The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

ECE342 – Computer Hardware
Lecture 19

Bruno Korst, P.Eng.

Agenda

- Memory
- A look at Lab 5
  - Camera module
  - Image
  - Functionality
- DMA

Memory

- Before the break, we had reviewed memory (from 243)
- Memory modules connections
  - Address lines
  - Data lines
  - Chip Select (enable)
  - Read/write

- Types
  - On-chip Static RAM, Dynamic RAM, Synchronous DRAM, DDR
  - Cache
  - ROM – read only (PROM, EPROM, EEPROM)

Memory

- In ECE243 (DE1SoC) you saw
  - OnChip (Static RAM)
    - Keeps info as long as power is applied
  - DRAM (asynch)
    - Dynamic, needs refreshing (frequent read/write)
    - Memory controller responsible for refreshing
  - Synchronous DRAM
    - Uses clock to incorporate controller
    - Refresher is built-in
    - Large memory units (DIMM modules)

Memory

- ECE243 cont'd
  - DDR – double data rate SDRAM
    - Data transferred both on rising and falling edges of the clock
    - Presently at DDR5 version
  - Cache
    - Reduce memory access time
    - Fast memory closer to processor, holds active parts of program and data
  - ROM – read only (PROM, EPROM, EEPROM)

Memory

- Add external memory to a microcontroller
  - STM32F7x comes with capability to interface external memory (FMC)
    - Example: MT48LC4M32B2 SDRAM – 128Mb (1Meg x 32 x 4banks)
      - Configuration of memory and microcontroller allows the external SDRAM to be seen as internal memory to the microcontroller
  - Other alternatives for other families
    - SPI Flash memory module, such as W25Q64BV (64Mb)
    - SD card via SPI

Memory

- What do we need to design a digital camera
  - STM32F4x – based board
  - Memory card
    - SD card, connecting via SPI
  - LCD display
    - Interfaces using flexible static memory controller (FSMC)
  - A button – via GPIO
  - A rotary encoder – if we want a timer…
  - A camera module
    - Interfaces via I2C (or a variation of it) and a digital camera interface

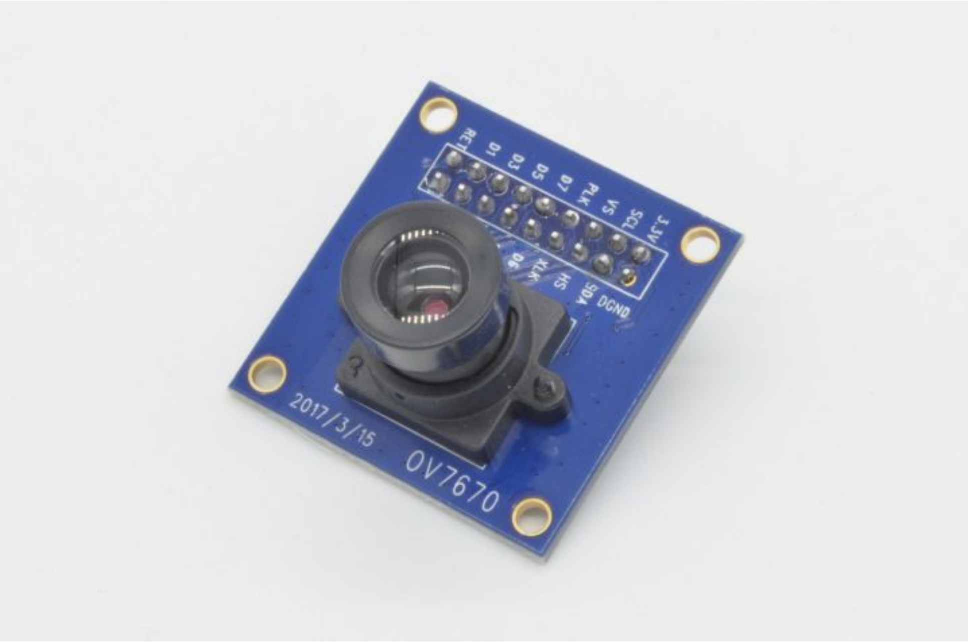
Let’s look at Lab 5

- For Lab5 you will interface a camera with the STM32
- You will use the DCMI (Digital Camera Interface) and will use DMA
- The camera module is the OV7670
  - Uses an I2C compatible protocol (SCCD)

Lab 5

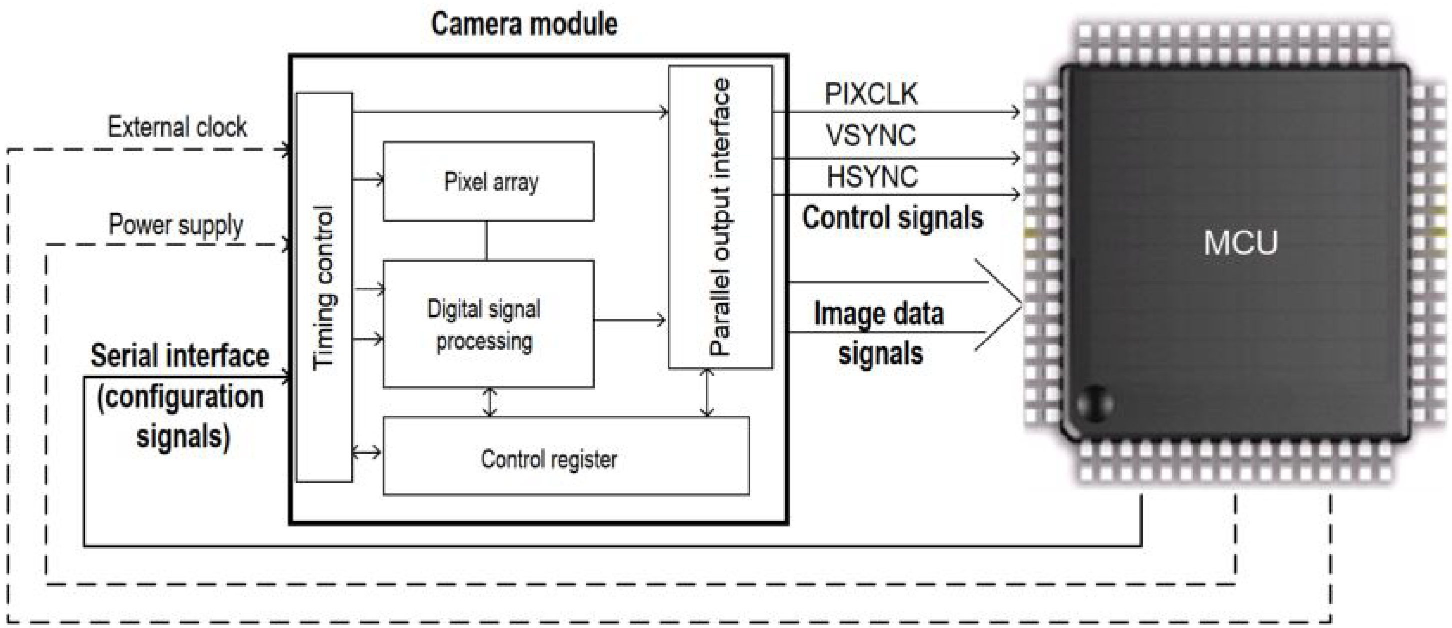
- Digital camera modules

  - Image sensor (CMOS or CCD)
  - Lens
  - PCB
  - Interconnect
    - Control signals
    - Camera configuration
    - Image data signals
    - Power supply



Lab 5

<u>DCMI</u>



- Control signals: clock and synchronization (horizontal and vertical)
  - Horizontal relates to line, vertical relates to frame
- Image data signals: these are the bits representing image pixels
- Configuration: resolution, format, frame rate, type of interface
- Power supply

Lab 5

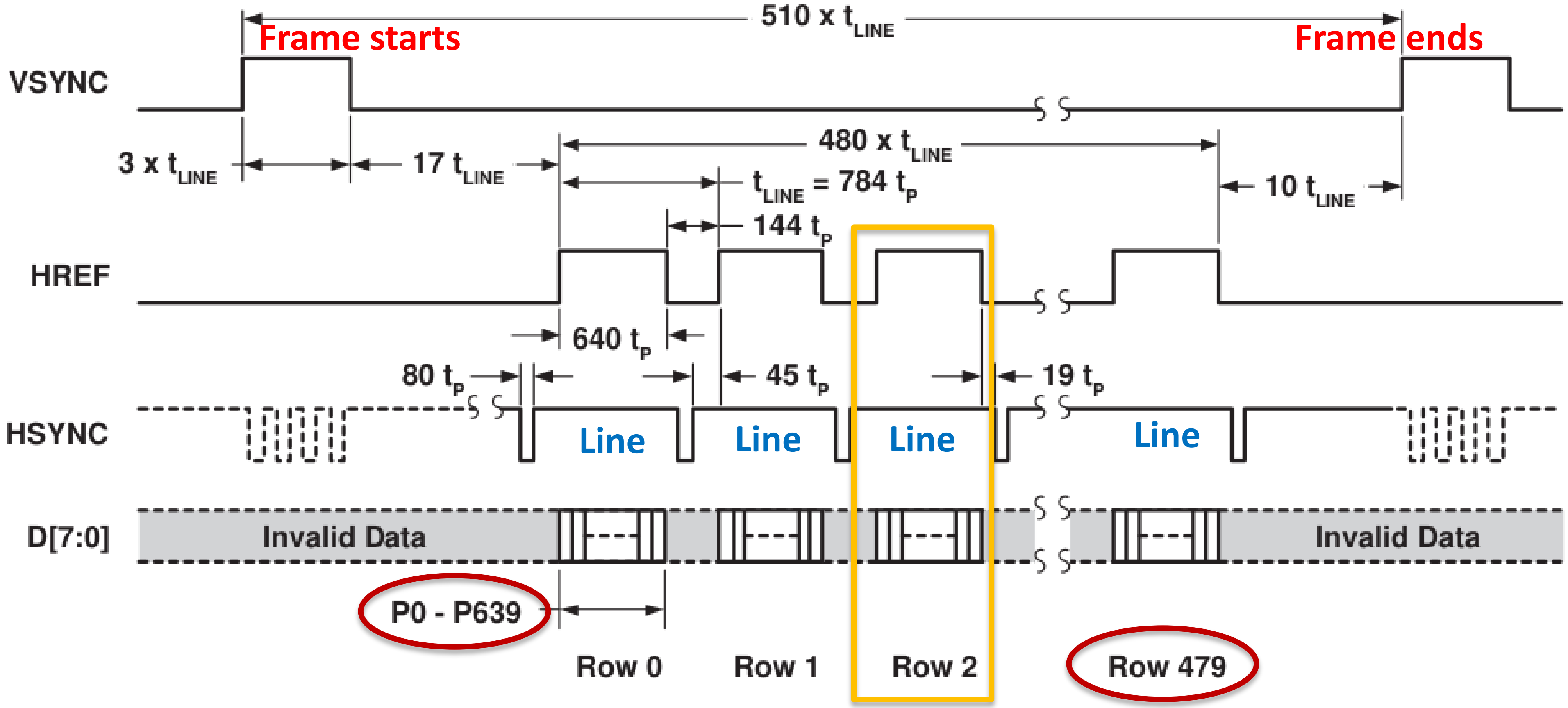
- Formatting images
  - A video is a succession of frames
  - A frame is comprised of lines
  - A line is comprised of pixels
- Pixels
  - monochrome (8 bit gray scale: 0 white, 255 black)
  - RGB – red, green, blue
    - Camera module OV7670 supports RGB565, RGB555 and RGB444
      - Ex: RGB444 → 4 bits red, 4 bits green, 4 bits blue
      - 0 represents dark (no light) and 15 (4 bits) full intensity
  - YCbCr – a format to encode RGB using luma (bright) and chroma (blue and red)

Lab 5

- Signaling
  - OV7670 uses parallel synchronous (needs clock)
  - Data is sampled at the rising edge of PCLK (pixel clock) only when HREF (horizontal synchronization) is high.
    - HREF rising edge → start of a line (falling edge → end of a line)
  - All bytes taken when HREF is high are the pixels in one line.
    - Depending on the format chosen, different number of bytes per pixel
  - VSYNCH will indicate start and end of frames
  - PCLK will dictate the frame rate (24MHz will produce 30 fps)

Lab 5

- Example: signaling for VGA frame format (640 x 480)



Lab 5

- The OV7670 Camera module
  - I2C refers to SCCD
  - Pixel data output is 8 bits in parallel

Pin	Type	Description
VDD**	Supply	Power supply
GND	Supply	Ground level
SDIOC	Input	SCCB clock
SDIOD	Input/Output	SCCB data
VSYNC	Output	Vertical synchronization
HREF	Output	Horizontal synchronization
PCLK	Output	Pixel clock
XCLK	Input	System clock
D0-D7	Output	Video parallel output
RESET	Input	Reset (Active low)
PWDN	Input	Power down (Active high)

Lab 5

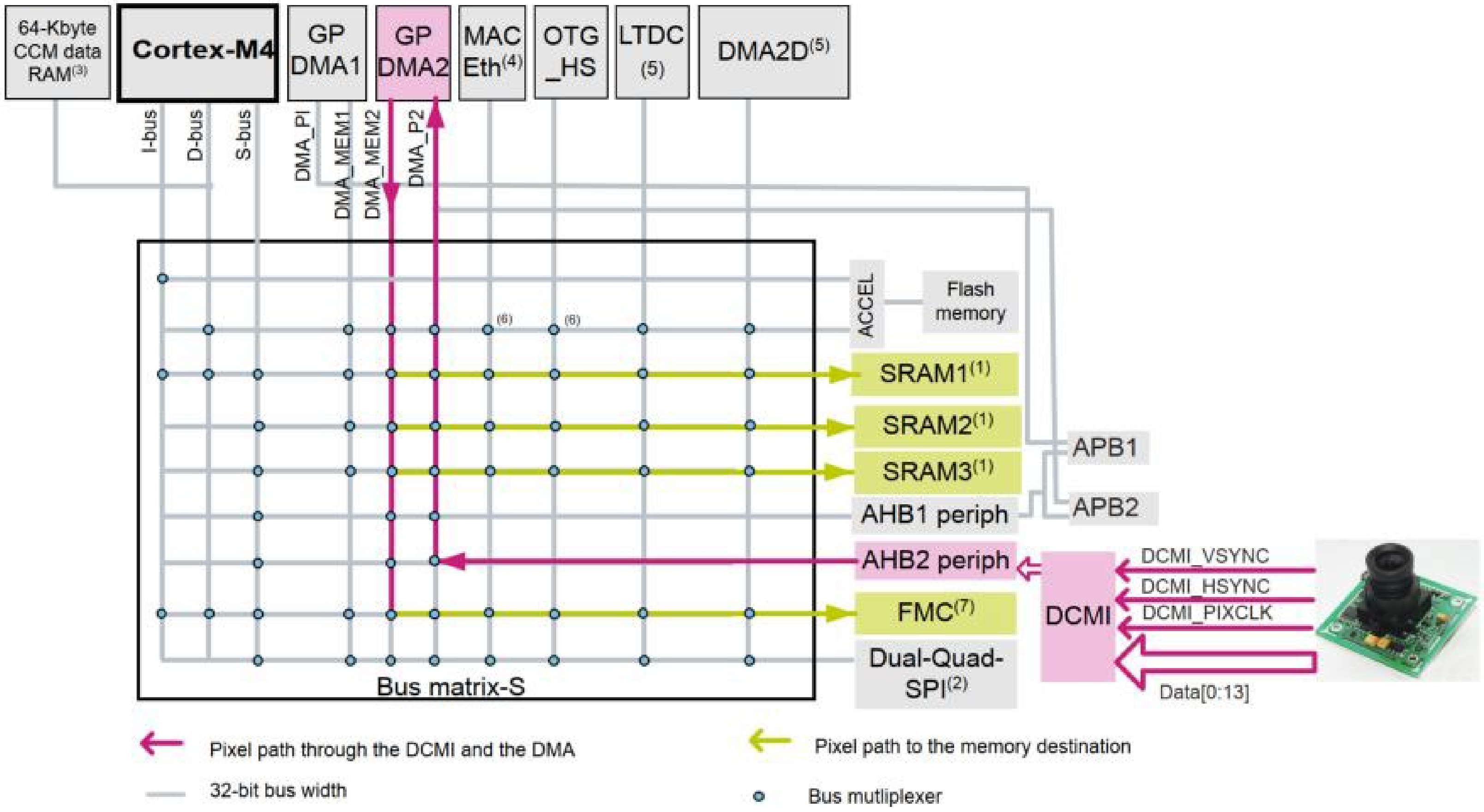
- Now we defined the specs for the picture, need to set up DCMI configuration
  - GPIO configuration
    - Need to configure the I2C, and enable interrupts via NVIC
  - Clock and timing configuration
  - Configure the DCMI peripheral
    - Capture mode, data format, image size and resolution
  - Configure the memory communication (DMA, seen below…)
  - Configure the camera module

Lab 5

- SCCB – serial camera control bus
  - It's an I2C compatible interface
  - Two wire interface: only one master, at least one peripheral
  - SIOC and SIOD
    - Serial bus clock signal (SCL), serial bus data signal (SDA)
  - All transmissions are initiated by the master

Lab 5

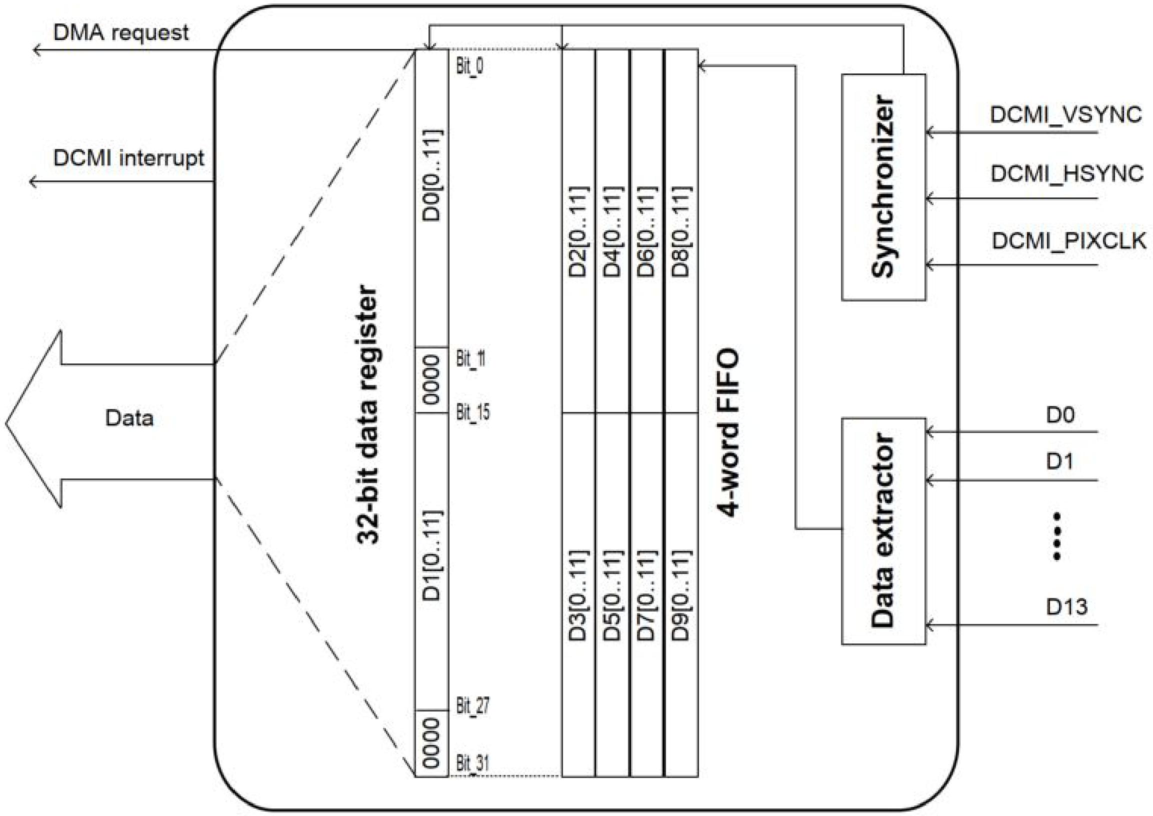
- Connection between camera and device is via DCMI to a memory controller



Lab 5

- The DCMI connects the module signals with the bus and a memory controller (DMA)

AHB2
here



Camera
Module
here

Lab 5

- DCMI functional description
  - Synchronizer controls data flow through data extractor, FIFO and 32 bit register
  - Data is packed into 4 word FIFO, then ordered into 32 bit register
  - A request to the memory controller is generated
  - Transfers the data to the corresponding memory destination
- Note: where is the CPU in all this?

Memory

- Suppose you have a system collecting data from an I/O port, and the data needs to be stored as it is collected.
    - CPU runs a program, part of it is to collect data from the I/O port
        - When that part of the program is reached, the port is read

    - Possible outcomes between the processor and external device
        - Program tries to read but device is not ready ("busy waiting")
        - Device is ready with data, program was running another part (did not try to read)
        - Program is made to read frequently (polling) so that it will not miss any data
        - Device interrupts CPU operation with data ready
            - » CPU goes to handle the interruption
            - » Collects data, stores it, goes back to doing what it was doing

Memory

- The options demand that the CPU stop its execution to handle the incoming data

- A **better option**: spare the CPU of having anything to do with the transfer

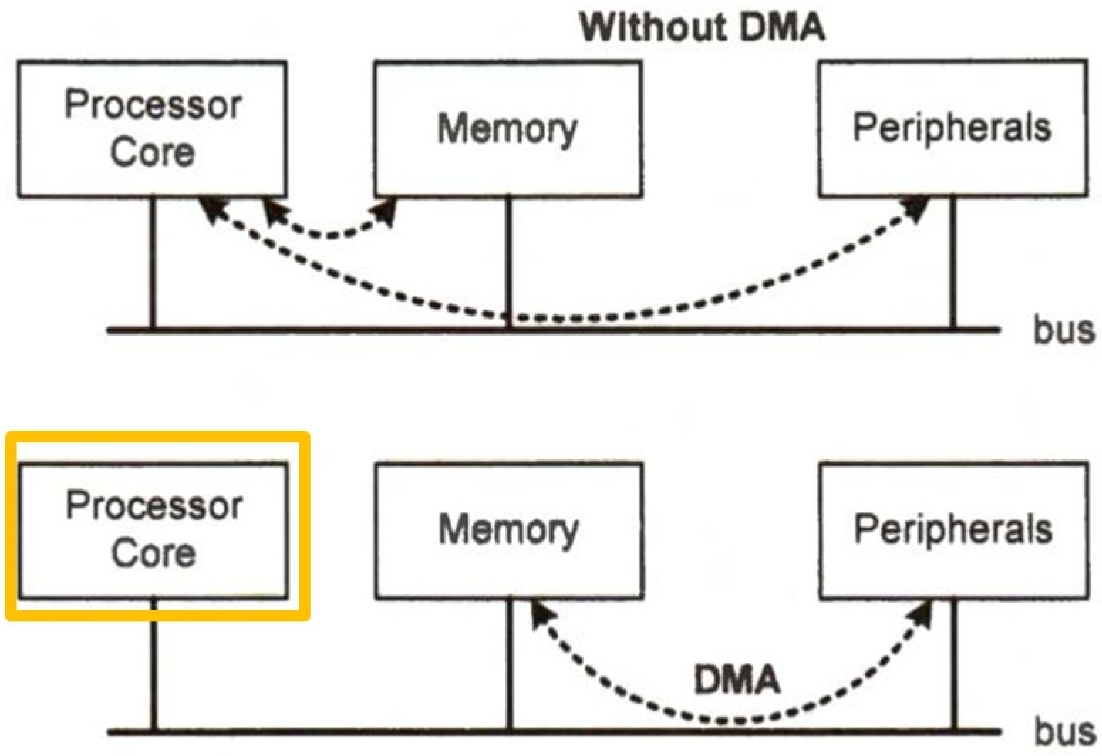
  - This is **Direct Memory Access (DMA)**

  - In this case, the CPU only gets involved when the data is all transferred and ready

Direct Memory Access

- With DMA, the transfer is done via the DMA controller
- Improves the energy efficiency of the system
  - It transfers between peripherals and memory and between memory and memory
- Two types of strategies
  - Cycle stealing – uses spare/idle cycles of the CPU
  - Independent DMA controller

Direct Memory Access



Processor is not involved and can execute other tasks

Direct Memory Access

- Offers an efficient way to interface peripherals

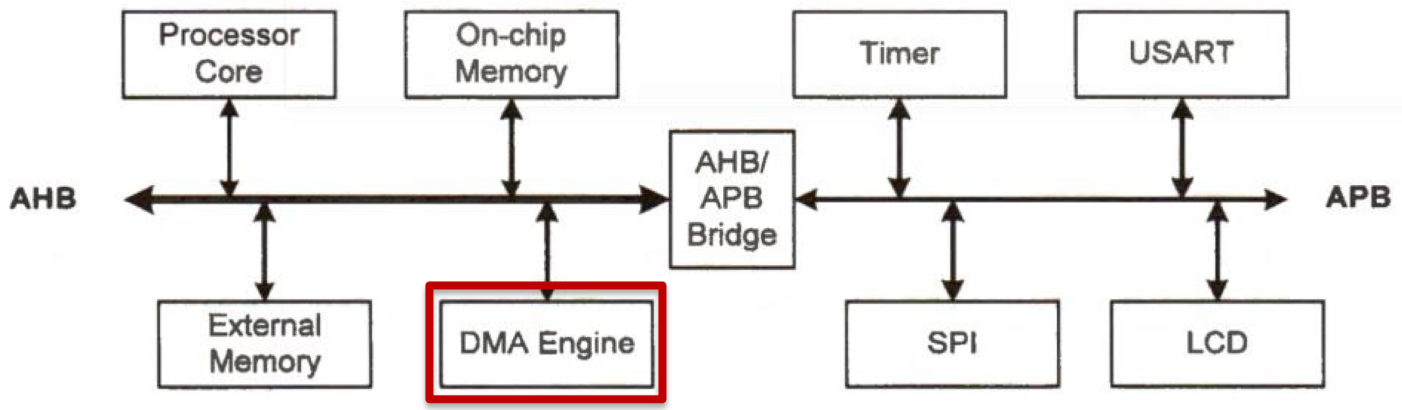
  - Slow peripherals (such as a sensor)
    - DMA releases the processor from waiting for data
    - This frees the CPU to perform other tasks

  - Fast peripherals (such as an external ADC)
    - DMA improves data throughput
    - Memory access does not involve the CPU
    - In high-speed, DMA helps to reduce interrupt rate, and its associated overhead

Direct Memory Access

- Storing data with DMA
  - Peripheral does not need local memory
  - Data is stored efficiently in data memory

- For the STM32 platform, recall the AMBA bus architecture



Direct Memory Access

- The On-chip DMA controller resides on the AHB bus
  - A high-performance bus, which allows for multiple masters (with arbitration)

- When data is coming from the APB bus (low speed peripherals)
  - Data goes through the bridge
    - The bridge will act as a slave for AHB, and as a master for APB
    - Provides buffering for address, control and data
      - No loss due to difference in speed between buses

Direct Memory Access

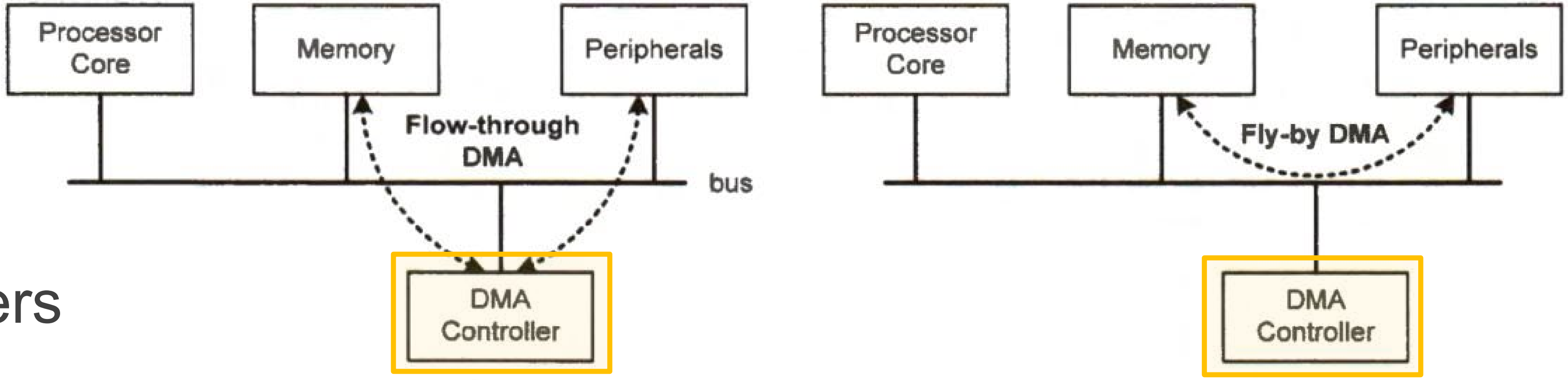
- On-chip DMA controller

  – Can act as bus master and bus slave
  – As a master
    - Initiates transfer within the AHB
    - Initiates data transfer across the AHB/APB bridge

  – As a slave
    - Takes data and commands from processor when DMA transfers are set up

  – It manages multiple channels simultaneously, each with own interface to peripherals

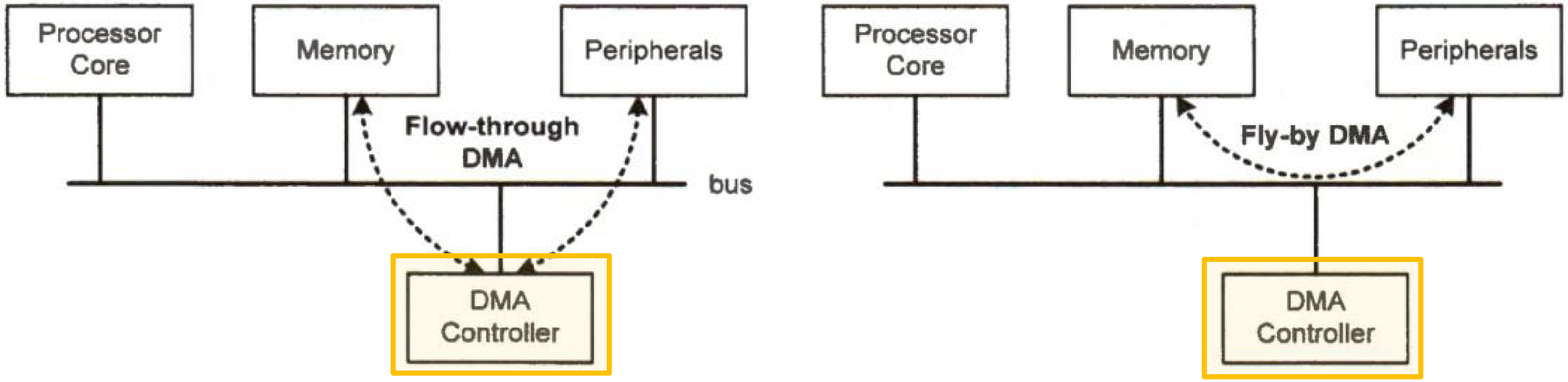
Direct Memory Access



- Types of DMA Transfers
  - Flow-through
    - Uses register in the controller
      - Data is read one at a time from source to register
      - Data is written from register into destination
    - Use when:
      - Devices have registers of different sizes
      - When memory to memory transfer cannot be read/written in one cycle

Direct Memory Access

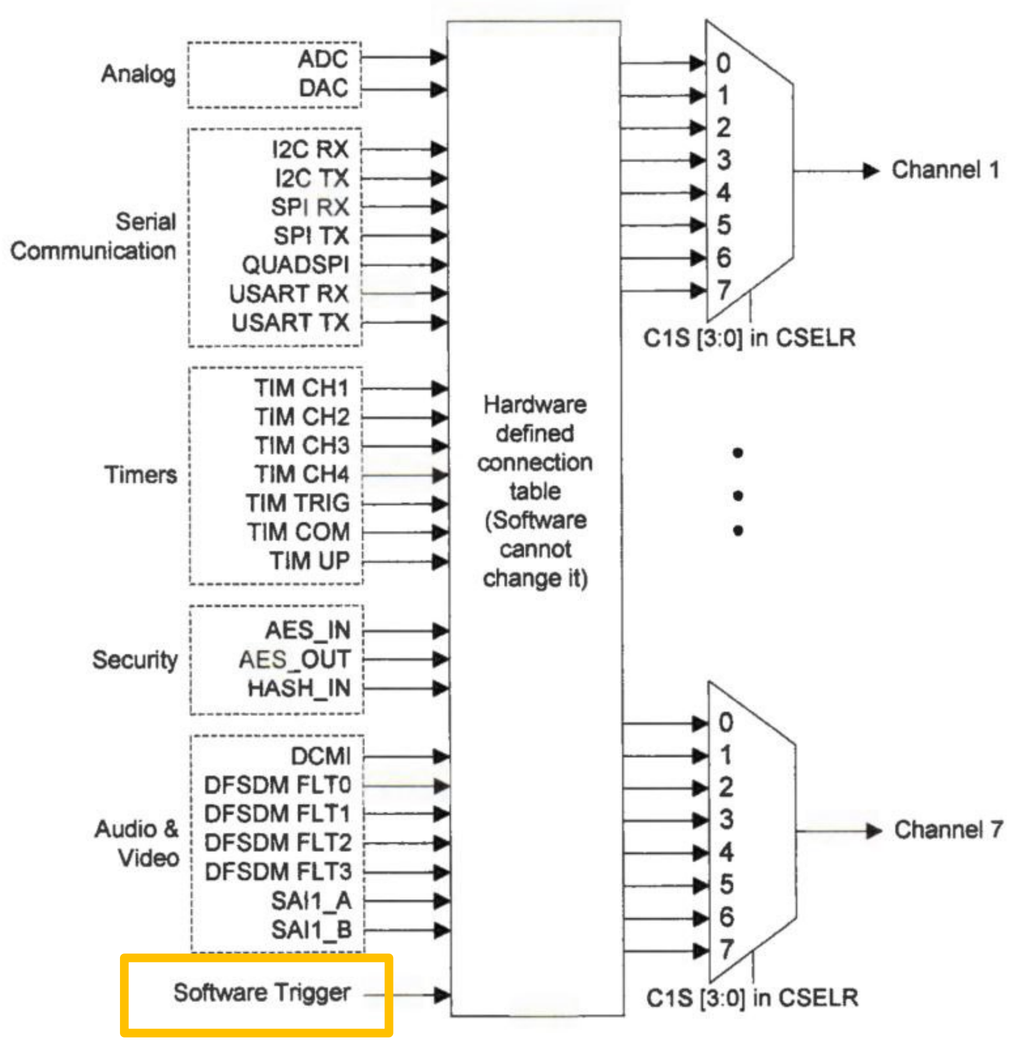
- Fly-by DMA
  - Data goes from source to destination without using controller register
  - More efficient than flow-through; it's used when flow-through is not imperative



Direct Memory Access

- DMA controllers offer multiple channels
  - Channels transfer data independently
  - Each has source, destination, transfer direction, transfer width, data amount and trigger
    - When the channel is enabled and the trigger is received, DMA transfers occur automatically
    - A register is used to select events for each channel

Direct Memory Access



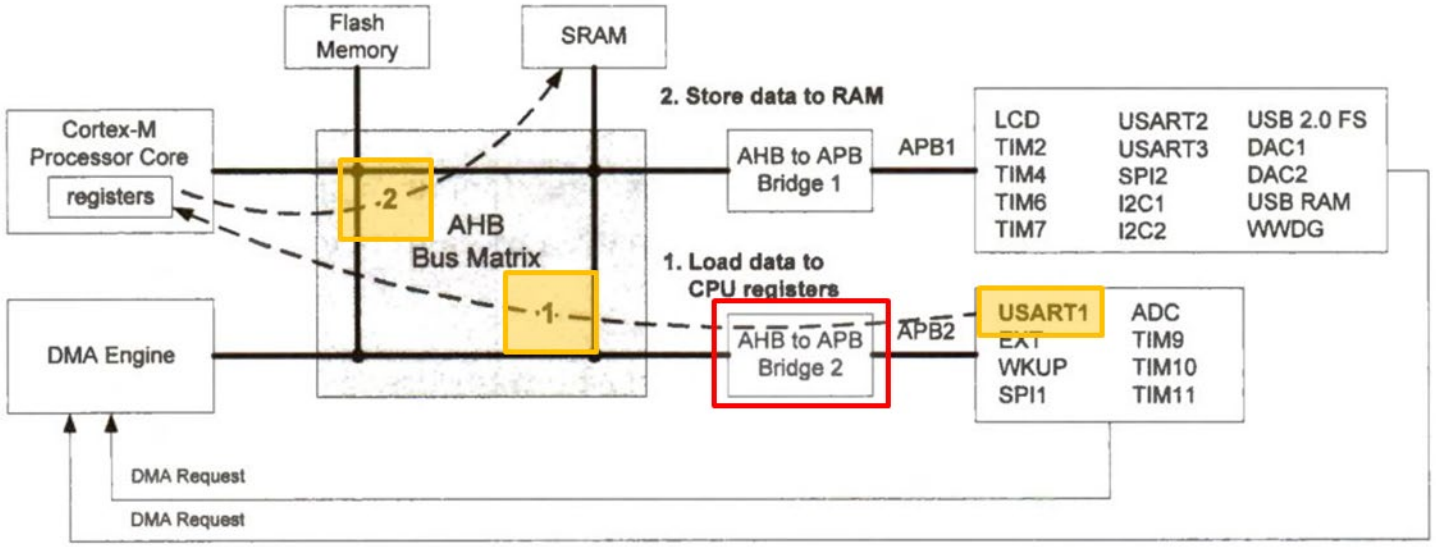
- For one DMA controller
  - Each channel performs transfers independently
  - One trigger is available per channel
    - 7 channels, 8 event types
  - Any of the event types can be selected as trigger
  - Trigger is selected in a register, plus a software trigger if needed.

Direct Memory Access

- When programming DMA
  - Which DMA controller should be used? (if multiple)
  - Which channel should be used?
    - Different channels have access to different resources (ADC, USART, I2C, etc)
  - Which trigger should be selected?
- Priorities
  - Hardware priority is higher, channel 1 is highest among them
  - If using software priority, highest should be channel demanding highest BW

Direct Memory Access

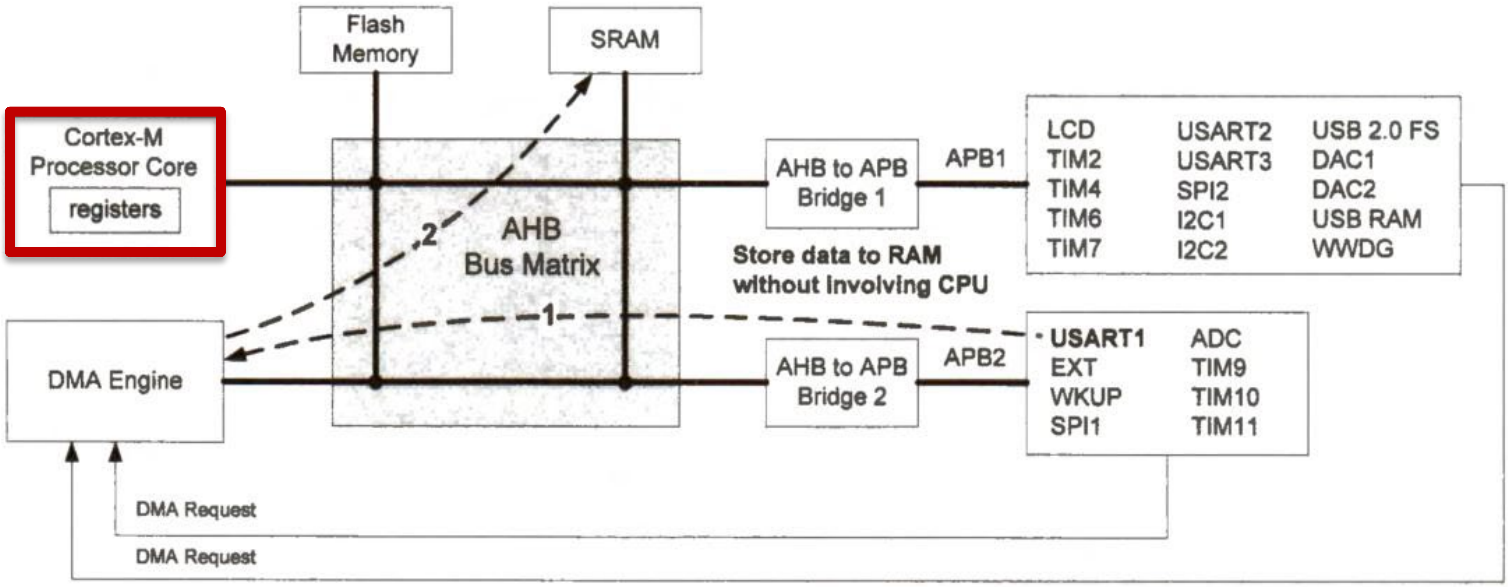
- Example: Transfer USART to RAM
  - First: **without DMA** (i.e.: CPU is occupied)
    1. Processor executes load data to CPU registers
    2. Processor executes store data to RAM



Direct Memory Access

- Example: Transfer USART to memory – with DMA
  - Processor programs DMA controller (i.e. sets up DMA transfer) and enables the channel

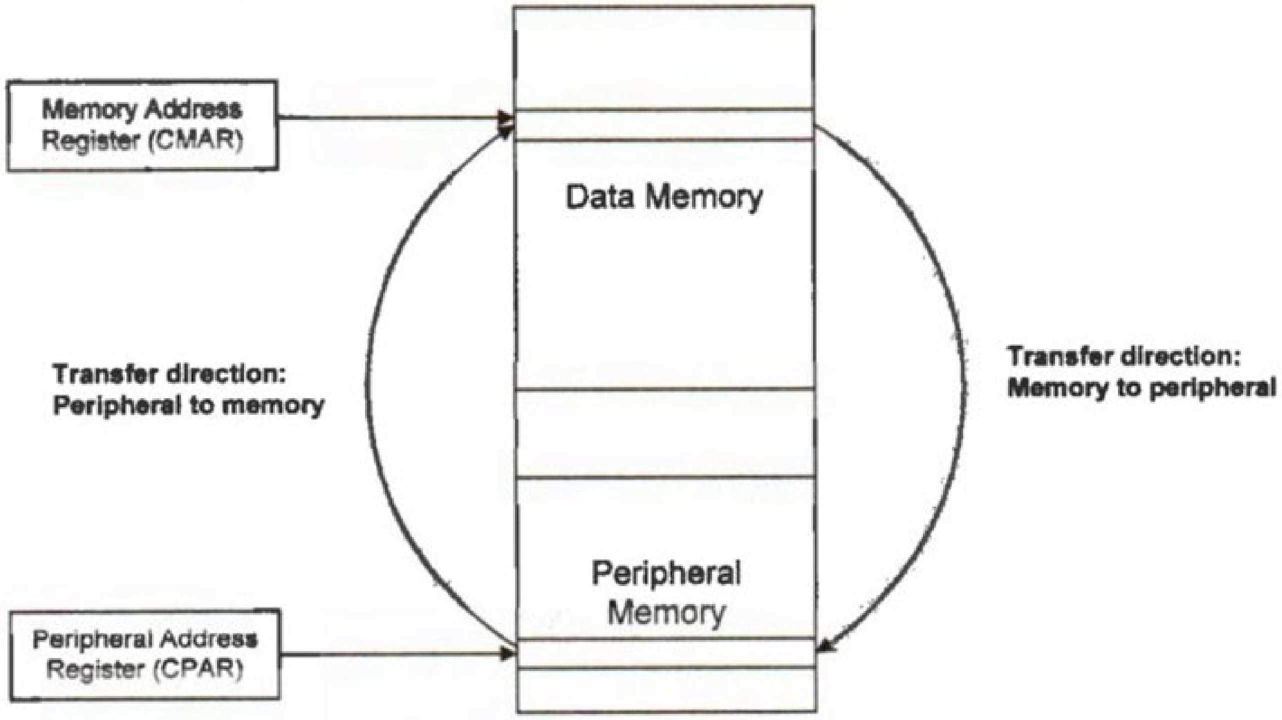
Not used!



  - At the end of transfer, DMA engine sends interrupt to inform data is saved

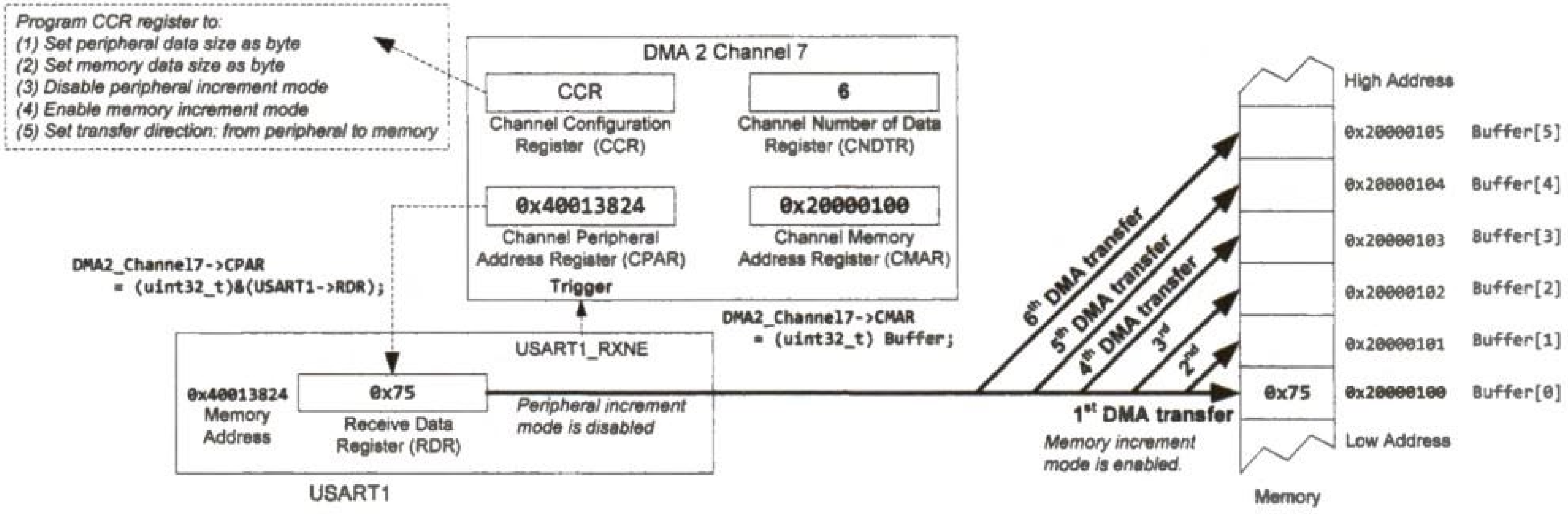
Direct Memory Access

- Each DMA channel has 4 registers, indicating
  - Start memory address
  - Start peripheral address
  - Transfer size (how much data)
  - Direction of transfer and other channel configuration



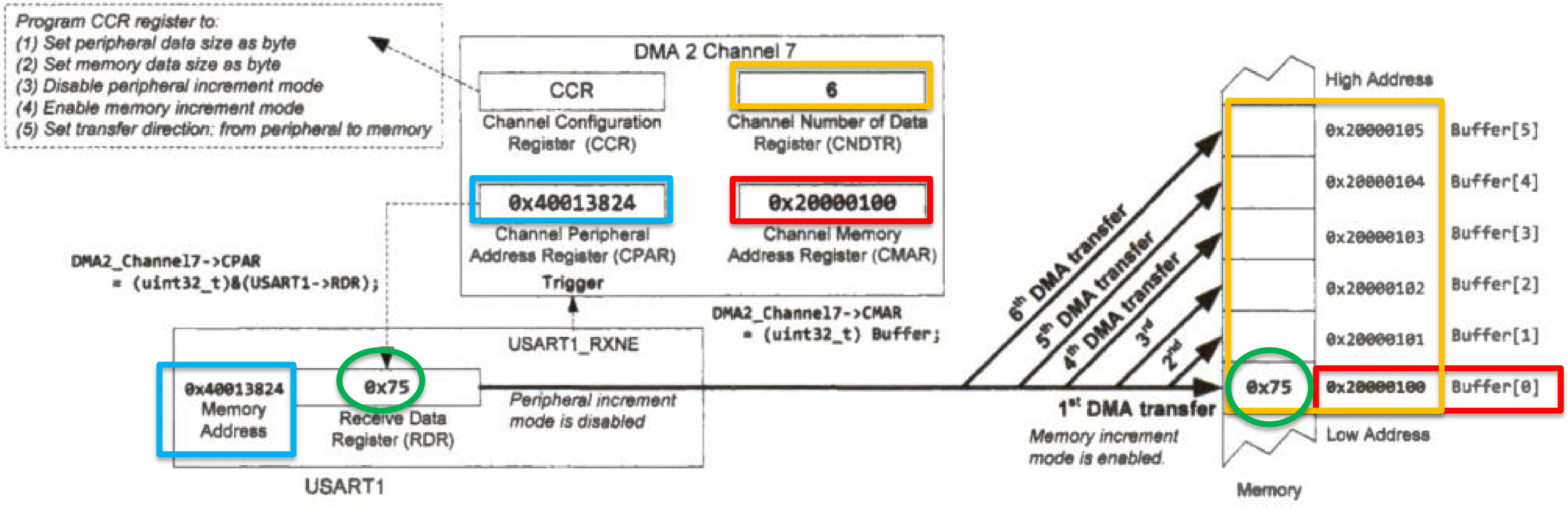
Direct Memory Access

- DMA controller 2, channel 7 – transferring 6 bytes
  - Peripheral address 0x40013824, memory (buffer) address 0x20000100



Direct Memory Access

- DMA controller 2, channel 7 – transferring 6 bytes
  - Peripheral address 0x40013824, memory (buffer) address 0x20000100



Direct Memory Access

- Note in setting up DMA
  - May use both TX and RX to two separate buffers (memory), which means two separate channels to set up
    - Each channel will be triggered by the peripheral when the respective register is “filled”
  - Interrupts can indicate DMA transfer finished/half finished/error
  - Buffer may be set up as “circular” for continuous data streams (when counter reaches the amount of data to be transmitter, address returns to start)