Test

Exam Review

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University

Some material taken from EECS370 at U. of Michigan

Announcements

- P7: Due next Friday.
- Exam: on Monday
- P8: After that.

Exam Details

- Main 5 sections
  - Multiple questions / section

- Some short answer
- Some fill-in-the blank/code/table

A “Cheat” Sheet is Allowed

- 2-sided
- 8.5”x11” paper
- Handwritten (not photocopied)

Major Topics

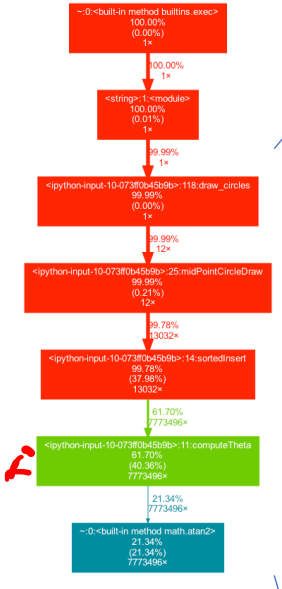
- Performance Profiling
- Data Structures

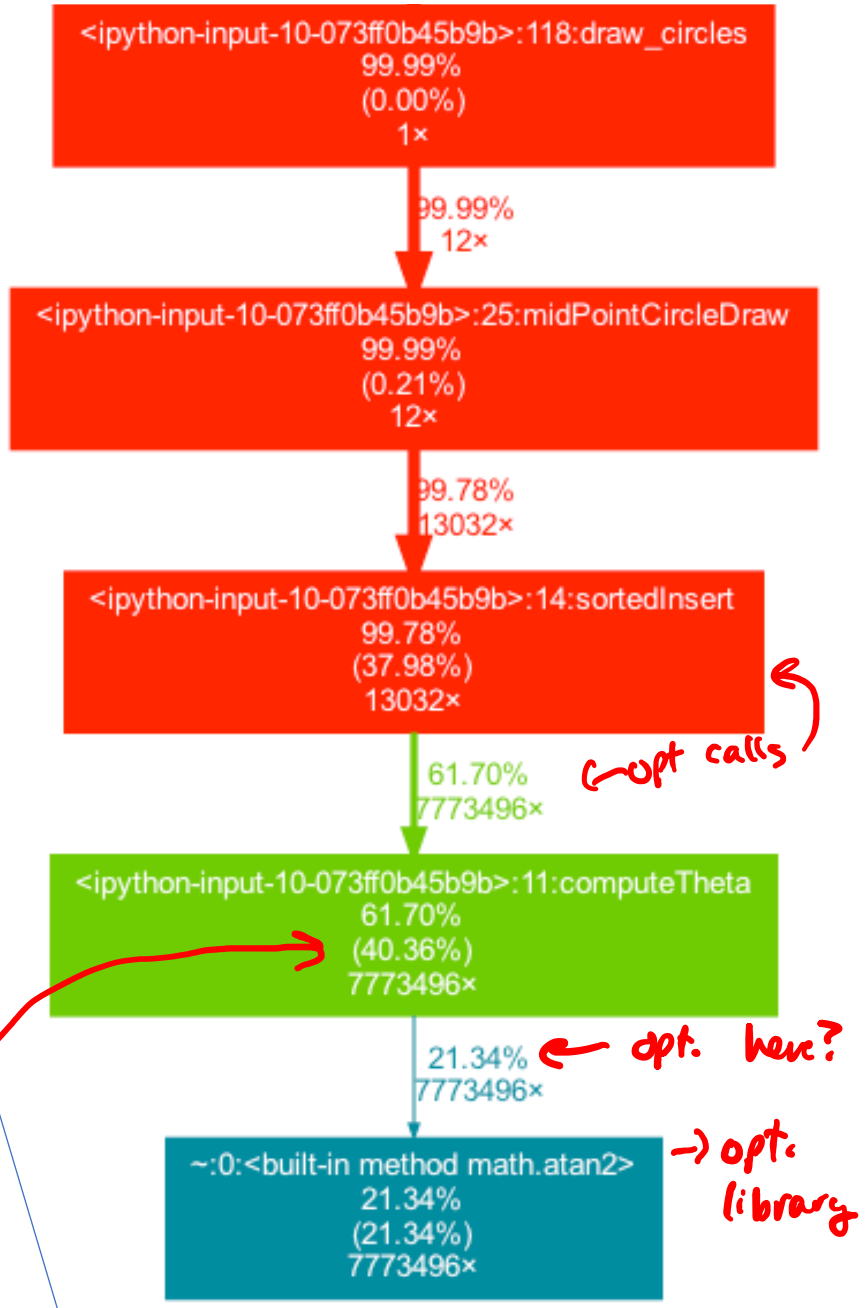
- Bus Interfaces
- MMIO
- DMA

- Pipelining

Performance Profiling

- What is profiling? performance
  Measurement of applications runtime, memory allocation
- What function should we be optimizing here?
  → Red : opt total calls
  Green: opt. runtime
- How do you know?
  → biggest fctn calls
  → biggest overall runtime



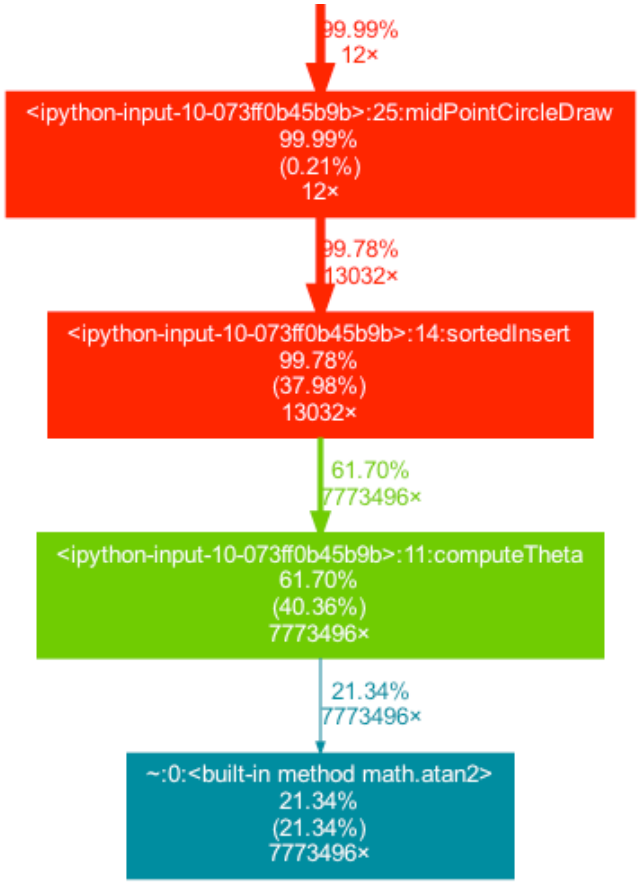


Algorithm Tuning

```
def computeTheta(self, x,y, x_centre, y_centre):
    return math.atan2(x-x_centre, y-y_centre)

def sortedInsert(self, theList, x, y, x_centre, y_centre):
    for index, value in enumerate(theList):
        oldTheta = self.computeTheta(value[0],value[1],x_centre,y_centre)
        newTheta = self.computeTheta(x,y, x_centre, y_centre)
        if oldTheta > newTheta:
            theList.insert(index, (x,y))
            return theList
    theList.append((x,y))
    return theList
```

value[2]

theta



Data Structures

- When is better here? list or array?
  - Inserting at the beginning? linked list
  - Accessing the element at position N (i.e. values[n]) array
  - Accessing elements sequentially? → doesnt matter, about the same
- What's funny about Python's lists?

→ wierd hybrid of lists & arrays

Bus Interfaces

- AXI4 “Full” vs. AXI4 Lite vs. AXI4 Stream

- What are the benefits of each?
- Where do we use them?

	Full	Lite	Stream
Ease of use	complex	medium	simple
MMFO?	MMFO	MMFO	NOT MMFO
Speed	fast	slow	fast

Bus Interfaces

P3 - EMA (Stream)

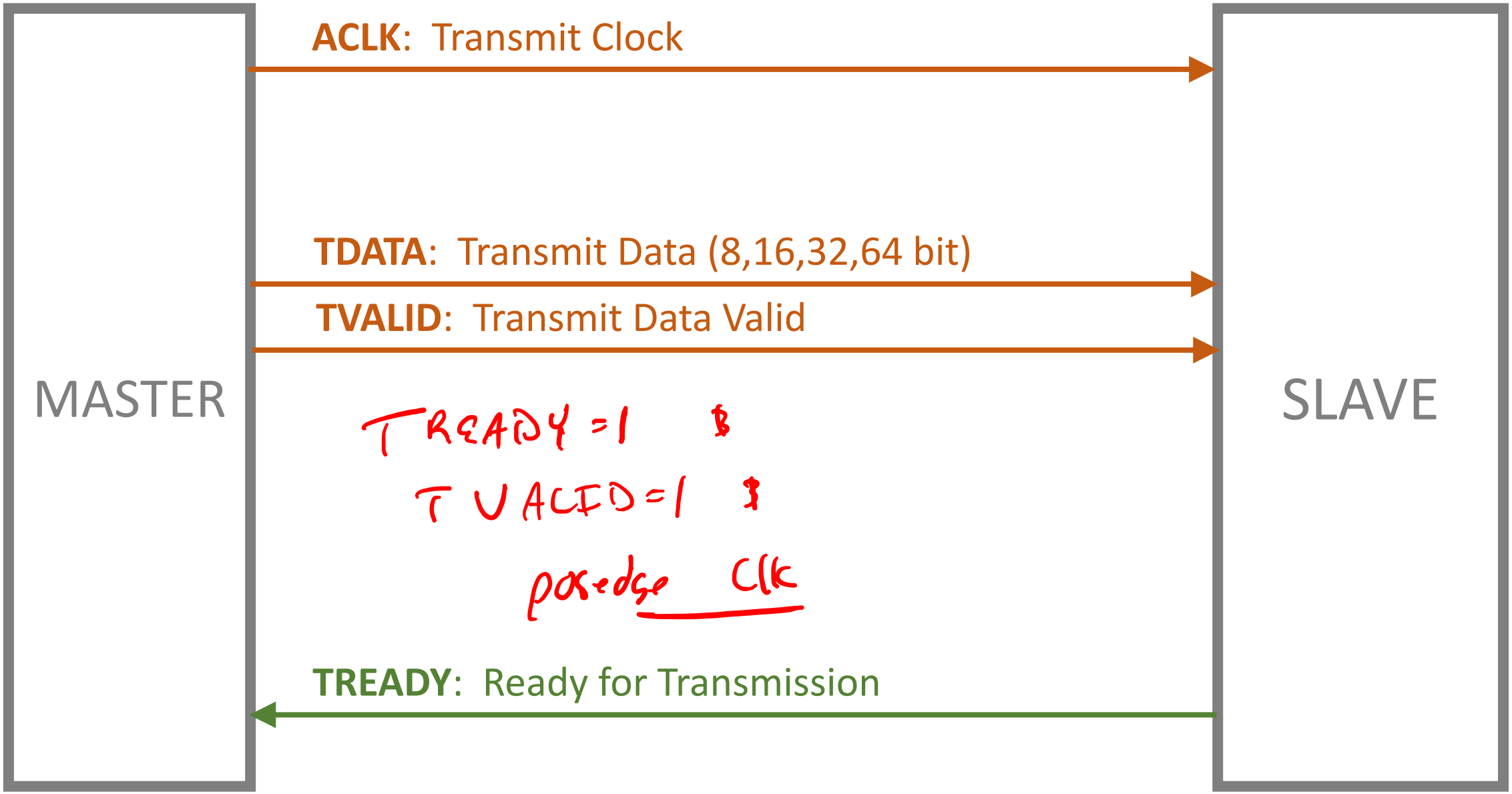
P4/P5 Popcut MMIO (Lite)

P6/P7 PC. DMA (Full)

- AXI4 "Full" vs. AXI4 Lite vs. AXI4 Stream

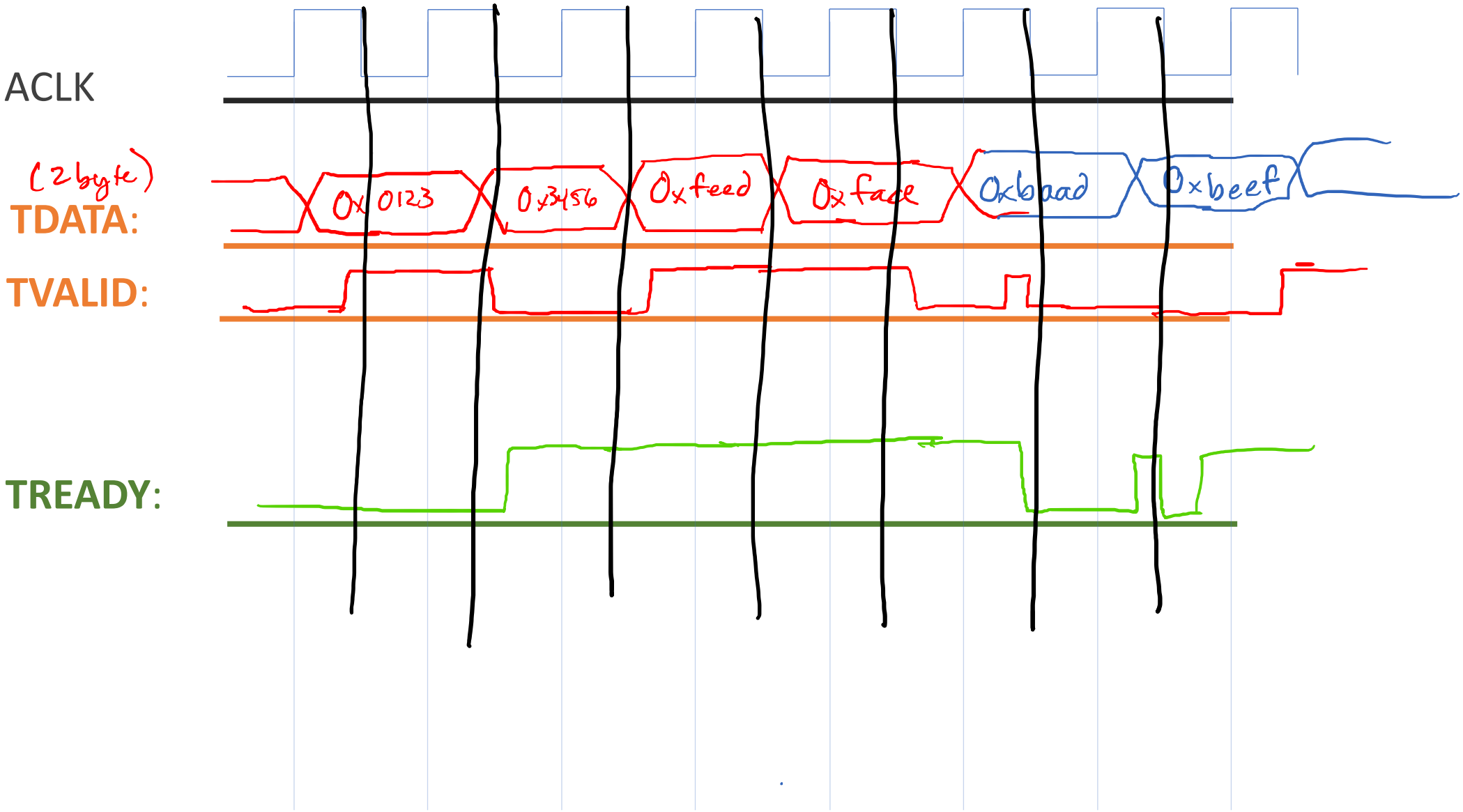
- What are the benefits of each?
- Where do we use them?

	Full	Lite	Stream
Benefits -	MMIO High. Perf.	MMIO Simple	Simple, High Perf.
Drawbacks -	High Complexity	low perf.	NO MMIO



When is a transmission valid?

Transferring data on a AXI4-Stream Bus.



MMIO

- Define MMIO? - memory mapped Inputs / Outputs
- What is MMIO? - inputs / outputs w/ addressable interface
- Why do we use it?

access I/O w/ traditional load + store instructions

MMIO Loads

- In ASM?

```
mov r2, 0x40000000 ;
ldr r3, [r2, 0x144];
```

- In C?

```
# define ADDR 0x40000144
uint32_t x = *((volatile uint32_t *)(ADDR))
= *((volatile uint32_t *)(0x40000144))
```

MMIO Loads and Stores

- In ASM?

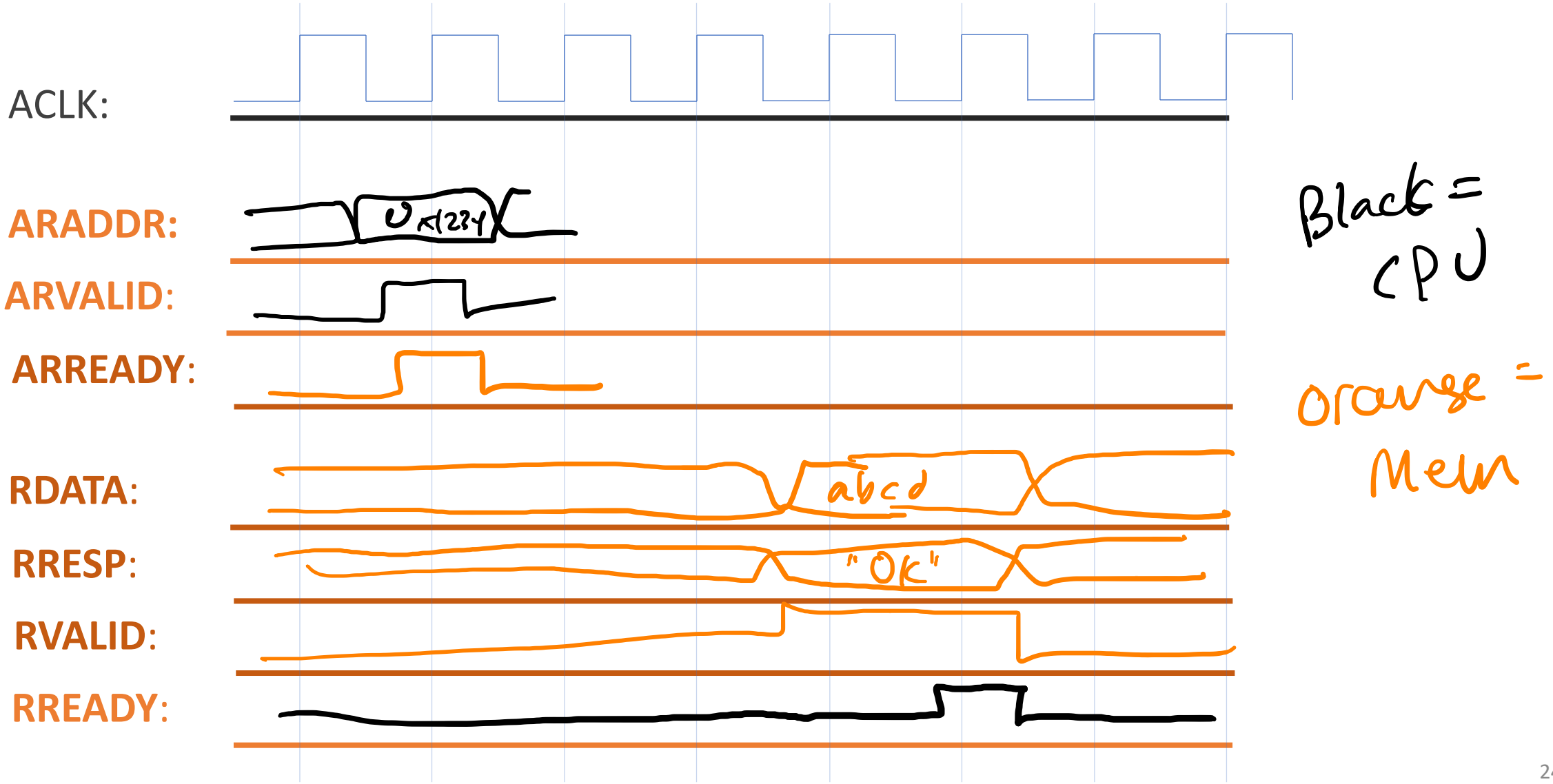
```
mov r2, 0x40000000 ;
ldr r3, [r2, 0x144];
```

- In C?

```
uint32_t  x = *(volatile uint32_t *)(0x40000144);
```

Store: *(volatile uint32_t *)(0x40000144) = 32;

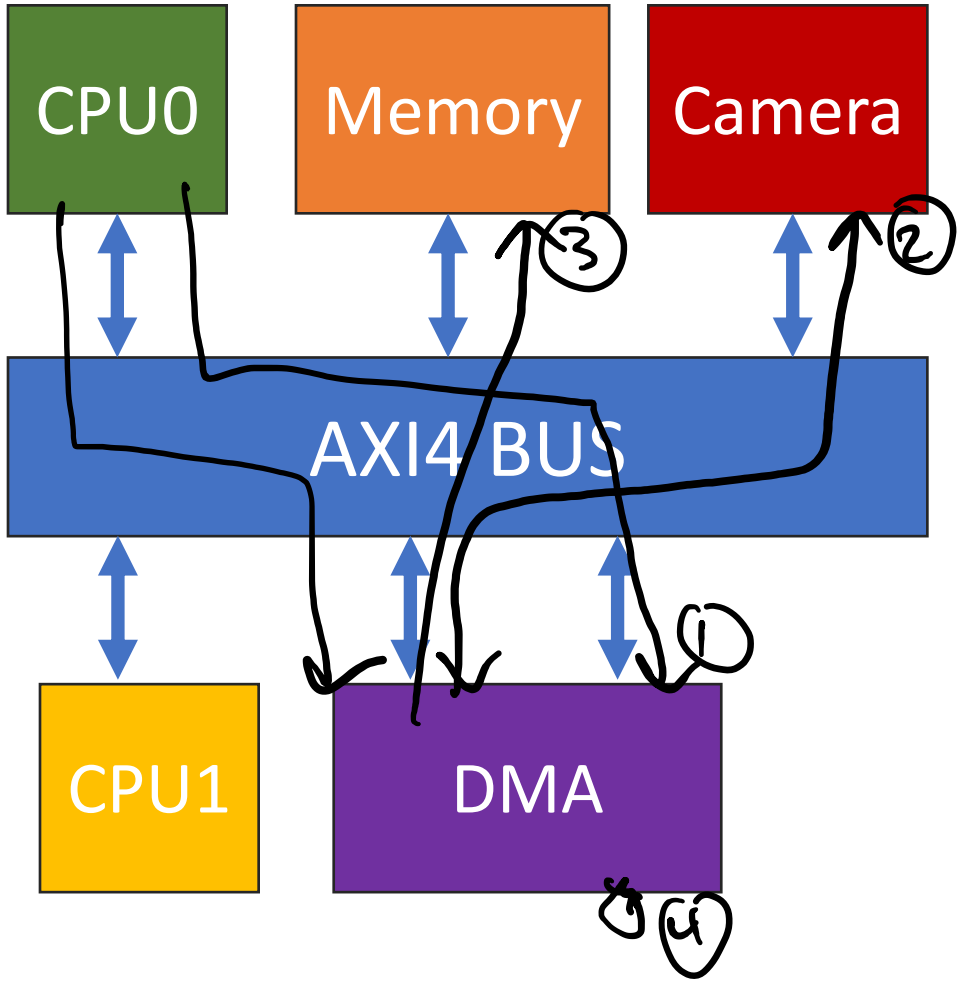
AXI4Lite: Load 0x1234, response: 0xabcd



Linux MMIO?

- What’s weird about C/MMIO with Linux?

DMA



- Define DMA?
- What's the goal of DMA?
- What steps are involved?

1. CPU tells DMA what/where
2. DMA loads data from source
3. DMA stores data to Dest.
4. Adjust Address
5. loop until done
6. set "done" flag
7. CPU polls for done flag

DMA Control Design

```
void dma (uint32_t * from,
          uint32_t * to,
          uint32_t size)
{
   register uint32_t reg;

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

      reg = from[i]; //load

      to[i] = reg; //store
   }
}
```

- What interfaces do you need?
- How do you start/stop DMA?
- Design a DMA state machine?

All DMA 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)

= 1 for done
= 0 for !done

DMA Control

```
void dma (uint32_t * from,
            uint32_t * to,
            uint32_t size)
{
   register uint32_t reg;

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

      reg = from[i]; //load

      to[i] = reg; //store
   }
}
```

- AXI4 Master Interface
  - Loads + Stores
- 5 MMIO registers
  - Control (Start)
  - Status (Done)
  - Source (`from`)
  - Destination (`to`)
  - `size` (in Bytes)

Using DMA from CPU's side:

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

*(volatile uint32_t *)

```
void dma (uint32_t * from,
          uint32_t * to,
          uint32_t size)
{
    ~(0x0408) = from;
    ~(0x040C) = to
    ~(0x0410) = size
    ~(0x0400) = 1;

    while (~(0404) != 0x0) {;}
}
```

Using DMA from CPU's side:

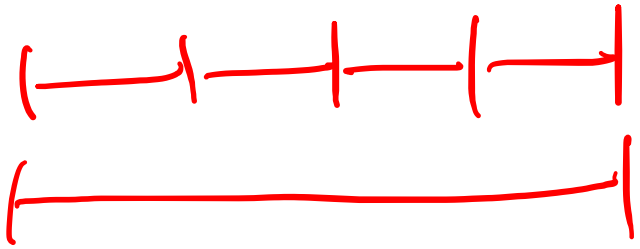
0x0400: Control Register
0x0404: Status Register
0x0408: Source Address
0x040C: Destination Address
0x0410: Transfer Size in Bytes

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

    *((volatile uint32_t*)(0x0408))=src;
    *((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){;}
}
```

0x01

DMA System Interface

Pipelining



- Define Latency

  time for 1 operation

- Define Throughput

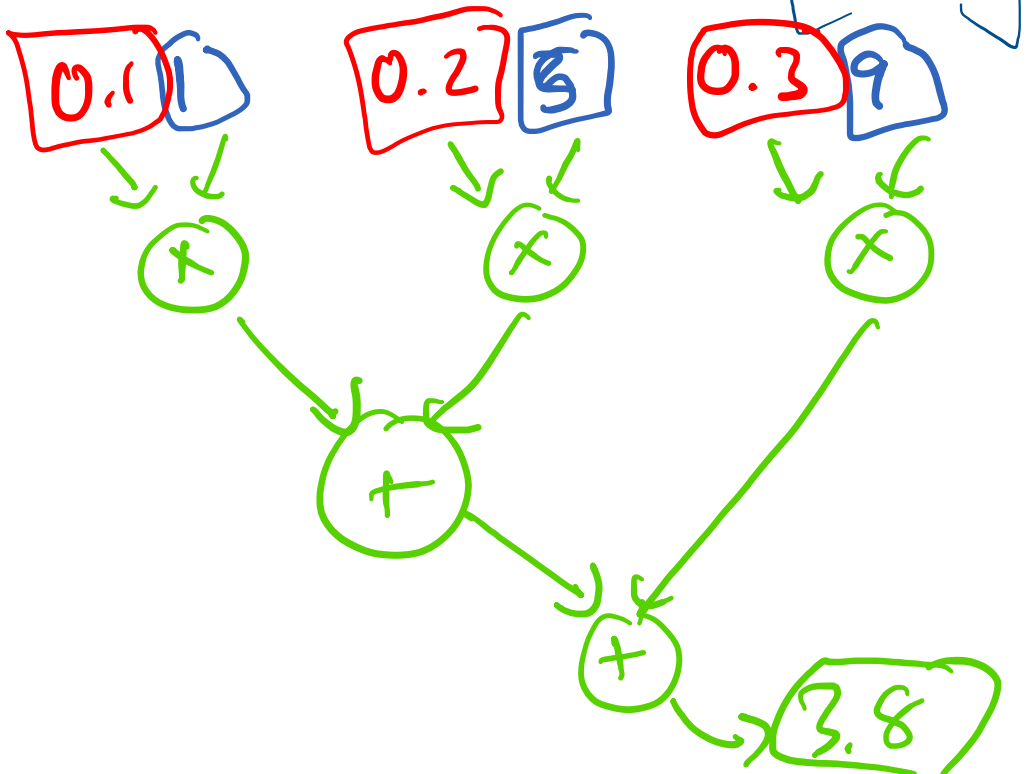
  average operations per time

  (per second)

  (per cycle)

Build a dependence graph for this computation

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 1 \\ 5 \\ 9 \end{bmatrix} = \begin{bmatrix} 3.8 \end{bmatrix}$$



Build a FMAC timeline for this computation

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 3.8 & 4.4 & 5 & 5.6 \end{bmatrix}$$

Exam Review

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University