

# ARM systems

Matteo SONZA REORDA  
Dip. Automatica e Informatica  
Politecnico di Torino

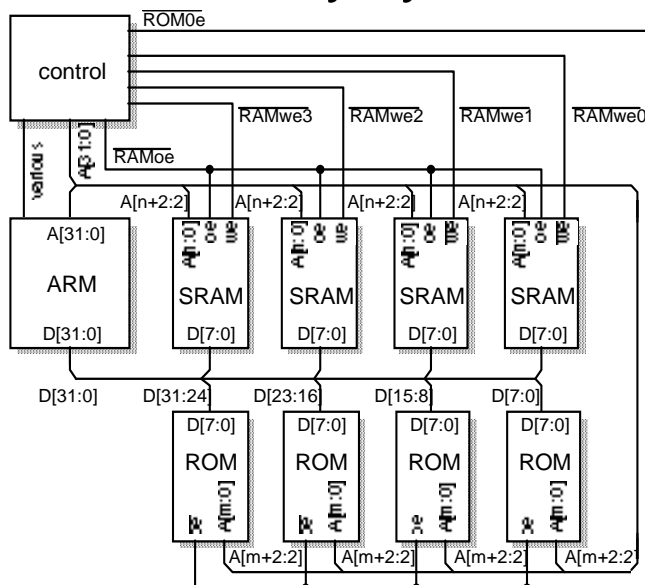
## The ARM memory interface

- It includes
  - A 32-bit address bus,  $A[31:0]$
  - A 32-bit bidirectional data bus,  $D[31:0]$
  - Several control signals to specify
    - whether the memory is needed ( $mreq$ ) and whether the address is sequential ( $seq$ ); these signals are issued by the processor during the previous cycle
    - the direction ( $r/w$ ) and size ( $b/w$ , or  $mas[1:0]$ ) of the transfer
    - Some timing and control information ( $abe$ ,  $ale$ ,  $ape$ ,  $dbe$ ,  $lock$ ,  $bl[3:0]$ ).
- In most recent ARM cores, this interface evolves into the AMBA bus.

# Simple memory interface

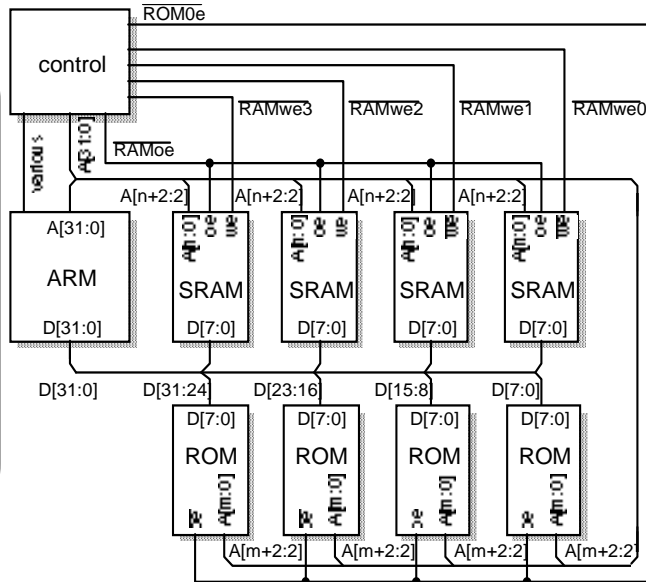
- This form is suitable to connect
  - ROMs
  - Static RAMs
- In this case
  - the address must be stable until the end of the cycle
  - Address and data buses may directly be connected to the memory modules.

# Simple ARM memory system



# Simple ARM memory system

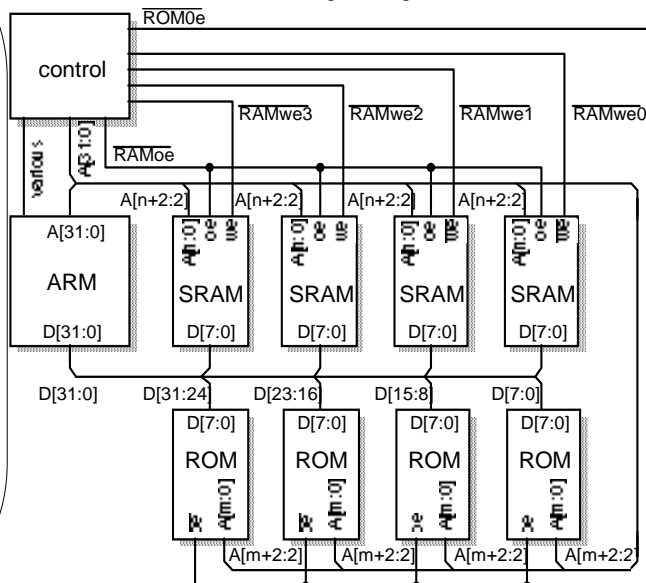
- Read operation always involve all the memory modules
- Write operations (on RAMs) only involve the right modules



Matteo SONZA REORDA

# Simple ARM memory system

- The control logic:
- Decides whether to activate the RAM or RAM bank
  - Controls the write enable signals
  - Ensures that the data is ready before the processor continues.

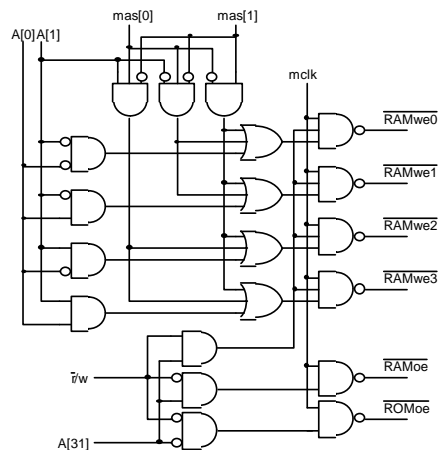


Matteo SONZA REORDA

# Control logic

Assumption:

- ROM occupies the lowest part of the memory, RAM the highest



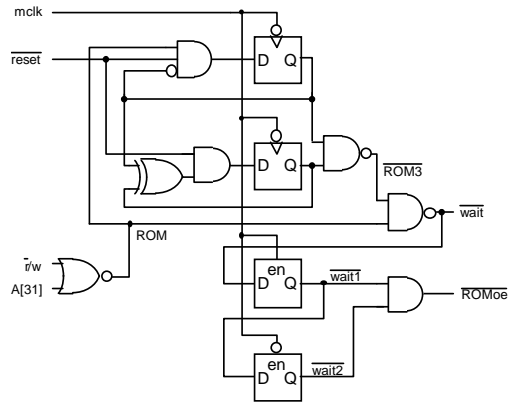
# Wait states

- If the different types of memories have different access times, it could be convenient not to tune the system clock to the slowest of them
- Alternatively, the *wait* signal may be asserted when a slow memory module is accessed
- When asserted, *wait* forces the processor to stretch the access time correspondingly.

# ROM wait state generator

Assumption:

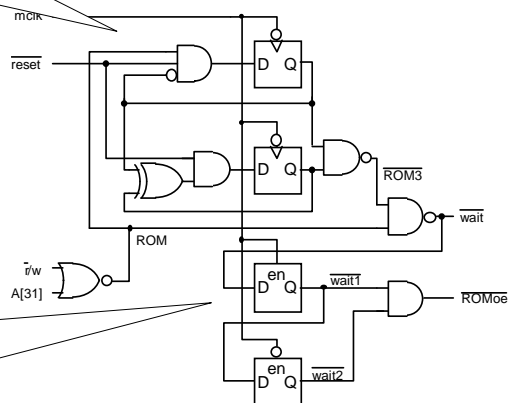
- The ROM access time corresponds to 4 clock cycles.



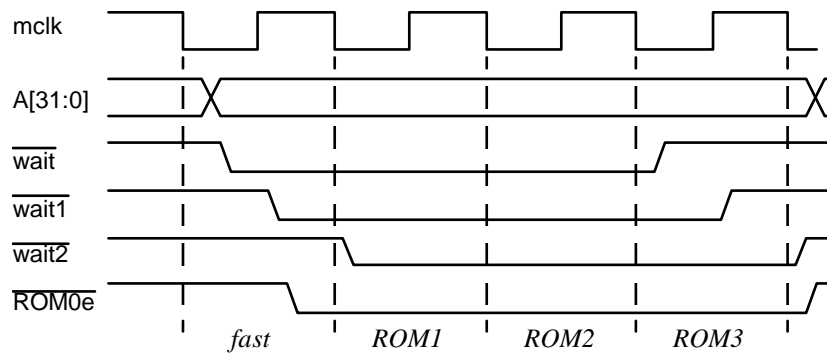
This circuitry implements a **wait state generator**

- The ROM access time corresponds to 4 clock cycles.

This circuitry is used to generate a stretched glitch-free ROMoe signal.



## Timing diagram for ROM accesses



## Peripheral devices

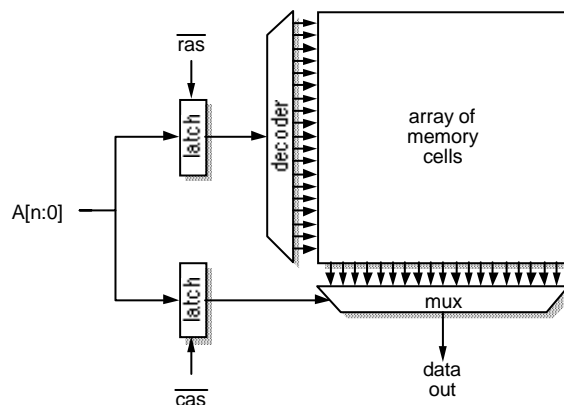
- Since they are connected using a memory-mapped scheme, access to them can be implemented as for the memory
- Since they are normally rather slow, a mechanism for generating the *wait* signal such as the one presented for the ROM may be exploited.

# DRAM accesses

- DRAM is currently the cheapest semiconductor memory technology
- Unfortunately, DRAMS
  - are slower than SRAMs, so they tend to slow-down the whole system
  - Require a specific access protocol, taking into account their internal architecture
  - Require refreshing cycles
- The memory controller in charge of DRAMs is more complex than for SRAMs.

# DRAM architecture

1. The row address is provided and latched, together with the RAS signal
2. The column address is provided and latched, together with the CAS signal
3. The memory matrix is accessed, and the operation performed.



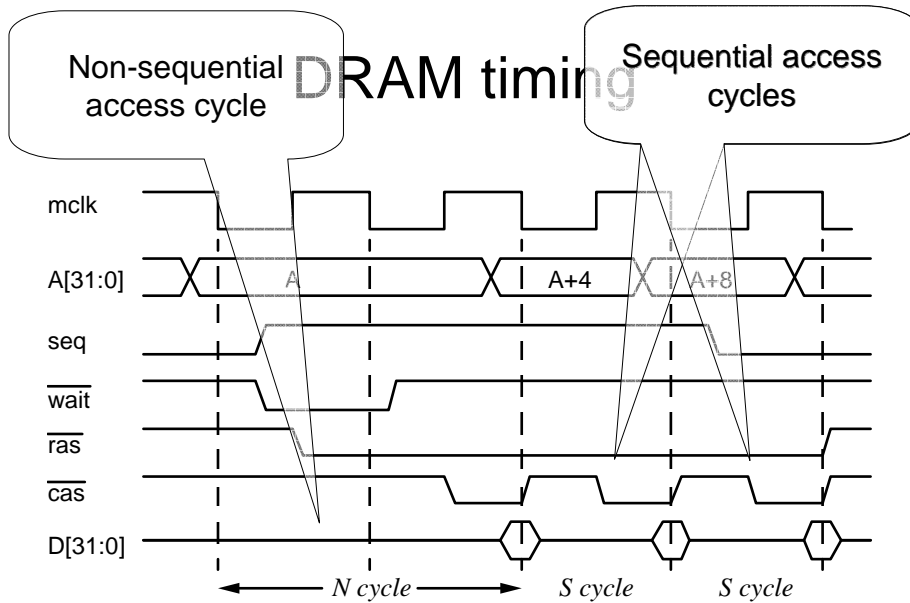
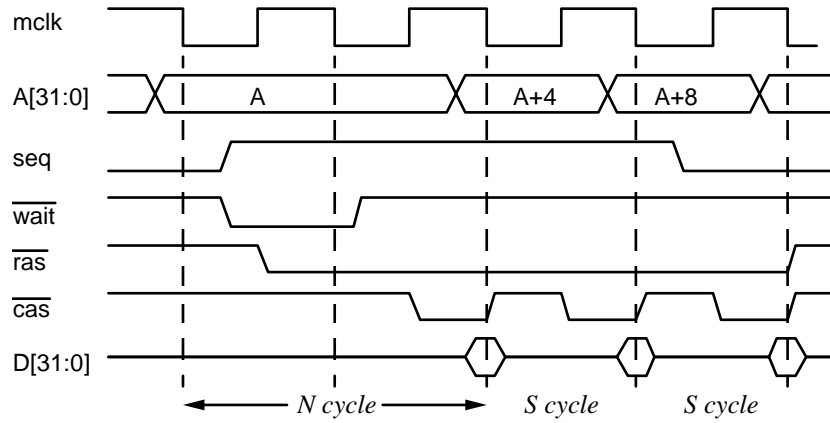
## Sequential accesses

- They correspond to the case, in which some consecutive accesses refer to data belonging to the same row
- This case amounts to about 75% of memory cycles
- In this case
  - Only the column address must be provided (together with the CAS signal)
  - The same row than in the previous access is accessed
  - The resulting access time is significantly lower.

## *seq* signal

- When the ARM processor produces an address corresponding to the incremented (or same) version of the previous one, it asserts the *seq* signal
- If the *seq* signal is asserted and we are not at the end of a row, a *cas*-only memory access can be performed
- Both the *seq* signal and the address are available half a clock cycle before the real memory cycle.

# DRAM timing



## Memory controller

- It is in charge of
  - Receiving the address and control signals from the CPU
  - Transforming them into the proper sequence of RAS, CAS and address signals to the memory.
- When the *seq* signal is activated a CAS-only access is performed, provided that the sequence does not cross different rows.

## Bus architecture

- ARM released a standard bus architecture (named AMBA, or *Advanced Microcontroller Bus Architecture*) to be used for developers of cores to be connected to ARM processors
- By following these specifications, the designer may produce cores that can easily be re-used in other ARM systems.

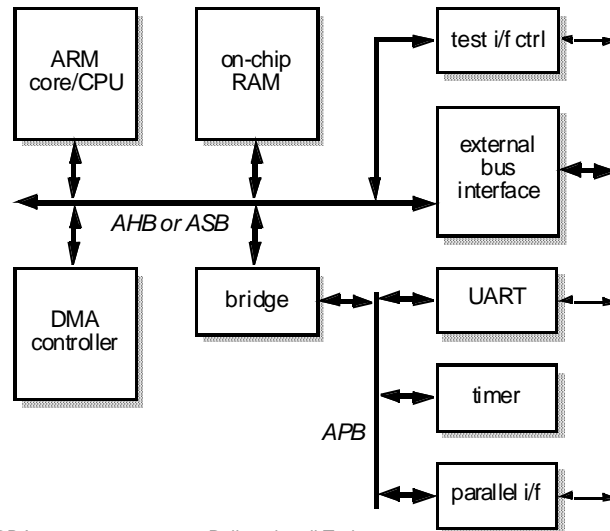
# Bus architecture

- The AMBA specification includes 3 busses:
  - The *Advanced High-performance Bus* (AHB): it is used to connect high-performance modules. It supports burst mode data transfers and split transactions. All timing is referenced to a single clock edge.

## Bus architecture (II)

- The *Advanced System Bus* (ASB): it is an old specification, to be substituted by AHB
- The *Advanced Peripheral Bus* (APB): offers a simpler interface for low-performance peripherals. APB is generally used as a local secondary bus which appears as a slave module on the AHB.

## Typical AMBA-based system



Matteo SONZA REORDA

Politecnico di Torino

23

## Bus arbitration

- Arbitration is performed in a centralized way using as many couples of signals  $AREQ_x/AGNT_x$  as the modules connected on the AHB
- The policy implemented by the arbiter is not specified by the standard.

Matteo SONZA REORDA

Politecnico di Torino

24

## Arbitration protocol

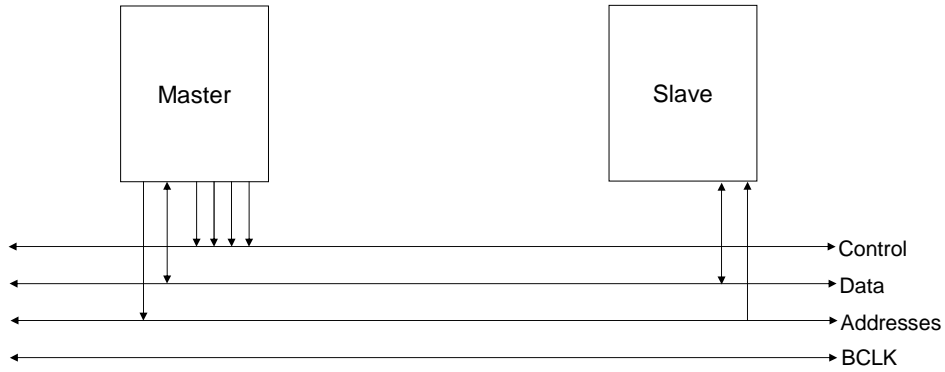
- A master  $x$  issues a request to the arbiter activating  $AREQ_x$
- When the bus is available, the arbiter issues a grant ( $AGNT_x$ ) to the master, which takes control of the bus.

A suitable signal ( $BLOK$ ) is available to guarantee atomic bus transactions.

## ASB

- ASB is used to connect the processor with memory and sophisticated peripherals.

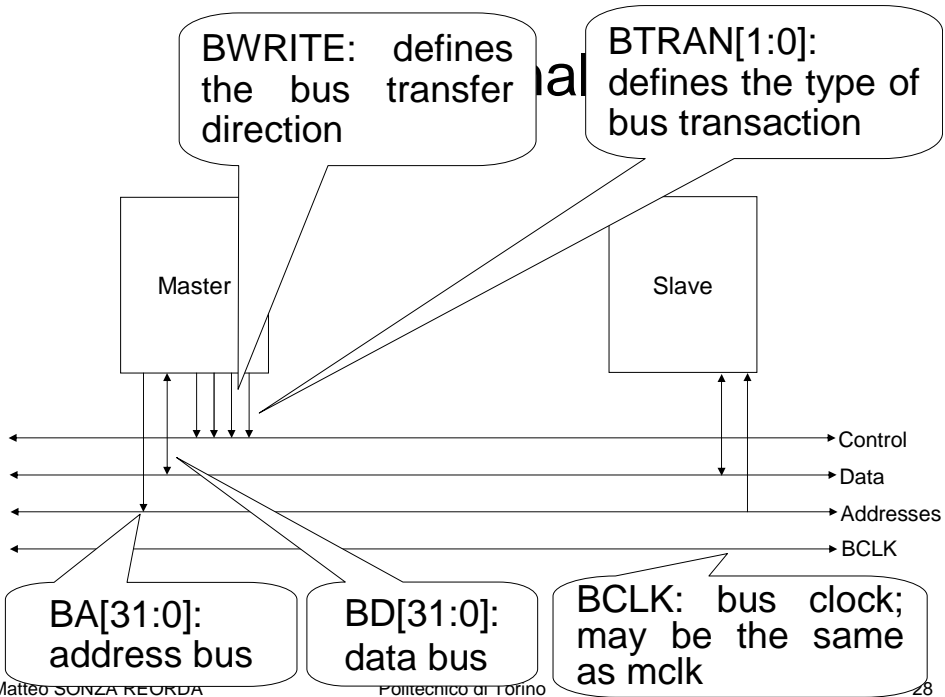
# ASB signals



Matteo SONZA REORDA

Politecnico di Torino

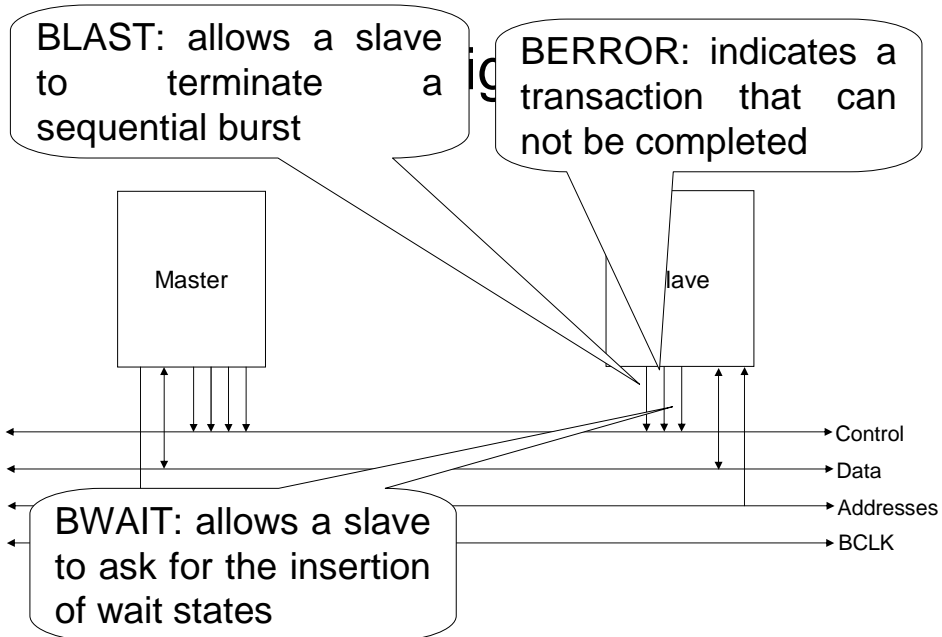
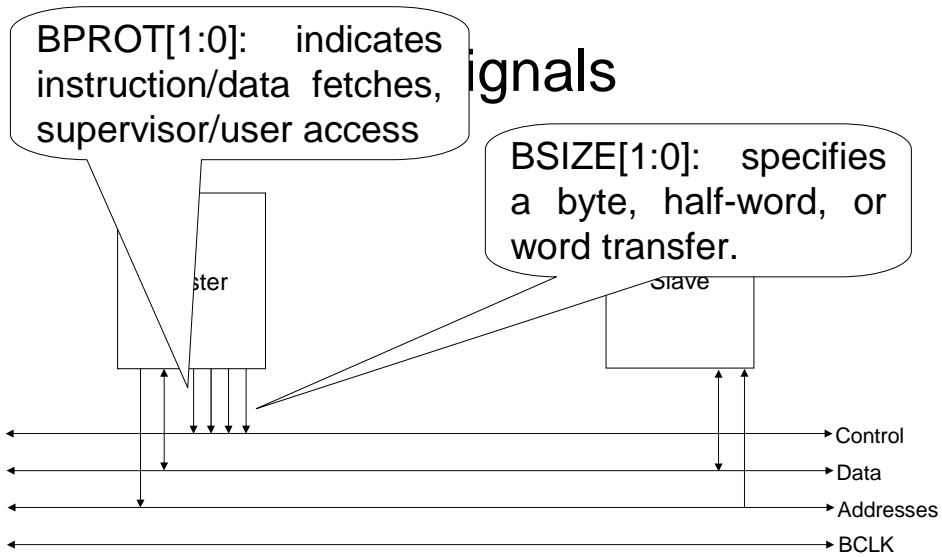
27



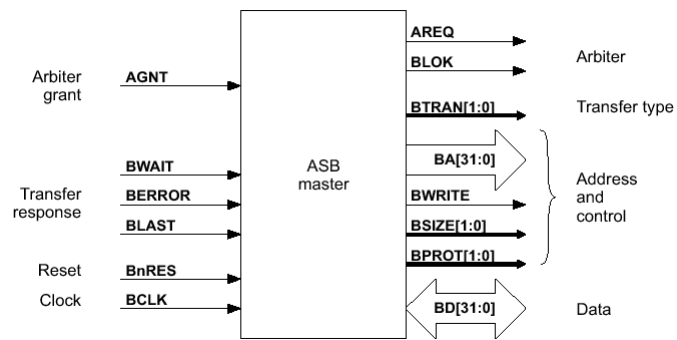
Matteo SONZA REORDA

Politecnico di Torino

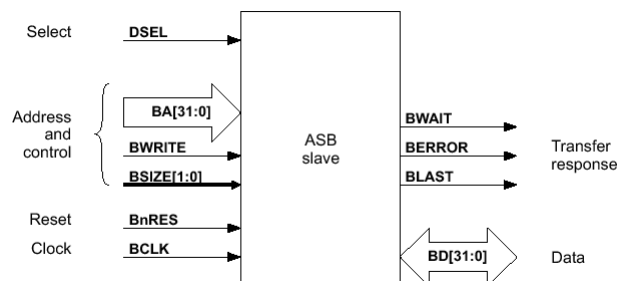
28



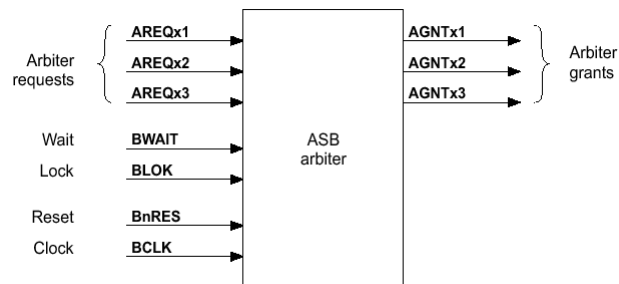
# ASB Master



# ASM Slave



# ASB Arbiter



## Test Interface Controller

- It allows accessing to single modules on the internal bus from the outside
- It appears as a bus master on the ASB
- It accesses the ASB through a 32-bit bidirectional port
- In most current ARM cores the controller is integrated in the standard JTAG interface for test.

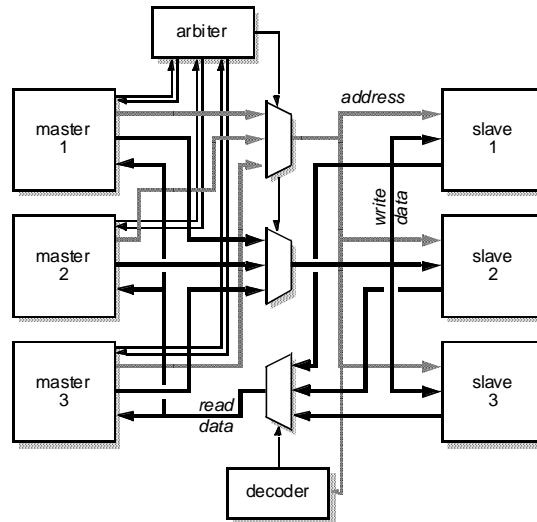
# APB

- The Advanced Peripheral Bus is used to connect simple peripheral modules
- It includes the following signals (the control signals are generated by the bridge)
  - *PADDR*[*n*:0]: address bus ( $n \leq 32$ )
  - *PRDATA*[*m*:0]: data bus ( $m=7, 15, \text{ or } 31$ )
  - *PWRITE*: direction indicator
  - *PSEL*<sub>*x*</sub>: individual peripheral select strobe
  - *PENABLE*: peripheral timing strobe
  - *PCLK*: APB clock
  - *PRESET*<sub>*n*</sub>: APB reset.

# AHB

- The *Advanced High-performance Bus* replaces the ASB in high performance systems
- Differences wrt ASB:
  - AHB supports split transactions
  - AHB has a single clock edge
  - AHB uses a centrally multiplexed bus scheme, rather than a bidirectional bus
  - AHB supports wider data bus configurations (64 or 128 bits).

# AHB architecture



Matteo SONZA REORDA

Politecnico di Torino

37

## AMBA reference peripheral specification

- If a system developer wishes to develop a system able to more easily support an existing operating system, he should follow the ARM reference peripheral specification, that defines the following components:
  - A memory map
  - An interrupt controller
  - A counter timer
  - A reset controller.

Matteo SONZA REORDA

Politecnico di Torino

38

# ARMulator

- This tool allows software to be developed before a system is manufactured
- It allows building a clock-cycle accurate software model of an ARM system including
  - Caches
  - MMU
  - Memory
  - Peripherals.

## Debugging mechanism

- Debugging a SoC is particularly difficult, since the developer has no access to internal signals and the code is often written in a ROM
- ARM provides a debug solution based on
  - An *embeddedICE* module, that can be programmed to halt the processor when a given instruction is executed
  - Exploiting the JTAG port for programming the embeddedICE and accessing internal core elements
  - An *embedded trace macrocell* that allows tracing the values passing on the busses.

