



## Memory-mapped I/O



- Instead of real memory, some addresses map to I/O devices instead
- But how do you make this happen?
  - **MAGIC** isn't a bad guess, but not very helpful

Let's start by looking at how a memory bus works

7

## Today...



Memory-Mapped I/O

Example Bus with Memory-Mapped I/O

Bus Architectures

AMBA APB

8

## Bus terminology



- Any given transaction have an "initiator" and "target"
- Any device capable of being an initiator is said to be a "bus master"
  - In many cases there is only one bus master (single master vs. multi-master).
- A device that can only be a target is said to be a slave device.

9

## Basic example

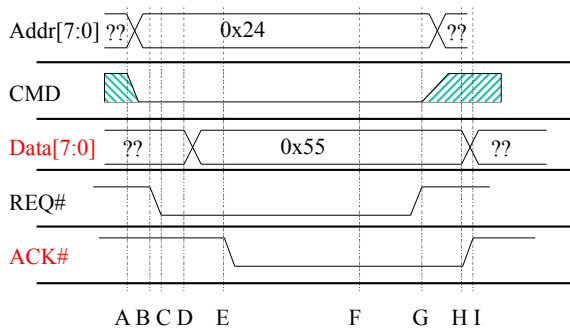


Let's demonstrate a hypothetical example bus

- Characteristics
  - Asynchronous (no clock)
  - One Initiator and One Target
- Signals
  - Addr[7:0], Data[7:0], CMD, REQ#, ACK#
    - CMD=0 is read, CMD=1 is write.
    - REQ# low means initiator is requesting something.
    - ACK# low means target has done its job.

## Read transaction

Initiator wants to read location 0x24



## A read transaction



- Say initiator wants to read location 0x24
  - A. Initiator sets Addr=0x24, CMD=0
  - B. Initiator *then* sets REQ# to low
  - C. Target sees read request
  - D. Target drives data onto data bus
  - E. Target *then* sets ACK# to low
  - F. Initiator grabs the data from the data bus
  - G. Initiator sets REQ# to high, stops driving Addr and CMD
  - H. Target stops driving data, sets ACK# to high terminating the transaction
  - I. Bus is seen to be idle

### A write transaction



- Say initiator wants to write 0xF4 location 0x31
  - A. Initiator sets Addr=0x24, CMD=1, Data=0xF4
  - B. Initiator *then* sets REQ# to low
  - C. Target sees write request
  - D. Target reads data from data bus  
(only needs to store in register, not write all the way to memory)
  - E. Target *then* sets ACK# to low.
  - F. Initiator sets REQ# to high, stops driving other lines
  - G. Target sets ACK# to high, terminating the transaction
  - H. Bus is seen to be idle.

### Returning to memory-mapped I/O



Now that we have an example bus, how would memory-mapped I/O work on it?

Example peripherals

0x00000004: Push Button - Read-Only

Pushed -> 1

Not Pushed -> 0

0x00000005: LED Driver - Write-Only

On -> 1

Off -> 0

### The push-button (if Addr=0x04 write 0 or 1 depending on button)



Addr[7]  
Addr[6]  
Addr[5]  
Addr[4]  
Addr[3]  
Addr[2]  
Addr[1]  
Addr[0]  
REQ#  
CMD

ACK#

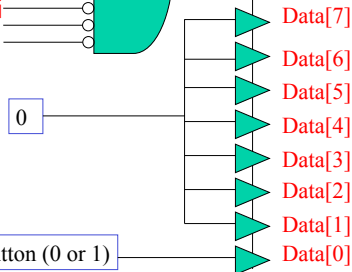
Data[7]  
Data[6]  
Data[5]  
Data[4]  
Data[3]  
Data[2]  
Data[1]  
Data[0]

Button (0 or 1)

### The push-button (if Addr=0x04 write 0 or 1 depending on button)



Addr[7]  
Addr[6]  
Addr[5]  
Addr[4]  
Addr[3]  
Addr[2]  
Addr[1]  
Addr[0]  
REQ#  
CMD



What about CMD?

### The LED (1 bit reg written by LSB of address 0x05)



Addr[7]  
Addr[6]  
Addr[5]  
Addr[4]  
Addr[3]  
Addr[2]  
Addr[1]  
Addr[0]  
REQ#  
CMD

ACK#

LED

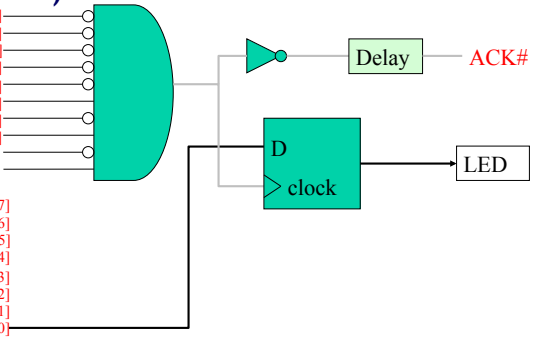
DATA[7]  
DATA[6]  
DATA[5]  
DATA[4]  
DATA[3]  
DATA[2]  
DATA[1]  
DATA[0]

### The LED (1 bit reg written by LSB of address 0x05)



Addr[7]  
Addr[6]  
Addr[5]  
Addr[4]  
Addr[3]  
Addr[2]  
Addr[1]  
Addr[0]  
REQ#  
CMD

DATA[7]  
DATA[6]  
DATA[5]  
DATA[4]  
DATA[3]  
DATA[2]  
DATA[1]  
DATA[0]



## Let's write a simple assembly program Light on if button is pressed.



### Peripheral Details

0x00000004: Push Button - Read-Only

Pushed -> 1

Not Pushed -> 0

0x00000005: LED Driver - Write-Only

On -> 1

Off -> 0

19

## Today...



Memory-Mapped I/O

Example Bus with Memory-Mapped I/O

Bus Architectures

AMBA APB

20

## Driving shared wires



- It is commonly the case that some shared wires might have more than one potential device that needs to drive them.
  - For example there might be a shared data bus that is used by the targets and the initiator. We saw this in the simple bus.
  - In that case, we need a way to allow one device to control the wires while the others “stay out of the way”
    - Most common solutions are:
      - using tri-state drivers (so only one device is driving the bus at a time)
      - using open-collector connections (so if any device drives a 0 there is a 0 on the bus otherwise there is a 1)

21

## Or just say no to shared wires.



- Another option is to not share wires that could be driven by more than one device...
  - This can be really expensive.
    - Each target device would need its own data bus.
    - That's a LOT of wires!
  - Not doable when connecting chips on a PCB as you are paying for each pin.
  - Quite doable (though not pretty) inside of a chip.

22

## Wire count



- Say you have a single-master bus with 5 other devices connected and a 32-bit data bus.
  - If we share the data bus using tri-state connections, each device has “only” 32-pins.
  - If each device that could drive data has it's own bus...
    - Each slave would need \_\_\_\_\_ pins for data
    - The master would need \_\_\_\_\_ pins for data
- Again, recall pins==\$\$\$\$\$.

23

## What happens when this “instruction” executes?



```
#include <stdio.h>
#include <inttypes.h>

#define REG_F00 0x40000140

main () {
    uint32_t *reg = (uint32_t *) (REG_F00);
    *reg += 3;

    printf(“0x%x\n”, *reg); // Prints out new value
}
```

24

**"\*reg += 3" is turned into a ld, add, str sequence**

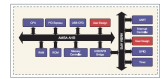


- Load instruction
  - A bus read operation commences
  - The CPU drives the address "reg" onto the address bus
  - The CPU indicated a read operation is in process (e.g. R/W#)
  - Some "handshaking" occurs
  - The target drives the contents of "reg" onto the data lines
  - The contents of "reg" is loaded into a CPU register (e.g. r0)
- Add instruction
  - An immediate add (e.g. add r0, #3) adds three to this value
- Store instruction
  - A bus write operation commences
  - The CPU drives the address "reg" onto the address bus
  - The CPU indicated a write operation is in process (e.g. R/W#)
  - Some "handshaking" occurs
  - The CPU drives the contents of "r0" onto the data lines
  - The target stores the data value into address "reg"

**Details of the bus "handshaking" depend on the particular memory/peripherals involved**



- SoC memory/peripherals
  - AMBA AHB/APB
- NAND Flash
  - Open NAND Flash Interface (ONFI)
- DDR SDRAM
  - JEDEC JESD79, JESD79-2F, etc.



**Why use a standardized bus?**



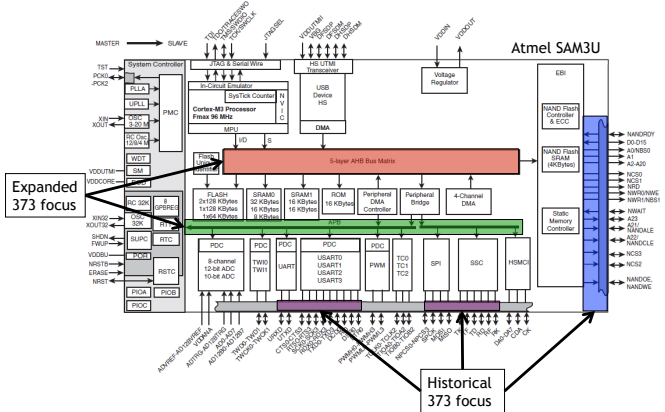
- Downsides
  - Have to follow the specification
  - Probably has actions that are unnecessary
- Upside
  - Generic systems
  - Allows modules to be reused on different systems

**Today...**

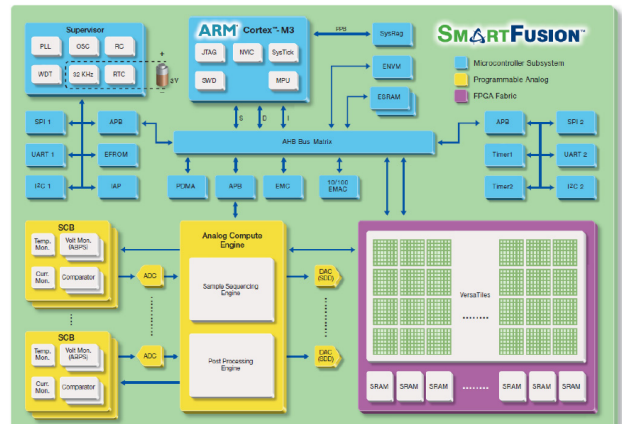


Memory-Mapped I/O  
 Example Bus with Memory-Mapped I/O  
 Bus Architectures  
 AMBA APB

**Modern embedded systems have multiple busses**



**Actel SmartFusion system/bus architecture**





Questions?

Comments?

Discussion?