CSE/ELE 664
Intro. To SoC Design

Lecture 24

ENGINEERING@SYRACUSE

1

Cortex-M3 Program Image

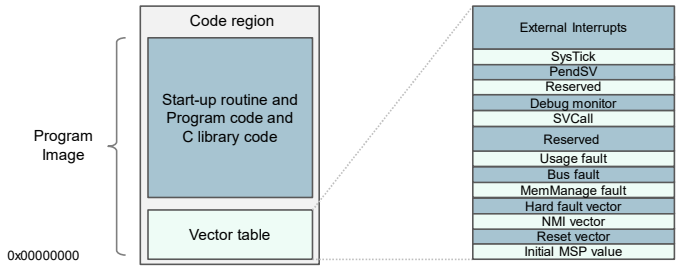
ENGINEERING@SYRACUSE

2

Cortex-M3 Program Image

The program image in Cortex-M3 contains

- Vector table: includes the starting addresses of exceptions (vectors), and the value of the main stack pointer (MSP)
- C start-up routine
- Program code: application code and data
- C library code: program codes for C library functions

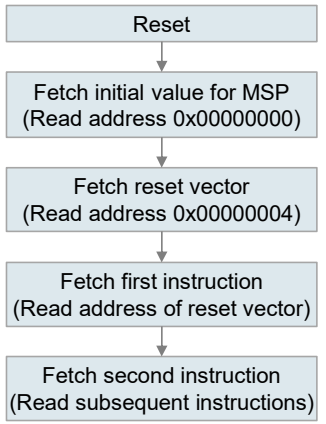


3

Cortex-M3 Program Image (cont.)

After reset, the processor:

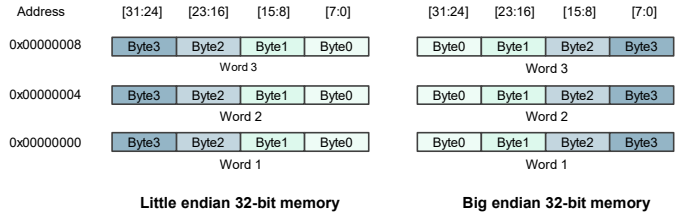
- First reads the initial MSP value
- Then reads the reset vector
- Branches to the start of the program execution address (reset handler)
- Subsequently executes program instructions



4

Cortex-M3 Endianness

- Endian refers to the order of bytes stored in memory
  - Little endian: lowest byte of a word-size data is stored in bit 0 to bit 7
  - Big endian: lowest byte of a word-size data is stored in bit 24 to bit 31
- Cortex-M3 supports both little endian and big endian



5

Cortex-M3 Instruction Set

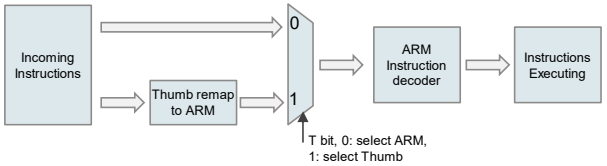
ENGINEERING@SYRACUSE

6

ARM and Thumb Instruction Set

Some processors support both ARM and Thumb code

- Benefit from both 32-bit ARM (high performance) and 16-bit Thumb-1 (high code density)
- A multiplexer is used to switch between two states: ARM state (32-bit) and Thumb state (32-/16-bit), which can be done using a branch



7

Cortex-M3 Instruction Set, Part I

ARM assembly syntax:

label

mnemonic operand1, operand2, … ; Comments

- Label is used as a reference to an address location
- Mnemonic is the name of the instruction
- Operand1 is the destination of the operation
- Operand2 is normally the source of the operation
- Comments are written after " ; ", which does not affect the program
- For example:

  MOVS R3, #0x11 ;Set register R3 to 0x11

- Note that the assembly code can be assembled by either ARM assembler (armasm), or assembly tools from a variety of vendors (e.g., GNU tool chain); when using GNU tool chain, the syntax for labels and comments is slightly different

8

Cortex-M3 Instruction Set, Part II

Functional grouping of instructions

- Load and store (memory accesses)
- Arithmetic (using architectural registers and flags)
- Branch (to control program flow)
- Special instructions (low power states, debug, and others)

9

Data Insertion and Alignment

- Insert data inside programs
  - DCD: insert a word-size data
  - DCB: insert a byte-size data
  - ALIGN:
    - Used before inserting a word-size data
    - Uses a number to determine the alignment size
- For example:

```
…
ALIGN               4                   ; Align to a word boundary
MY_DATA             DCD     0x12345678  ; Insert a word-size data
MY_STRING           DCB     “Hello”, 0  ; Null terminated string
…
```

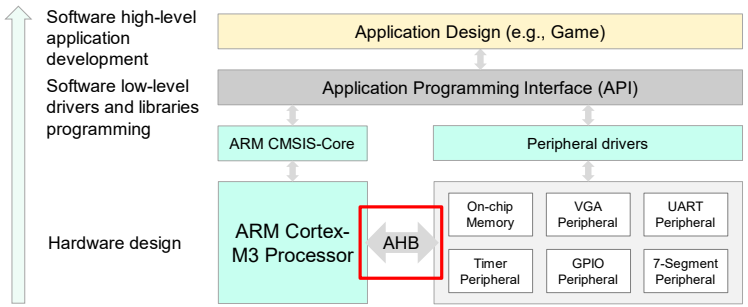
10

On-Chip Bus Architecture

ENGINEERING@SYRACUSE

11

Building a System on a Chip



12

What Is Bus?

- Traditionally, a bus is a communication system that allows data to be transferred between different components in a system.
- The infrastructures is defined in both hardware and software:
  - Hardware infrastructure includes the physical implementation, such as cables or wires; e.g., the PCI uses a PCI cable to connect components inside a desktop.
  - Software infrastructure includes the bus protocol, e.g., PCI bus protocol.

PCI socket on a mother board

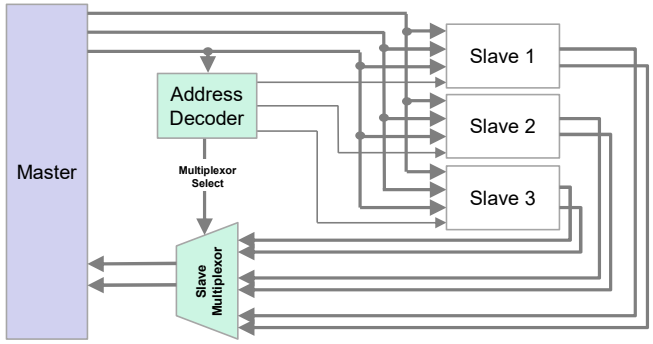
PCI bus cable

Should not use an off-chip bus architecture for on-chip communication or vice versa

	Off-chip bus	On-chip bus
Examples	PCI, PCI-X, PCI-express	CoreConnect, AMBA
Width	Cannot have many wires (cost limitations), serial bus, time multiplexed, or tri-state bus	Wide bus is acceptable
Synchronization	Needs extra effort for synchronization	Synchronized

13

Bus Terminology



Bus transaction: either read or write operations

Master: the module that initiates the transaction request

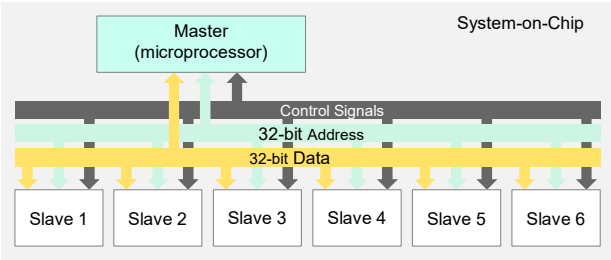
Slave: the module that responds to the request

14

Bus Operation in General

A bus typically consists of three types of signal lines:

1. **Data bus** is used to exchange data information.
2. **Address bus** is used to select one of the peripherals (or one register of a peripheral).
3. **Control signals** are used to synchronize and identify transactions, such as ready, write/read, transfer mode signals.

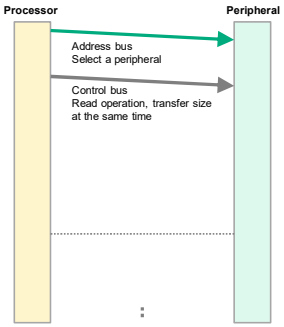


15

A Typical Bus Operation Example, Part I

A typical operation to access a peripheral mainly consists of:

1. The master (e.g., a processor) selects one peripheral (or one register) by giving the address to the address bus. At the same time, it sets control signals, such as read or write, transfer size, and so forth.

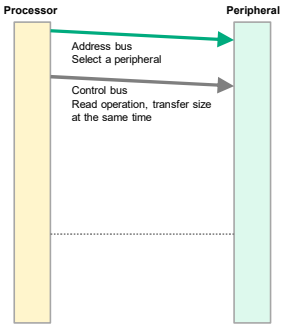


16

A Typical Bus Operation Example, Part II

A typical operation to access a peripheral mainly consists of:

1. The master (e.g., a processor) selects one peripheral (or one register) by giving the address to the address bus. At the same time, it sets control signals, such as read or write, transfer size, and so forth.
2. The master waits for the slave (e.g., peripheral) to respond.

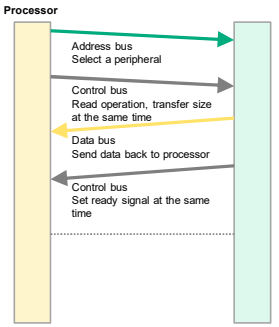


17

A Typical Bus Operation Example, Part III

A typical operation to access a peripheral mainly consists of:

1. The master (e.g., a processor) selects one peripheral (or one register) by giving the address to the address bus. At the same time, it sets control signals, such as read or write, transfer size, and so forth.
2. The master waits for the slave (e.g., peripheral) to respond.
3. Once the slave is ready, it sends back the requested data to the processor; at the same time, it sets the ready signal on the control bus.

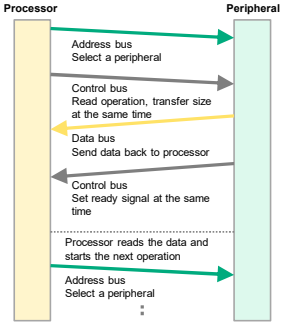


18

A Typical Bus Operation Example, Part IV

A typical operation to access a peripheral mainly consists of:

1. The master (e.g., a processor) selects one peripheral (or one register) by giving the address to the address bus. At the same time, it sets control signals, such as read or write, transfer size, and so forth.
2. The master waits for the slave (e.g., peripheral) to respond.
3. Once the slave is ready, it sends back the requested data to the processor; at the same time, it sets the ready signal on the control bus.
4. Finally, the master reads the transmitted data, and starts another communication cycle.

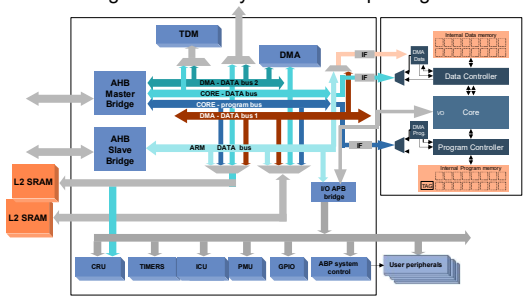


19

Communication Architecture Standards

What do we need communication standards for?

- Modular design approach
- Allows design reuse
- Facilitates IPs' integration into a system on a chip design



Picture source: http://www.ecs.soton.ac.uk/ (SoC Advance design Technique)

20

Overview of ARM AMBA

ENGINEERING@SYRACUSE

21

ARM AMBA System Bus

AMBA: Advanced Microcontroller Bus Architecture

- AMBA protocol is an open standard on-chip interconnect specification
- Provides the interface standard that enables IP re-use
- Facilitates right-first-time development of multi-processor designs with large numbers of controllers and peripherals
- Widely used in modern portable mobile devices, such as tablets and smartphones

AMBA®
Interconnect Standards from ARM

22

ARM AMBA Bus Families

AMBA family	Bus protocol
AMBA 5	CHI
	AHB5, AHB-Lite
AMBA 4	ACE, ACE-Lite
	AXI4, AXI4-Lite, AXI4-Stream
AMBA 3	AXI
	AHB **(AHB-Lite)**
	APB
	ATB
AMBA 2	AHB, APB
AMBA 1	ASB, APB

As of Sept 2017

23

AMBA® 3 AHB-Lite Bus Architecture

ENGINEERING@SYRACUSE

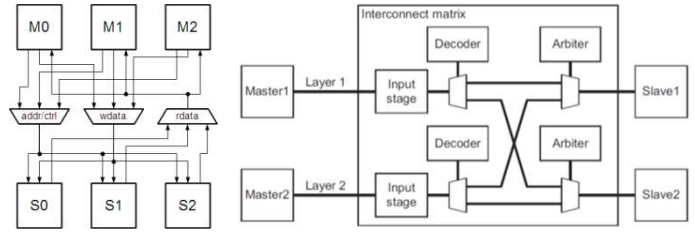
24

AMBA 3 AHB-Lite Bus

- AHB: Advanced High-Performance Bus
  - High-performance synthesizable designs
  - Supports multiple bus masters arbitrated to access a single channel
  - Provides high-bandwidth operation
- AHB-Lite:
  - A subset of AHB
  - Simplifies the design of AHB bus, e.g., typically with a single master
  - Multilayer implementations allow several masters to access several slaves simultaneously

25

Multilayer AHB



Multiplexed AHB
(AMBA 2)

One transaction at a time

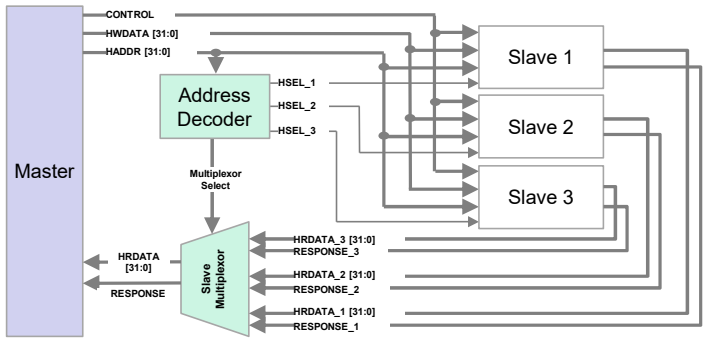
Multilayer AHB
(AMBA 3)

Different masters can access multiple different slaves simultaneously

Adapted from: Rabeay, J. 2003. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall/Pearson.

26

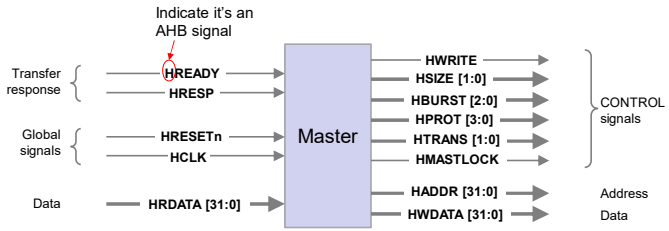
AHB-Lite Bus Block Diagram



27

AHB-Lite Master Interface

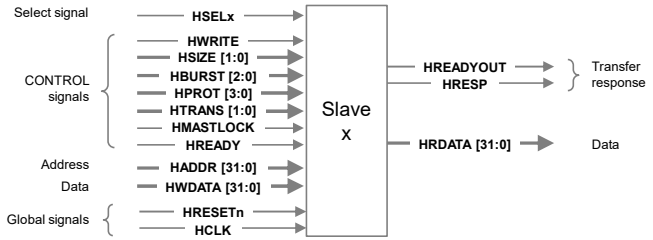
- The AHB-Lite master provides address and control information to initiate read and write operations.
- The master also receives the response from the slave, including data, ready and response signal.



28

AHB-Lite Slave Interface

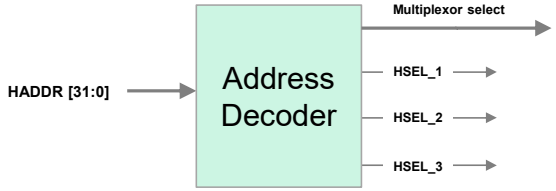
- An AHB-Lite slave responds to transfer initiated by the master in the system.
- The signal HSELx is the output from the address decoder, which is used to select one of the slaves at one time.



29

Address Decoder

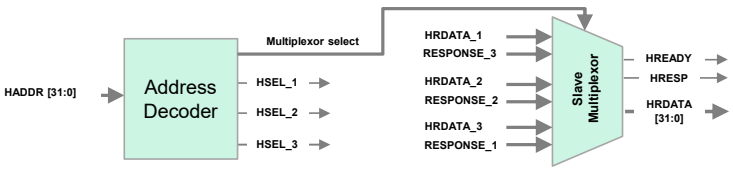
- Selects one of the slaves depending on the current address bus
- Also informs the slave multiplexor



30

Slave Multiplexor

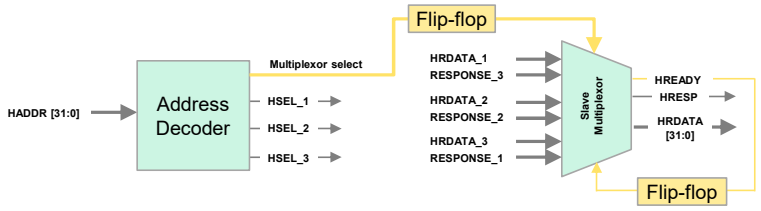
- Inputs the response signals (HRDATA, HREADY and HRESP) from all the slaves, and outputs one of them depending on the selecting signal from the address decoder



31

Hardware Implementation

- Due to the pipelined operation, some signals have to be deliberately delayed including:
  - The selecting signals from the decoder to the multiplexor are delayed for one clock cycle
  - The HREADY signal is delayed for one clock cycle before it is fed back to the multiplexor
- The detailed implementation can be referred in the code



32