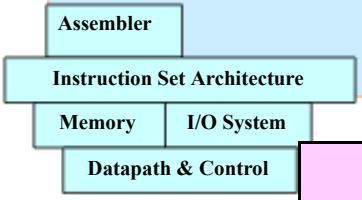
ELEC 3300
Introduction to Embedded Systems

Topic 10

Direct Memory Access (DMA)

Prof. Vinod Prasad

Course Overview



Introduction to Embedded Systems

More about Embedded Systems

Basic Computer Structure

MCU Main Board

Digital and Analog Interfacing

USART, IEEE1394, USB, I2C and SPI

Microcontroller Structure

A/D Port

Serial Port

CPU

External Memory Port

External Interrupt Port

Interrupt Organization

Timer and Counter

External Timer Port

Simple I/O Port

Buffering and Direct Memory Access (DMA)

Memory, Interfacing to Memory, Memory Timing and applications

Interfacing LCD

Motor Interfacing

In this course, STM32 is used as a driving vehicle for delivering the concepts.

To be covered | In progress | Done

Expected Outcomes

- On successful completion of this topic, you will be able to
  - Classify three different I/O modules
  - Describe the basic operations of the three different I/O modules
  - Understand the hardware configurations of DMA
  - Illustrate the operation of DMA block transfer mode

What is the function of Direct Memory Access (DMA)?

DMA provides high-speed data transfers between:

-Peripherals and memory
-Memory and memory

Data can be quickly moved by the DMA without any CPU action.

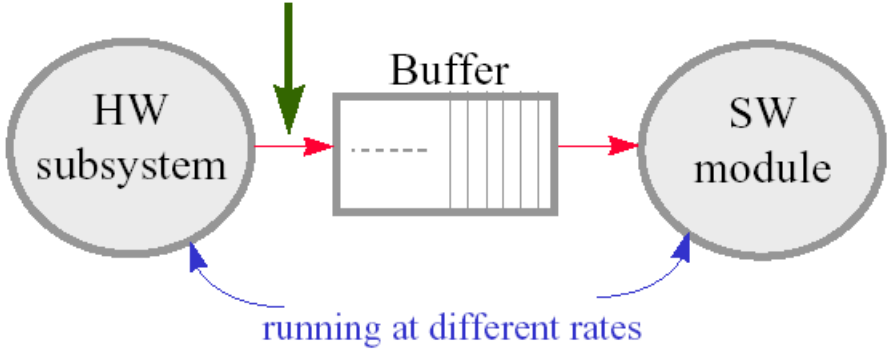
minimal CPU action

This keeps CPU resources free for other operations.

What is a buffer?

- An area of memory that is used to store temporary data

Depending on what the hw subsystem is, the rate at which a hardware event is generated can be unpredictable. For example, if the hardware subsystem is a keypad, the time interval between two keystrokes generally varies. If both the input rate and output rate of the buffer is known, we can easily calculate the size of the buffer such that no overflow or underflow can ever occur.

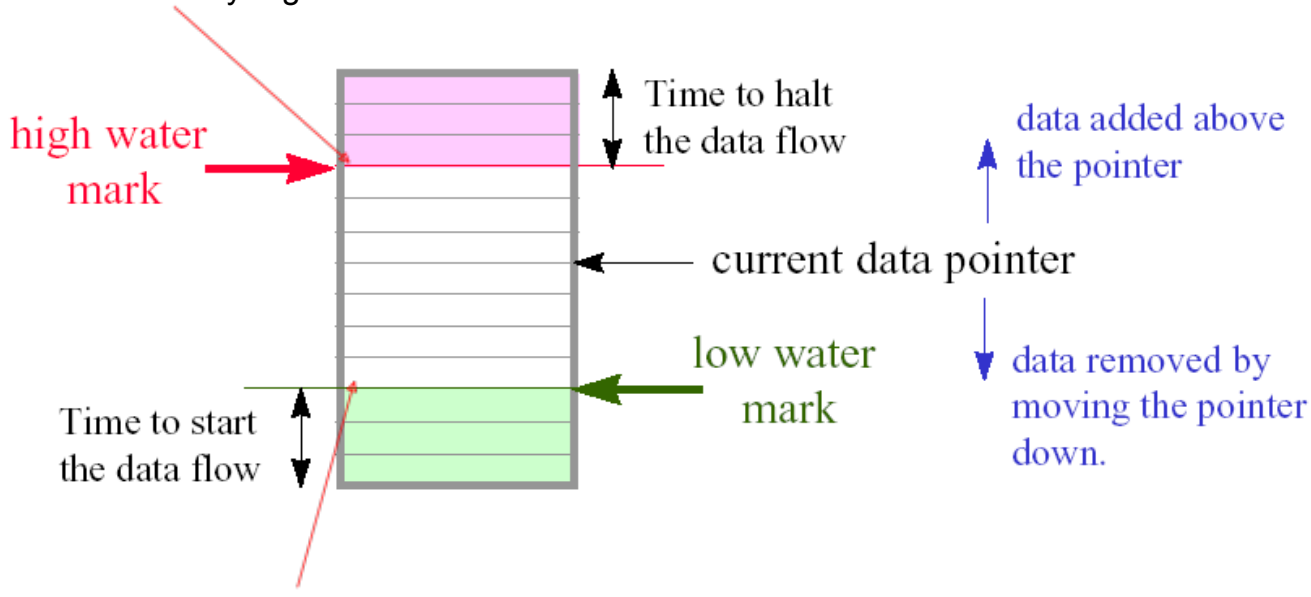


HW typically runs much faster than software

without a buffer sw module cannot keep up with data input from hw module ⇓ overflow of data

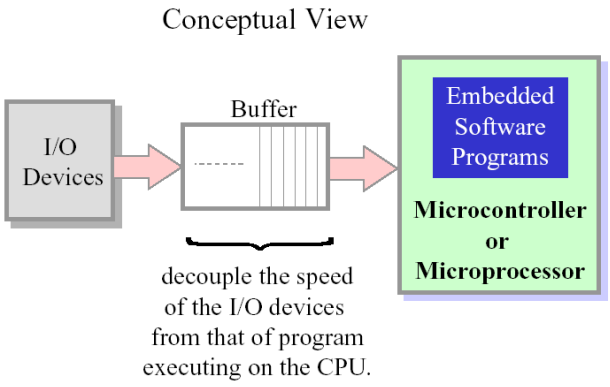
The Buffer

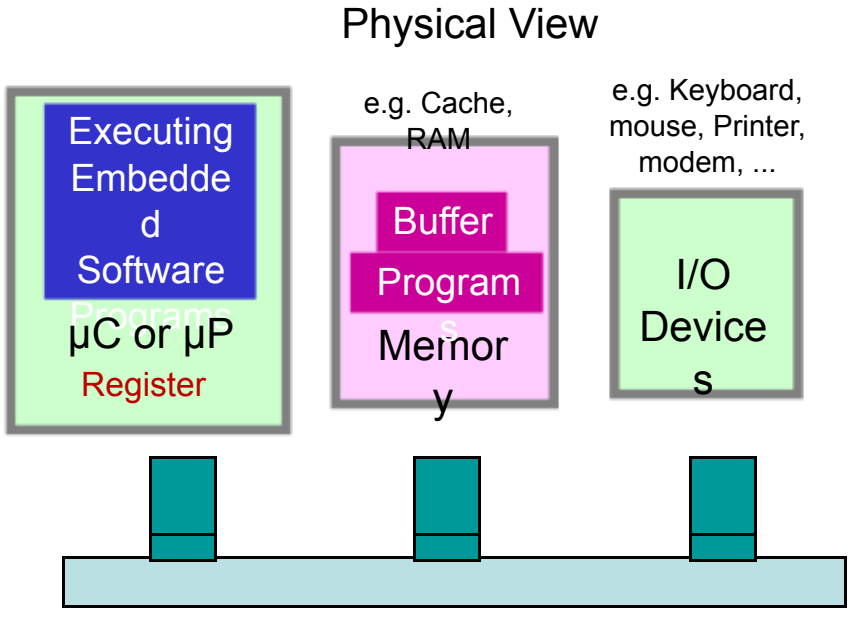
Caution – boundary region of data overflow starts from here.



Caution – boundary region of data underflow starts from here.

Concurrency Between I/O and Programs

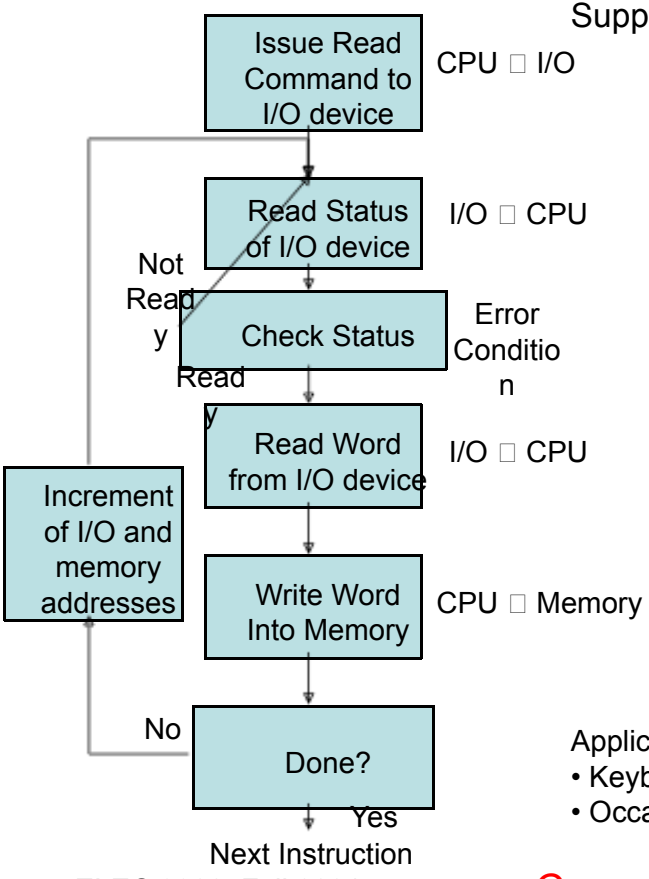


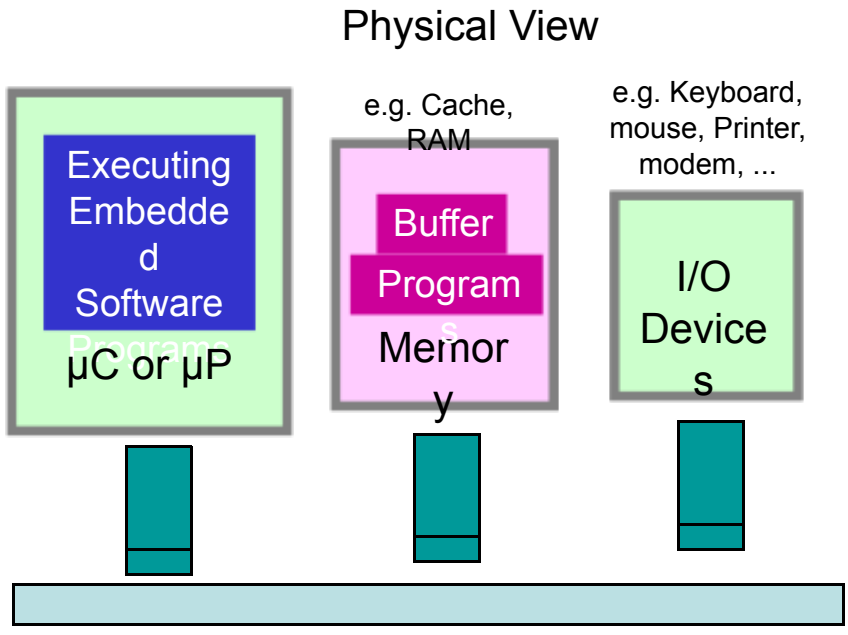


How to write a block of data from disk to memory ?
- Three different I/O modules

I/O Module: Programmed I/O

Suppose this is done one-byte at a time under the software control.



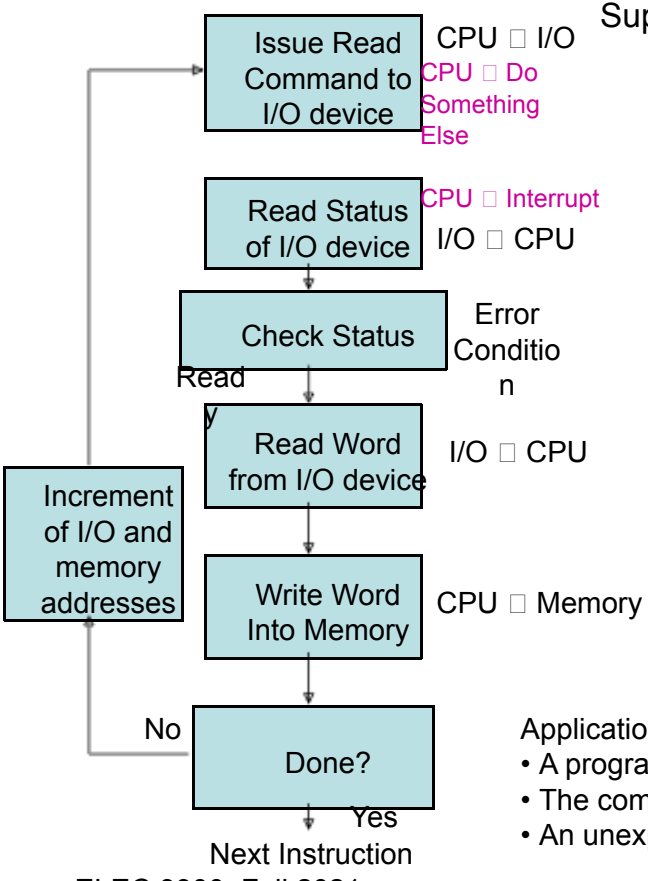


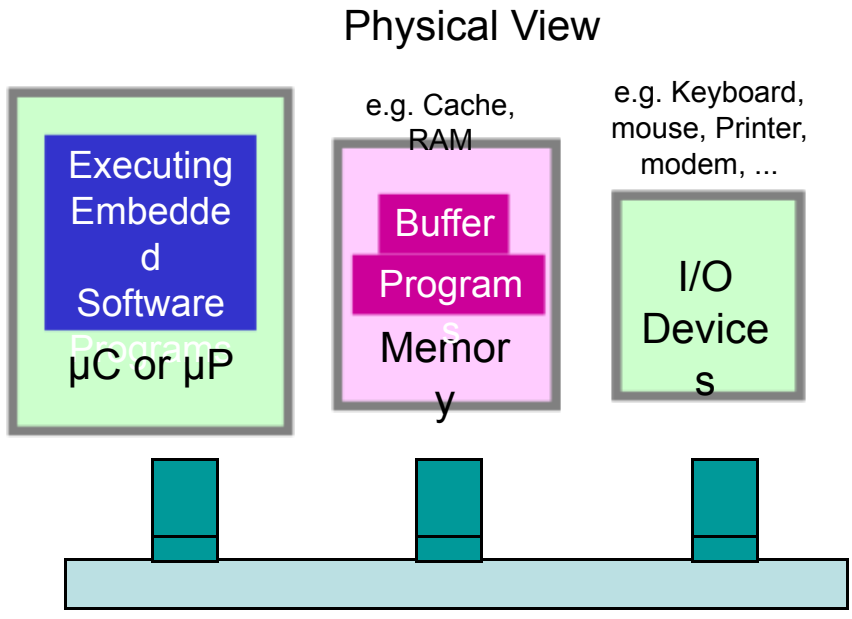
Applications:
- Keyboards interfacing
- Occasional application to other simple character-based data transfer

Cons: CPU is heavily involved.

I/O Module: Interrupt I/O

Suppose this is done one-byte at a time under the software control.

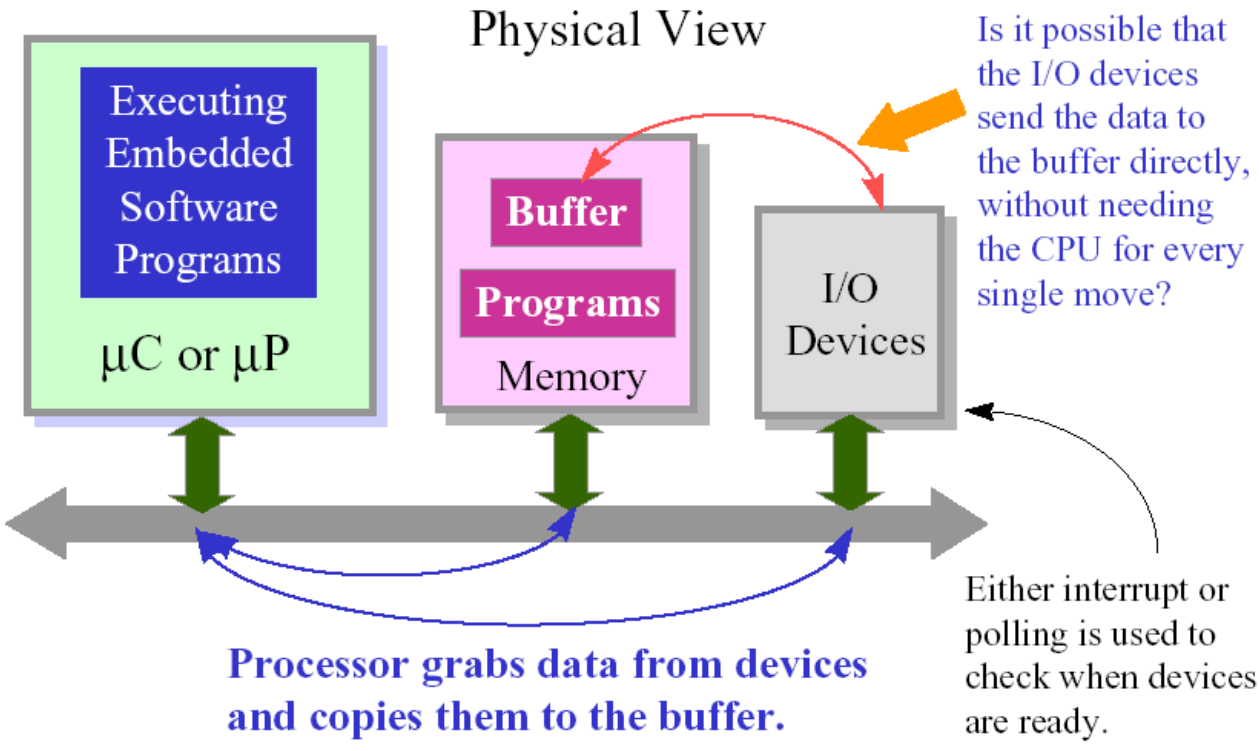




Applications:

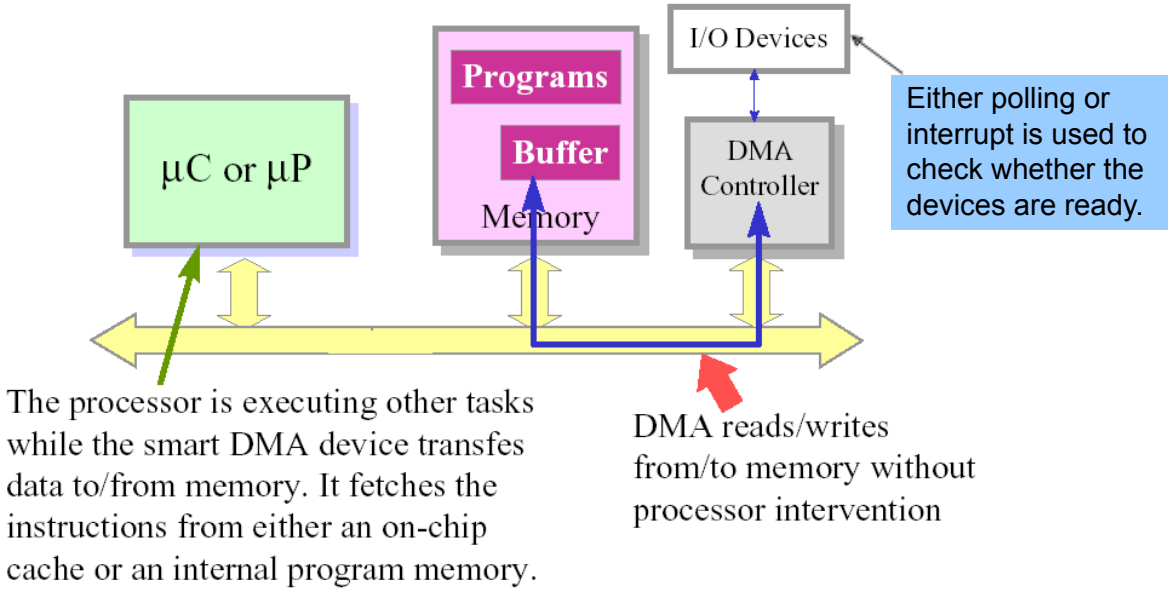
- A program to execute an illegal instruction
- The completion of an I/O task initiated previously a program
- An unexpected user command from keyboard such as resetting the computer.

Increasing Concurrency Between I/O and Programs



I/O Module: Direct Memory Access (DMA)

- What if we make the I/O devices a little smarter?
  - Make the device capable of moving data to/from memory itself
  - Advantage: it would no longer need the processor to move the data
  - Overhead: DMA controller and Additional Control logics



DMA – How

- The device can buy memory access cycles from the bus while the processor is not reading or writing.

- Generally, a block move is set up, say 512 bytes per block
  - The processor initiates the block data transfer by memory-mapped I/O
  - A whole block of data is then transferred without further processor intervention
  - To do this, the DMA controller takes control of the bus (instead of the processor) to sequentially copy all the bytes in the block
  - On completion, the DMA controller restores control of the bus to the CPU

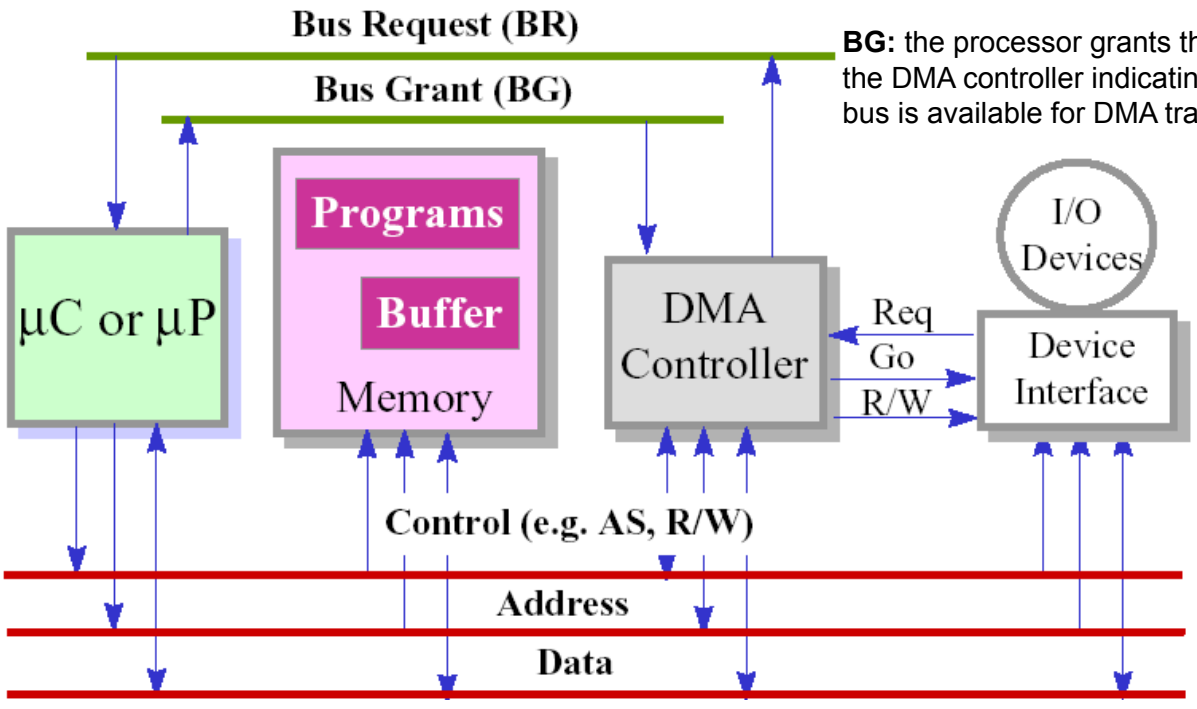
there can be 1 bit to specify whether it is a memory or IO device

Memory-mapped I/O: Uses the same address space to address both memory and I/O devices. The memory and registers of the I/O devices are mapped to address values. So when an address is accessed by the CPU, it may refer to a portion of physical RAM, or it can instead refer to memory of the I/O device. Thus, the CPU instructions used to access the memory can also be used for accessing devices.

DMA Hardware

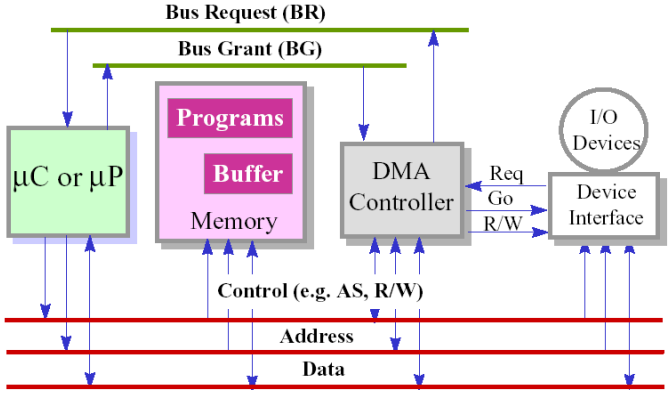
BR: the DMA controller requests use of the bus for a DMA transfer

BG: the processor grants the bus to the DMA controller indicating that the bus is available for DMA transfer



DMA Controller

- When inactive, the DMA controller looks like a normal interface (i.e. the internal registers can be accessed by the processor using a normal read/write bus cycle)
- But when the DMA controller is granted the bus, it generates its own read/write cycles (i.e. it issues address and control signals for the memory and device interfaces)
- This allows the DMA controller to transfer a byte (or word) between memory and device every bus cycle if desired



DMA Operation (Block Transfer Mode)

As an example, consider reading a block from disk (i.e., I/O device):

Processor tells the Device Interface the address of block on disk.

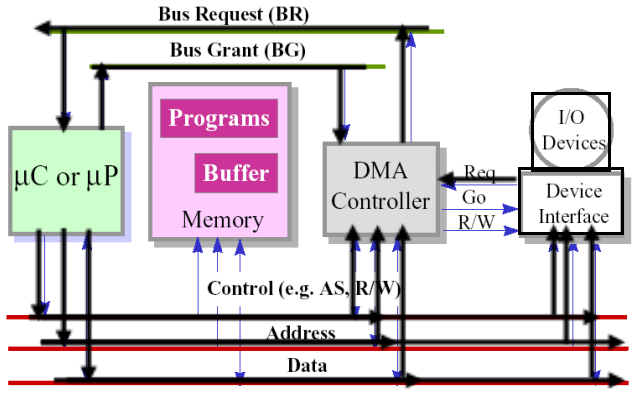
Processor tells the DMA Controller the address and the size of block in memory.

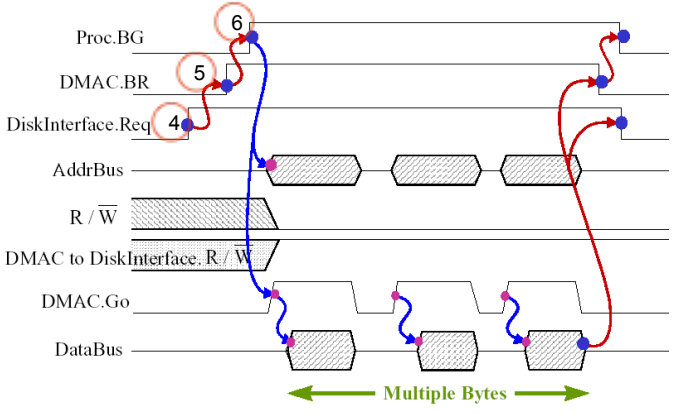
Device Interface reads block from disk to its internal buffer, while CPU continues normally.

Once the device interface completes reading the block, it asserts REQ to DMA Controller.

DMA Controller asserts BR.

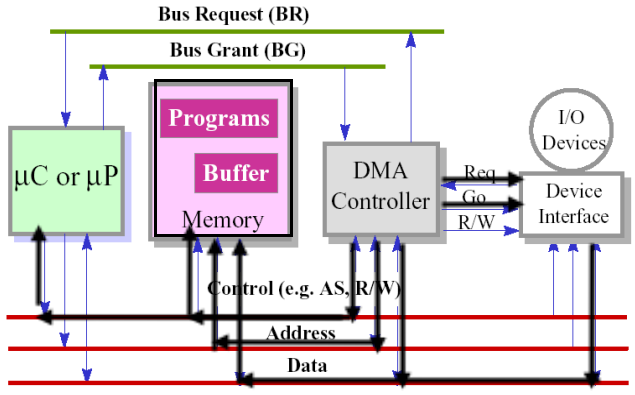
Processor asserts BG.

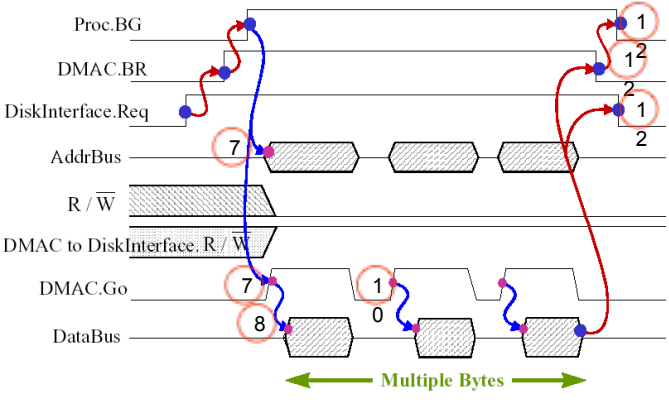




DMA Operation (Block Transfer Mode)

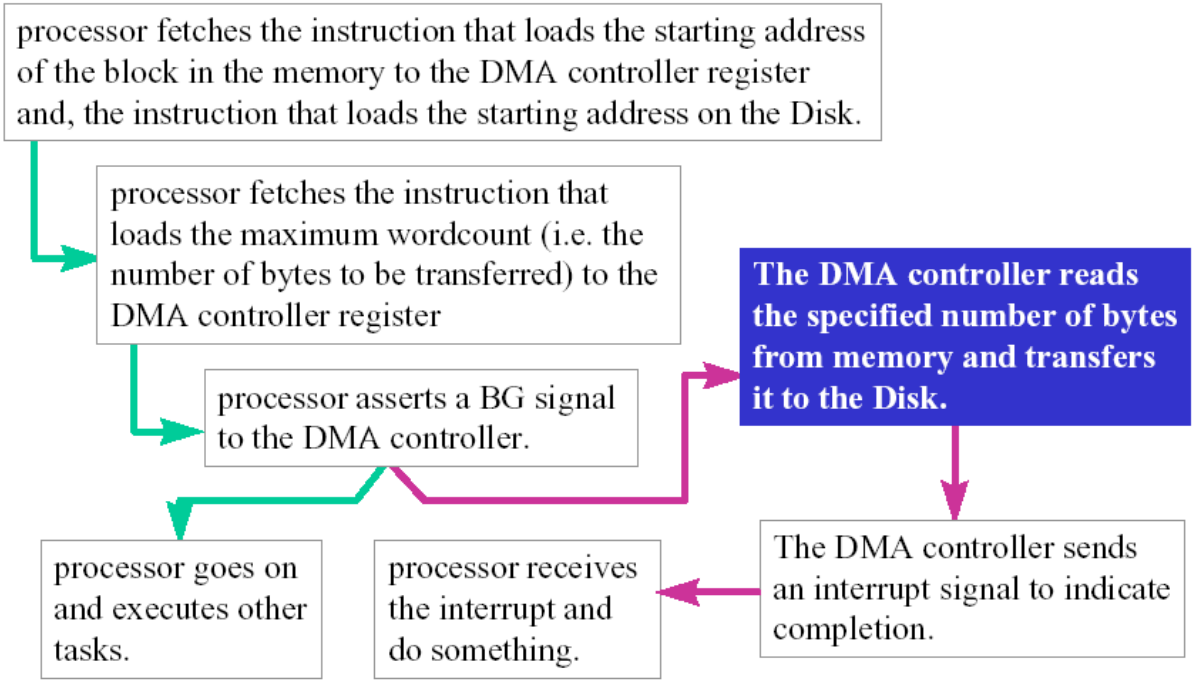
1. DMA Controller puts memory address on Address Bus, asserts control signals to handshake with memory, assert Read and GO to Device Interface
2. Device Interface outputs byte of data from its buffer to Data Bus
3. Memory uses the data and address on the bus to store data
4. DMA Controller de-asserts GO, and Device Interface disconnects from Data Bus
5. Repeat Step 7 to 10 until whole block transferred.
6. DMA Controller de-asserts BR and the processor regains control of bus
7. DMA Controller sends an interrupt to Processor





DMA Operation (Block Transfer Mode)

As an example, consider reading a block from disk:



DMA – Why

- The reason to use DMA are
  - Higher performance: since it requires only one interrupt per *N*-bytes transfer to check the device's readiness rather than one interrupt per byte (as in interrupt I/O).
  - Higher performance: executing *N* move instructions to transfer *N*-bytes of data is slow compared to stealing a cycle from the bus and doing the transfer
  - Used for slower devices that requires a large number of block transfer such as disk, tapes, etc

block size

DMA in STM32

Figure 48. DMA block diagram in connectivity line devices

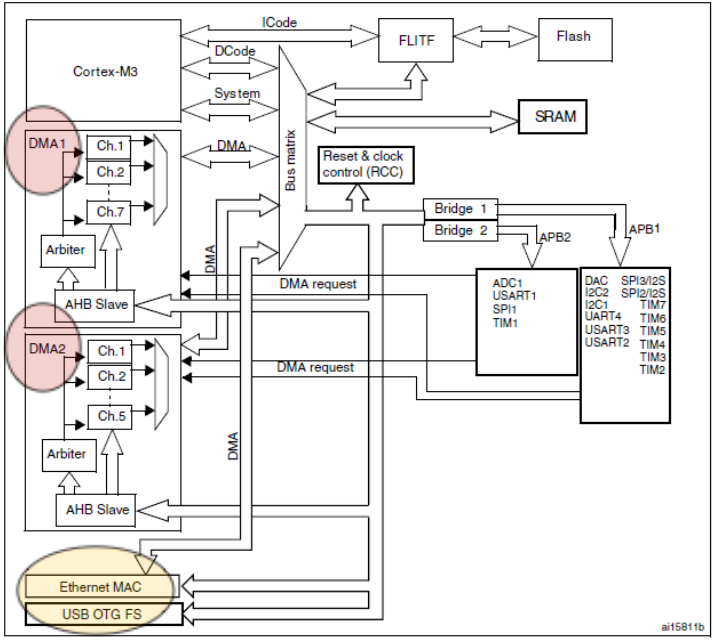
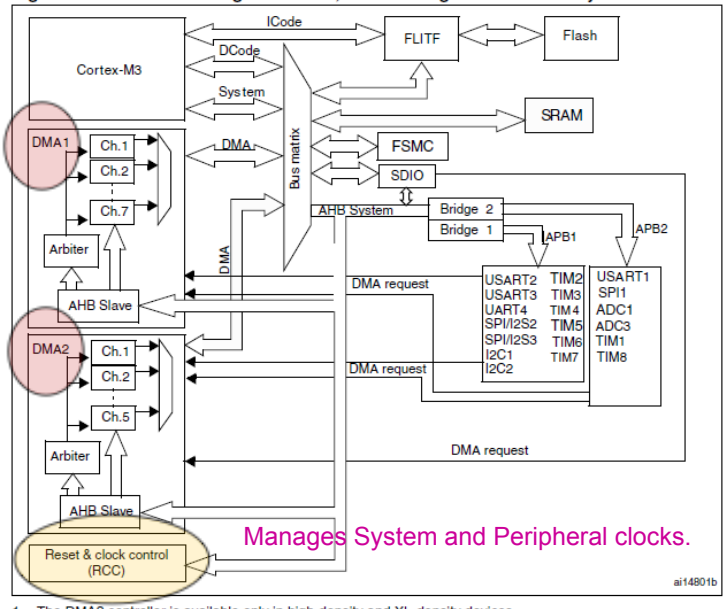


Figure 49. DMA block diagram in low-, medium- high- and XL-density devices



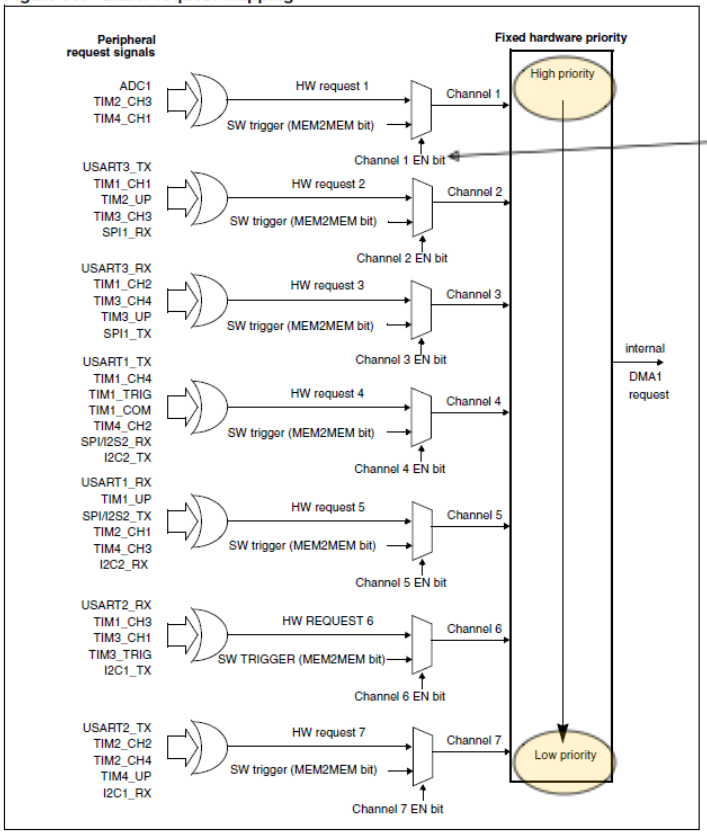
Manages System and Peripheral clocks.

1. The DMA2 controller is available only in high-density and XL-density devices.
1. ADC3, SPI/I2S3, UART4, SDIO, TIM5, TIM6, DAC, TIM7, TIM8 DMA requests are available only in high-density devices

RCC: Power save mode: Switch off on-chip peripherals by removing access to their master clocks.

DMA in STM32

Figure 50. DMA1 request mapping



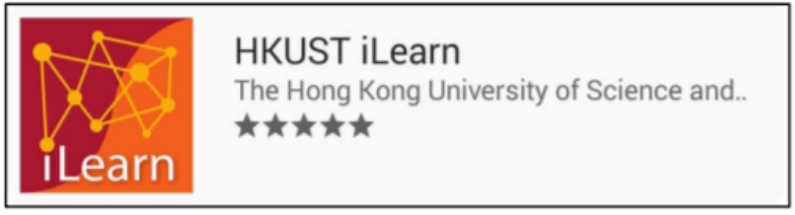
Must ENABLE the channel.

Table 78. Summary of DMA1 requests for each channel

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1						
SPI/I^2S		SPI1_RX	SPI1_TX	SPI2/I2S2_RX	SPI2/I2S2_TX		
USART		USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I^2C				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

In-class activities (Question 1 – 2)

For Android devices, search **HKUST iLearn** at Play Store.



For iOS devices, search **HKUST iLearn** at App Store.

A project demo on DMA application

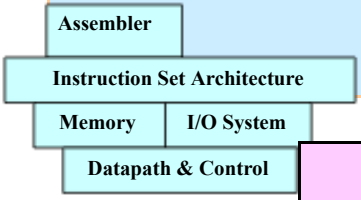
- https://www.youtube.com/watch?v=q50A47aAI9M

This test was taking 84.326 seconds no DMA and no Optimization. It took only 33.71 seconds with DMA+optimization.

Reflection (Self-evaluation)

- Do you ….
  - Distinguish three different I/O modules and their operations?
  - List some examples of applying different I/O in modern computer ?
  - List the hardware configurations of DMA ?
  - Understand the operations of block transfer mode of DMA ?

Course Overview



Introduction to Embedded Systems

More about Embedded Systems

Basic Computer Structure

MCU Main Board

Digital and Analog Interfacing

USART, IEEE1394, USB, I2C and SPI

Microcontroller Structure

A/D Port

Serial Port

CP U

External Memory Port

External Interrupt Port

Interrupt Organization

Timer and Counter

External Timer Port

Simple I/O Port

Buffering and Direct Memory Access (DMA)

Memory, Interfacing to Memory, Memory Timing and applications

Interfacing LCD

Motor Interfacing

In this course, STM32 is used as a driving vehicle for delivering the concepts.

To be covered | In progress | Done