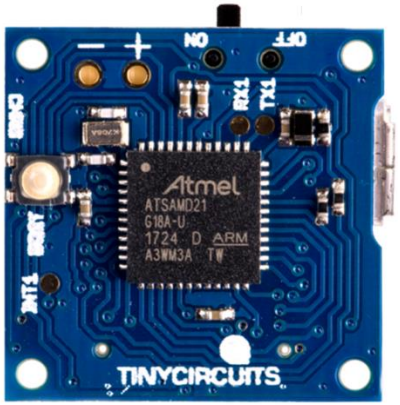
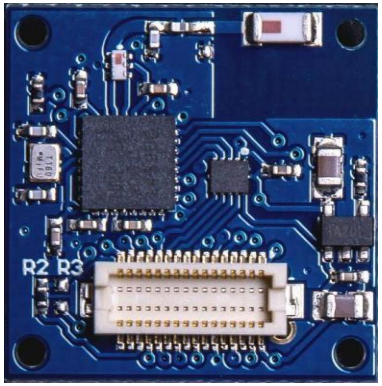
CSE190 Winter 2025



Lecture 14

Direct Memory Access



Wireless Embedded Systems

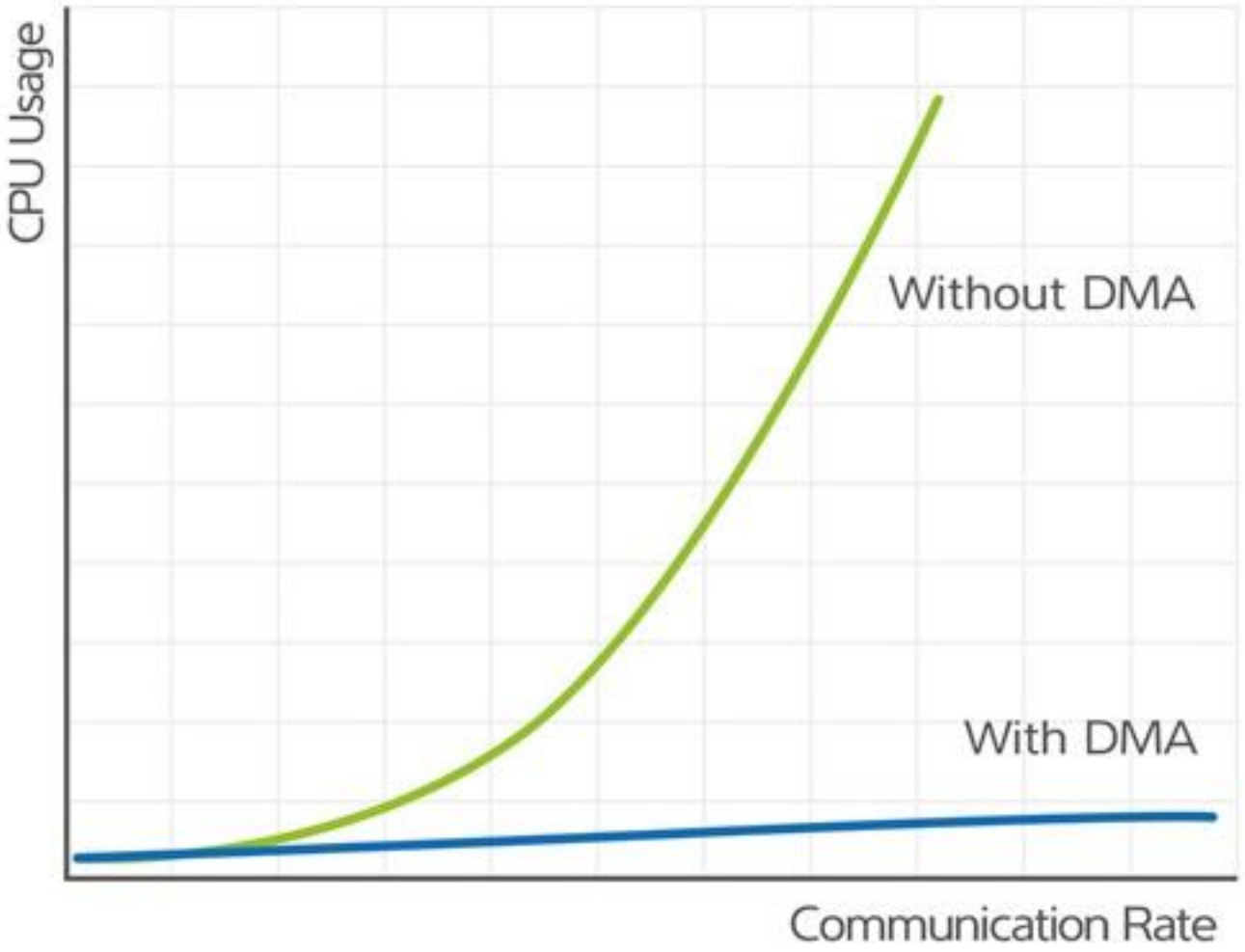
Aaron Schulman

How do you move data to and from peripherals in MMIO?

- Moving data to a peripheral?
  - CPU instructions that write data from RAM to MMIO
  - `I2C->DATA = X[1];`

- Moving data from a peripheral?
  - CPU instructions that read MMIO to RAM
  - `X[2] = I2C->DATA;`

Why do we need DMA?



Why do we need DMA?

Polling and Interrupt driven I/O concentrates on data transfer between the processor and I/O devices.

An instruction to transfer (mov data,R0) only occurs after the processor determines that the device is ready

- Either by polling a status flag in the device register or
- Waits for the device to send an interrupt request.

Why do we need DMA?

Considerable overhead is incurred by MMIO, because several program instructions must be executed for each data word transferred.

Moving things is a waste of CPU instructions:

– Instructions are needed to increment memory address and keeping track of how many bytes are moved.

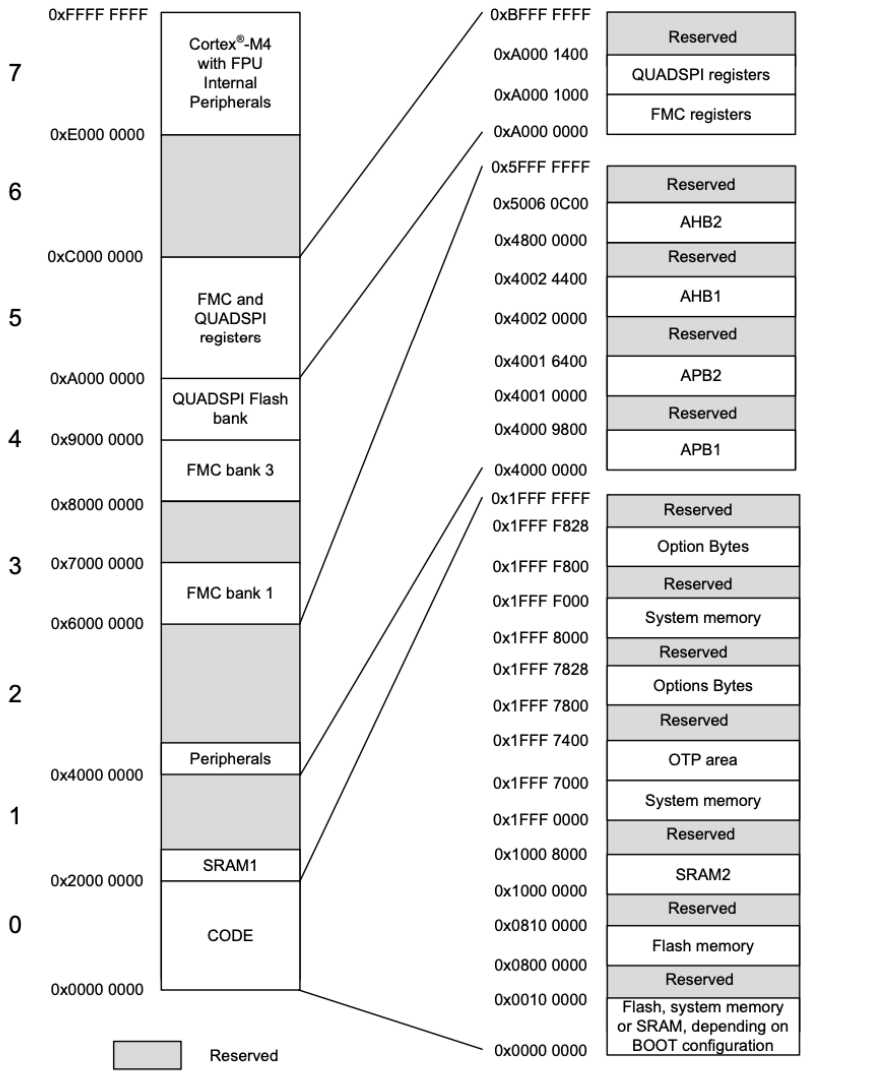
Direct Memory Access (DMA)

- To transfer large blocks of data at high speed, an alternative approach is used, the DMA periperhal.

- Blocks of data are transferred between an external device and the main memory, without continuous intervention by the processor.

- ***It's just another peripheral, but it's only job is moving data.***

DMA is possible because of linear memory addressing



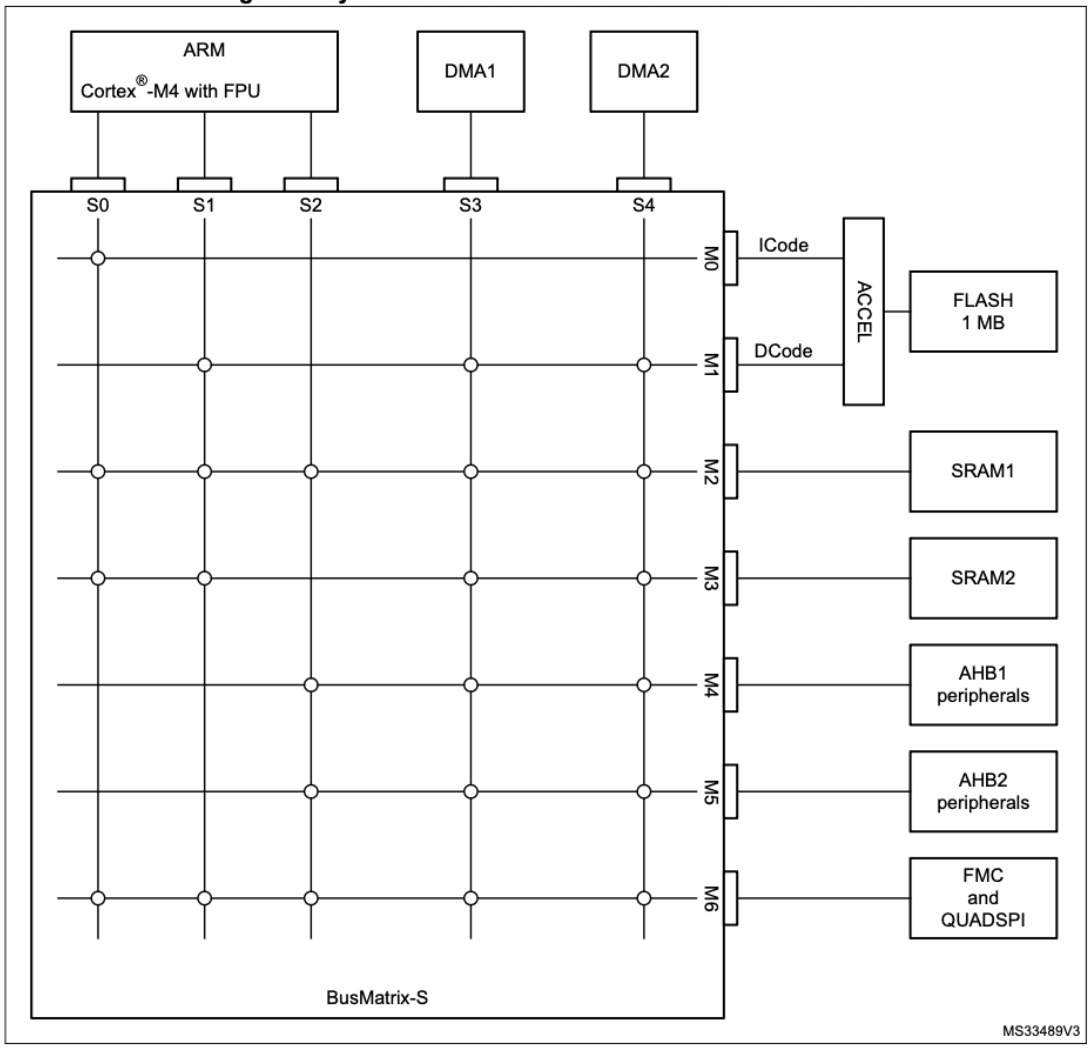
DMA Controller

- DMA controller is connected to the internal I/O bus.

- Performs the functions that would normally be carried out by the processor when access main memory. For each word transferred, it provides the memory address and all the bus signals that control data transfer.

Memory Architecture

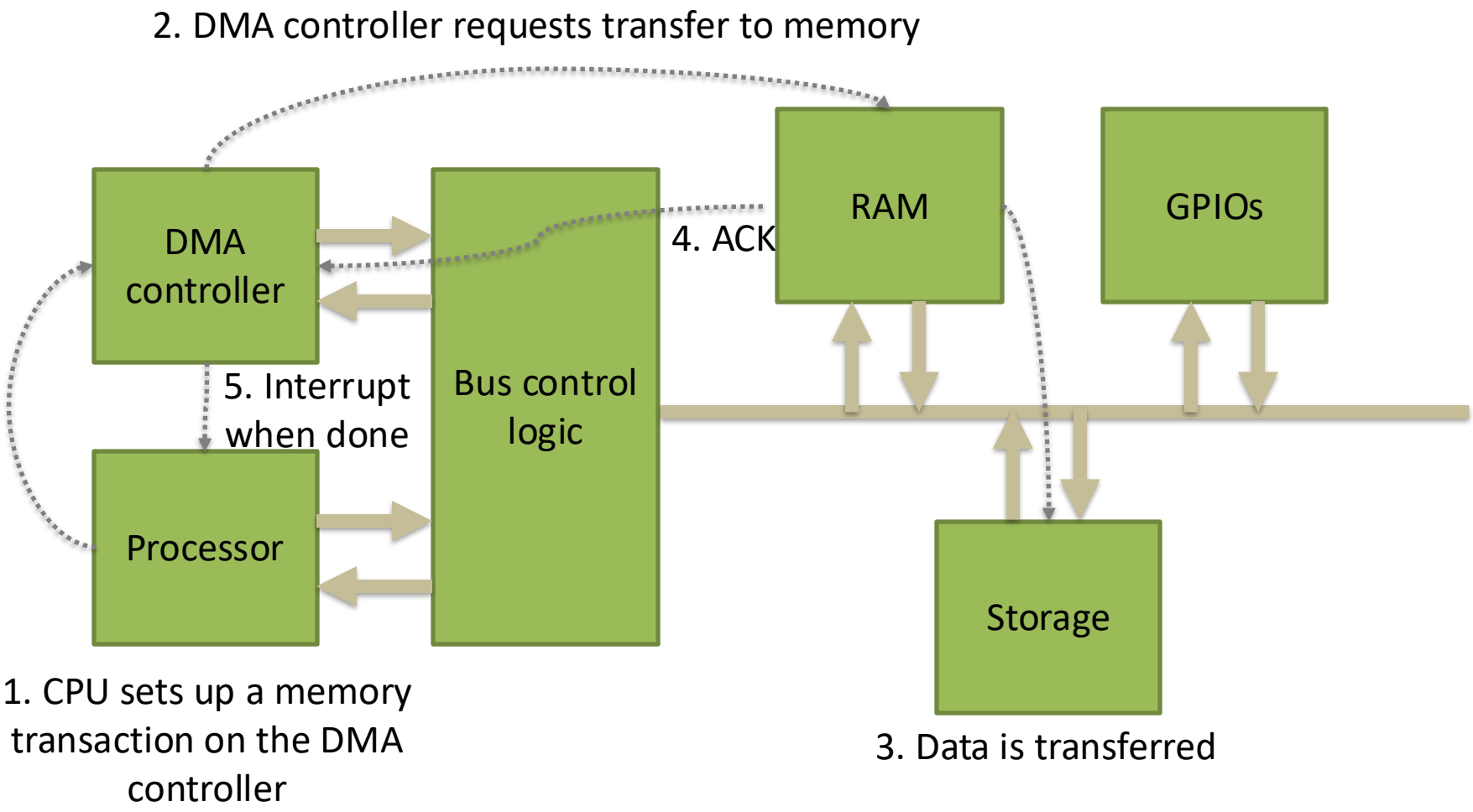
Figure 1. System architecture for STM32L47x/L48x devices



The DMA Transaction in a nutshell

1. Device wishing to perform DMA asserts the microcontroller's bus request signal.
2. Processor completes the current bus cycle and then asserts the bus grant signal to the device.
3. The device then asserts the bus grant ack signal.
4. The DMA device performs the transfer from the source to destination address.
5. Once the DMA operations have been completed, the device releases the bus by asserting the bus release signal.
6. Microcontroller acknowledges the bus release and resumes its bus cycles from the point it left off.

Use of DMA Controllers



Buffers and Arbitration

- Most DMAs have a data storage buffer – peripherals can send a burst of data faster than RAM memory can handle (as long as this happens infrequently)

- Bus Arbitration is needed to resolve conflicts with more than one device (2 DMAs or DMA and processor, etc..) try to use the bus to access main memory.

Bus Arbitration

- Bus Primary – the device that is allowed to initiate bus transfers on the bus at any given time. When the current primary relinquishes control, another device can acquire this status.

- Bus Arbitration – the process by which the next device to become bus primary is selected and bus primaryship is transferred to it.

Arbitration Approaches

- Centralized – a single arbiter performs the arbitration.
- Distributed – all devices participate in the selection of the next bus master.

Cache coherency problems

Imagine a CPU equipped with a cache and an external memory that can be accessed directly by devices using DMA. When the CPU accesses location X in the memory, the current value will be stored in the cache. Subsequent operations on X will update the cached copy of X, but not the external memory version of X, assuming a write-back cache. If the cache is not flushed to the memory before the next time a device tries to access X, the device will receive a stale value of X.