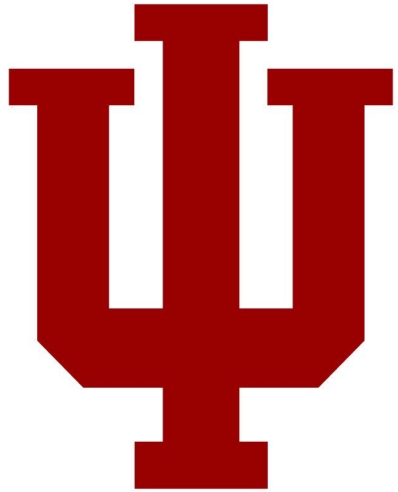
07: MMIO II

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University



Some material taken from:
https://github.com/trekhleb/homemade-machine-learning/tree/master/homemade/neural_network
http://cs231n.github.io/neural-networks-1/

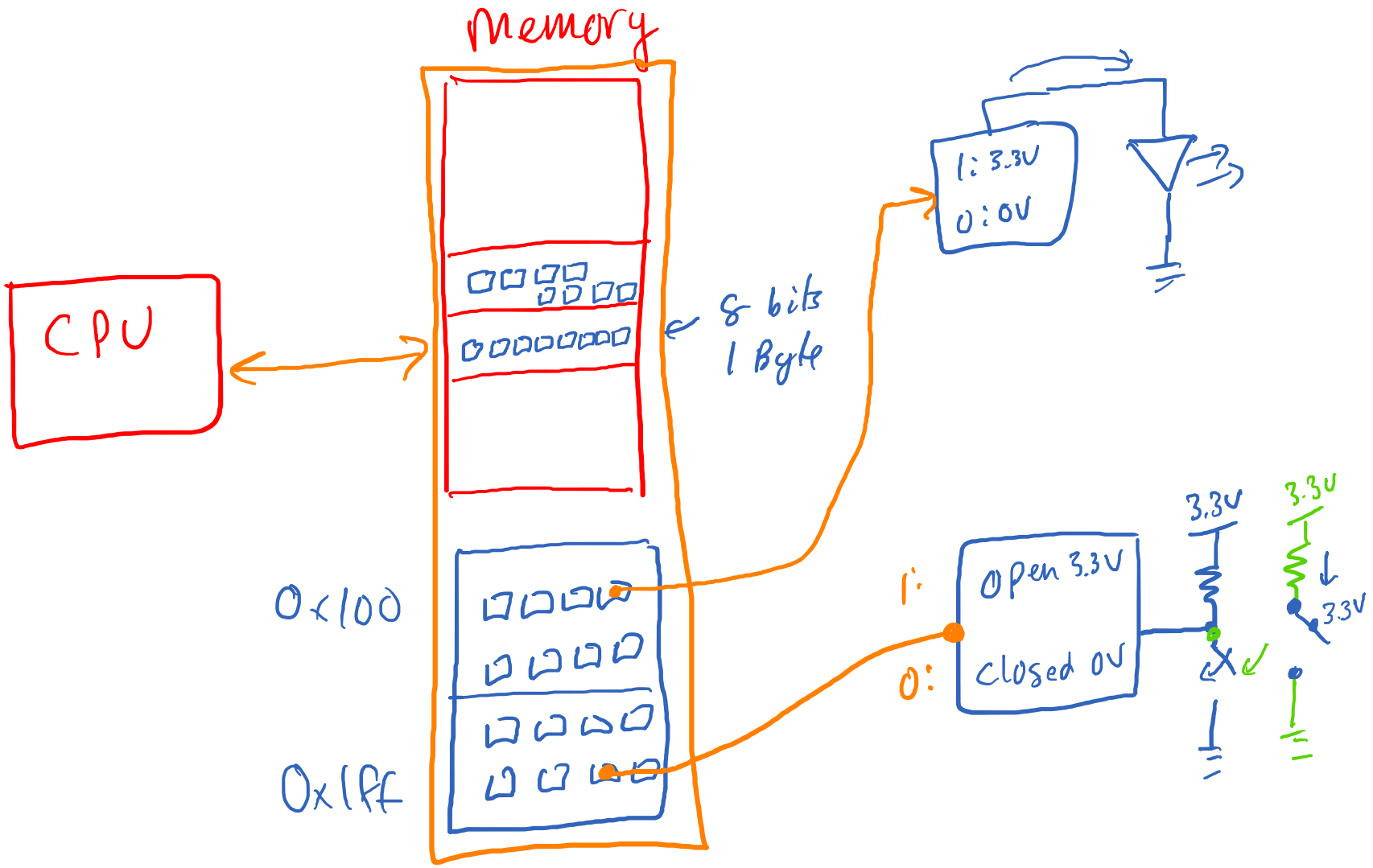
Announcements

- P2: Demo due by Friday
- P3: Out now!
  - Need a Pynq
  - Groups of 2 allowed
- P4: Out soon…

Optimizations thus far

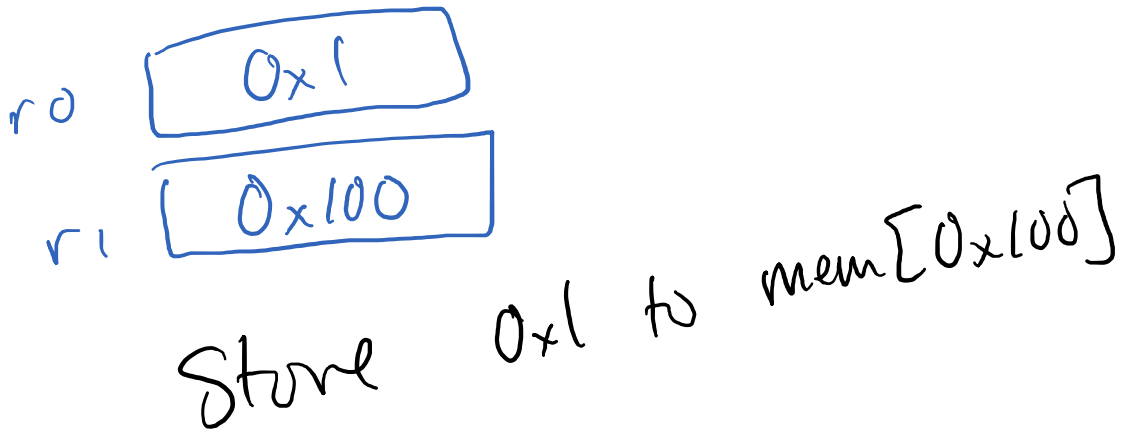
- Algorithmic complexity
- Removing redundant computation
- ~~Multithreading / Multiprocessing*~~
- Python/C/Asm Interfacing
- Map to Hardware

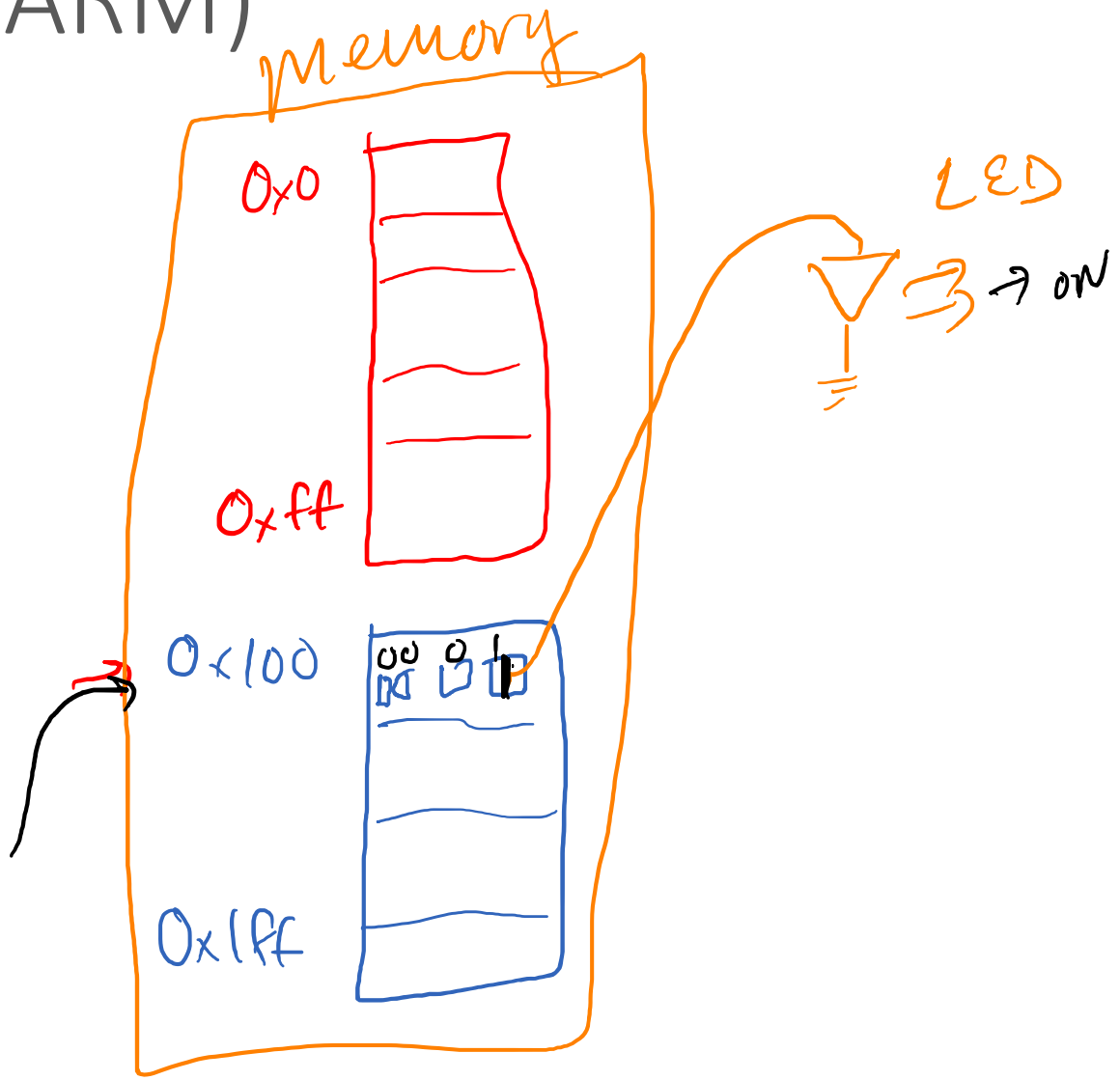
Review: Memory-Mapped I/O



MMIO Store from ASM (ARM)

```
mov r0, 0x1
mov r1, 0x100
str r0, [r1]
```

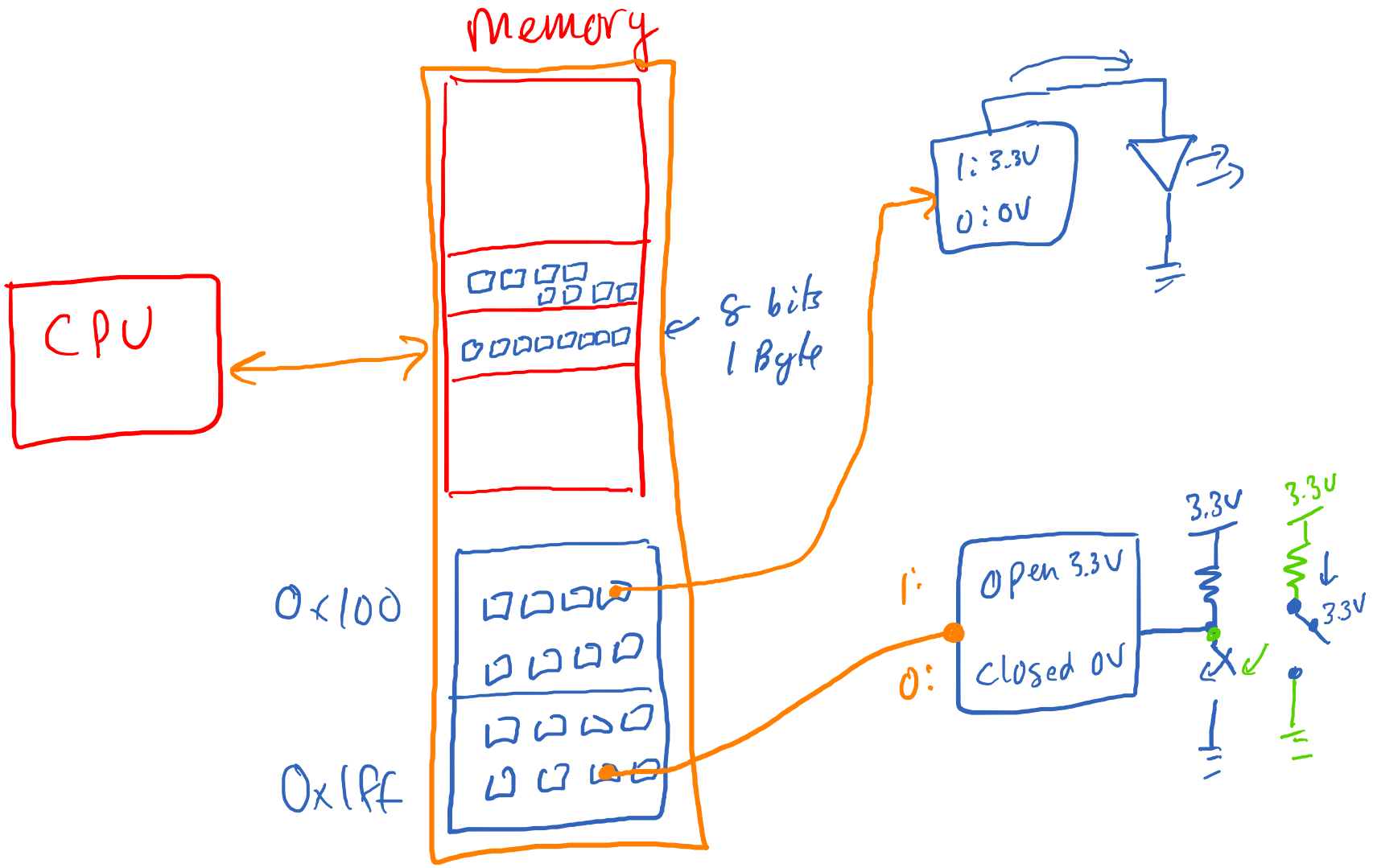




MMIO Store from C

```
#define LED_ADDR 0x1000
uint32_t * LED_REG = (uint32_t *)(LED_ADDR);
*LED_REG = 0x1;
```

Review: Memory-Mapped I/O



MMIO Load from ASM (ARM)

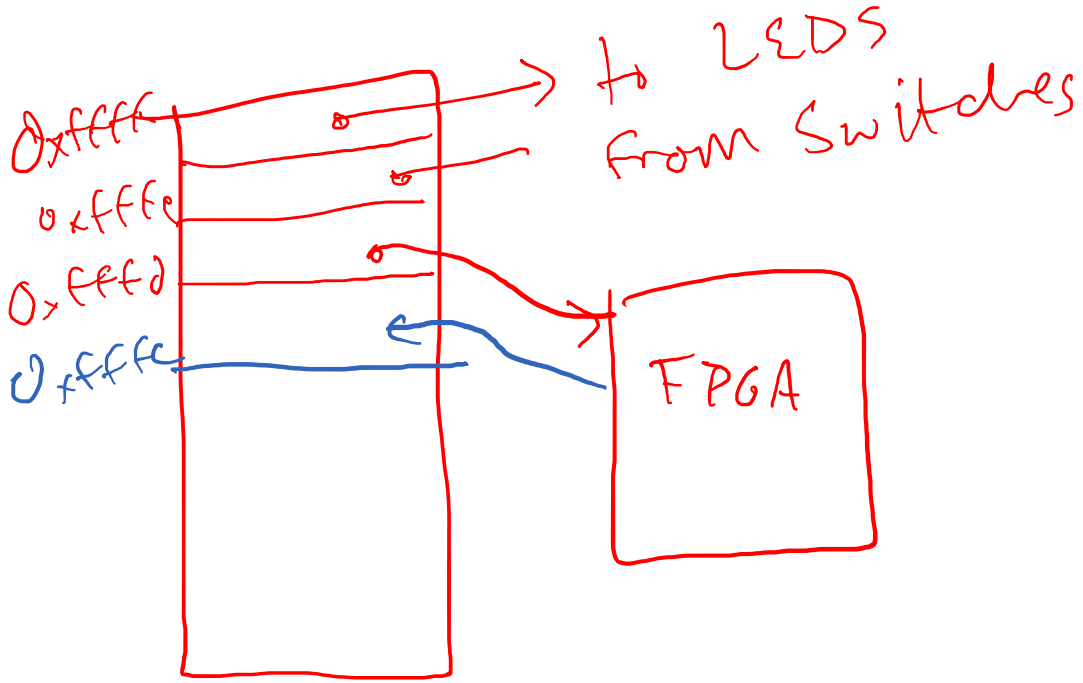
```
mov r0, 0x1ff
ldr r1, [r0]
```

MMIO Load from C

```
#define SW_ADDR 0x1ff
uint32_t * SW_REG = (uint32_t *)(SW_ADDR);
int y = (*SW_REG);
```

Memory-Mapped I/O

- I/O devices pretend to be memory
- Devices accessed with native CPU load/store instructions



Question: What happens here?

```
int y = 0;

int quit = y;
while(!quit)
{
     //more code
     quit = y;
}
```

Problem: The compiler is “helping”. (-O3 edition)

```
int y = 0;

int quit = y;
while(!quit)
{
     //more code
     quit = y;
}
```

Problem: The compiler is “helping”. (-O3 edition)

```
int y = 0;

int quit = y;
while(!quit)
{
     //more code
     quit = y;
}
```

```
int y = 0;

int quit = y;
while(1 !quit)
{
     //more code
     quit = y;
}
```

What's the difference here?

```
int y = 0;



int quit = y;
while(!quit)
{
     //your code
     quit =y ;
}
```

```
int y = 0;
uint32_t * SW_REG = &y;

int quit = (*SW_REG);
while(!quit)
{
     //your code
     quit = (*SW_REG);
}
```

What's the problem with this?

```
int y = 0;
uint32_t * SW_REG = &y;

int quit = (*SW_REG);
while(1 !quit)
{
      //your code
      quit = (*SW_REG); //stop case?
}
```

Use `volatile` for MMIO addresses!

```
#define SW_ADDR 0xfffe
volatile uint32_t * SW_REG = (uint32_t * SW_ADDR);

int quit = (*SW_REG);
while(!quit)
{
    //more code
    quit = (*SW_REG);
}
```

`volatile` Variables

- `volatile` keyword tells compiler that the memory value is subject to change randomly.

- **Use `volatile` for all MMIO memory. The values change randomly!**

Use **`volatile`** for all MMIO memory.

(hint: P4)

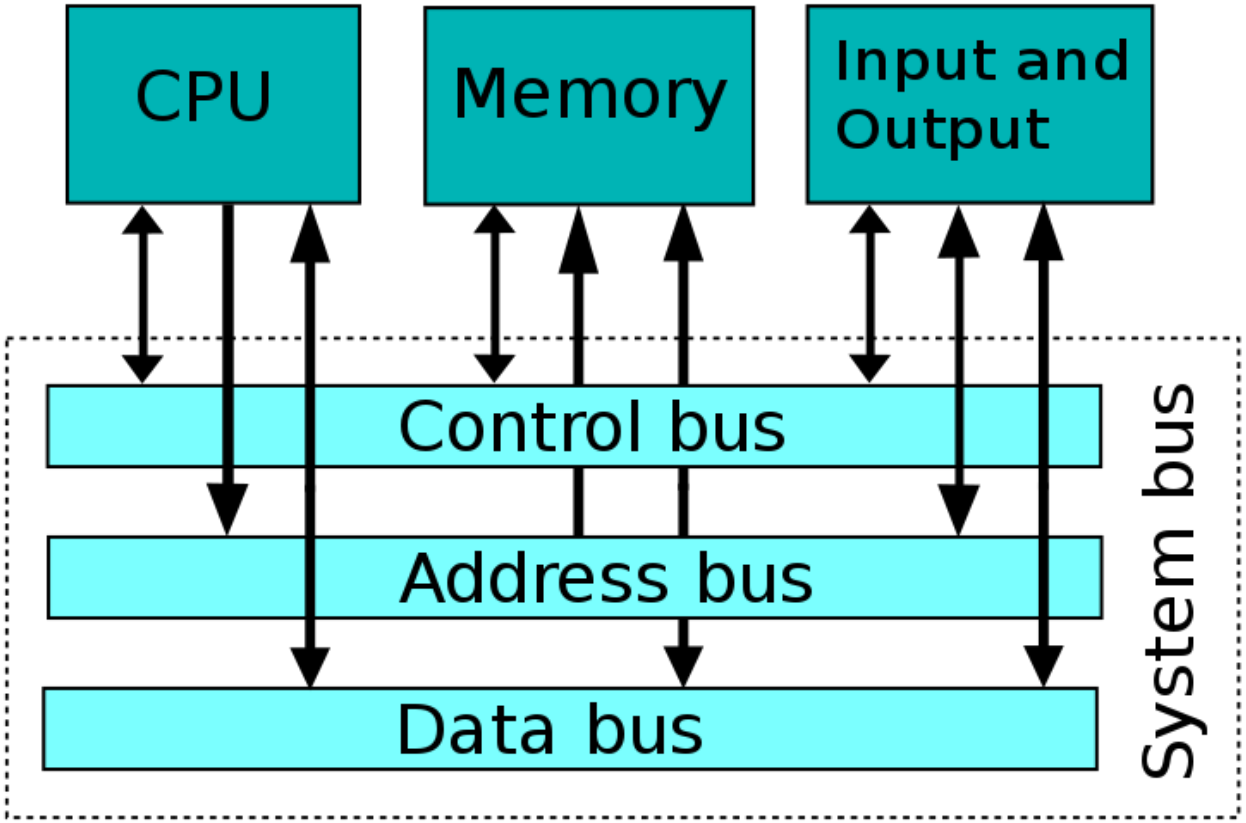
Demo Time

MMIO Bus Interface

What do the CPU and Memory need to communicate?

	CPU->Mem	Mem->CPU
Loads		
Stores		

The System Bus

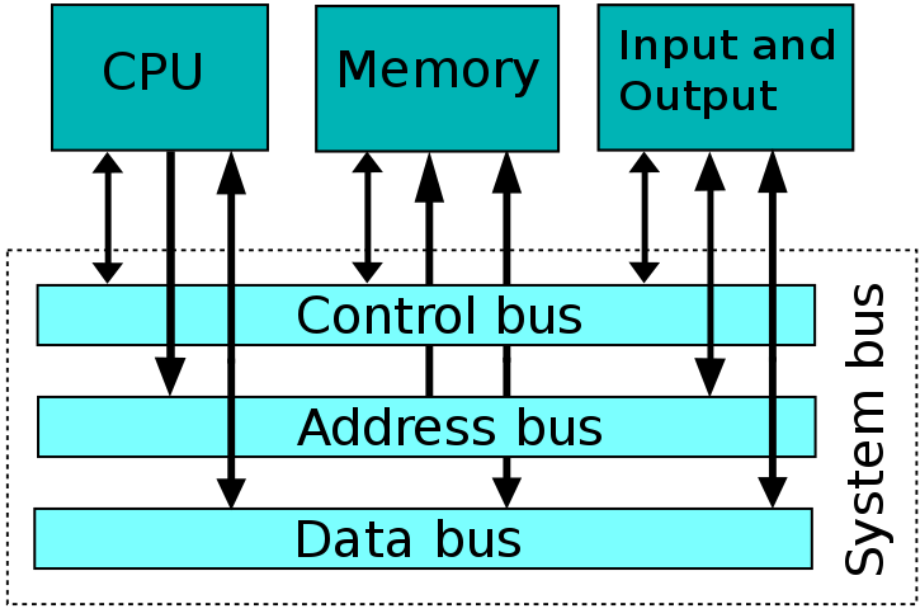


Bus terminology review

- A "transaction" occurs between 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".

Transaction Steps

- CPU wants to load data from Memory
- CPU wants to store data to I/O



Hypothetical Bus Example

- Characteristics
  - Asynchronous (no clock) – hay, why no?
  - 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 is acknowledging the job is done.

Read transaction

Initiator wants to read location 0x24

CMD=0 is read, CMD=1 is write.
REQ# low means initiator is requesting.
ACK# low means target is acknowledging.

I: Addr[7:0]

I: CMD

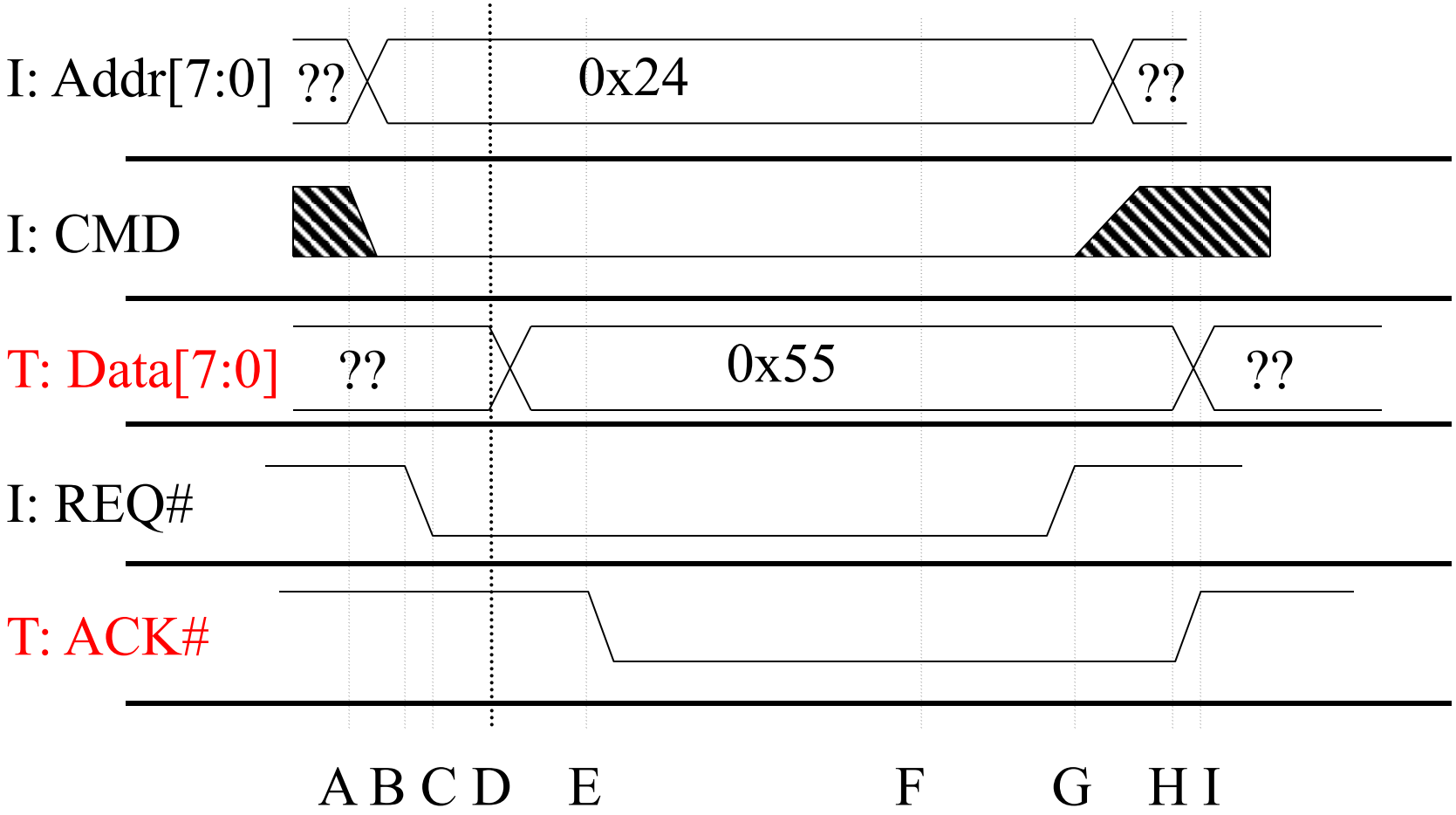
T: Data[7:0]

I: REQ#

T: ACK#

Read transaction

Initiator wants to read location 0x24



A read transaction

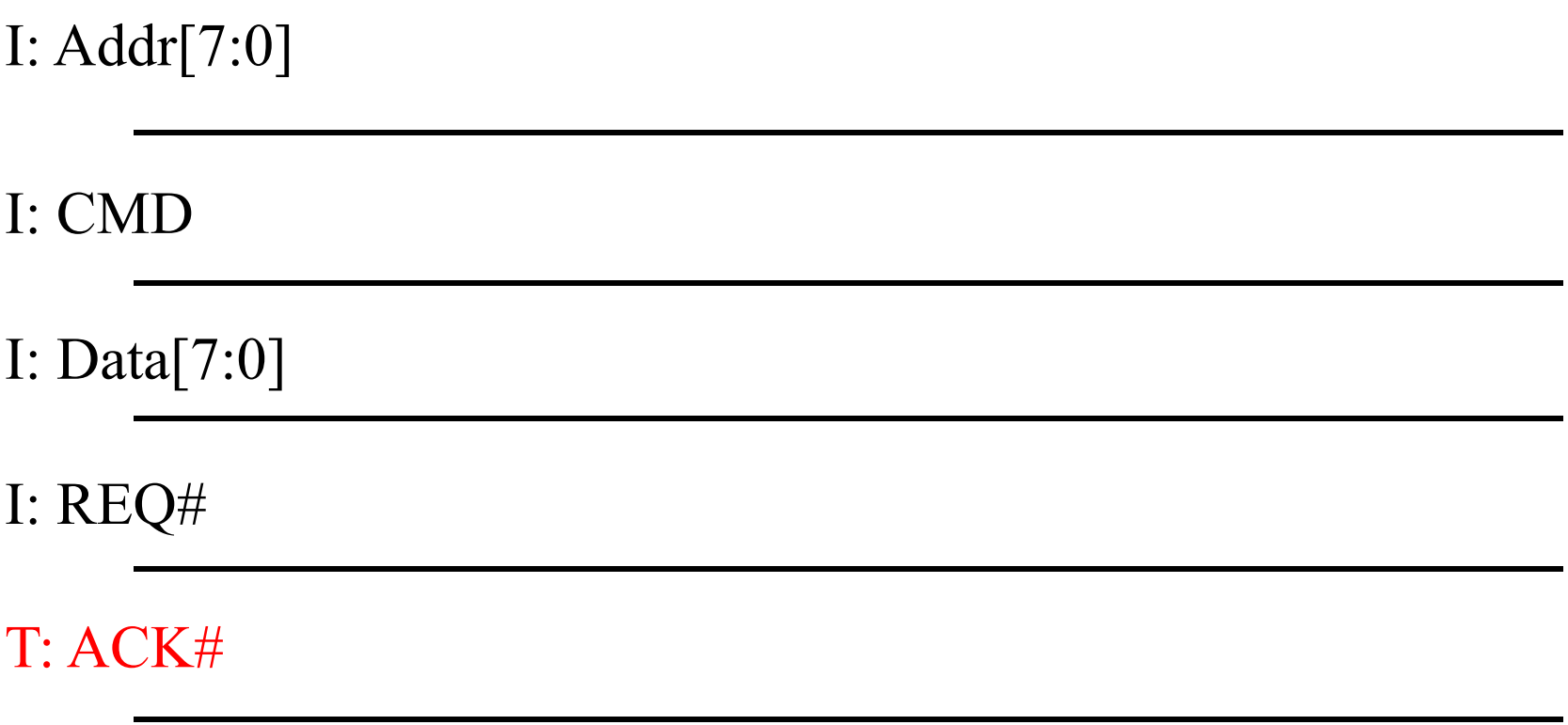
Say initiator wants to read location 0x24

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

Write transaction

Initiator wants to write 0x56 to location 0x24

CMD=0 is read, CMD=1 is write.
REQ# low means initiator is requesting.
ACK# low means target is acknowledging.



Tri-State Buffer

- Drives output when enabled
- Otherwise does not drive output (high-impedance)

Tri-State Buffer

Can MMIO behave as memory?

Example peripherals

0x04: Push Button - Read-Only

Pushed -> 1

Not Pushed -> 0

0x05: LED Driver - Write-Only

On -> 1

Off -> 0

The push-button

(if Addr=0x04 read 0 or 1 depending on button)

CMD=0 is read, CMD=1 is write.
REQ# low means initiator is requesting.
ACK# low means target is acknowledging.

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

ACK#

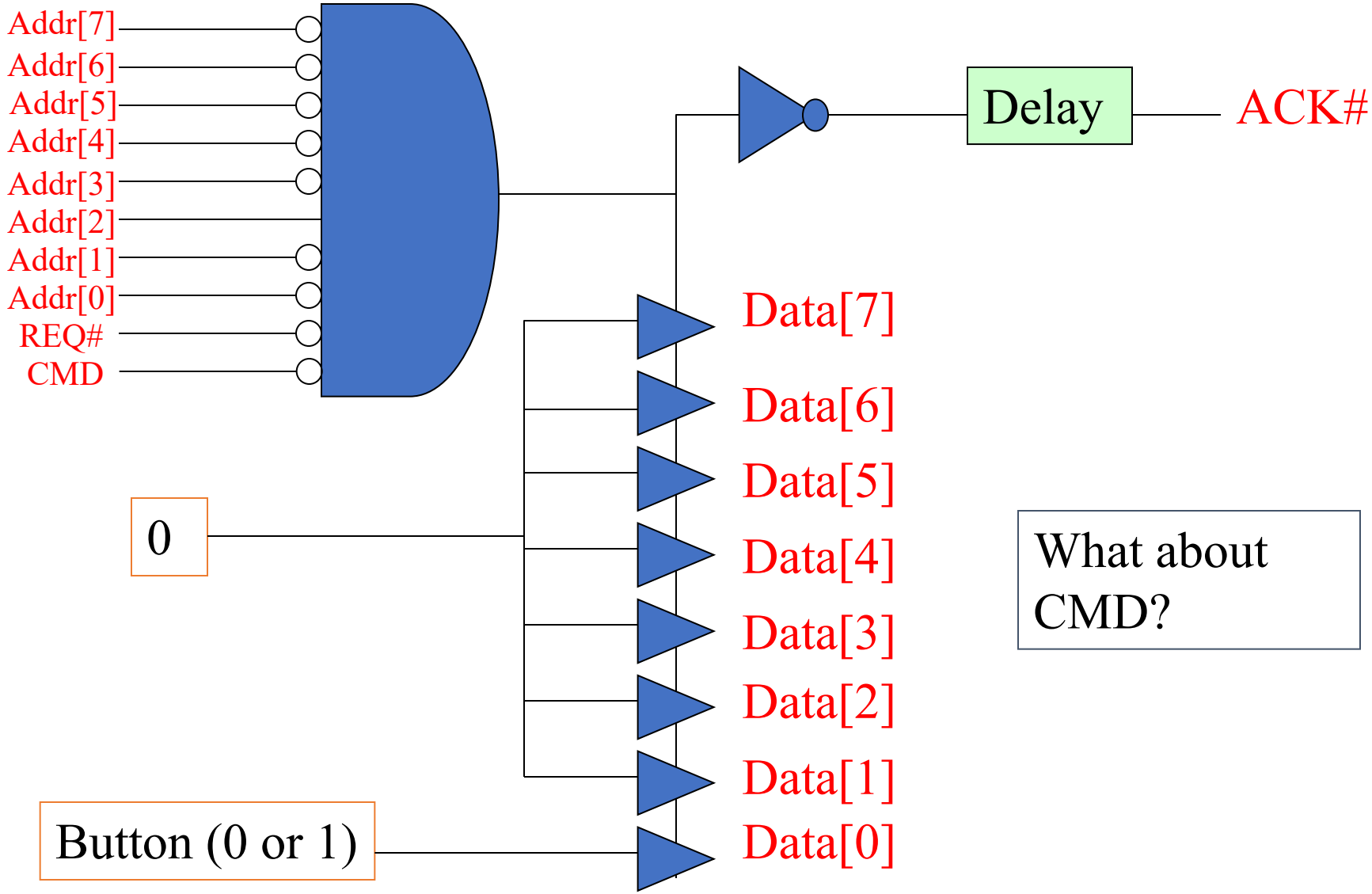
Data[7]
Data[6]
Data[5]
Data[4]
Data[3]
Data[2]
Data[1]
Data[0]

Button (0 or 1)

What about CMD?

The push-button

(if Addr=0x04 read 0 or 1 depending on button)



The LED

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

CMD=0 is read, CMD=1 is write.
REQ# low means initiator is requesting.
ACK# low means target is acknowledging.

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

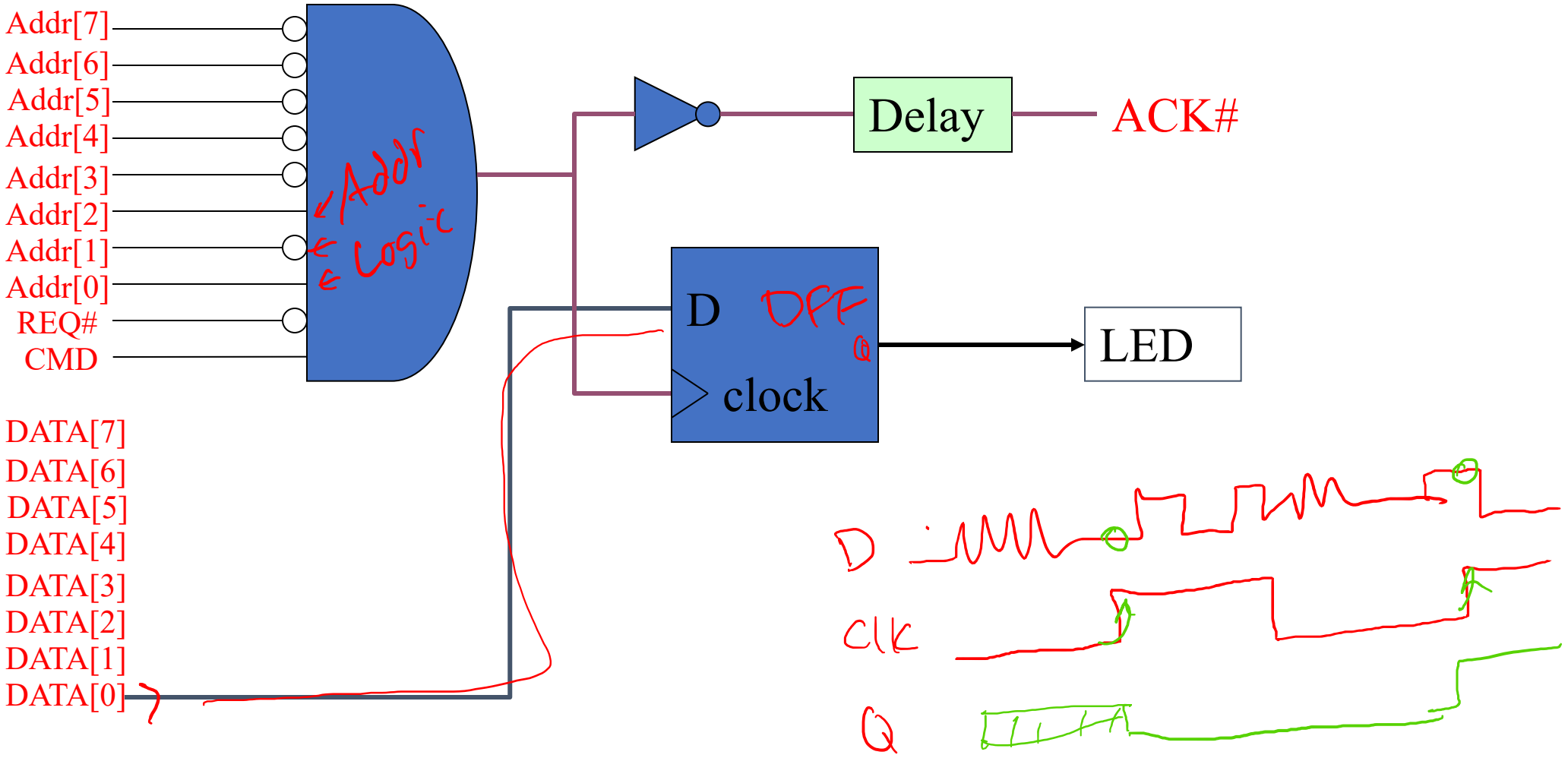
DATA[7]
DATA[6]
DATA[5]
DATA[4]
DATA[3]
DATA[2]
DATA[1]
DATA[0]

ACK#

LED

The LED

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



Let’s write a simple C program to turn the LED on if button is pressed.

Peripheral Details

0x04: Push Button - Read-Only

- Pushed -> 1
- Not Pushed -> 0

0x05: LED Driver   - Write-Only

- On -> 1
- Off -> 0

In ASM:

```
        mov r0, #0x4   % PB
        mov r1, #0x5   % LED
loop:   ldr r2, [r0, #0]
        str r2 [r1, #0]
        b loop
```

Let's write a simple C program to turn the LED on if button is pressed.

Peripheral Details

0x04: Push Button - Read-Only

- Pushed -> 1
- Not Pushed -> 0

0x05: LED Driver - Write-Only

- On -> 1
- Off -> 0

Next Time

- Real bus architectures

References

- https://www.youtube.com/watch?v=okiTzvihHRA
- https://web.eecs.umich.edu/~prabal/teaching/eecs373/
- https://en.wikipedia.org/wiki/File:Computer_system_bus.svg
- https://www.realdigital.org/doc/a9fee931f7a172423e1ba73f66ca4081

07: MMIO Buses

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University