### EECS 373 Design of Microprocessor-Based Systems

Ron Dreslinski University of Michigan

Lecture 4: Bit of assembly, Memory-mapped I/O, APB September 19, 2016

# Admin

- HW2 is now posted.
- HW1 answers posted as well.

# Today

- Memory-mapped I/O
- Bus Architectures
- ARM AHB-Lite

#### Memory-mapped I/O



- The idea is really simple
  - Instead of real memory at a given memory address, have an I/O device respond.
    - Huh?
- Example:
  - Let's say we want to have an LED turn on if we write a "1" to memory location 5.
  - Further, let's have a button we can read (pushed or unpushed) by reading address 4.
    - If pushed, it returns a 1.
    - If not pushed, it returns a 0.





- How do you get that to happen?
  - We could just say "magic" but that's not very helpful.
  - Let's start by detailing a simple bus and hooking hardware up to it.
- We'll work on a real bus next time!

# Basic example

- Discuss a basic bus protocol
  - Asynchronous (no clock)
  - Initiator and Target
  - REQ#, ACK#, Data[7:0], ADS[7:0], CMD
    - CMD=0 is read, CMD=1 is write.
    - REQ# low means initiator is requesting something.
    - ACK# low means target has done its job.

## A read transaction

- Say initiator wants to read location 0x24
  - A. Initiator sets ADS=0x24, CMD=0.
  - B. Initiator *then* sets REQ# to low. (why do we need a delay? How much of a delay?)
  - 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 ADS and CMD
  - H. Target stops driving data, sets ACK# to high terminating the transaction.
  - I. Bus is seen to be idle.



### A write transaction (write 0xF4 to location 0x31)

- Initiator sets ADS=0x31, CMD=1, Data=0xF4
- Initiator *then* sets REQ# to low.
- Target sees write request.
- Target reads data from data bus. (Just has to store in a register, need not write all the way to memory!)
- Target *then* sets ACK# to low.
- Initiator sets REQ# to high & stops driving other lines.
- Target sets ACK# to high terminating the transaction



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



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



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



#### Outline



- Finish up example
- Bus Architectures
- ARM AHB-Lite

What happens when this "instruction" executes?



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

#define REG\_F00 0x40000140

```
main () {
    uint32_t *reg = (uint32_t *)(REG_FOO);
    *reg += 3;
    printf("0x%x\n", *reg); // Prints out new value
}
```

#### "\*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.









#### Modern embedded systems have multiple busses







### Advanced Microcontroller Bus Architecture (AMBA)

- Advanced High-performance Bus (AHB)
- Advanced Peripheral Bus (APB)



#### AHB

- High performance
- Pipelined operation
- Burst transfers
- Multiple bus masters
- Split transactions

#### APB

- Low power
- Latched address/control
- Simple interface
- Suitable of many peripherals

#### Actel SmartFusion system/bus architecture





#### **Bus terminology**



- Any given transaction have an "<u>initiator</u>" and "<u>target</u>"
- Any device capable of being an initiator is said to be a "<u>bus master</u>"
  - 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.
- Some wires might be shared among all devices while others might be point-to-point connections (generally connecting the master to each target).

#### **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)

#### 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.

#### 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==\$\$\$\$\$.

#### Outline



- Finish up example
- Bus Architectures
- ARM APB

# APB is a fairly simple bus designed to be easy to work with.



- Low-cost
- Low-power
- Low-complexity
- Low-bandwidth
- Non-pipelined
- Ideal for peripherals

# Let's just look at APB writes (Master writing to device) as a starting point.

• We'll add reads shortly.

#### Notation





#### **APB bus signals**



- PCLK
  - Clock
- PADDR
  - Address on bus
- PWRITE
  - 1=Write, 0=Read
- PWDATA
  - Data written to the I/O device.
     Supplied by the bus master/ processor.



#### **APB bus signals**



- PSEL
  - Asserted if the current bus transaction is targeted to *this* device
- PENABLE
  - High during entire transaction *other than* the first cycle.
- PREADY
  - Driven by target. Similar to our #ACK. Indicates if the target is <u>ready</u> to do transaction. Each target has it's own PREADY



#### So what's happening here?





#### Example setup



- For the next couple of slides, we will assume we have one bus master "CPU" and two slave devices (D1 and D2)
  - D1 is mapped to address
     0x00001000-0x0000100F
  - D2 is mapped to 0x00001010-0x0000101F

# Say the CPU does a store to location 0x00001004 with no stalls







# What if we want to have the LSB of this register control an LED?

PWDATA[31:0]



We are assuming APB only gets lowest 8 bits of address here...

#### Reg A should be written at address 0x00001000 Reg B should be written at address 0x00001004





We are assuming APB only gets lowest 8 bits of address here...

**T4** 

#### Reads...





The key thing here is that each slave device has its own read data (PRDATA) bus!

Recall that "R" is from the initiator's viewpoint-the device drives data when read.

#### Let's say we want a device that <u>provides</u> data from a switch on a read to any address it is assigned. (so returns a 0 or 1)





PREADY



#### Device provides data from switch A if address 0x00001000 is read from. B if address 0x00001004 is read from



PRDATA[31:0]

PWRITE PENABLE PSEL Switch A PADDR[7:0] Switch B

PREADY

#### All reads read from register, all writes write...





We are assuming APB only gets lowest 8 bits of address here...

#### Things left out...



- There is another signal, PSLVERR (APB Slave Error) which we can drive high if things go bad.
  - We'll just tie that to 0.
- Notice we are assuming that our device need not stall.
  - We **could** stall if we needed.
    - I can't find a limit on how long, but I suspect at some point the processor would generate an error.

#### Verilog!



#### /\*\*\* APB3 BUS INTERFACE \*\*\*/

| input PCLK,                    | // cl | ock                                       |
|--------------------------------|-------|---|
| input PRESERN,                 | // sy | stem reset                                |
| input PSEL,                    | // pe | ripheral select                           |
| input PENABLE,                 | // di | stinguishes access phase                  |
| output wire PREADY,            | // pe | ripheral ready signal                     |
| output wire PSLVERR,           | // er | ror signal                                |
| input PWRITE,                  | // di | stinguishes read and write cycles         |
| <pre>input [31:0] PADDR,</pre> | // I/ | 0 address                                 |
| input wire [31:0] PWDATA,      | // da | ta from processor to I/O device (32 bits) |
| output reg [31:0] PRDATA,      | // da | ta to processor from I/O device (32-bits) |

#### /\*\*\* I/O PORTS DECLARATION \*\*\*/

```
output reg LEDOUT, // port to LED
input SW // port to switch
);
assign PSLVERR = 0;
assign PREADY = 1;
```

#### 42

#### **APB state machine**

- IDLE
  - Default APB state
- SETUP
  - When transfer required
  - PSELx is asserted
  - Only one cycle
- ACCESS
  - PENABLE is asserted
  - Addr, write, select, and write data remain stable
  - Stay if PREADY = L
  - Goto IDLE if PREADY = H and no more data
  - Goto SETUP is PREADY = H and more data pending

We'll spend a bit more time on this next week...







#### Questions?

#### Comments?

Discussion?