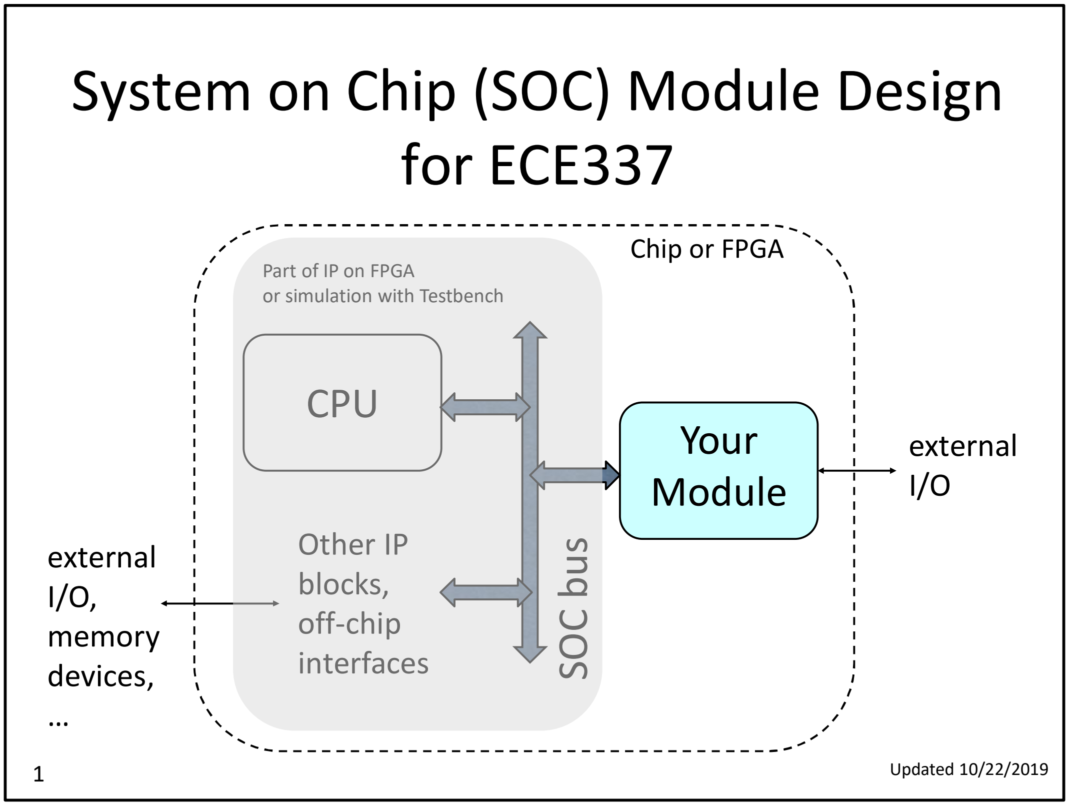


System on Chip (SOC) Module Design for ECE337

Chip or FPGA
Part of IP on FPGA
or simulation with Testbench
CPU
Your Module
external I/O
Other IP blocks, off-chip interfaces
SOC bus
external I/O, memory devices, …

Updated 10/22/2019

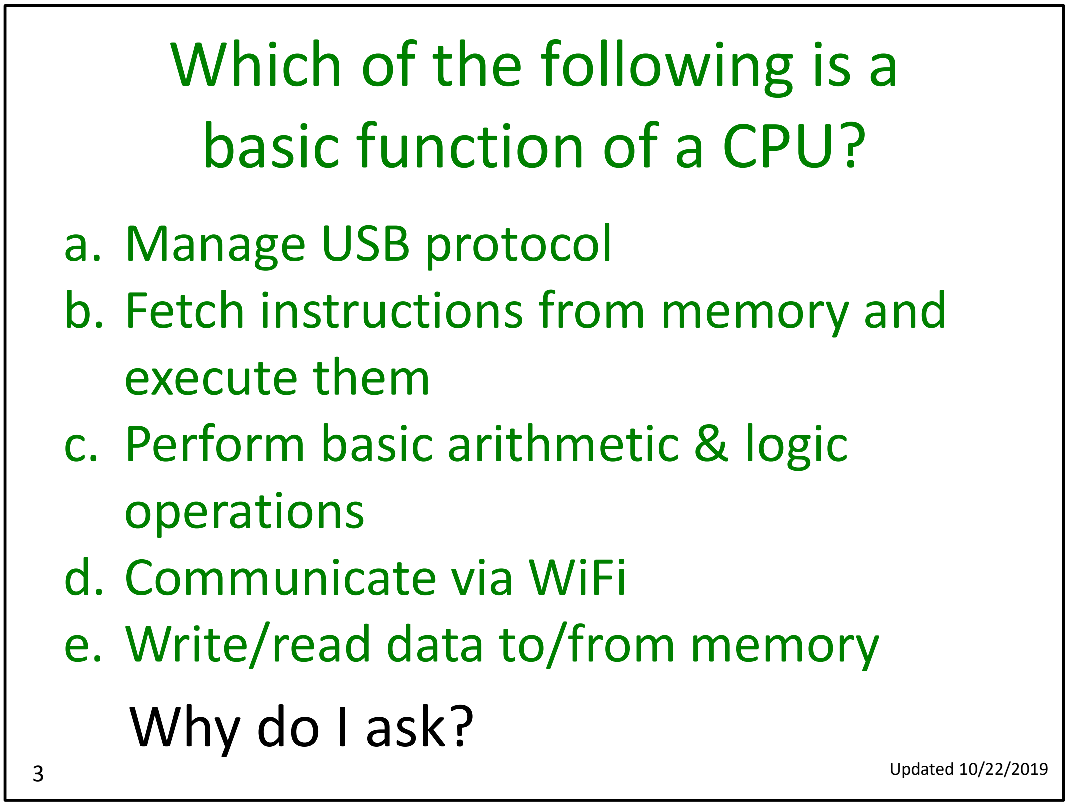
This illustration presents the usual scope of an SOC oriented ECE337 project design. In general you will be designing a single module that is require to communicate via something generically called an SOC or system bus. You will use your test bench or existing IP blocks to simulate or implement other parts of an SOC.

Goals, Expectations

Part 1. Understand operation, architecture of a minimalist SoC (mostly lecture)

Part 2. Understand operation of APB, AHB-lite SoC busses (lecture and lab)

Lab. Design some modules that work with an SoC bus.



Which of the following is a basic function of a CPU?

a. Manage USB protocol
b. Fetch instructions from memory and execute them
c. Perform basic arithmetic & logic operations
d. Communicate via WiFi
e. Write/read data to/from memory

Why do I ask?

Updated 10/22/2019

If you don’t have an idea what can/cannot be done by a CPU, you will not understand the purpose of an SoC bus which is to augment the capabilities of a CPU.

What signals/busses are typically needed to use memory?

a. Address

b. Data

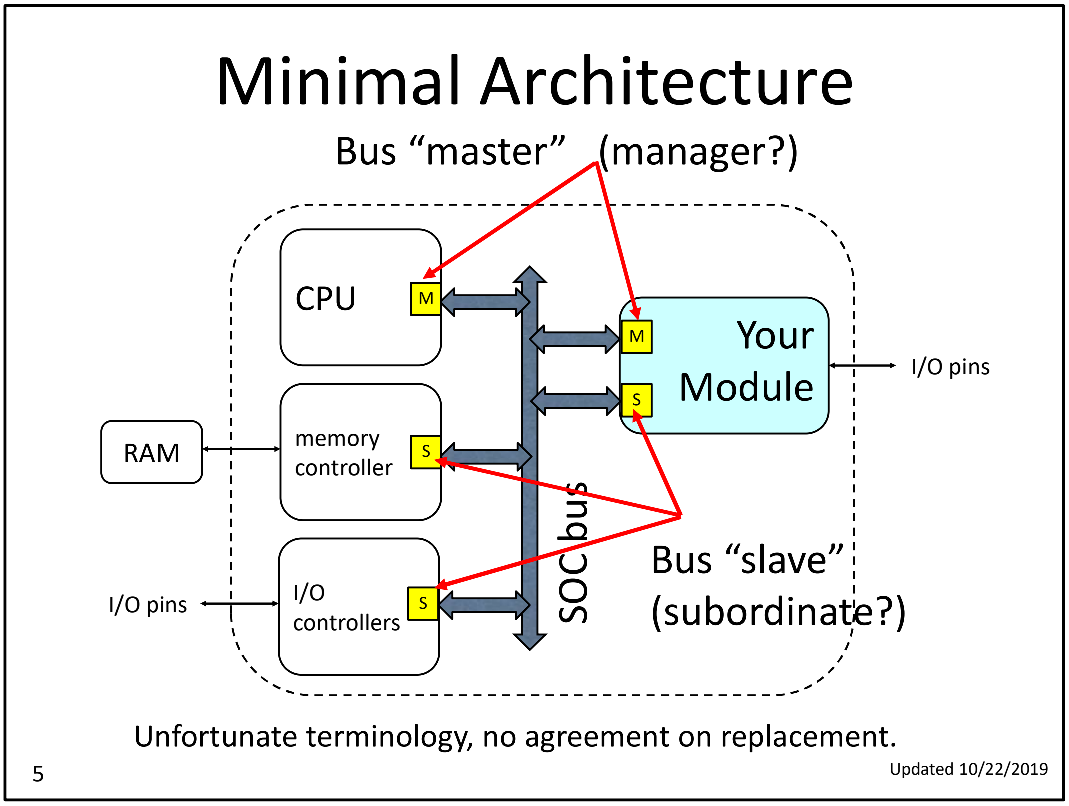
c. Read enable

d. Write enable

e. Shift enable

Multiple are correct.
Choose any one that is.

Communication with memory is similar to communicating with devices on an SoC bus



Minimal Architecture

Bus “master” (manager?)

CPU

M

memory controller

S

I/O controllers

S

RAM

I/O pins

SOC bus

M

S

Your Module

I/O pins

Bus “slave” (subordinate?)

Unfortunate terminology, no agreement on replacement.

5

Updated 10/22/2019

CPU – On every SOC there is a processor, commonly something in the ARM family of processors as usually found on most mobile devices. All input and output to the CPU except possibly for clock, reset, and interrupts goes through the SOC bus. The CPU reads and writes memory using the SOC bus. It sends and receives data on I/O pins by sending commands to various I/O controllers by way of the SOC bus. The module you are creating for your ECE337 project receives read or write commands from the CPU by via the SOC bus.

If you are prototyping your design on the de2i-150 boards in EE215, one of two situations are likely. 1. The Atom processor on the development which is connected to the FPGA will take the place of the CPU. A PCIe bridge (you will learn about if you use the de2-150 tutorial) will connect the Atom to the SOC bus on the FPGA.

A note on clocking. For ECE337 projects, you will normally use the same clock for all components in the design. In large real world designs, you are likely to have multiple clocks due to the difficulty of distributing a very fast clock reliably across a large chip. However, multiple clocks can add considerably to the challenge of designing and verifying a design.

SOC bus – An SOC bus, in its simplest form, consists of a collection of wires that resemble a simple memory interface. It has address wires for the CPU to specify which slave device (memory, I/O, your module, etc) it wants to send or receive data from. It has wires for the data to be written to a slave device. It has another set of wires for data to be read from a slave device. Finally, the SOC bus has wires for control signals. The simplest control signals are a “write” signal to indicate that data is to be written to a slave device and a “read” signal to indicate that data is to be read from the slave device. Depending on the particular SOC bus standard you are using, there will be other control signals that determine the mode of operation of the bus, particular kinds of read/write transactions, etc.

In practice, an SOC bus includes more than just wires. It also includes address decoding logic which provides a signal to each slave device to indicate when that device is the destination of a bus transaction. The bus may also include multiplexing logic in order to avoid the need for bidirectional tri-state wires. In ASIC design, one usually avoids tri-state outputs internal to the chip as such interfaces are slower and demand more power than unidirectional interfaces.

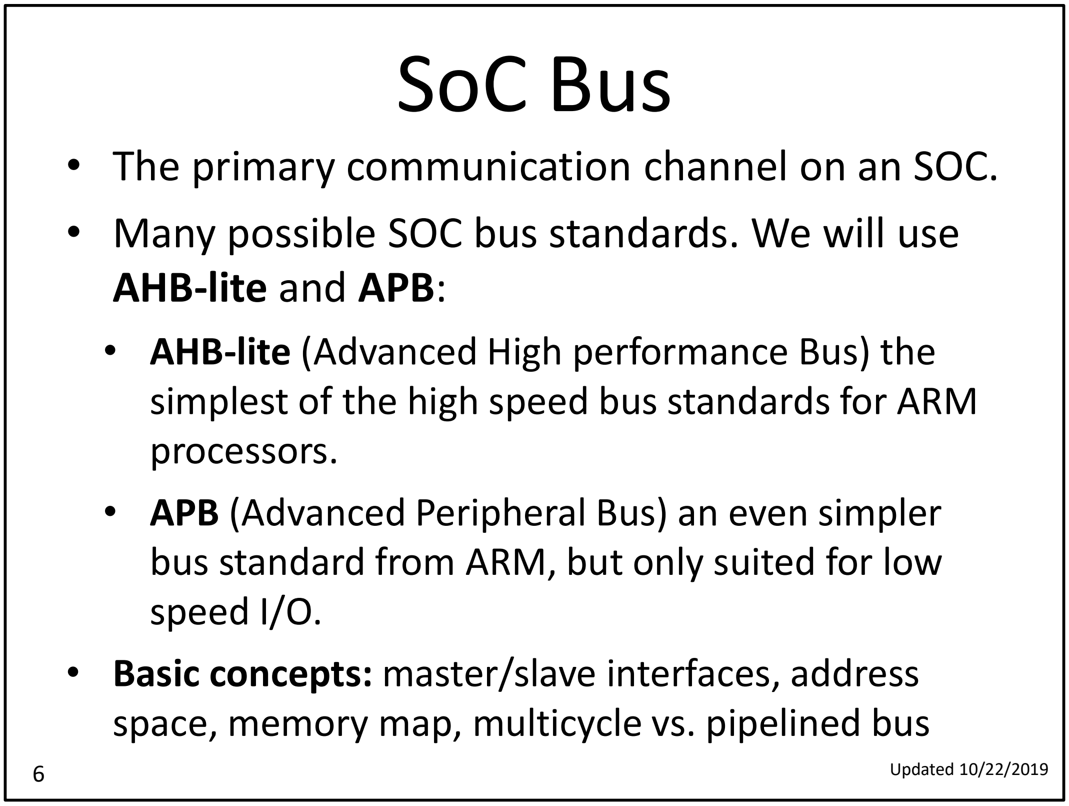
Bus master – a module used by or built into components that need to be able to issue read/write commands on the SOC bus. The simplest SOC's have only one bus master.

Bus slave – a module used by or built into components that only need to be able to respond to read or write commands from a bus master. Most modules connected to an SOC will act as a bus slave.

Your module – Your design will at least need a slave interface (unless it is itself a CPU), but you might also need a master interface if your module needs to directly read or write data to other devices connected to the SOC bus.

Memory controller – this is a slave device that enables the CPU or other bus masters to read and write RAM. The RAM could be internal or external, but in this illustration it is external to the SOC.

I/O controller – this is a slave device that either sends data to external I/O pins or reads data from external I/O pins in response to read or write commands on the SOC bus. There are numerous types of I/O controllers to handle the various types of I/O interfaces that may be found on an SOC (or microcontroller). Examples include general purpose I/O (GPIO), SPI, I2C, UART, etc.



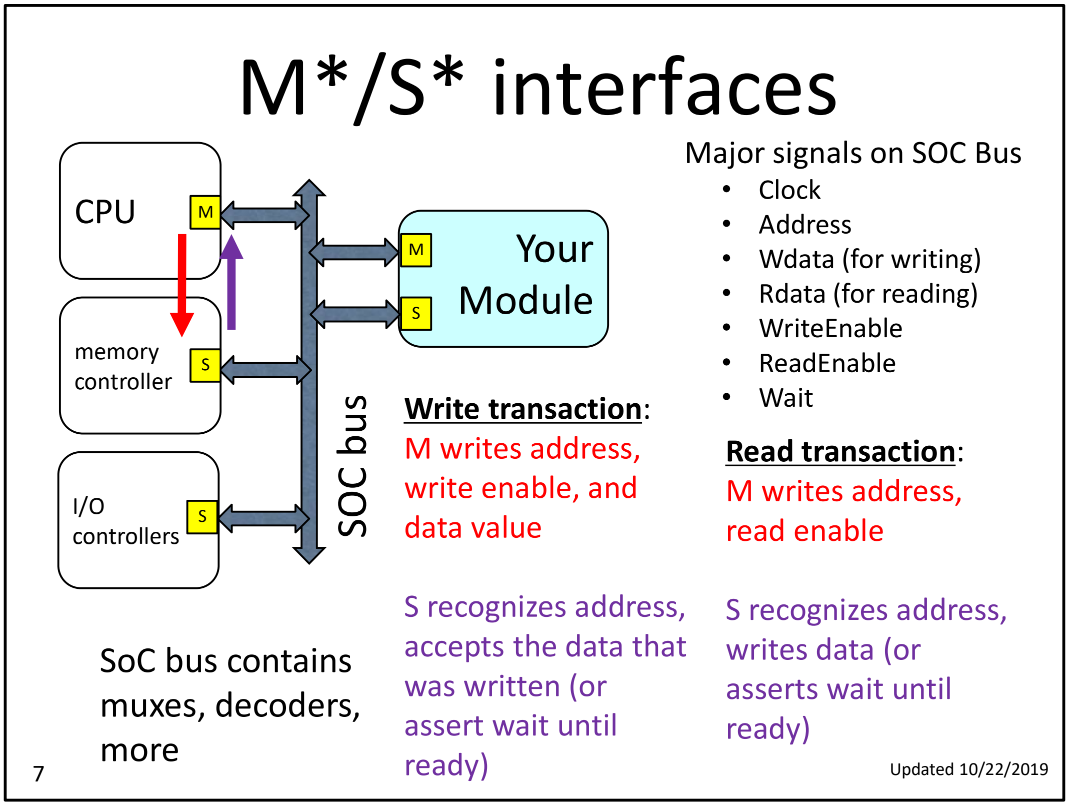
SoC Bus

- The primary communication channel on an SOC.
- Many possible SOC bus standards. We will use **AHB-lite** and **APB**:
  - **AHB-lite** (Advanced High performance Bus) the simplest of the high speed bus standards for ARM processors.
  - **APB** (Advanced Peripheral Bus) an even simpler bus standard from ARM, but only suited for low speed I/O.
- **Basic concepts:** master/slave interfaces, address space, memory map, multicycle vs. pipelined bus

Updated 10/22/2019

Avalon is actually a family of interface standards used on Intel (formerly Altera) FPGAs. The standard closest to what we will use in ECE337 is the Avalon Memory Mapped Interface (Avalon-MM). Avalon-MM is a general purpose SOC bus standard. Other types of Avalon busses are designed for special purposes such as streaming media data, external tri-state peripheral interfaces, etc.

AHB-lite and APB are two bus standards that are part a family of bus standards under the name Advanced Microcontroller Bus Architecture.



M*/S* interfaces

Major signals on SOC Bus

- Clock
- Address
- Wdata (for writing)
- Rdata (for reading)
- WriteEnable
- ReadEnable
- Wait

Write transaction:
M writes address, write enable, and data value

S recognizes address, accepts the data that was written (or assert wait until ready)

Read transaction:
M writes address, read enable

S recognizes address, writes data (or asserts wait until ready)

SoC bus contains muxes, decoders, more

7

Updated 10/22/2019

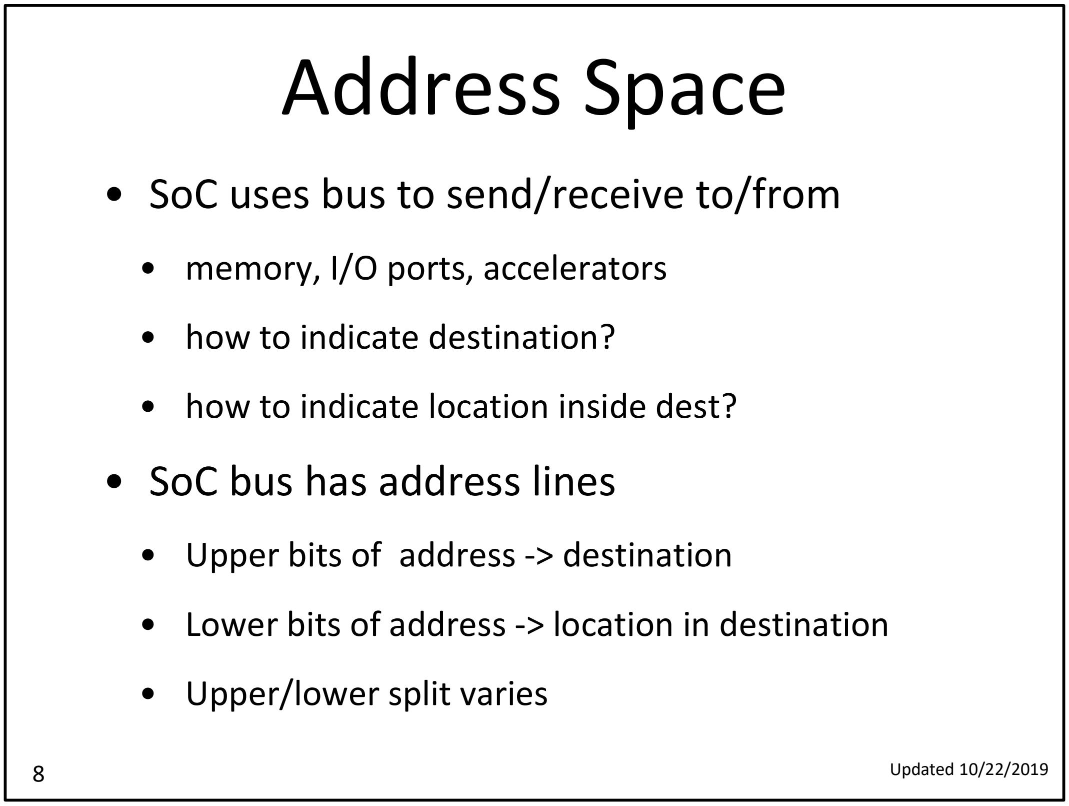
Major signals on SOC bus: The actual names will vary but any SOC bus will have signals that correspond to these. Depending on the type of bus you are using, there may be a variety of additional control signals to control the mode of operation for the bus, enforce security mechanisms, handle arbitration between multiple bus masters. etc.

Clock: SOC busses are synchronous in operation. Devices connected to the bus will only receive data on the active edge of the clock. I.e., imagine registers connected to the bus for which the bus clock serves as the clock for the register.

Address: The address wires on the bus are used to identify which slave device is to be the destination for a command on the bus. Typically a given slave will respond to an entire range of addresses, usually contiguous. More will be said about this on a slide regarding the **address space.** In ECE337, your SOC bus will be 32 bits wide, i.e. 32 parallel wires on the chip or FPGA.

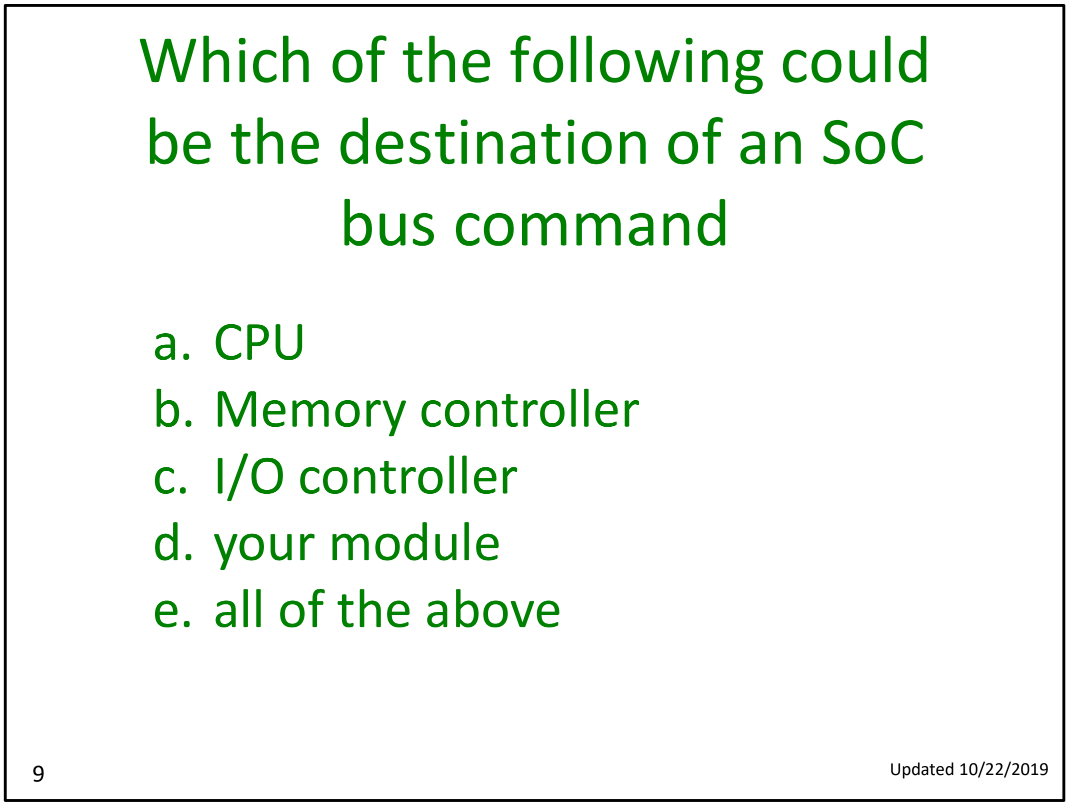
Wdata: This is a set of wires in the bus used by a bus master to transmit data to slave devices. Typically the minimum width of Wdata is 32 bits. Wdata can be larger. The number of wires will be a power of 2.

Rdata: This is a site of wires used by a bus slave to send data back to the bus master. Like Wdata, Rdata is at 32 bits or some higher power of 2.



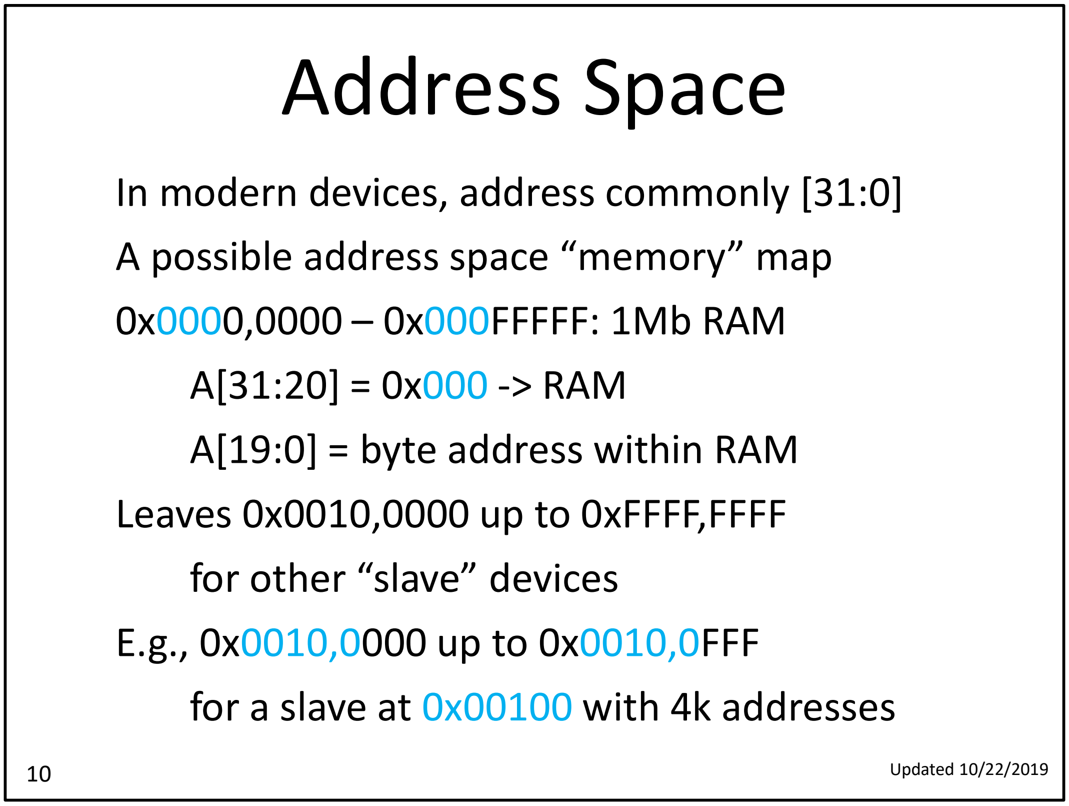
Address Space

- SoC uses bus to send/receive to/from
  - memory, I/O ports, accelerators
  - how to indicate destination?
  - how to indicate location inside dest?
- SoC bus has address lines
  - Upper bits of address -> destination
  - Lower bits of address -> location in destination
  - Upper/lower split varies



Which of the following could be the destination of an SoC bus command

a. CPU
b. Memory controller
c. I/O controller
d. your module
e. all of the above



Address Space

In modern devices, address commonly [31:0]

A possible address space “memory” map

0x0000,0000 – 0x000FFFFF: 1Mb RAM

A[31:20] = 0x000 -> RAM

A[19:0] = byte address within RAM

Leaves 0x0010,0000 up to 0xFFFF,FFFF

for other “slave” devices

E.g., 0x0010,0000 up to 0x0010,0FFF

for a slave at 0x00100 with 4k addresses

This example is deliberately contrived so that the upper/lower split happens on a 4 bit boundary, making it easier to show on the slide.

Given: 32 bit address bits on SoC bus.

If each device on the bus uses address range of 0x0000_0000 to 0x1FFF_FFFF.

How many devices can there be on the bus?

a. 1 b. 2 c. 4 d. 8 e. more

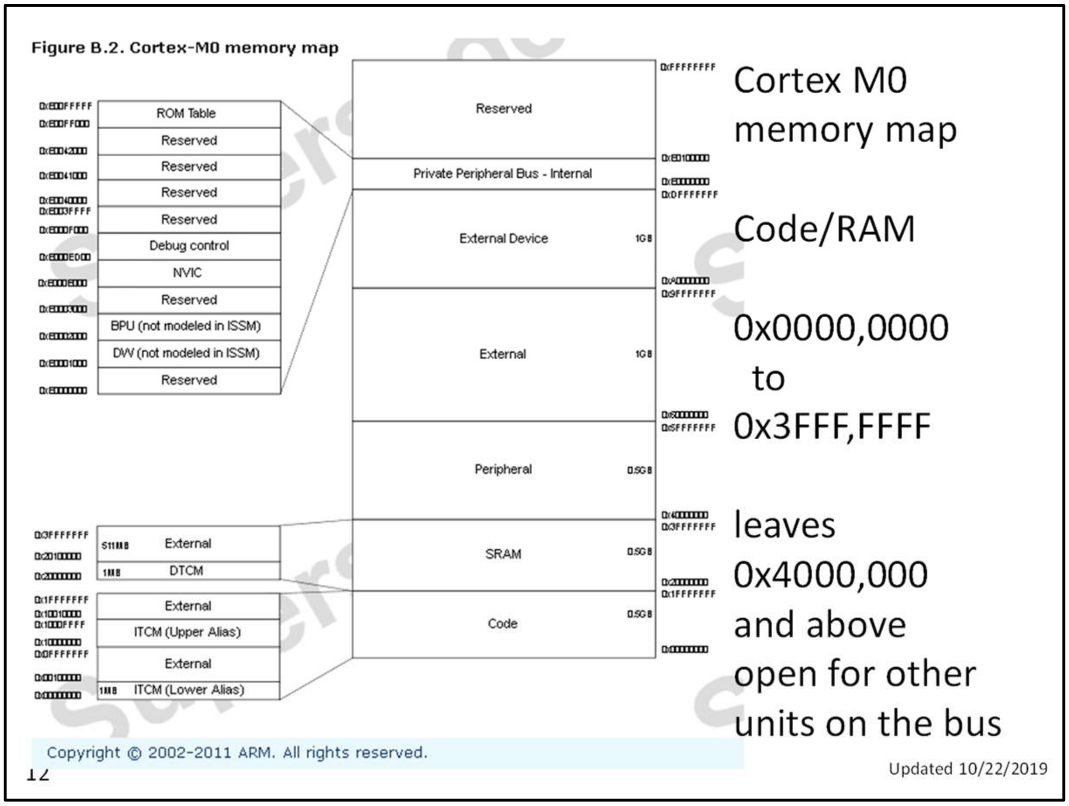


Figure B.2. Cortex-M0 memory map

Cortex M0 memory map

Code/RAM

0x0000,0000
to
0x3FFF,FFFF

leaves
0x4000,000
and above
open for other
units on the bus

12

Updated 10/22/2019

In this case, the addressable memory is 1G (30 address bits), so the upper 2 bits 0b00 indicates code/RAM and 0b01 and above selects other peripheral and external locations

A typical peripheral device will have an address space or 2kb (11 bits address) or 4kb (12 bits address) even if it doesn’t need nearly that many address. In such a case, the upper 20 or 21 bits identify the peripheral.