Chapter 1 Introduction

* 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, across the entire address range. 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 contained 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 having a peripheral device control a processor's memory bus directly. 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 processor. Thus DMA enables more efficient use of interrupts, increases data throughput, and potentially reduces hardware costs by eliminating the need for peripheral-specific FIFO buffers.

The DMA controller asserts a DMA request signal to the CPU. The DMA controller requests the permission to use the AHB bus. The CPU complete 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. Once the transfers complete DMA stop driving the bus and assert the DMA request the signal. The CPU then remove its DMA acknowledge signal and resume control of the bus

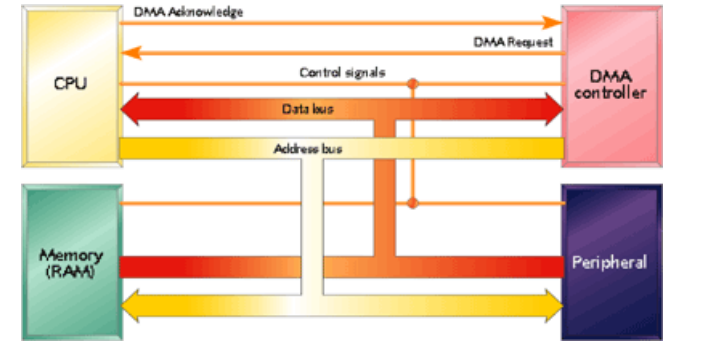


Figure 1.1 DMA controllers the processor bus. access from [*http://www.embedded.com/electronics-blogs/beginner-s-corner/4024879/Introduction-to-direct-memory-access*](http://www.embedded.com/electronics-blogs/beginner-s-corner/4024879/Introduction-to-direct-memory-access)

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 total number of data transfer. This value is decremented after each peripheral event or each beat of the burst. 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. All the possible DMA configuration had been shown in figure 1.2.

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 next section will discuss the programming concept.

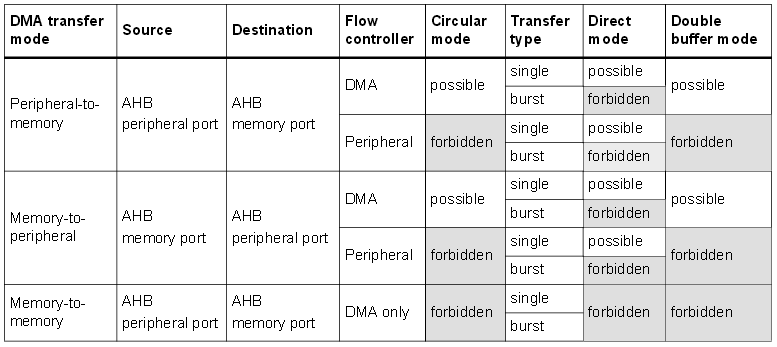


Figure 1.2 Possible DMA Configurations

2.2 Programming Concept

As example lets write a simple program which transfers data between two arrays mean memory to memory transfer mode. To make it more interesting let’s do same task using DMA with difference configuration. Then can compare the transfer data in both arrays. The comparison both arrays will discuss in nest section. The code has been shown in appendix.

Configuring a channel consists of setting appropriate parameters through a DMA\_reg structure:



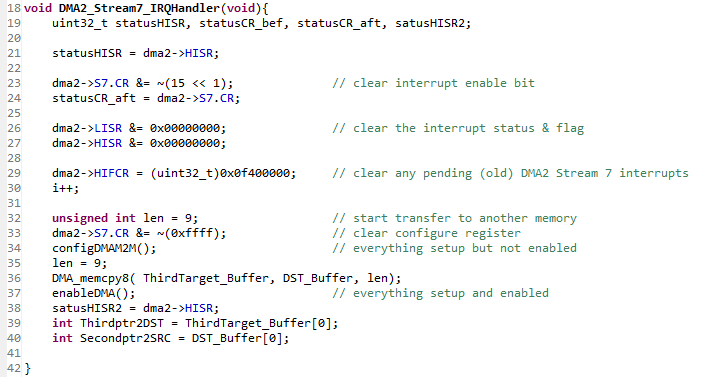
First of all create two arrays: SRC\_Const\_Buffer and DST\_Buffer. Size of length is determined by len variable which loaded with value 9. The following sequences configure DMA 7 stream in channel 0. Before configure the DMA should un-reset enable clock. The function DMAUnresetEnableClock() was reset the DMA2 and turn on DMA2 clock. The function configDMAM2M() was configure DMA in memory to memory mode. Before configure the DMA must disable the stream.

The following sequence explain configDMAM2M() function. Select the DMA channel 0 (request) using CHSEL[2:0] in the DMA\_S7CR register. Select 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]. Then we select data size to be transferred (32-bit word). This need to be done for both – peripheral and memory addresses. 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. The incremented size based on the DMA\_NDTR register. The priority mode assign very high. Let’s make it simple program, disable the FIFO and select the single memory burst transfer using MBURST. The last thing is enable the entire interrupt bit using TCIE, HTIE, TEIE and DMEIE.

The function DMA\_memcpy8 passing pDstAddr, pSrcAddr and uSize parameter. The pDstAddr variable is the destination address. The pSrcAddr variable is source address. The uSize is size of the number data. This function first checked the stream in enable and disable. If stream is enabled then disable it. Need 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 transferred, and some sort of signal telling the DMA controller when the next byte is ready to be transferred. Once the initialization structure is defined, it’s time to initialize the DMA stream, and turn it on. It should then be ready to receive and react to DMA requests on the specified channel.

Then load destination, source addresses and amount of data to be sent. After load these values using DMA\_memcpy8 function. After this operation DMA is prepared to do transfers. Any time DMA can be fired usingenableDMA() function. The function enableDMA() is enable the DMA. The HAL\_NVIC\_EnableIRQ (DMA2\_Stream7\_IRQn) function was genearated interrupt then DMA2\_Stream7\_IRQHandler function handler the interrupt.

The interrupt handler code looks general like shown on figure. Once it enter interrupt handler function then clear the DMA\_HIFCR register with write "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.



Chapter 3 Result and Discussion

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.

*If one of memory sizes would be different, say source 16-bit and destination 8- bit – then DMA would cycle two times in 8 bit chunks as describe in figure a . In order to get the full transfer data use equation 1 to find number of times.*

DMA\_SxNDTR = Multiple of ((Mburst beat) × (Msize)/(Psize)) equation 1

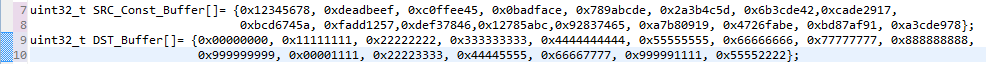




When select the difference memory data size and peripheral size without changing the number of data register (DMA\_SxNDTR) as describe in figure

Memory address pointer is incremented after each data transfer (increment is done according to MSIZE)

Peripheral address pointer is incremented after each data transfer (increment is done according to PSIZE



|  |  |
| --- | --- |
|  | 1 word = 4 x byte  9 byte = 2.25 words |
|  | 1 word = 2 x halfword  5 halfword = 2.5 word |
|  | 3 word |

|  |  |  |
| --- | --- | --- |
|  |  | Byte  Burst incr4  Full fifo  Len 16  4 word |
|  |  | Byte  Incr8  Full fifo  Len 40  Transmit 10 word |
|  |  | Byte  Incr16  Full fifo  Len 50  11.5 word |
|  |  | Halfword  Incr4  Full fifo  30  15 word |
|  |  | Byte  Incr4  ¼ fifo  Len 20  5 word |
|  |  | Byte  Incr4  ¾ fifo  Len = 15  3.75 word |
|  |  | Byte  Inr4  ½ fifo  Len 10  2.5 word |
|  |  | Word  Incr4  Full fifo  Len 14  14 word |
|  |  | Byte  Incr16  Full fifo  Len 21  \*Word 4.125 |

Appendix

request mapping figure 1.3

register figure .15

