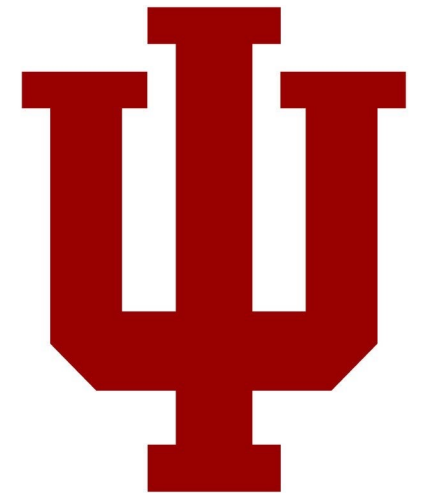


11: Direct Memory Access (DMA)

DMA

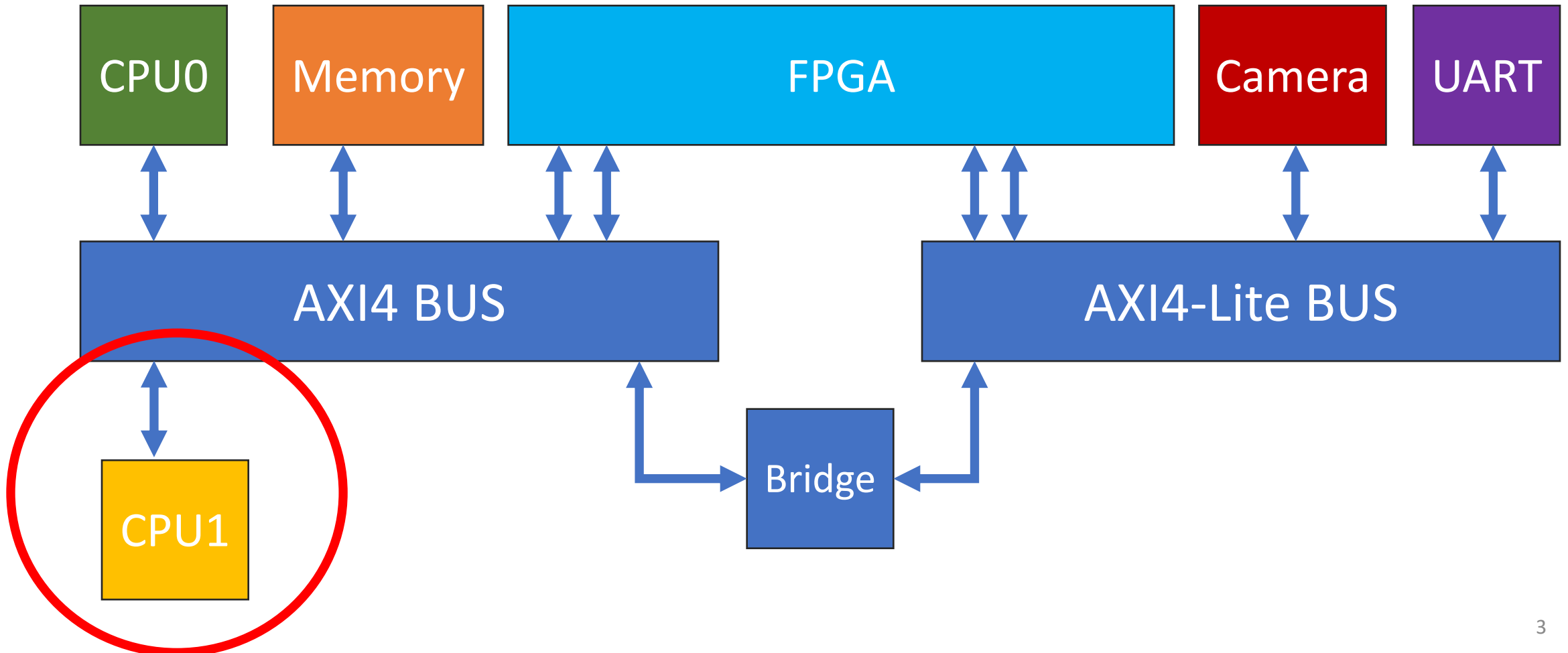
Engr 315: Hardware / Software Codesign
Andrew Lukefahr
Indiana University



Announcements

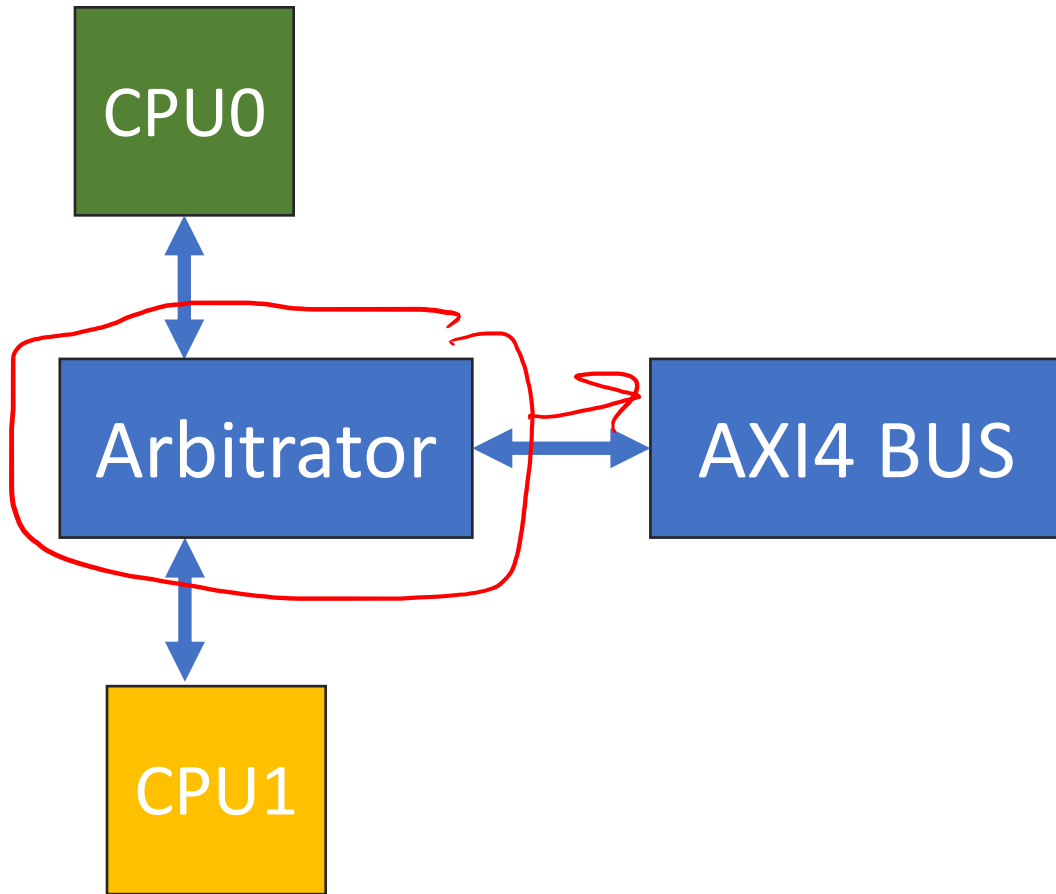
- P5 is out!
 - Due Monday after break
- P6 out soon (hopefully)

Review: Multi-Master Buses



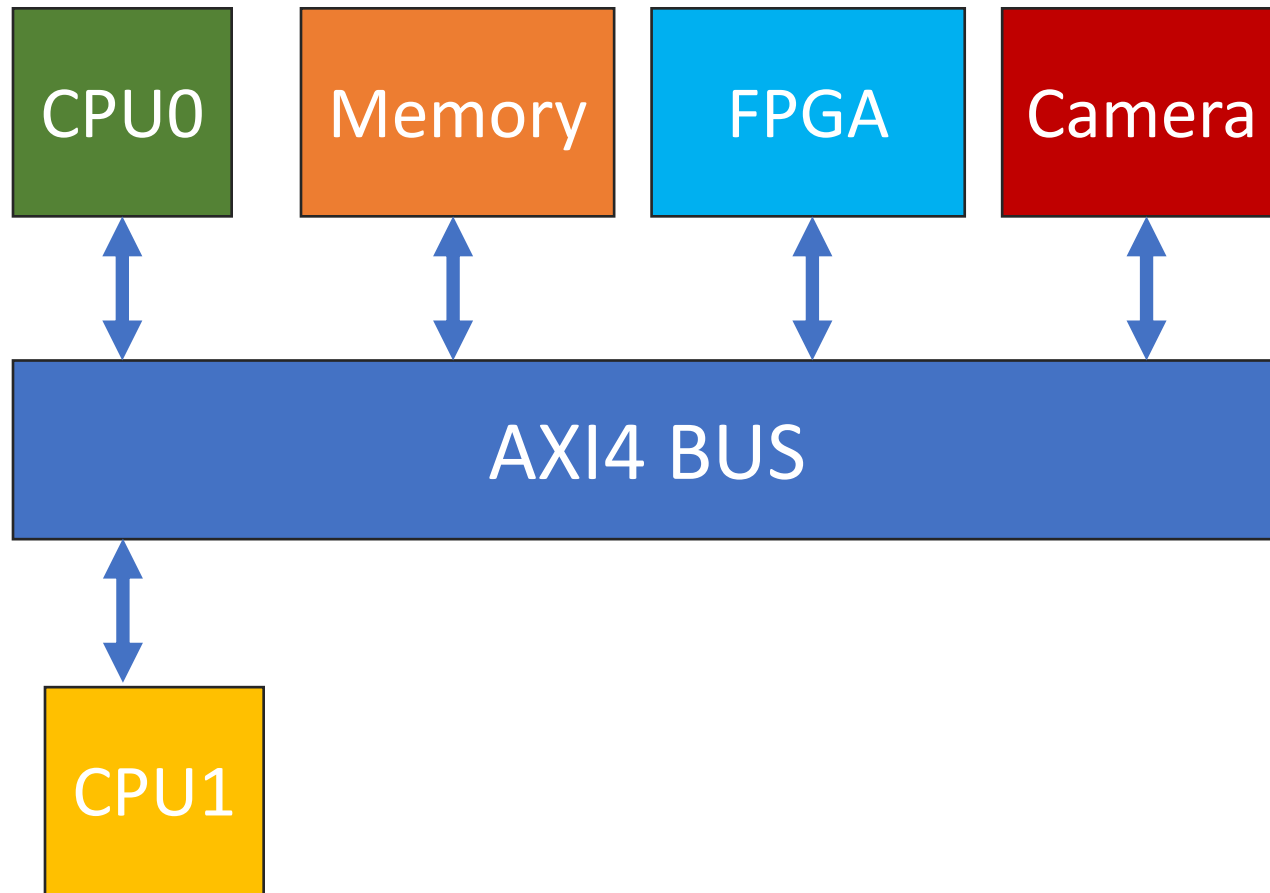
An **Arbitrator** selects who gets to use the bus

- What happens if both request a transaction at the same time?



- **Arbitration:**
 - Fixed-Priority → *fast*
 - Round Robin → *fair*
 - Many more...

Q: How do I move data between the Camera and Memory?



A: The CPU copies data from Camera to Memory

```
#define CAMERA_MMIO_ADDR 0x40000004
volatile uint32_t * camera =
    (uint32_t *) (CAMERA_MMIO_ADDR);
#define BUF_SIZE 1024;
uint32_t buf[BUF_SIZE];

int main () {

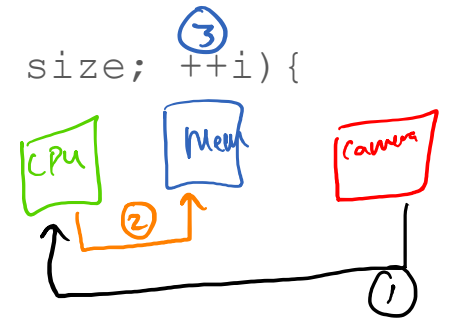
    while (true){
        copy_image(camera, buf, BUF_SIZE);
        detect_face(buf);
    }
}
```

```
void copy_image (uint32_t * source,
                uint32_t * dest,
                uint32_t size)
{
    register uint32_t reg;

    for (int i = 0; i < size; ++i){
        ① reg = *source;
        ② dest[i] = reg;
        ④ }
    }
```

① load
② store

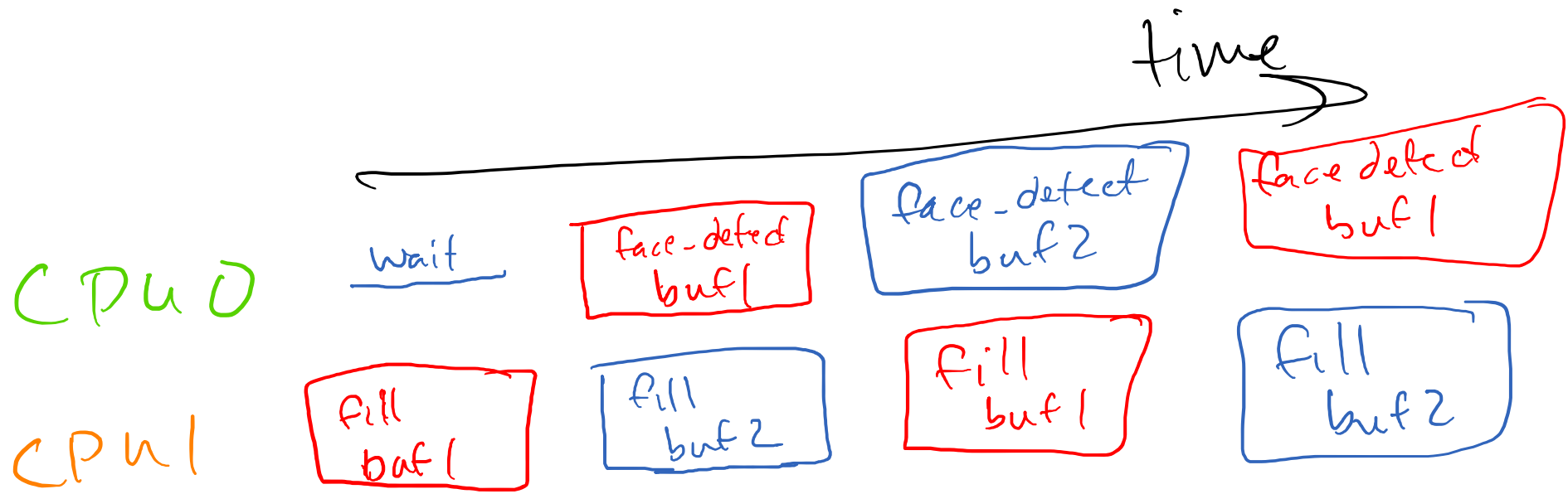
③ increment
④ branch



Double-Buffering

Copy on CPU1, Version 2.

```
int main () {  
  
    ask_cpul_to_copy_image(camera, buf1, BUF_SIZE);  
    wait_for_cpul_done();  
  
    while (true){  
        ask_cpul_to_copy_image(camera, buf2, BUF_SIZE);  
        detect_face(buf1);  
        wait_for_cpul_done();  
  
        ask_cpul_to_copy_image(camera, buf1, BUF_SIZE);  
        detect_face(buf2);  
        wait_for_cpul_done();  
    }  
}
```

Why are we wasting an entire CPU for this?

```
void copy_image (uint32_t * source,
                 uint32_t * dest,
                 uint32_t size)
{
    register uint32_t reg;

    for (int i = 0; i < size; ++i){

        reg = *source;

        dest[i] = reg;
    }
}
```

DMA: Direct Memory Access

- A mini-CPU that does copy for you:

```
void copy (uint32_t * source,
           uint32_t * dest,
           uint32_t size)
{
    register uint32_t reg;

    for (int i = 0; i < size; ++i){

        reg = *source;

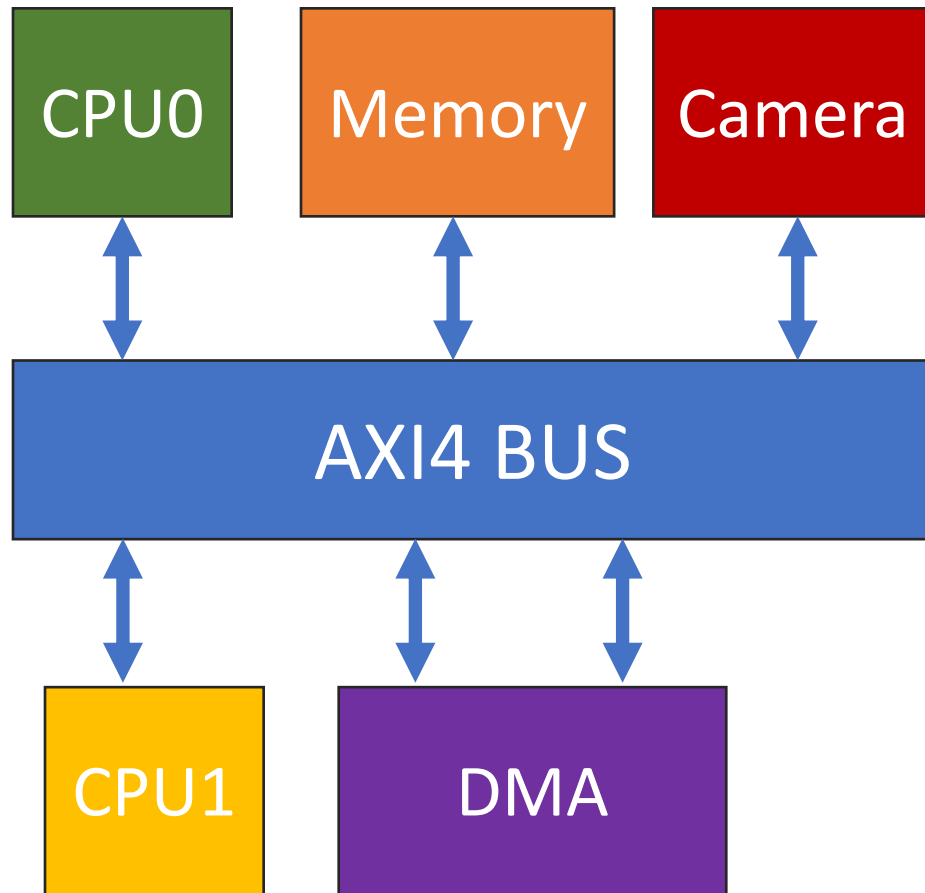
        dest[i] = reg;

    }
}
```

Using DMA from C:

```
int main () {  
  
    dma_start_copy (camera, buf1, BUF_SIZE);  
    dma_wait_for_done();  
  
    while (true){  
        dma_start_copy (camera, buf2, BUF_SIZE);  
        detect_face(buf1);  
        dma_wait_for_done();  
  
        dma_start_copy (camera, buf1, BUF_SIZE);  
        detect_face(buf2);  
        dma_wait_for_done();  
    }  
}
```

DMA has 2 interfaces



- Interface 1: Copy Memory
 - Data-Intensive Interface
 - AXI4 Master
 - Initiates Loads / Stores
- Interface 2: Tell DMA what to copy
 - Control Interface
 - AXI4 Slave
 - Responds to Loads/Stores

What's needed to do this in Hardware?

```
void dma_copy (uint32_t * source,
               uint32_t * dest,
               uint32_t size)
{
    register uint32_t reg;

    for (int i = 0; i < size; ++i){

        reg = *source;

        dest[i] = reg;
    }
}
```

Hardware Interface Needs:

```
void dma_copy (uint32_t * source,
               uint32_t * dest,
               uint32_t size)
{
    register uint32_t reg;

    for (int i = 0; i < size; ++i){

        reg = *source;

        dest[i] = reg;
    }
}
```

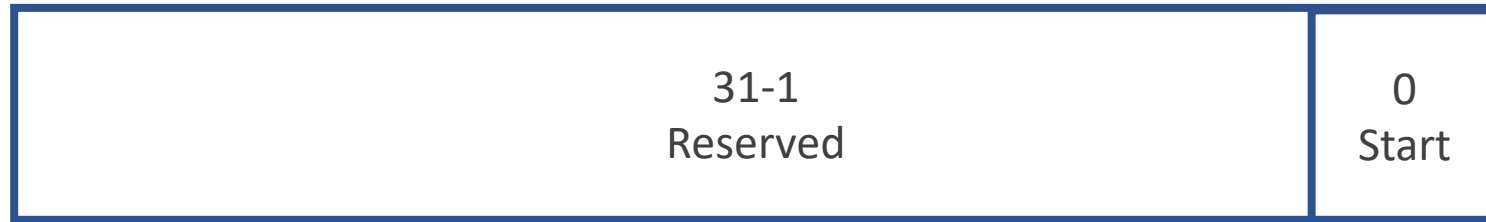
MyDMA MMIO Interface

- 0x0400: Control Register
- 0x0404: Status Register
- 0x0408: Source Address
- 0x040C: Destination Address
- 0x0410: Transfer Size in Bytes

MMIO Control Register

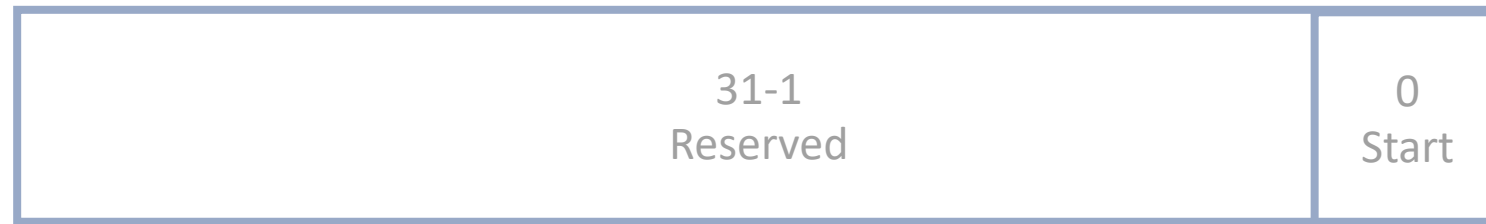
MMIO Control Register

Control - 0x0400

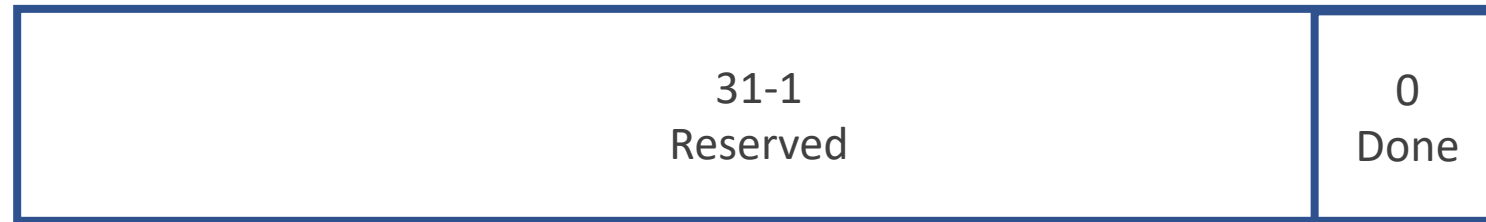


MMIO Status Register

Control - 0x0400
Read/Writer



Status - 0x0404
Read - Only



MMIO Data Registers

Source - 0x0408

31-0
DMA Source Address

Destination - 0x040C

31-1
DMA Destination Address

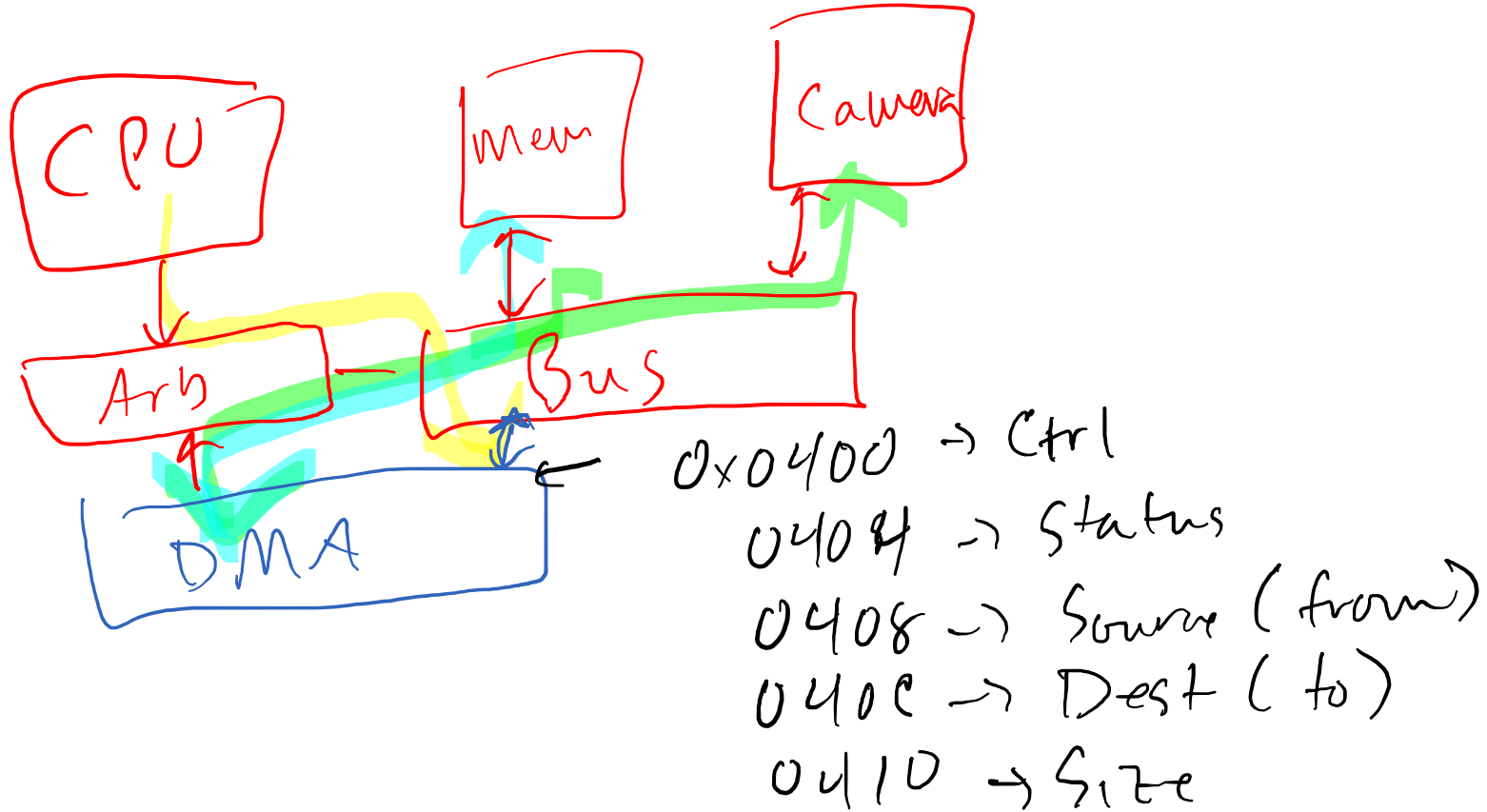
Size - 0x0410

31-16 Reserved	15-0 DMA Transfer Size (in Bytes)
-------------------	--------------------------------------

All MMIO Registers

Control - 0x0400	31-1 Reserved	0 Start
Status - 0x0404	31-1 Reserved	0 Done
Source - 0x0408	31-0 DMA Source Address	
Destination - 0x040C	31-1 DMA Destination Address	
Size - 0x0410	31-16 Reserved	15-0 DMA Transfer Size (in Bytes)

MyDMA Interface



MyDMA Internals

MyDMA Internals

- IDLE: Status[Done]=1, wait for Control[Start]
- START: Status[Done] = 0, i = 0;
- LOAD: tmp = [Source+i]
- STORE: Dest+i = tmp

Does the AXI4 Full Interface have an address?

Does the AXI4 Full Interface have an MMIO Address?

- Is pretending to be memory, or a CPU?
- Does a CPU have a memory address?
- **No.**
- MMIO is for SLAVE interfaces.

Using DMA from the CPU:

0x0400: Control Register
0x0404: Status Register
0x0408: Source Address
0x040C: Destination Address
0x0410: Transfer Size in Bytes

```
void dma_copy ( uint32_t * source,  
                uint32_t * dest,  
                uint32_t size) {  
  
    register uint32_t reg;  
  
    for (int i = 0; i < size; ++i){  
  
        reg = *source; //load  
        dest[i] = reg; //store  
    }  
  
    //code me!  
  
}
```

Using DMA from the CPU:

0x0400: Control Register
0x0404: Status Register
0x0408: Source Address
0x040C: Destination Address
0x0410: Transfer Size in Bytes

```
void dma_copy ( uint32_t * source,
                uint32_t * dest,
                uint32_t size) {

    *((volatile uint32_t *) (0x0408)) = source;
    *((volatile uint32_t *) (0x040C)) = dest;
    *((volatile uint32_t *) (0x0410)) = size;
    *((volatile uint32_t *) (0x0400)) = 0x1; //start

    //spin until copy done
    while( *((volatile uint32_t *) (0x0404)) != 0x1) {;}
}
```

Using DMA from the CPU:

0x0400: Control Register
0x0404: Status Register
0x0408: Source Address
0x040C: Destination Address
0x0410: Transfer Size in Bytes

```
void dma_start_copy (    uint32_t * source,
                        uint32_t * dest,
                        uint32_t size){

    *((volatile uint32_t *) (0x0408))= source;
    *((volatile uint32_t *) (0x040C))= dest;
    *((volatile uint32_t *) (0x0410))= size;
    *((volatile uint32_t *) (0x0400))= 0x1; //start
}

void dma_wait_for_done(){
    //spin until copy done?
    while( *((uint32_t) (0x0404)) != 0x1){;}
}
```

volatile keyword omitted!

Using DMA from C:

```
int main () {  
  
    dma_start_copy (camera, buf1, BUF_SIZE);  
    dma_wait_for_done();  
  
    while (true){  
        dma_start_copy (camera, buf2, BUF_SIZE);  
        detect_face(buf1);  
        dma_wait_for_done();  
  
        dma_start_copy (camera, buf1, BUF_SIZE);  
        detect_face(buf2);  
        dma_wait_for_done();  
    }  
}
```

Real DMA

Register Address Map

Table 2-6: AXI CDMA Register Summary

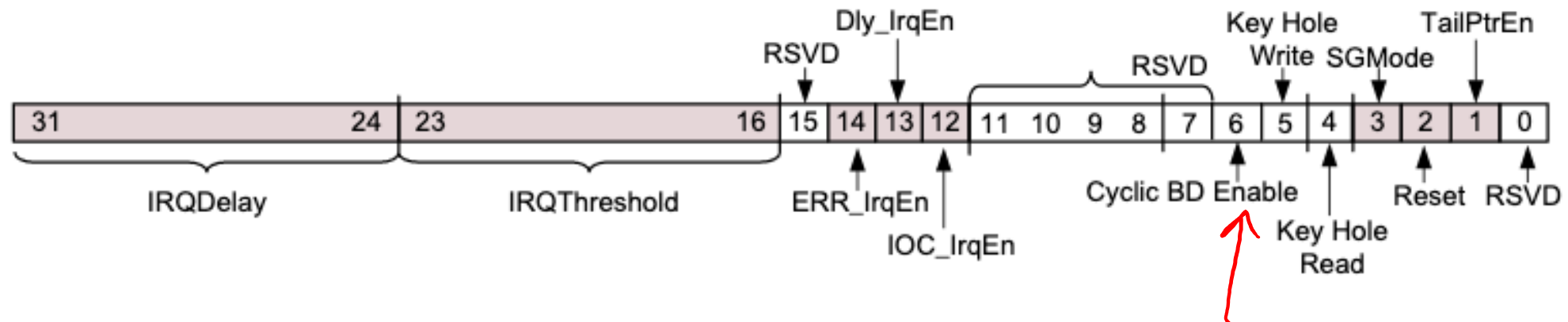
Address Space Offset ⁽¹⁾	Name	Description
00h	CDMACR	CDMA Control
04h	CDMASR	CDMA Status
08h	CURDESC_PNTR	Current Descriptor Pointer
0Ch ⁽²⁾	CURDESC_PNTR_MSB	Current Descriptor Pointer. MSB 32 bits. Applicable only when the address space is greater than 32.
10h	TAILDESC_PNTR	Tail Descriptor Pointer
14h ⁽²⁾	TAILDESC_PNTR_MSB	Tail Descriptor Pointer. MSB 32 bits. Applicable only when the address space is greater than 32.
18h	SA	Source Address
1Ch ⁽²⁾	SA_MSB	Source Address. MSB 32 bits. Applicable only when the address space is greater than 32.
20h	DA	Destination Address
24h ⁽²⁾	DA_MSB	Destination Address. MSB 32 bits. Applicable only when the address space is greater than 32.
28h	BTT	Bytes to Transfer

Real DMA

Register Details

CDMACR (CDMA Control – Offset 00h)

This register provides software application control of the AXI CDMA.



Other DMA tweaks

```
void dma_copy (uint32_t * source,
               uint32_t * dest,
               uint32_t size,
               uint32_t inc_from, uint32_t inc_to)
{
    register uint32_t reg;

    for (int i = 0; i < size; ++i){

        reg = (inc_from ? *source[i] : *source);

        if (inc_to) dest[i] = reg;
        else       dest = reg;
    }
}
```

Other DMA tweaks

- Interrupts (not in E315)
- Repeat the transfer?

References

- https://www.xilinx.com/support/documentation/ip_documentation/axi_cdma/v4_1/pg034-axi-cdma.pdf

11: Direct Memory Access (DMA)

Engr 315: Hardware / Software Codesign
Andrew Lukefahr
Indiana University

