The CPU completes its current bus activity, stop driving the bus and return a DMA acknowledge signal to the DMA controller. The next step DMA controller reads and writes one or more memory byte, then driving the address, data and signals as itself the CPU. At one time the transfers complete DMA stop driving the bus and assert the DMA request the signal. The CPU then removes its DMA acknowledge signal and resume control of the bus

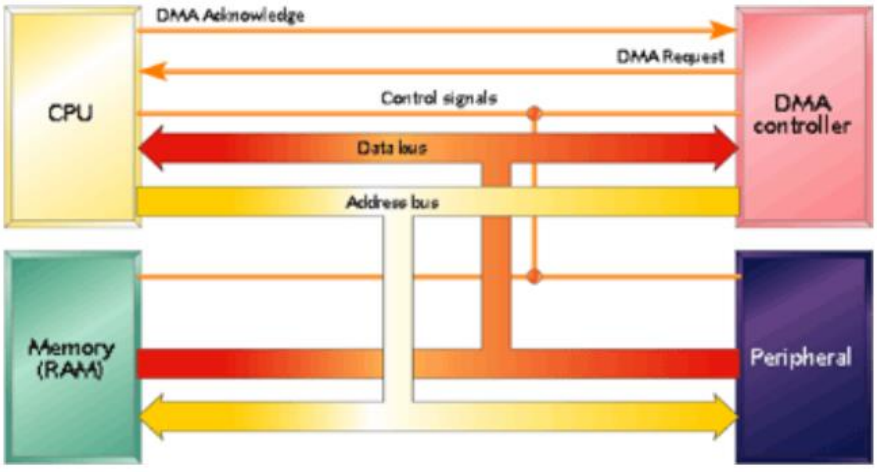


Figure 1.1 DMA controllers the processor bus. [Accessed from *http://www.embedded.com/electronics-blogs/beginner-s-corner/4024879/Introduction-to-direct-memory-access*]

Chapter 2 Methodology

2.1 Introduce the DMA feature in stm32f429 Discovery board.

In the stm32f429 Discovery board there have two DMA controllers almost identical. The main difference between them is DMA2 can performance memory to memory transfer mode.

DMA controller can transfer data from a peripheral to a memory, memory to peripheral or memory to memory. DMA request that can be selected out of 8 possible channel requests and each stream contain 8 possible channels. The selection of the channel and stream can be select from the DMA request mapping as shown in appendix. If select the transfer mode was memory to memory then can chose any column from the DMA2 request mapping.

The circular mode, direct mode and double buffer mode not allow in memory to memory transfer mode. DMA_SXCR register is the register which configured the DMA controller. DMA_NDTR register setting the number of data transfer. This value is decremented after each peripheral event or each beat of the burst. DMA_NDTR register is the data width, such as 8, 16 and 32 bits generally with the same peripheral FIFO. The burst size is the DMA once can transfer through a several data size from source to destination depend on PBURST and PSIZE or MBURST bits and MSIZE bits in the DMA_SXCR register.

The interrupt status register (DMA_LISR) for stream zero to three. DMA_HISR register for stream four to seven. All the DMA registers had been shown in appendix. The entire possible DMA configuration had been shown in figure 1.2.

The flow controller is determines the number of data elements for transmission for one DMA transaction. There are 2 types of flow control that can be selected using PFCTRL in the register DMA_SxCR [5]. The flow controller DMA is the number of data elements to be transmitted to install software (programmer) to register DMA_SxNDTR. The SxNDTR does not specify the number of bytes to transfer and it does depend on data size (MSIZE and PSIZE). The peripheral controller is the number of items of data transmitted is not known in advance. Peripheral hardware itself tells DMA controller when last transmitted data elements. The value that is loaded to DMA_SxNDTR does not matter.

The circular mode is used for transmission of circular data (scanning DAC). This mode is enabled by setting bit CIRC in the register DMA_SxCR[8]. The DMA after each transaction the number of transmitted data elements are automatically reloaded initial value NDTR, programmed during configuration flow. After each transaction data DMA transfer is not interrupted, and begins again with the previously established parameters.

Peripheral port register address set in the DMA_SxPAR register. DMA_SxMA0R register (and in the DMA_SxMA1R register in the case of a Double-buffer mode) set the memory address in the DMA. The data will be written to or read from this memory after the peripheral event. The figure in appendix describes the corresponding source (SRC) and destination (DST) addresses.

Implementing direct memory access is straightforward, once know how it works and how to configure your DMA controller. Each channel can be controlled using four registers: Memory address, peripheral address, number of data and configuration. And all channels

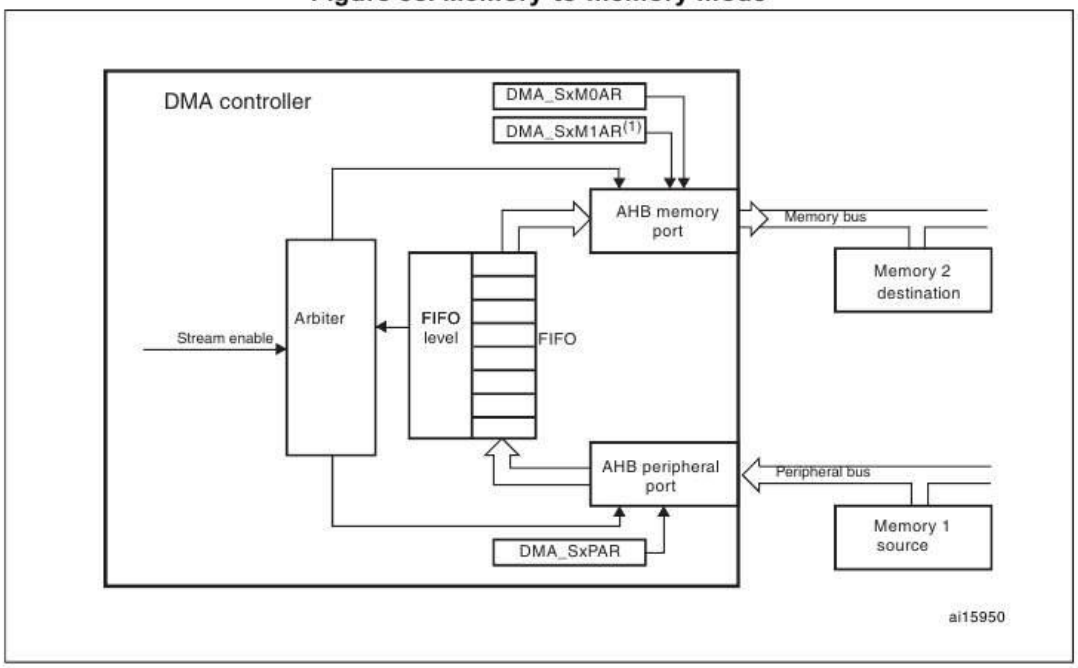
have two dedicated registers: DMA interrupt status register and interrupt flag clear register. Once set DMA takes care of memory address increment without disturbing CPU. DMA channels can generate three interrupts: transfer finished, half-finished and transfer error. The appendix will presented all the register had been used in this program. The next section will discuss the programming concept.

DMA transfer mode	Source	Destination	Flow controller	Circular mode	Transfer type	Direct mode	Double buffer mode
Peripheral-to-memory	AHB peripheral port	AHB memory port	DMA	possible	single	possible	possible
					burst	forbidden	
			Peripheral	forbidden	single	possible	forbidden
					burst	forbidden	
Memory-to-peripheral	AHB memory port	AHB peripheral port	DMA	possible	single	possible	possible
					burst	forbidden	
			Peripheral	forbidden	single	possible	forbidden
					burst	forbidden	
Memory-to-memory	AHB peripheral port	AHB memory port	DMA only	forbidden	single	forbidden	forbidden
					burst		

Figure 1.2 Possible DMA Configurations

2.2 Programming Concept

Figure 38. Memory-to-memory mode



1. For double-buffer mode.

As an example, let's write a simple program which transfers data between two arrays mean memory to memory transfer mode. To cook it more interesting, let's do the same task using DMA with different configuration. And so can compare the transfer information in both arrays. The comparison, both arrays will discuss in next section. The code has been shown in the appendix.

Configuring a channel consists of determining appropriate parameters through a DMA_reg structure:

```
typedef struct DMA_Type DMA_reg;
typedef struct{
      volatile uint32_t CR;
      volatile uint32_t NDTR;
      volatile uint32_t PAR;
      volatile uint32_t M0AR;
      volatile uint32_t M1AR;
      volatile uint32_t FCR;
}Stream_t;

struct DMA_Type{
  volatile uint32_t LISR;
  volatile uint32_t HISR;
  volatile uint32_t LIFCR;
  volatile uint32_t HIFCR;
  Stream_t  S0;
  Stream_t  S1;
  Stream_t  S2;
  Stream_t  S3;
  Stream_t  S4;
  Stream_t  S5;
  Stream_t  S6;
  Stream_t  S7;
};

#define dma1   ((DMA_reg*)0x40026000)
#define dma2   ((DMA_reg*)0x40026400)
```

Beginning of all creates two arrays: SRC_Const_Buffer and DST_Buffer. The SRC_Const_Buffer is used for source buffer address and the DST_Buffer is used for the destination buffer address. The number of data is determined by len variable which loaded with value 16. The following sequences configure DMA 7 stream in channel 0. Before configuring the DMA should un-reset enable clock. The function DMAUnresetEnableClock() was reset the DMA2 and turn on DMA2 clock. The function configDMAM2M() was configure DMA in memory to memory mode. Before configuring the DMA must disable the stream.

The following sequence explains configDMAM2M () subroutine. Take the DMA channel 0 (request) using CHSEL [2:0] in the DMA_S7CR register. Pick out the DMA transfer mode using DIR [1:0] in the DMA_S7CR. Select 32 bit memory and peripheral data size using MSIZE [1:0] and PSIZE [1:0]. In order to transfer more than one array data, then enable the memory and peripheral increment mode using MINC and PINC. The memory increment mode is a memory address pointer to incremented after each data transfer. Increment the index depends on the size of the data element, which is set in PSIZE or MSIZE. For example one byte in size, the pointer is incremented by one for a half-word (2 bytes), for the word (4 bytes). The priority mode assign very high. The final thing is enable the entire interrupt bit using TCIE, HTIE, TEIE and DMEIE.

The function DatasizeBurstFIFO passing the Memory Size, PeripheralSize, Burst and FIFO parameter. The choice for the data size to be transferred is 32-bit (1 word). This need to be done for both – peripheral and memory addresses so that the data can be transferred completely without changing the DMA_SxNDTR register. Let's fix it simple program, disable the FIFO and select the single memory burst transfer using MBURST.

The function DMA_memcpy8 passing pDstAddr, pSrcAddr and uSize parameter. The pDstAddr variable is the destination address. The pSrcAddr variable is the source address. The uSize is size of the number data. This function first checked the stream in enable and disable. If the stream is enabled, then disable it. Require a "source" a memory address where you're copying data from, a "destination" where you're copying to, some information about the size of the data chunks to be changed, and some sort of signal telling the DMA controller when the following byte is ready to be transfered. In one case the initialization structure is set, it's time to initialize the DMA stream, and turn it on. It should then be ready to pick up and react to DMA requests on the assigned channel.

Then load destination, source addresses and amount of data to be transmitted. After load these values using DMA_memcpy8 function. Later on this operation DMA is prepared to perform transfers. Any time DMA can be fired using enableDMA () subroutine. The function enableDMA () enables the DMA. The HAL_NVIC_EnableIRQ (DMA2_Stream7_IRQn) function was generated interrupt, then the DMA2_Stream7_IRQHandler function handler the interrupt.

The interrupt handler code looks generally like shown on figure 1. At one time it enters interrupt handler function, and then clears the DMA_HIFCR register with writing "1" to clear the interrupt flag to avoid re-entering the interrupt handler function. It's important to ensure the interrupt clear has actually happened before you exit your interrupt code. The easy way to do this is to clear the interrupt at the very top of your interrupt handler, before you do anything else. Inside the interrupt handler function can observe the status register.

```
void DMA2_Stream7_IRQHandler(void)
{
    uint32_t statusHISR;

    dma2->S7.CR &= ~(15 << 1);              // clear interrupt enable bit
    dma2->LISR &= 0x00000000;               // clear the interrupt status & flag
    dma2->HISR &= 0x00000000;
    dma2->HIFCR = (uint32_t)0x0f400000;     // clear any pending (old) DMA2 Stream 7 interrupts

    // start transfer to another memory
    dma2->S7.CR &= ~(0xffff);               // clear configure register

    configDMAM2M();                         // everything setup but not enabled
    unsigned int len = 9;
    DMA_memcpy8( ThirdTarget_Buffer, DST_Buffer, len);
    DatasizeBurstFIFO (DMA_MemoryDataSize_Word, DMA_PeripheralDataSize_Word, DMA_MemoryBurst_Single, Full_FIFO);
    enableDMA();                            // everything setup and enabled

    satusHISR = dma2->HISR;
    int Thirdptr2DST = ThirdTarget_Buffer[0];
    int Secondptr2SRC = DST_Buffer[0];
}
```

Figure 1 DMA2_Stream7_IRQHandler function

Chapter 3 Result and Discussion

3.1 Discuss about the difference configuration of PSIZE and MSIZE

MSIZE (Memory data size) and PSIZE (Peripheral data size) can be select difference data size to be transferred. When PSIZE and MSIZE are not equal, the DMA performs some data alignments as described in Table 47. The memory data size is 16 bit and the peripheral data size is 8 bit then DMA would cycle two times in 8 bit chunks as describe in figure 2. In order to get the full transfer data use the equation 1 to find out the number of times.

DMA_SxNDTR = Multiple of ((Mburst beat) × (M size) / (Psize)) equation 1

The table 1 had shown the select the difference memory data size and peripheral data size in DataSizeBurstFIFO subroutine and varying the number of data register (DMA_SxNDTR). The figure 4 had shown the DataSizeBurstFIFO subroutine.

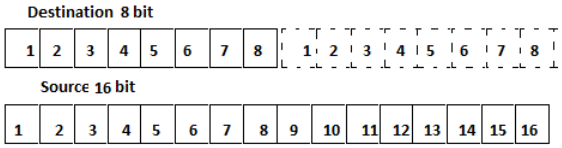


Figure 2 MSIZE is 16 bit and PSIZE data size is 8 bit

Table 47. Restriction on NDT versus PSIZE and MSIZE

PSIZE[1:0] of DMA_SxCR	MSIZE[1:0] of DMA_SxCR	NDT[15:0] of DMA_SxNDTR
00 (8-bit)	01 (16-bit)	must be a multiple of 2
00 (8-bit)	10 (32-bit)	must be a multiple of 4
01 (16-bit)	10 (32-bit)	must be a multiple of 2

```
uint32_t SRC_Const_Buffer[]=    {0x12345678, 0xdeadbeef, 0xc0ffee45, 0x0badface, 0x789abcde, 0x2a3b4c5d, 0x6b3cde42,0xcade2917,0xbcd6745a,
                                0xfadd1257, 0xdef37846, 0x12785abc,0x92837465, 0xa7b80919, 0x4726fabe, 0xbd87af91};
uint32_t DST_Buffer[]=          {0x00000000, 0x11111111, 0x22222222, 0x333333333, 0x4444444444,0x55555555, 0x66666666,0x77777777,0x888888888,
                                0x999999999, 0x00001111,0x22223333, 0x44445555, 0x66667777, 0x999991111, 0x55552222};
uint32_t ThirdTarget_Buffer[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
```

Figure 3 arrays had been created in this program

The figure 3 shown the arrays had been made in this program as SRC_Const_Buffer, DST_Buffer, ThirdTarget_Buffer. The ThirdTarget_Buffer had been used in interrupt and will discuss in next section.

```
void DatasizeBurstFIFO (uint32_t MemorySize, uint32_t PeripheralSize, uint32_t Burst, uint32_t FIFO){

    //source and destination data size word=32bit
        dma2->S7.CR &= ~(2 << 13);
        dma2->S7.CR |= (MemorySize << 13);

        dma2->S7.CR &= ~(2 << 11);
        dma2->S7.CR |= (PeripheralSize << 11);

        //FIFO
        dma2->S7.FCR &= (11 << FTH);
        dma2->S7.FCR |= (FIFO << FTH);

        dma2->S7.CR &= ~(2 << 23);
        dma2->S7.CR |= (Burst << 23);
}
```

Figure 4 DataSizeBurstFIFO subroutine

Table 1 Configure with different Memory data size and Peripheral data size

	Result	**Configuration the MSIZE and PSIZE in DataSizeBurstFIFO subroutine**	
1.	Name / Value DST_Buffer 0x20000044 DST_Buffer[0] 0x12345678 DST_Buffer[1] 0xdeadbeef DST_Buffer[2] 0x00000045 DST_Buffer[3] 0x33333333 DST_Buffer[4] 0x44444444 DST_Buffer[5] 0x55555555 DST_Buffer[6] 0x66666666 DST_Buffer[7] 0x77777777 DST_Buffer[8] 0x88888888 DST_Buffer[9] 0x99999999 DST_Buffer[10] 0x00001111 DST_Buffer[11] 0x22223333 DST_Buffer[12] 0x44445555 DST_Buffer[13] 0x66667777 DST_Buffer[14] 0x99991111 DST_Buffer[15] 0x55552222 len 9	`DatasizeBurstFIFO (DMA_MemoryDataSize_Word, DMA_PeripheralDataSize_byte, DMA_MemoryBurst_Single, Full_FIFO);`	DMA_SxNDTR = 9 Transfer 9 byte to the destination buffer.
2.	Name / Value DST_Buffer 0x20000044 DST_Buffer[0] 0x12345678 DST_Buffer[1] 0xdeadbeef DST_Buffer[2] 0x0000ee45 DST_Buffer[3] 0x33333333 DST_Buffer[4] 0x44444444 DST_Buffer[5] 0x55555555 DST_Buffer[6] 0x66666666 DST_Buffer[7] 0x77777777 DST_Buffer[8] 0x88888888 DST_Buffer[9] 0x99999999 DST_Buffer[10] 0x00001111 DST_Buffer[11] 0x22223333 DST_Buffer[12] 0x44445555 DST_Buffer[13] 0x66667777 DST_Buffer[14] 0x99991111 DST_Buffer[15] 0x55552222 len 5	`DatasizeBurstFIFO (DMA_MemoryDataSize_Word, DMA_PeripheralDataSize_halfword, DMA_MemoryBurst_Single, Full_FIFO);`	DMA_SxNDTR = 5 Transfer 5 halfword to the destination buffer.
3.	Name / Value DST_Buffer 0x20000044 DST_Buffer[0] 0x12345678 DST_Buffer[1] 0xdeadbeef DST_Buffer[2] 0xc0ffee45 DST_Buffer[3] 0x33333333 DST_Buffer[4] 0x44444444 DST_Buffer[5] 0x55555555 DST_Buffer[6] 0x66666666 DST_Buffer[7] 0x77777777 DST_Buffer[8] 0x88888888 DST_Buffer[9] 0x99999999 DST_Buffer[10] 0x00001111 DST_Buffer[11] 0x22223333 DST_Buffer[12] 0x44445555 DST_Buffer[13] 0x66667777 DST_Buffer[14] 0x99991111 DST_Buffer[15] 0x55552222 len 3	`DatasizeBurstFIFO (DMA_MemoryDataSize_Word, DMA_PeripheralDataSize_Word, DMA_MemoryBurst_Single, Full_FIFO);`	DMA_SxNDTR = 3 Transfer 3 words to the destination buffer.

All the result from the table 1 set the FIFO to full FIFO, MSIZE is set to word (32 bits) and the burst size is single burst size. From the result 1 the data size (PSIZE) is set to byte (8 bits) and data width (DMA_NDTR) is loaded with value 9.The result 1 had been shown the 9 bytes data had been success transfer to the destination. The result 2 the PSIZE is set to half-word (16 bits) and the data width is loaded with value 5. The result shown the 5 half-word data had been success transfer to the destination. The result the PSIZE and MSIZE is set to word (32 bit) and data width is loaded with value 3. The result shown the 3 word had been success transfer to the destination.

Section 3.2 Transfer to another buffer done in interrupts

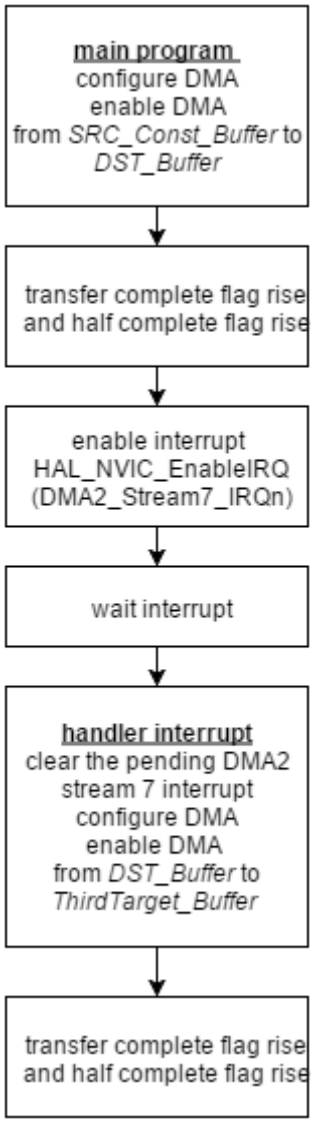


Figure 5 flow chart of DMA program with interrupt

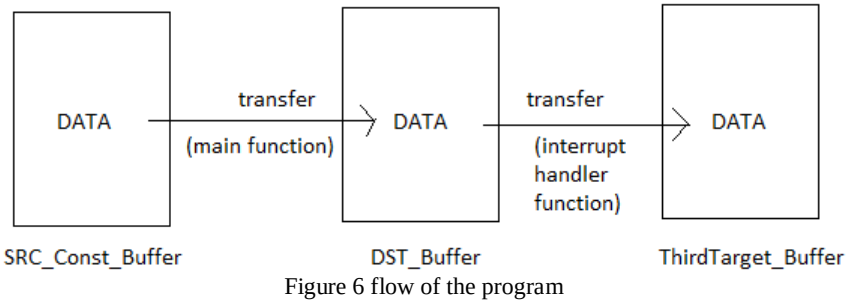


Figure 6 flow of the program

The DMA transfer the data through the interrupt. The flow chart of the DMA transfers the data through the interrupt as shown in figure 5. At the beginning the data is transferred from SRC_Const_Buffer to DST_Buffer through the main program then clear the pending interrupt flag.

After clearing the interrupt flag and waiting the interrupt. The data transfer from Dst_buffer to ThirdTarget_Buffer through the interrupt as describe in figure 6. The figure 7 shown the data had been success transfer to the ThirdTarget_Buffer.

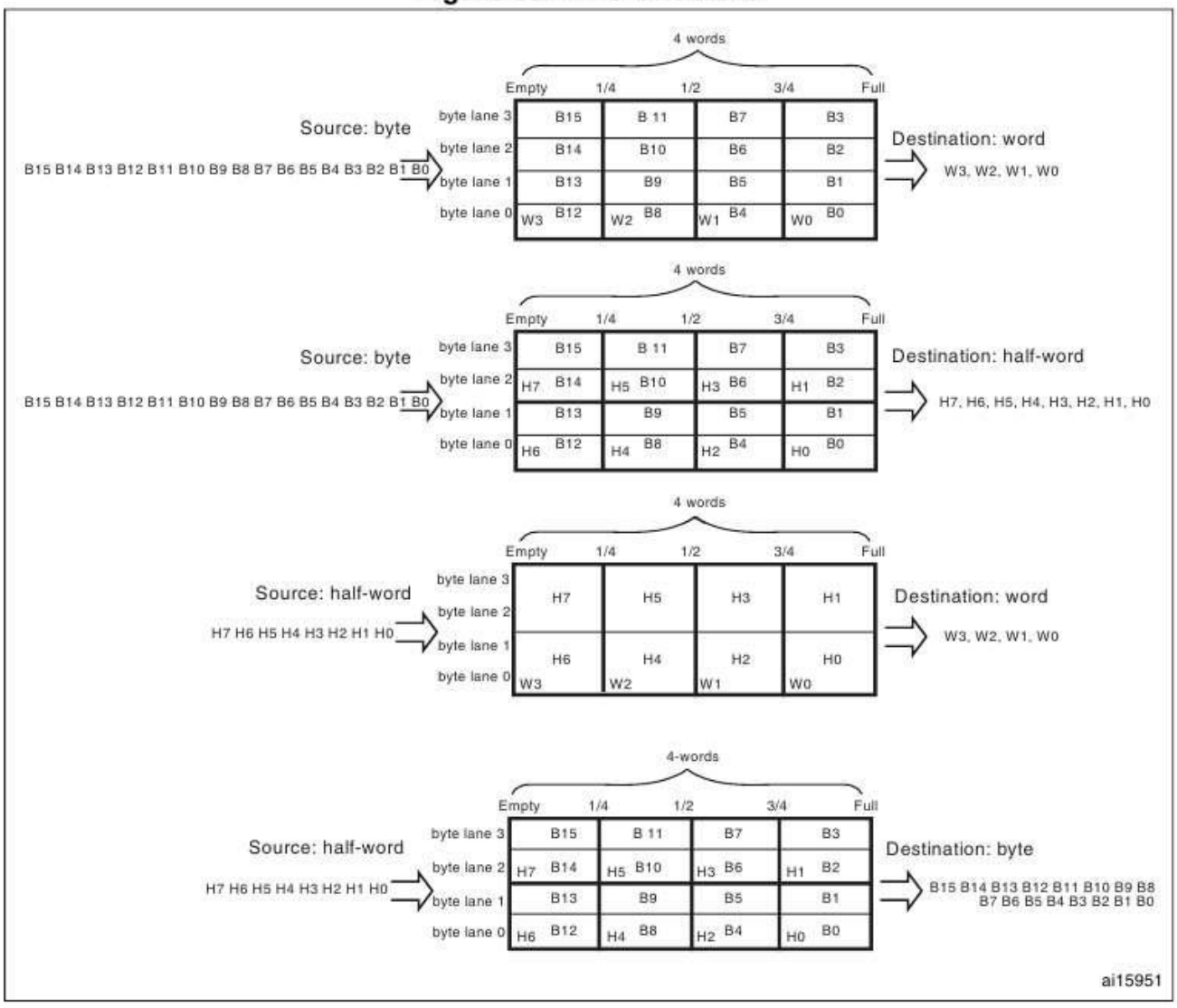
Name	Value
ThirdTarget_Buffer	0x200004dc
ThirdTarget_Buffer[0]	0x12345678
ThirdTarget_Buffer[1]	0xdeadbeef
ThirdTarget_Buffer[2]	0xc0ffee45
ThirdTarget_Buffer[3]	0x0badface
ThirdTarget_Buffer[4]	0x789abcde
ThirdTarget_Buffer[5]	0x2a3b4c5d
ThirdTarget_Buffer[6]	0x6b3cde42
ThirdTarget_Buffer[7]	0xcade2917
ThirdTarget_Buffer[8]	0xbcd6745a
ThirdTarget_Buffer[9]	0
ThirdTarget_Buffer[10]	0
ThirdTarget_Buffer[11]	0
ThirdTarget_Buffer[12]	0
ThirdTarget_Buffer[13]	0
satusHISR2	0x0c000000
len	9

Figure 7 after transfer data to the ThirdTarget_buffer

Section 3.3 Discuss the difference burst size and FIFO threshold level

FIFO is used for temporary storage of data coming from the source and destination to the next. Each stream has a 16-byte FIFO buffer having programmable threshold (1/4, 1/2, 3/4 and full). To enable transmission after filling up the threshold level should be set a bit in a register DMDIS DMA_SxFCR. FIFO structure differs depending on the width of the data source and destination. The structure of the FIFO buffer as describe in figure 39. The selection the threshold of FIFO (bits FTH [1: 0] register DMA_SxFCR) and portion size data memory (MBURST [1: 0] register DMA_SxCR) must be take care.

Figure 39. FIFO structure

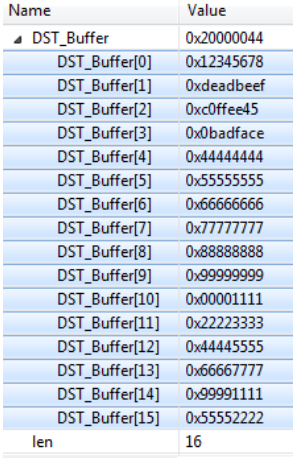
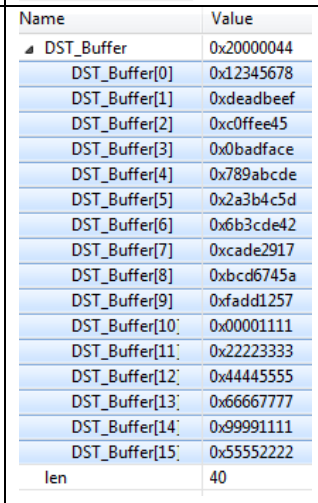
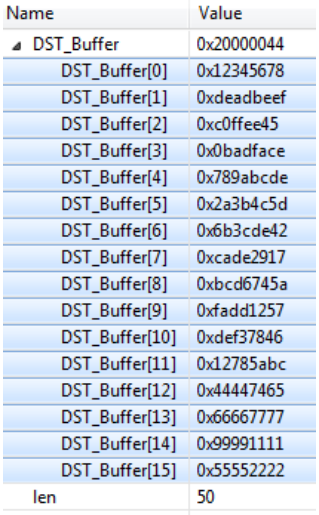


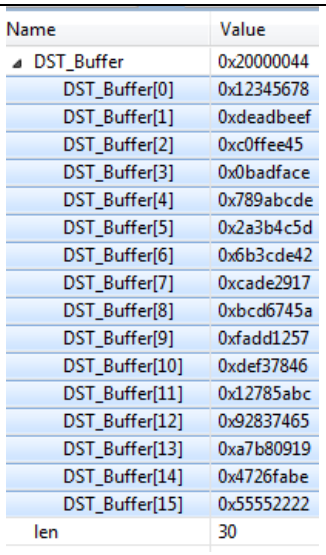
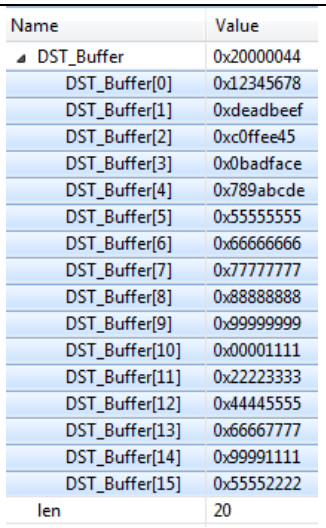
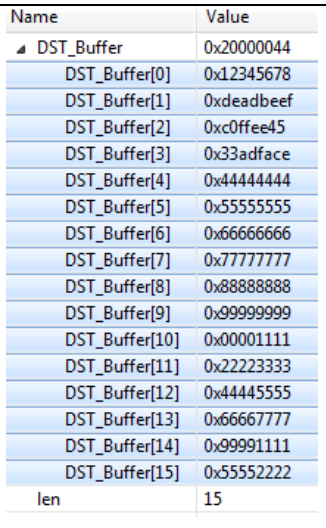
The burst mode is available only when FIFO mode is enabled. Burst mode allows configuring the amount of data to be transferred without CPU or DMA interrupt. The burst mode is set, the FIFO threshold should be compatible with burst size as describe in table 48.

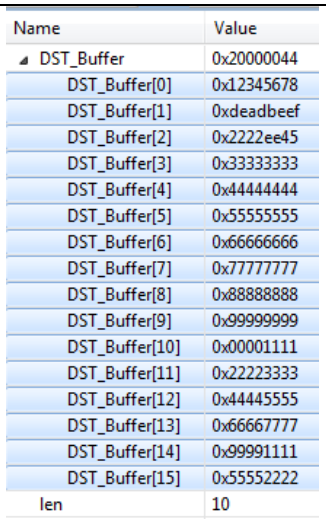
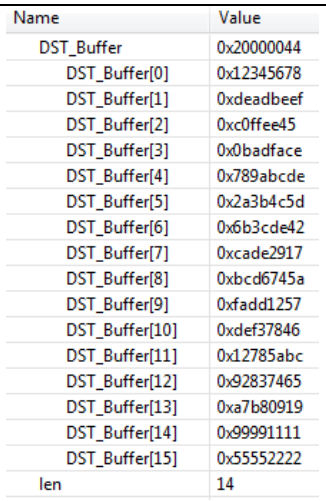
Table 48. FIFO threshold configurations

MSIZE	FIFO level	MBURST = INCR4	MBURST = INCR8	MBURST = INCR16
Byte	1/4	1 burst of 4 beats	forbidden	forbidden
	1/2	2 bursts of 4 beats	1 burst of 8 beats	
	3/4	3 bursts of 4 beats	forbidden	
	Full	4 bursts of 4 beats	2 bursts of 8 beats	1 burst of 16 beats
Half-word	1/4	forbidden	forbidden	forbidden
	1/2	1 burst of 4 beats		
	3/4	forbidden		
	Full	2 bursts of 4 beats	1 burst of 8 beats	
Word	1/4	forbidden	forbidden	
	1/2			
	3/4			
	Full	1 burst of 4 beats		

The table 2 had shown the select the difference burst size and FIFO threshold level with result.

	Result	Configuration	
1		`DatasizeBurstFIFO (DMA_MemoryDataSize_byte, DMA_PeripheralDataSize_byte, DMA_MemoryBurst_Incr4, Full_FIFO);`	Transfer 4 words to the destination
2		`DatasizeBurstFIFO (DMA_MemoryDataSize_byte, DMA_PeripheralDataSize_byte, DMA_MemoryBurst_Incr8, Full_FIFO);`	Transfer 10 words to the destination
3		`DatasizeBurstFIFO (DMA_MemoryDataSize_byte, DMA_PeripheralDataSize_byte, DMA_MemoryBurst_Incr16, Full_FIFO);`	Transfer 50 byte to the destination

4		`DatasizeBurstFIFO (DMA_MemoryDataSize_halfword, DMA_PeripheralDataSize_halfword, DMA_MemoryBurst_Incr4, Full_FIFO);`	Transfer 15 word to the destination
5		`DatasizeBurstFIFO (DMA_MemoryDataSize_byte, DMA_PeripheralDataSize_byte, DMA_MemoryBurst_Incr4, Quater_full_FIFO);`	Transfer 5 words to the destination
6		`DatasizeBurstFIFO (DMA_MemoryDataSize_byte, DMA_PeripheralDataSize_byte, DMA_MemoryBurst_Incr4, ThreeperFour_full_FIFO);`	Transfer 15 byte to the destination.

7		`DatasizeBurstFIFO (DMA_MemoryDataSize_byte, DMA_PeripheralDataSize_byte, DMA_MemoryBurst_Incr4, Half_full_FIFO);`	Transfer 10 words to the destination.
8		`DatasizeBurstFIFO (DMA_MemoryDataSize_Word, DMA_PeripheralDataSize_Word, DMA_MemoryBurst_Incr4, Full_FIFO);`	Transfer 14 words to the destination.

From the table 2 the result 1, 2, 3, 4 and 8 are used the full FIFO threshold level with the varying burst size. The result 1 shown the burst size is set to 4, MSIZE is byte (4 bursts of 4 beats) and the data width (NDTR) is 16. The result 1 shown the 4 words (16 byte) transfers are performed from DMA FIFO to destination as describe in figure 8. The calculation shown:

$$\text{DMA\_NDTR} \,/_{\text{multiple of word}} = \text{number of word transfer data to the destination}$$

Example calculation for result 1: 16/4 = 4 words transfer to the destination.

The result 2 set the burst size is 8, MSIZE is byte (2 burst of 8 beats) and the data width is 40. The result 2 shown the 10 words (40 byte) transfers are performed from DMA FIFO to destination. The result 3 had shown the 50 bytes data transfer to the destination. The result 4 had shown the 15 words data transfers to the destination. The following the result shown the difference structure FIFO.

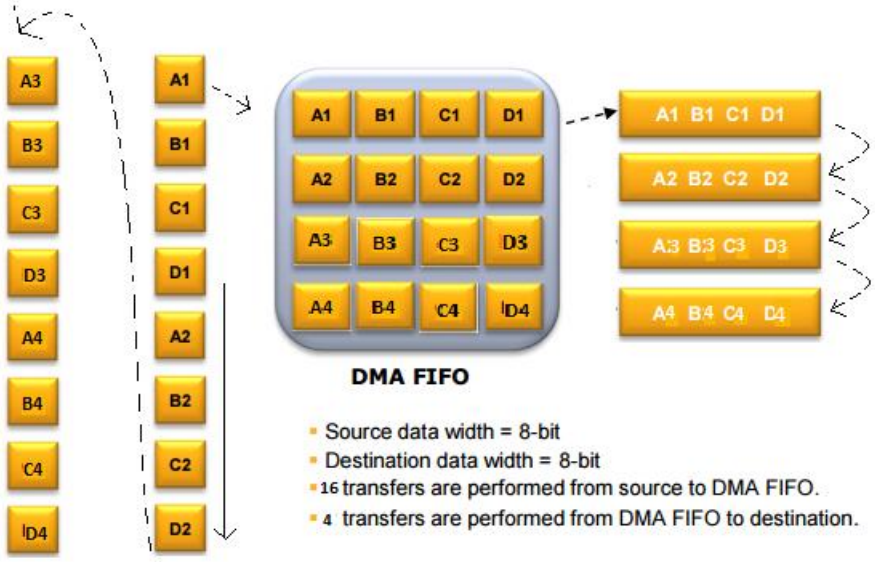


Figure 8 full FIFO structure buffer

Chapter 4 Reference

- lanmanck (24/12/2009) Several concepts dma use. burst, burst size, length [Accessed from: http://blog.csdn.net/lanmanck/article/details/5069618].
- StackOverflow (2015) STM32 DMA (concurrent stream, FIFO, burst mode,double buffer) [Online] Available from:http://stackoverflow.com/questions/24051720/stm32-dmaconcurrent-stream-fifo-burst-mode-double-buffer [Accessed: 30Th December 2015].

Chapter 5 Appendix

1. Request mapping

Table 43. DMA1 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX		SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX		SPI3_TX
Channel 1	I2C1_RX		TIM7_UP		TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1		I2S3_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S3_EXT_TX	TIM4_UP	TIM4_CH3
Channel 3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5	UART8_TX(1)	UART7_TX(1)	TIM3_CH4 TIM3_UP	UART7_RX(1)	TIM3_CH1 TIM3_TRIG	TIM3_CH2	UART8_RX(1)	TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2		TIM5_UP	
Channel 7		TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX

1. These requests are available on STM32F42xxx and STM32F43xxx only.

Table 44. DMA2 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1	SAI1_A(1)	TIM8_CH1 TIM8_CH2 TIM8_CH3	SAI1_A(1)	ADC1	SAI1_B(1)	TIM1_CH1 TIM1_CH2 TIM1_CH3	
Channel 1		DCMI	ADC2	ADC2	SAI1_B(1)	SPI6_TX(1)	SPI6_RX(1)	DCMI
Channel 2	ADC3	ADC3		SPI5_RX(1)	SPI5_TX(1)	CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX		SPI1_RX	SPI1_TX		SPI1_TX		
Channel 4	SPI4_RX(1)	SPI4_TX(1)	USART1_RX	SDIO		USART1_RX	SDIO	USART1_TX
Channel 5		USART6_RX	USART6_RX	SPI4_RX(1)	SPI4_TX(1)		USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
Channel 7		TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	SPI5_RX(1)	SPI5_TX(1)	TIM8_CH4 TIM8_TRIG TIM8_COM

1. These requests are available on STM32F42xxx and STM32F43xxx.

2. DMA Register

10.5.1 DMA low interrupt status register (DMA_LISR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				TCIF3	HTIF3	TEIF3	DMEIF3	Reserved	FEIF3	TCIF2	HTIF2	TEIF2	DMEIF2	Reserved	FEIF2
r	r	r	r	r	r	r	r		r	r	r	r	r		r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				TCIF1	HTIF1	TEIF1	DMEIF1	Reserved	FEIF1	TCIF0	HTIF0	TEIF0	DMEIF0	Reserved	FEIF0
r	r	r	r	r	r	r	r		r	r	r	r	r		r

10.5.2 DMA high interrupt status register (DMA_HISR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				TCIF7	HTIF7	TEIF7	DMEIF7	Reserved	FEIF7	TCIF6	HTIF6	TEIF6	DMEIF6	Reserved	FEIF6
				r	r	r	r		r	r	r	r	r		r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				TCIF5	HTIF5	TEIF5	DMEIF5	Reserved	FEIF5	TCIF4	HTIF4	TEIF4	DMEIF4	Reserved	FEIF4
				r	r	r	r		r	r	r	r	r		r

10.5.5 DMA stream x configuration register (DMA_SxCR) (x = 0..7)

This register is used to configure the concerned stream.

Address offset: 0x10 + 0x18 × *stream number*

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				CHSEL[2:0]			MBURST [1:0]		PBURST[1:0]		Reserved	CT	DBM or reserved	PL[1:0]	
				rw	rw	rw	rw	rw	rw	rw		rw	rw or r	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PINCOS	MSIZE[1:0]		PSIZE[1:0]		MINC	PINC	CIRC	DIR[1:0]		PFCTRL	TCIE	HTIE	TEIE	DMEIE	EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

10.5.6 DMA stream x number of data register (DMA_SxNDTR) (x = 0..7)

Address offset: 0x14 + 0x18 × *stream number*

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NDT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

10.5.10 DMA stream x FIFO control register (DMA_SxFCR) (x = 0..7)

Address offset: 0x24 + 0x24 × *stream number*

Reset value: 0x0000 0021

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								FEIE	Reserved	FS[2:0]			DMDIS	FTH[1:0]	
								rw		r	r	r	rw	rw	rw