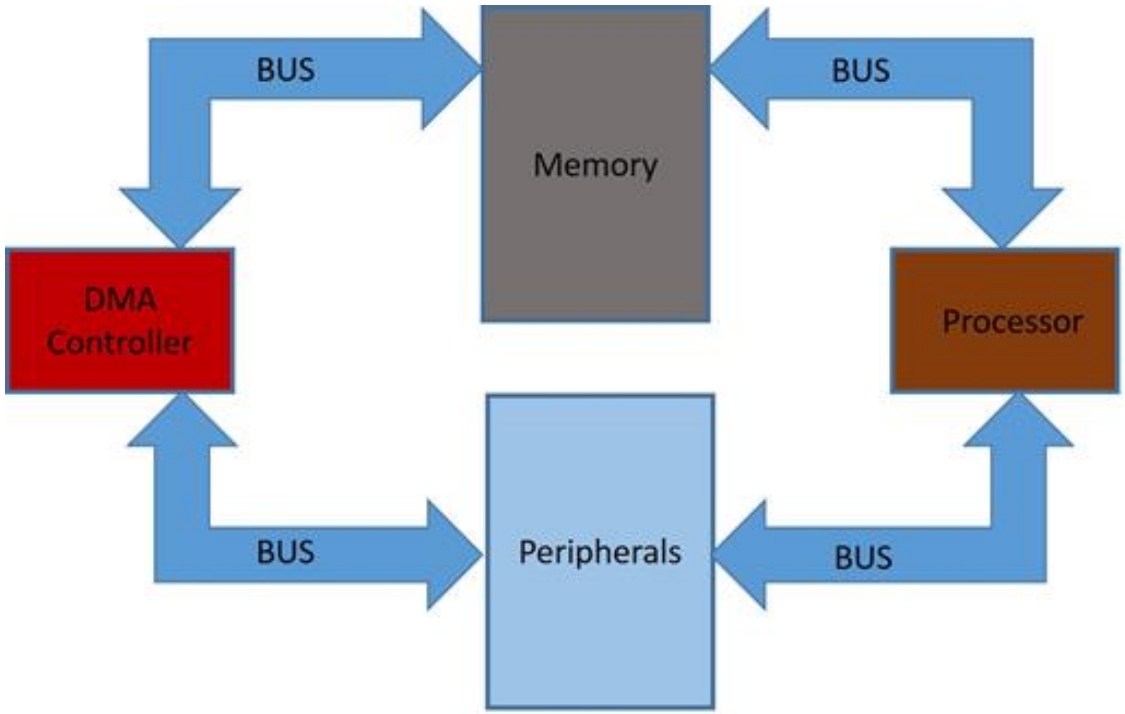
Embedded Systems Interfacing

Lecture one

Digital Input Output Part 1

Direct Memory Access (DMA)

DMA, also known as "Direct Transfer Controller", is a peripheral that is used to provide high-speed data transfer between peripherals and memory as well as memory to memory. Data can be quickly moved by DMA without any CPU actions, this keeps CPU resources free for other operations .



Direct Memory Access

The processor has a role in this process, it programs the DMA controller to set the required data then it orders the DMA to start. Once the DMA starts, the processor is free to do other tasks and it will not be involved by any means in the process of data transfer.

When the process of data transfer is complete, the DMA controller fires an interrupt to notify the processor, then the processor handles this event.

This feature is useful at any time when the CPU cannot keep up with the rate of data transfer, or when the CPU needs to perform work while waiting for a relatively slow I/O data transfer.

How DMA works

1. Some event (such as an incoming data-available signal from a UART) notifies a separate device called the DMA controller that data needs to be transferred to memory.

2. The DMA controller asserts a DMA request signal to the CPU, asking its permission to use the bus.

3. The CPU completes its current bus activity, stops driving the bus, and returns a DMA acknowledge signal to the DMA controller.

4. The DMA controller then sends an acknowledge signal to the peripheral, then it reads and writes one or more memory bytes, driving the address, data, and control signals as if it were itself the CPU.

5. When the transfer is complete, the DMA controller stops driving the bus and deasserts the DMA request signal, at this moment the DMA also can fire an interrupt if enabled.

6. The CPU can then remove its DMA acknowledge signal and resume control of the bus.

DMA Configuration

Our microcontroller stm32f103 has one DMA controller, called DMA1, this controller has 7 channels, each channel can be programmed to execute a different transfer. To program a channel to execute a specific data transfer, some configurations should be set.

Please remember to set all your configurations while the channel is disabled.

1- Source memory address :-

This is the memory that holds the data required to be transferred, so the transfer starts by reading data from this memory.

This memory can be a peripheral memory, hence called "peripheral", or it can be any other memory like SRAM or flash, hence called only "memory".

2- Destination memory address :-

This is the target memory which the data will be transferred to, so the transfer finishes after storing data to this memory.
The same as the source memory, this memory can be a peripheral memory or any other memory.

3- Destination memory address :-

Our microcontroller doesn't have registers for source memory address or destination memory address, however it has registers called Channel Memory Address Registers (DMA_CMARx) and Channel Peripheral Address Registers (DMA_CPARx), so when the case is transferring data from memory to peripheral or from peripheral to memory, the peripheral memory address is set in CPAR and the memory address is set in CMAR, then the direction is set by setting or clearing a bit called "DIR" in the Channel Configuration Register (DMA_CCRx), if cleared then the direction is from peripheral to memory, if set then the direction is from memory to peripheral.
If the case is transferring data from memory to memory then 2 things are required:
Choosing the memory-to-memory mode by setting the MEM2MEM bit in the CCR register.
Set one memory address in the CPAR register, and the other memory address in the CMAR register, then decide the value of the DIR bit by checking whether the CPAR holds the source address or the destination address, for example if it holds the source memory address the DIR value should be 0.

DMA Configuration

4- Data Size :-

This configuration determines the size of data read from source memory in one transfer operation, and the size of data stored to destination memory in one transfer operation. Three configurations are available: 8 bits, 16 bits, 32 bits.

Source port width	Destination port width	Number of data items to transfer (NDT)	Source content: address / data	Transfer operations	Destination content: address / data
8	8	4	@0x0 / B0 @0x1 / B1 @0x2 / B2 @0x3 / B3	1: READ B0[7:0] @0x0 then WRITE B0[7:0] @0x0 2: READ B1[7:0] @0x1 then WRITE B1[7:0] @0x1 3: READ B2[7:0] @0x2 then WRITE B2[7:0] @0x2 4: READ B3[7:0] @0x3 then WRITE B3[7:0] @0x3	@0x0 / B0 @0x1 / B1 @0x2 / B2 @0x3 / B3
8	16	4	@0x0 / B0 @0x1 / B1 @0x2 / B2 @0x3 / B3	1: READ B0[7:0] @0x0 then WRITE 00B0[15:0] @0x0 2: READ B1[7:0] @0x1 then WRITE 00B1[15:0] @0x2 3: READ B3[7:0] @0x2 then WRITE 00B2[15:0] @0x4 4: READ B4[7:0] @0x3 then WRITE 00B3[15:0] @0x6	@0x0 / 00B0 @0x2 / 00B1 @0x4 / 00B2 @0x6 / 00B3
8	32	4	@0x0 / B0 @0x1 / B1 @0x2 / B2 @0x3 / B3	1: READ B0[7:0] @0x0 then WRITE 000000B0[31:0] @0x0 2: READ B1[7:0] @0x1 then WRITE 000000B1[31:0] @0x4 3: READ B3[7:0] @0x2 then WRITE 000000B2[31:0] @0x8 4: READ B4[7:0] @0x3 then WRITE 000000B3[31:0] @0xC	@0x0 / 000000B0 @0x4 / 000000B1 @0x8 / 000000B2 @0xC / 000000B3

DMA Configuration

4- Data Size :-

to configure the data size of the source memory and the destination memory we have the MSIZE bits for setting the memory size and the PSIZE bits for the peripheral size. MSIZE and PSIZE are 2 bits each, it has 3 available options as we mentioned before: 8 bits, 16 bits, 32 bits. The 4th value of the 2 bits is reserved.

Source port width	Desti-nation port width	Number of data items to transfer (NDT)	Source content: address / data	Transfer operations	Destination content: address / data
32	8	4	@0x0 / B3B2B1B0 @0x4 / B7B6B5B4 @0x8 / BBBAB9B8 @0xC / BFBEBDBC	1: READ B3B2B1B0[31:0] @0x0 then WRITE B0[7:0] @0x0 2: READ B7B6B5B4[31:0] @0x4 then WRITE B4[7:0] @0x1 3: READ BBBAB9B8[31:0] @0x8 then WRITE B8[7:0] @0x2 4: READ BFBEBDBC[31:0] @0xC then WRITE BC[7:0] @0x3	@0x0 / B0 @0x1 / B4 @0x2 / B8 @0x3 / BC
32	16	4	@0x0 / B3B2B1B0 @0x4 / B7B6B5B4 @0x8 / BBBAB9B8 @0xC / BFBEBDBC	1: READ B3B2B1B0[31:0] @0x0 then WRITE B1B0[7:0] @0x0 2: READ B7B6B5B4[31:0] @0x4 then WRITE B5B4[7:0] @0x1 3: READ BBBAB9B8[31:0] @0x8 then WRITE B9B8[7:0] @0x2 4: READ BFBEBDBC[31:0] @0xC then WRITE BDBC[7:0] @0x3	@0x0 / B1B0 @0x2 / B5B4 @0x4 / B9B8 @0x6 / BDBC
32	32	4	@0x0 / B3B2B1B0 @0x4 / B7B6B5B4 @0x8 / BBBAB9B8 @0xC / BFBEBDBC	1: READ B3B2B1B0[31:0] @0x0 then WRITE B3B2B1B0[31:0] @0x0 2: READ B7B6B5B4[31:0] @0x4 then WRITE B7B6B5B4[31:0] @0x4 3: READ BBBAB9B8[31:0] @0x8 then WRITE BBBAB9B8[31:0] @0x8 4: READ BFBEBDBC[31:0] @0xC then WRITE BFBEBDBC[31:0] @0xC	@0x0 / B3B2B1B0 @0x4 / B7B6B5B4 @0x8 / BBBAB9B8 @0xC / BFBEBDBC

5- Number of Data :-

This is the number of data to be transferred. This number is independent of the data size, for example if the source memory data size is 16 bits and the number of data is 2, a total of 4 bytes will be loaded from the source memory.

The number of data is configured by setting your value in the Channel Number of Data Register (DMA_CNDTRx), but note that the maximum number of data 65535, which is the maximum number that can be represented in 16 bits, as only the least significant 16 bits are available in this register and the most significant 16 bits are reserved.

This register can be written while the channel is disabled, but when the channel is on it is a read-only register where you can read the remaining number of data to be transmitted as it decrements after every transfer.

6- Memory Increment:-

We know that the first transfer operation will work on the addresses set in the CMAR, CPAR as discussed before, but what about the next transfer operation then the after next, …etc. will they work on the same addresses or will they need to increment the address after each transfer?

This can be set by the PINC and MINC (Peripheral increment and memory increment) bits in the DMA_CCRx register. If incremented mode is enabled, the address of the next transfer will be the address of the previous one incremented by 1, 2 or 4 depending on the chosen data size.

This means that source memory incrementation option is independent of destination memory incrementation, so you can choose to set none of them, any of them or both of them.

DMA Configuration

6- Memory Increment Ex :-

Assume that source memory address is 0x100, destination memory address is 0x3000, number of data is 3, Data size of both source and destination memories is 8 bits, let's have a look at the results of different MINC and PINC settings as shown in the able .

no	Source memory Incrementation	Destination Memory Incrementation	Description	Application examples
1	Disabled	Disabled	0x100 → 0x3000 0x100 → 0x3000 0x100 → 0x3000	UART data register to SPI data register (communication bridge)
2	Disabled	Enabled	0x100 → 0x3000 0x100 → 0x3001 0x100 → 0x3002	1- Array initialisation with a constant value 2- Read from UART data register and put in array
3	Enabled	Disabled	0x100 → 0x3000 0x101 → 0x3000 0x102 → 0x3000	Write array to UART data register
4	Enabled	Enabled	0x100 → 0x3000 0x101 → 0x3001 0x102 → 0x3002	Copying array to array

7- Circular Mode :-

This configuration is enabled when we need to continuously repeat the transfer operations, for example to repeat the 3 transfer operations as in the previous example.

So circular mode is available to handle circular buffers and continuous data flows. This feature can be enabled using the CIRC bit in the DMA_CCRx register.

When circular mode is activated, the number of data to be transferred is automatically reloaded with the initial value programmed during the channel configuration phase, and the DMA requests continue to be served.

Circular mode can be used to implement circular buffers and continuous data streams.

DMA Configuration

8- Interrupts :-

An interrupt can be produced on a Half-transfer, Transfer complete or Transfer error for each DMA channel. Every event of those 3 events has its own interrupt enable bit.

The transfer error can be generated by reading from or writing to a reserved address space. When a DMA transfer error occurs during a DMA read or a write access, the faulty channel is automatically disabled through a hardware clear of its EN bit in the corresponding channel configuration register DMA_CCRx.

TEIF in the DMA_IFR register is set and an interrupt is generated if the transfer error interrupt enable bit TEIE in the DMA_CCRx register is set.

Interrupt event	Event flag	Enable control bit	Flag clear bit
Half transfer	HTIF	HTIE	CTCIF
Transfer complete	TCIF	TCIE	CHTIF
Transfer error	TEIF	TEIE	CTEIF
Global interrupt flag	GIF	-	CGIF

DMA Configuration

9- Channel Trigger Source :-

The trigger source means the event at which a new transfer operation will take place by the DMA channel.
When memory to memory mode is selected, the DMA channel works in free running mode which means it continuously operates DMA transfers depending on its clock, but when memory to peripheral or peripheral to memory mode is selected, the DMA controller should wait until the peripheral's event happens in order to get data ready, i.e. UART peripheral receives new data.

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1	-	-	-	-	-	-
SPI/I²S	-	SPI1_RX	SPI1_TX	SPI2/I2S2_RX	SPI2/I2S2_TX	-	-
USART	-	USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I²C	-	-	-	I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	-	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

DMA Configuration

9- Channel Priority :-

Our DMA controller (DMA1) has 7 different channels, although it is one controller only.

Channels are independent in configuration, which means that more than one channel can be enabled at the same time and configured to work on different transfer operations.

A channel may be waiting for its triggering event as discussed before , so another channel can have the chance to work while the other is waiting,

but what happens when more than one channel want to execute their work at the same time?

The answer is that an arbitration process is executed by an arbiter circuit, this circuit uses the channel priority level to determine the order of bus access.

Each channel has a **software** priority and a **hardware** priority.

DMA Configuration

9.1- Software Priority :-

Each channel priority can be configured in the PL[1:0] bits in the DMA_CCRx register. There are four levels:

- **Very high priority**
- **High priority**
- **Medium priority**
- **Low priority**

9.2- Hardware Priority :-

If 2 channel requests have the same software priority level, the channel with the lowest number will get priority versus the channel with the highest number. For example, channel 2 gets priority over channel 4.

The End …

IMT SCHOOL

www.imtschool.com

ww.facebook.com/imaketechnologyschool/