04: Bus Interfaces

Engr 315: Hardware / Software Codesign

Andrew Lukefahr

Indiana University

Announcements

- Slack – See Website
- Office Hours – See Website / Syllabus
- P2: Due next Wednesday → 13th Demo: Fri, 15th
  - ~~(New Project, Could be some bumps)~~
  - Need a Pynq
  - Groups of 2 allowed
- P3: Out now!

Failed Login & Disk Space

- If you can’t log into the Linux machines:
  - It might be you are out of disk drive space

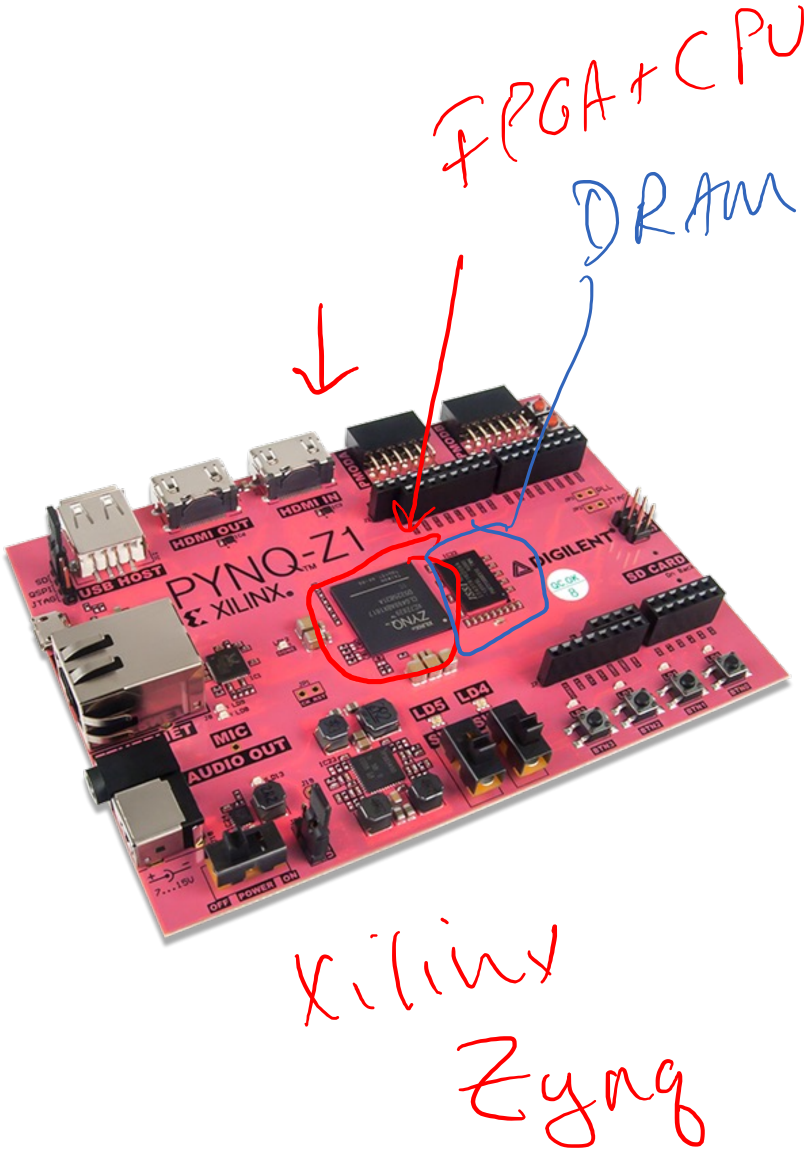
- ssh into silo/kj and do the following:

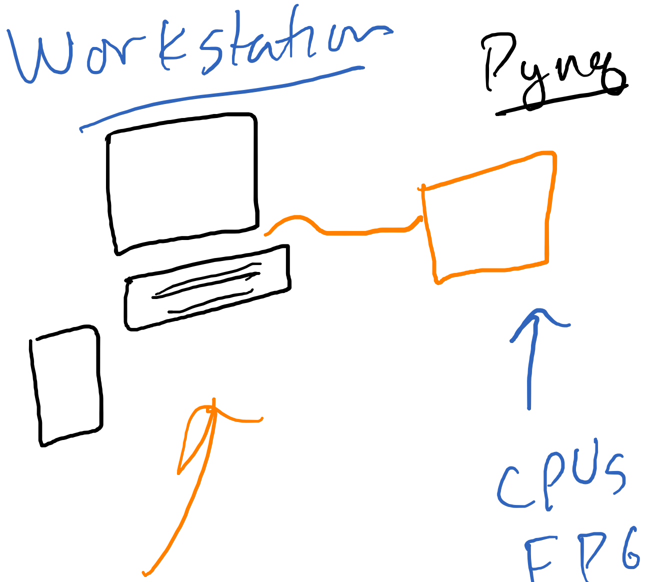
```
$ quota
```

If overfull, remove some things, then try to log in again.

The Pynq

- Used for P2 onward
- System-on-Chip
  - SoC - "S-O-C" or "Sock"
- Contains both FPGA and CPU
- Runs Linux + Python





Linux

→ Vivado

Internet

CPUs (2)

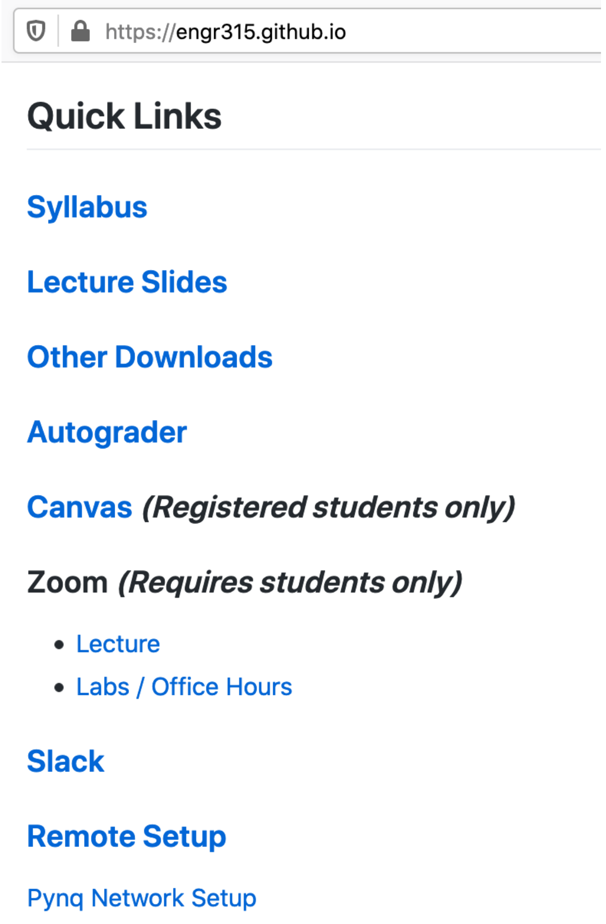
FPGA (1)

Linux (ubuntu)

Jupyter notebook

Setup Notes

- 4111 is best.
  - Everything already set up
- Can work from home
  - need Pynq networked
  - “Some” effort support
- ~~Pure-Remote students~~
  - ~~Email me~~



https://engr315.github.io

Quick Links

Syllabus

Lecture Slides

Other Downloads

Autograder

Canvas ***(Registered students only)***

Zoom ***(Requires students only)***

- Lecture
- Labs / Office Hours

Slack

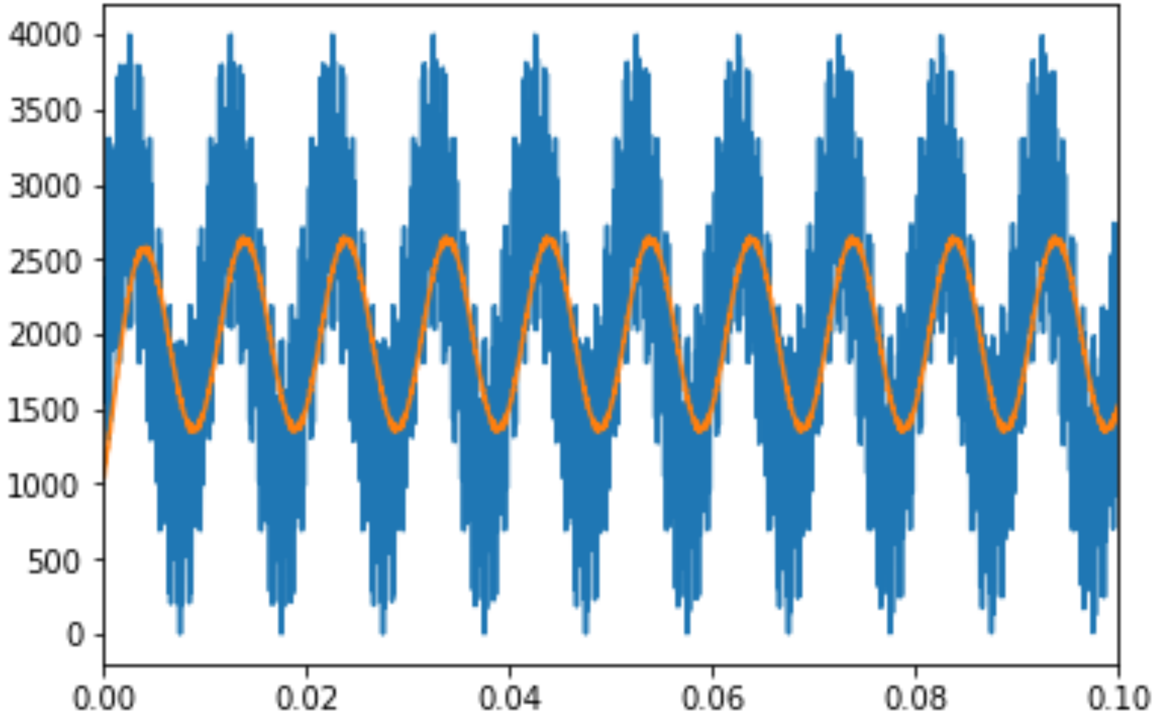
Remote Setup

Pynq Network Setup

Let’s talk P2 (and P3)

- What is EMA?
- Pynq Setup
  - The password is ‘iuxilinx’
- e315helper.py
- Vivado Setup

Signal Filtering



EMA is an IIR Filter.

$$y[n] = \underset{\substack{\uparrow\\ \text{scale factor}}}{\alpha}\, \underset{\substack{\uparrow\\ \text{current input}}}{x[n]} + \underset{\substack{\uparrow\\ \text{scale factor}}}{(1-\alpha)} \cdot \underset{\substack{\uparrow\\ \text{last output}}}{y[n-1]}$$

α = alpha

$$\boxed{y[-1] = 0 \text{ by definition}^{*}}$$

1000 for P2

EMA Example

$y[-1] = 0$

$y[n] = \alpha x[n] + (1-\alpha) \cdot y[n-1]$

$\alpha = 0.5$ $x = [0, 10, 0, 0, 10]$

n	y	x	αx		$(1-\alpha)\ y[n-1]$
-1	0	—	—		—
0	0	0	0	$+$	0
1	5	10	$(0.5)(10)$	$+$	0
2	2.5	0	0	$+$	$(0.5)(5) = 2.5$
3	1.25	0	0	$+$	$(0.5)(2.5) = 1.25$
4	5.625	10	$(0.5)(10)$	$+$	$(0.5)(1.25)$

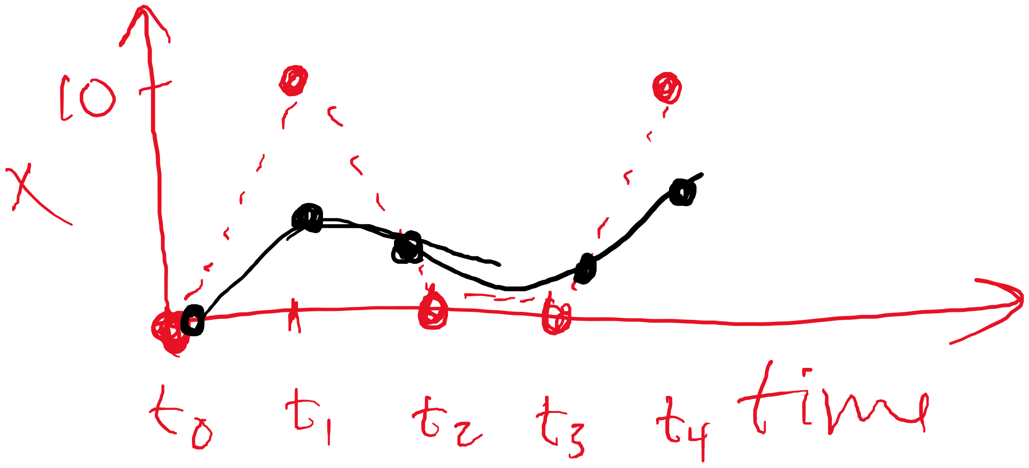
EMA Example

$y[-1] = 0$

$y[n] = \alpha\, x[n] + (1-\alpha)\, y[n-1]$

$\alpha = 0.5$

$x = [0, 10, 0, 0, 10]$



ASSEMBLY DEMO

```
int popcount_asm(uint64_t num)
{
    uint64_t result;
    asm (
        "POPCNT %1, %0   \n"
        : "=r" (result)
        : "mr" (num)
        : "cc"
    );
    return result;
}
```

```
gcc -Wall -march=nehalem -o test.o test.c popcount.c
```

Optimizations thus far

- Algorithmic complexity
- Function calls
- Data structures
- Libraries / Lower-level programming

- Explicitly Skip: Multicore (E201)

Optimizations thus far

- Algorithmic complexity
- Function calls
- Data structures
- Libraries / Lower-level programming

- Explicitly Skip: Multicore (E201)
- Hardware acceleration

Can we improve performance with Python->Verilog interfacing?

Oh Yes!

We could also map popcount to hardware

```
import cPopcount
print (cPopcount.cPopcount(0))

import hwPopcount
print hwPopcount.hwPopcount(0))
```

First we need to understand how CPUs and hardware communicate.

We’re going on an excursion into buses.

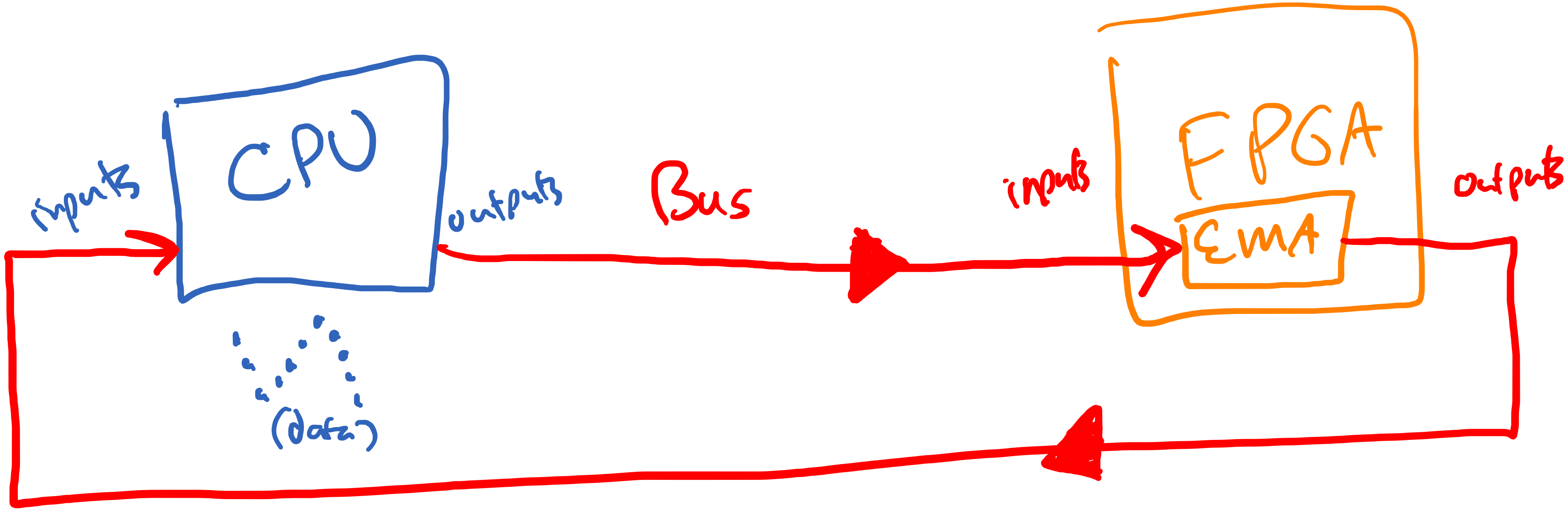
Buses: The hardware interface

- Standardized hardware interfaces for transferring data

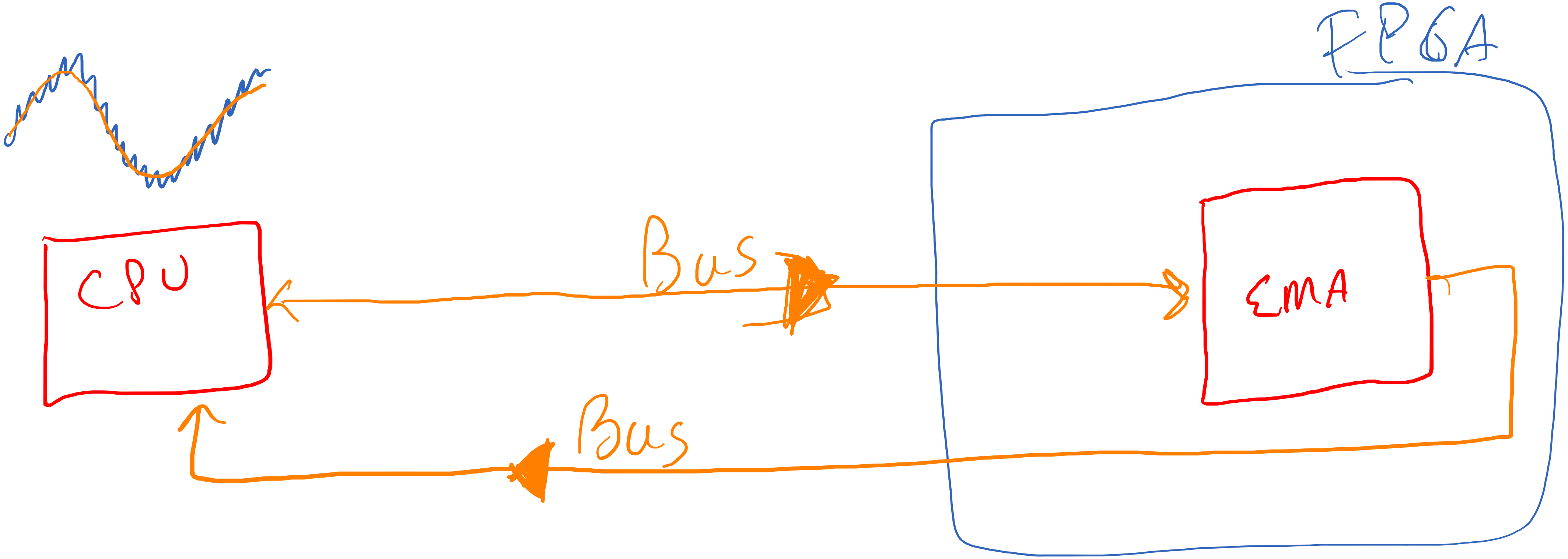
- Off-chip Buses: UART, I2C, SPI, RS-485, CAN, Ethernet
- On-Chip Buses: Wishbone, AHB/APB, AXI

- We’re going to study 2.
  - AXI4-Stream
  - AXI4-Lite

P3 “EMA” uses two buses to move data between CPU + hardware

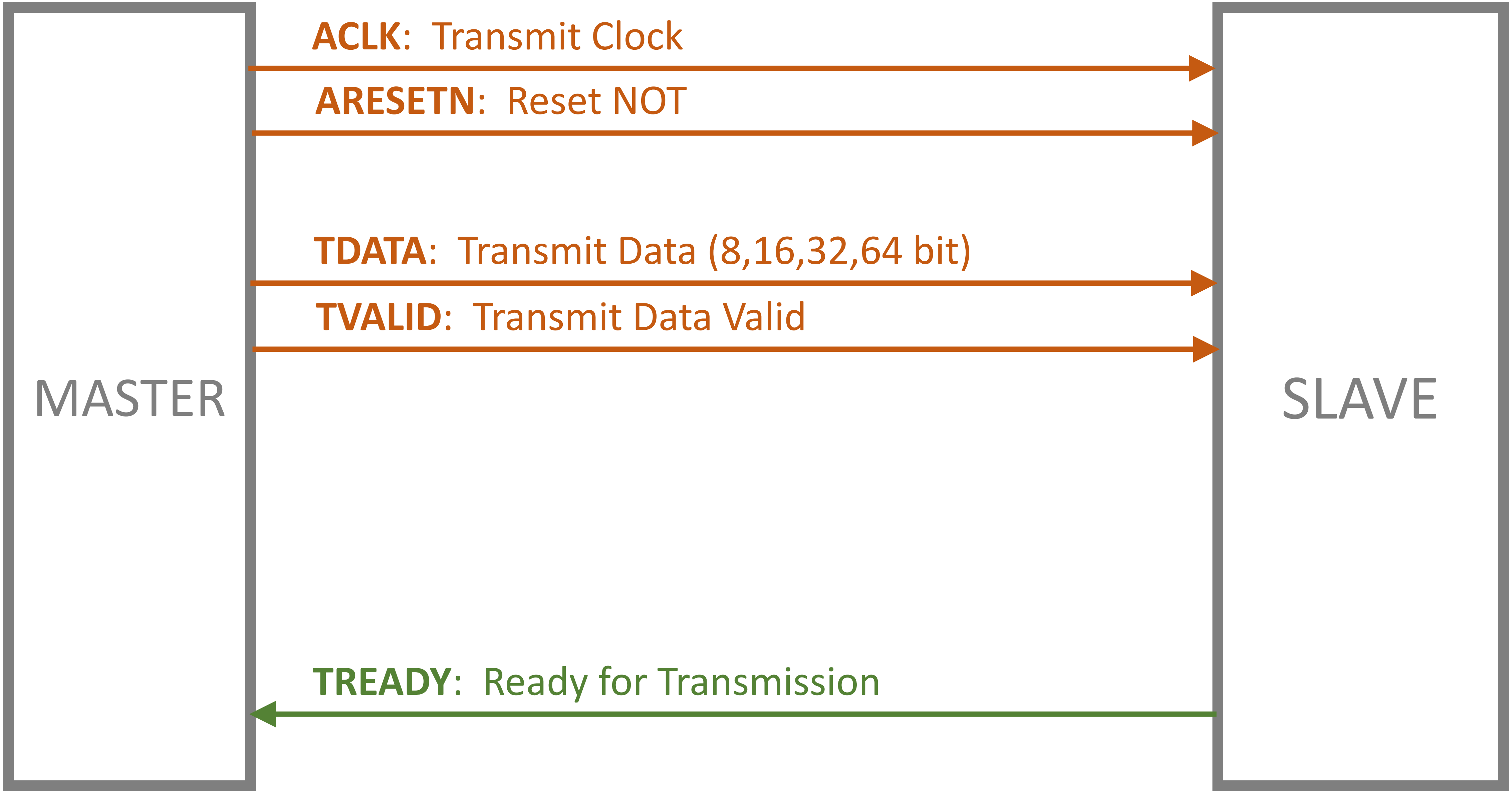


P3 “EMA” uses two buses to move data between CPU + hardware



Bus terminology

- A "transaction" occurs between an "*initiator*" and "*target*"
- Any device capable of being an initiator is said to be a "*bus master*"
    - Usually only one bus master (*single master* vs. *multi-master*).
- A device that can only be a target is said to be a "slave device".



MASTER

ACLK: Transmit Clock

ARESETN: Reset NOT

TDATA: Transmit Data (8,16,32,64 bit)

TVALID: Transmit Data Valid

TREADY: Ready for Transmission

SLAVE

AXI Stream Interface

ARESETN:

AXI Reset NOT

Data (TDATA) is only transferred when

TVALID is **1.**

This indicates the **MASTER** is trying to transmit new data.

TREADY is **1.**

This indicates the **SLAVE** is ready to receive the data.

If either **TVALID** or **TREADY** are 0, **no data is transmitted**.

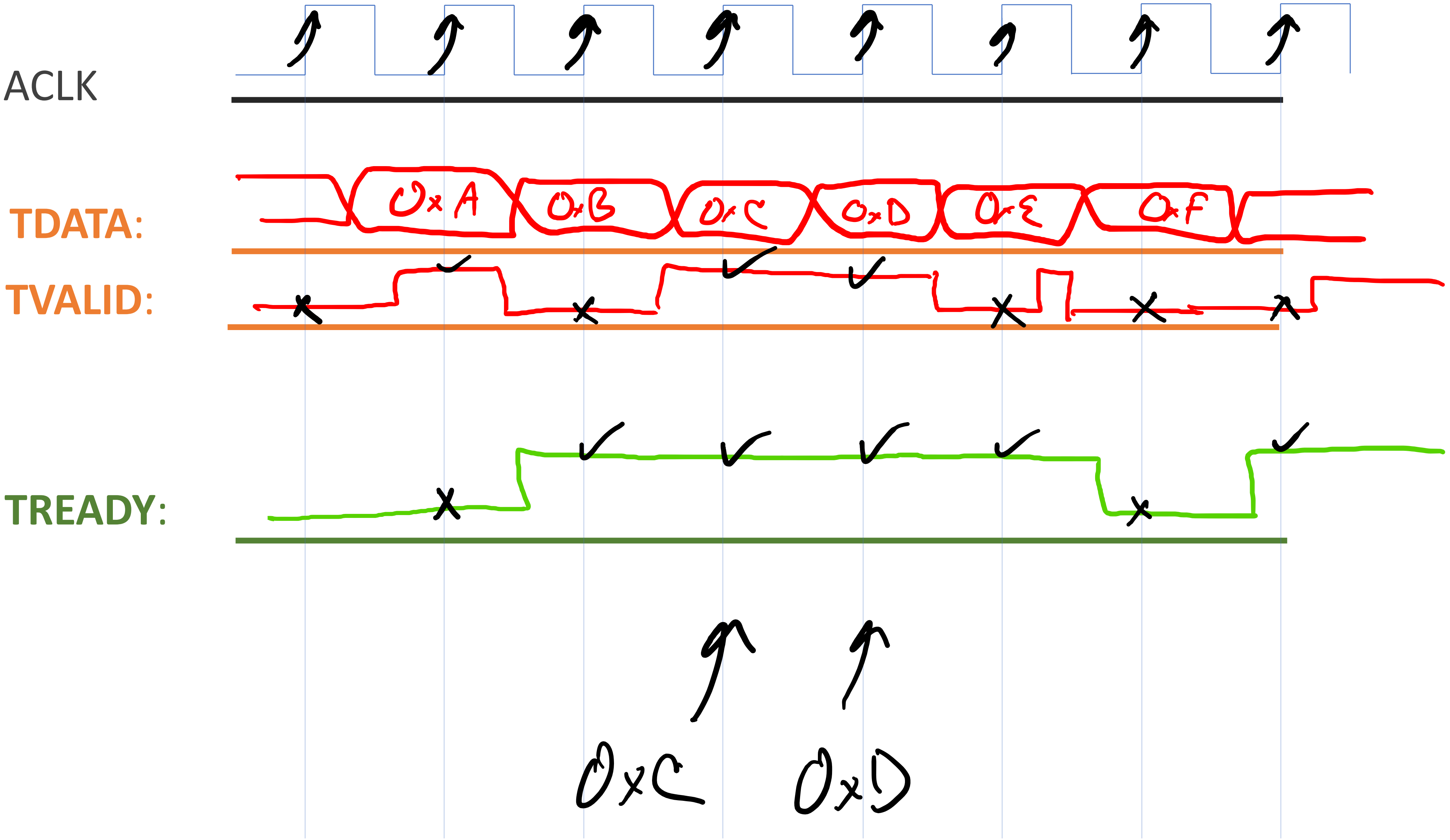
If **TVALID** and **TREADY** are 1, **TDATA** is transmitted

at the positive edge of ACLK

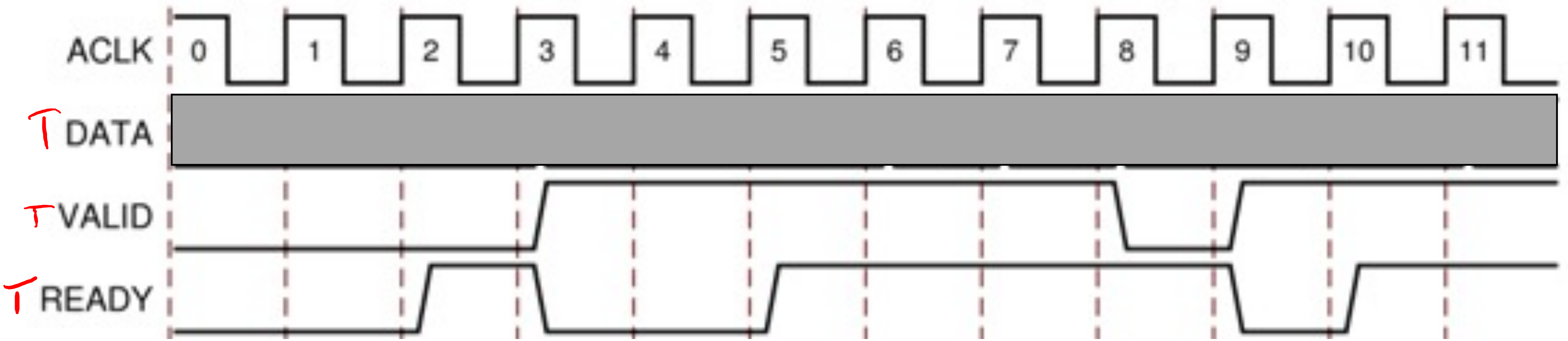
If **TVALID** and **TREADY** are 1, **TDATA** is transmitted

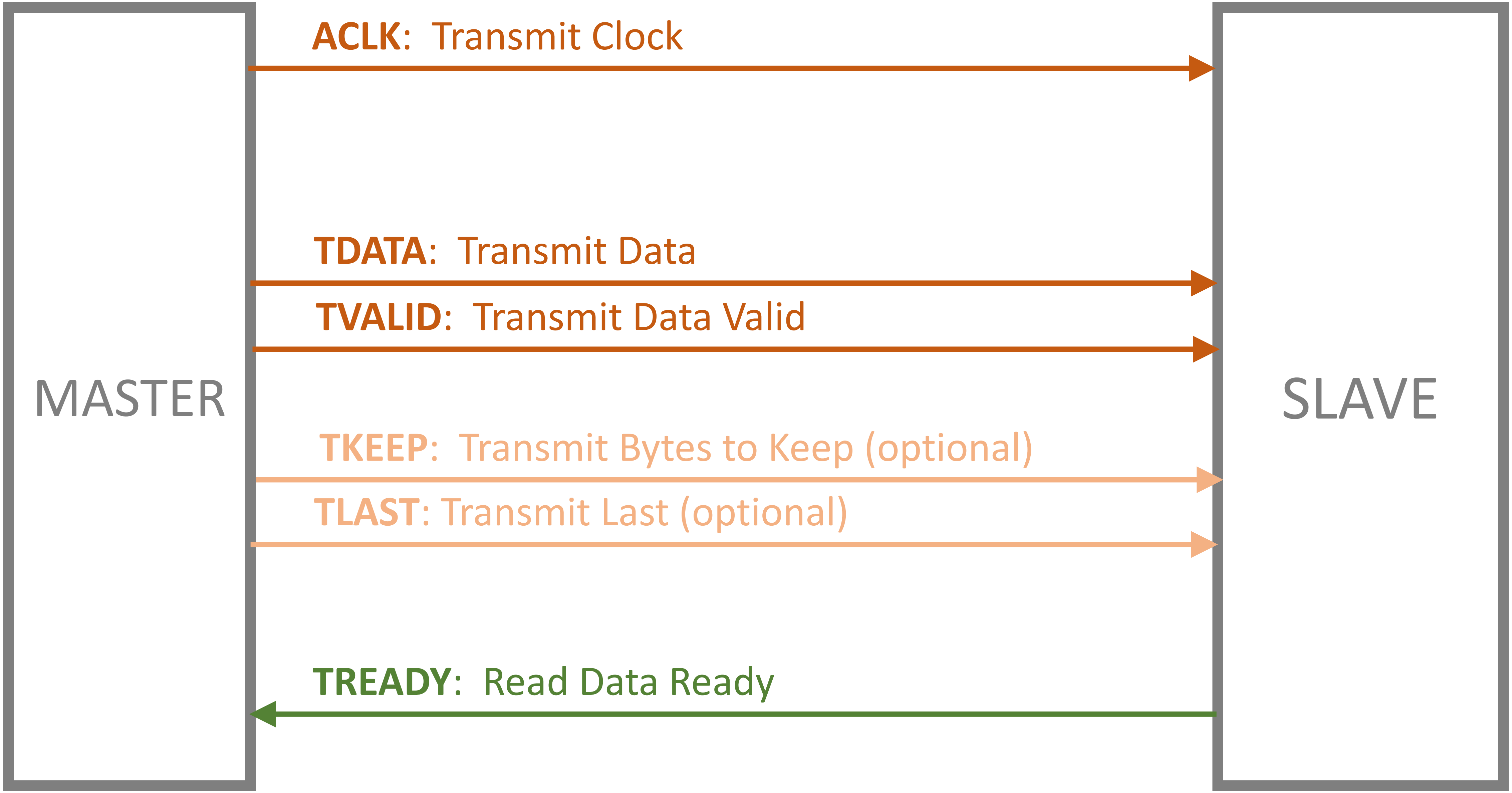
at the positive edge of ACLK

Transferring data on a AXI4-Stream Bus.



When is data transmitted?



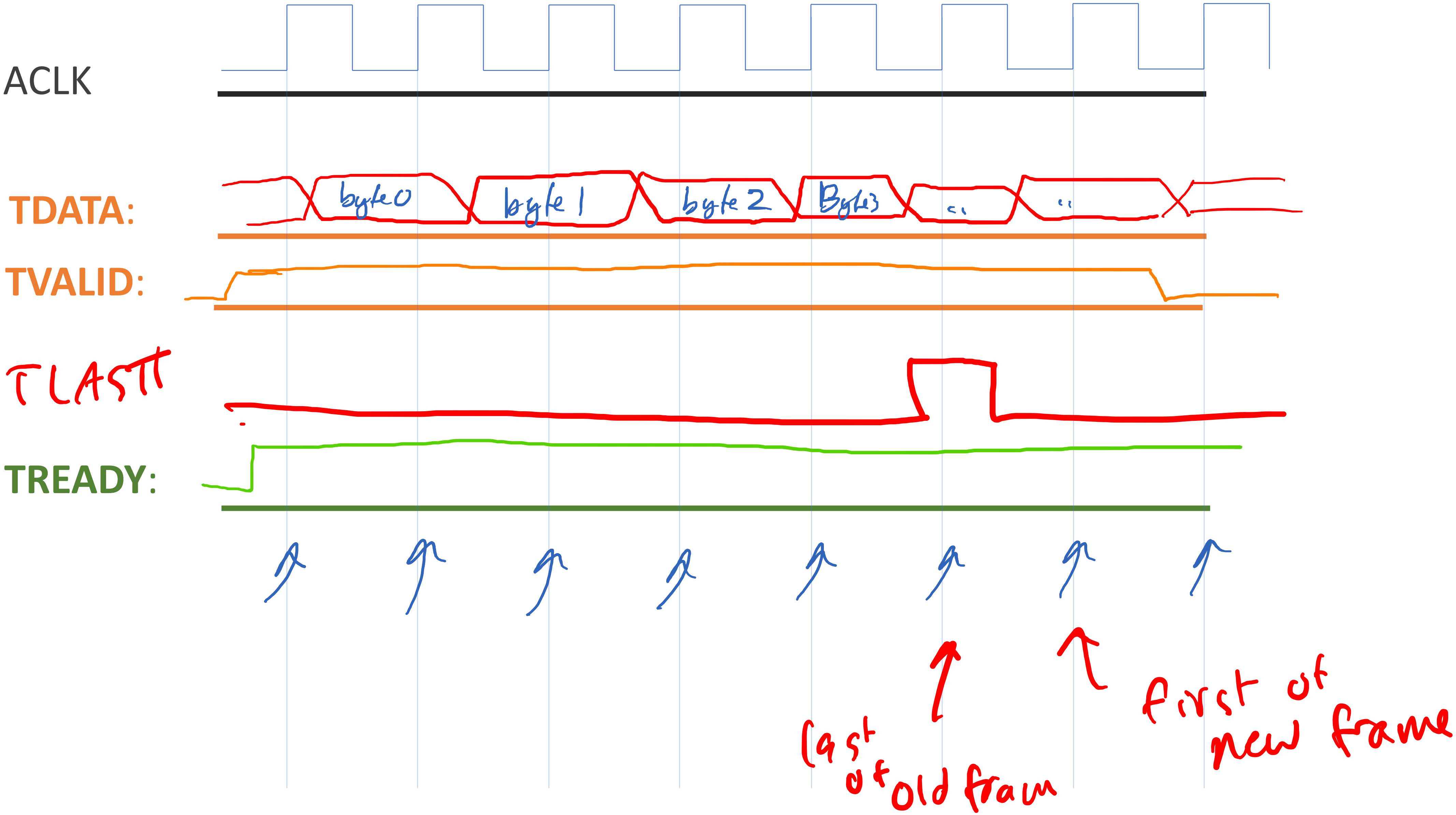


MASTER
SLAVE
ACLK: Transmit Clock
TDATA: Transmit Data
TVALID: Transmit Data Valid
TKEEP: Transmit Bytes to Keep (optional)
TLAST: Transmit Last (optional)
TREADY: Read Data Ready

TLAST

- Special signal to indicate a group or “burst” of transmissions is complete.
- “Indicates the boundary of a packet”

Transferring data on a AXI4-Stream Bus.



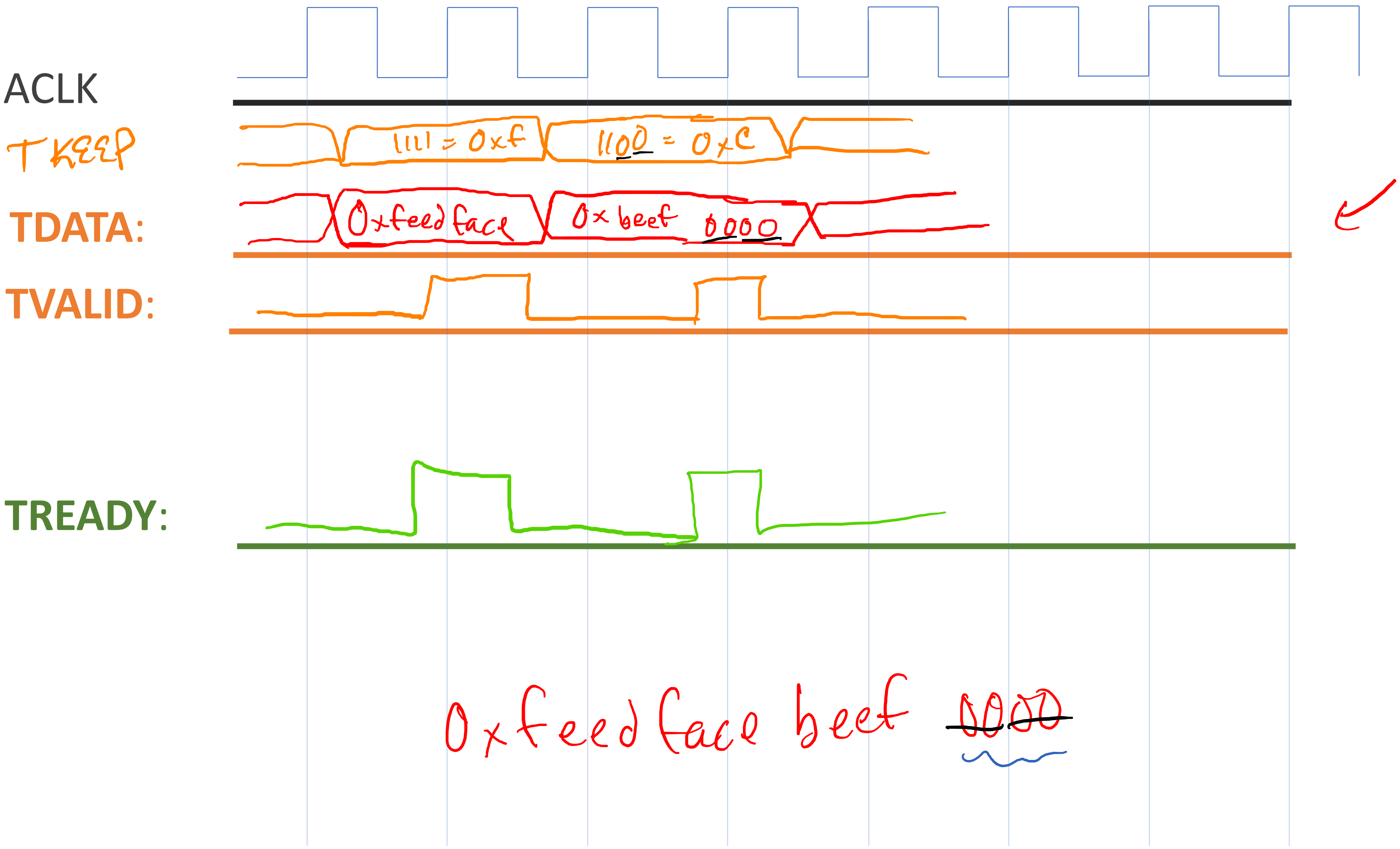
TKEEP

- What if TDATA is 32-bits (4 bytes) wide, and I want to send 6 bytes?

- **TKEEP** let’s me specify which bytes to “keep”.

- **TKEEP** ==1111 -> Keep all 4 bytes (32-bits)
- **TKEEP** ==1100 -> Keep first 2 bytes (16-bit)
- **TKEEP** ==1000 -> Keep first byte (8-bit)

Transferring data on a AXI4-Stream Bus.


ACLK

TKEEP

TDATA:

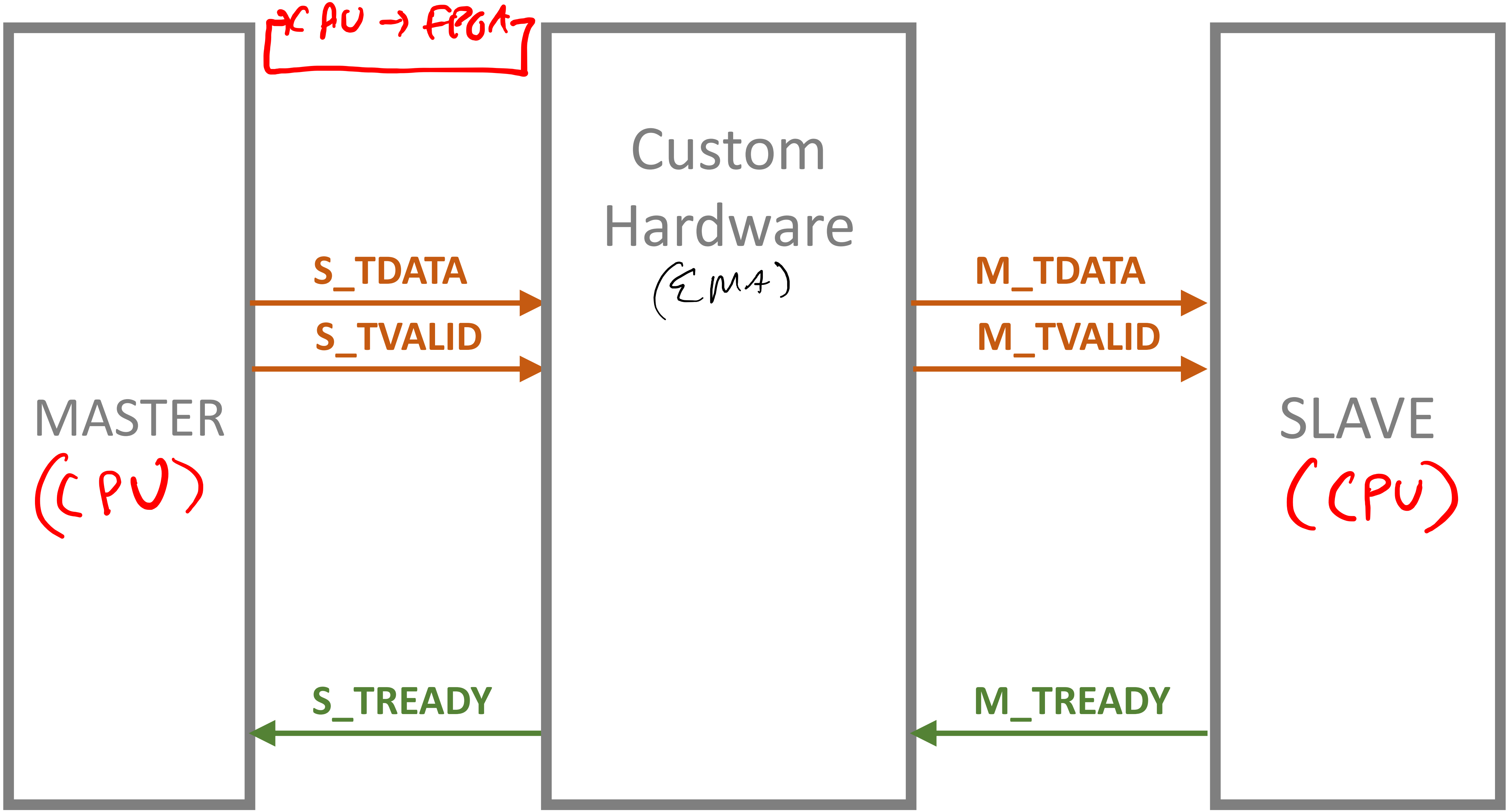
TVALID:

TREADY:

0xfeed face beef ~~0000~~

32-bit (4B)

6B



CPU → FPGA

MASTER
(CPU)

S_TDATA

S_TVALID

S_TREADY

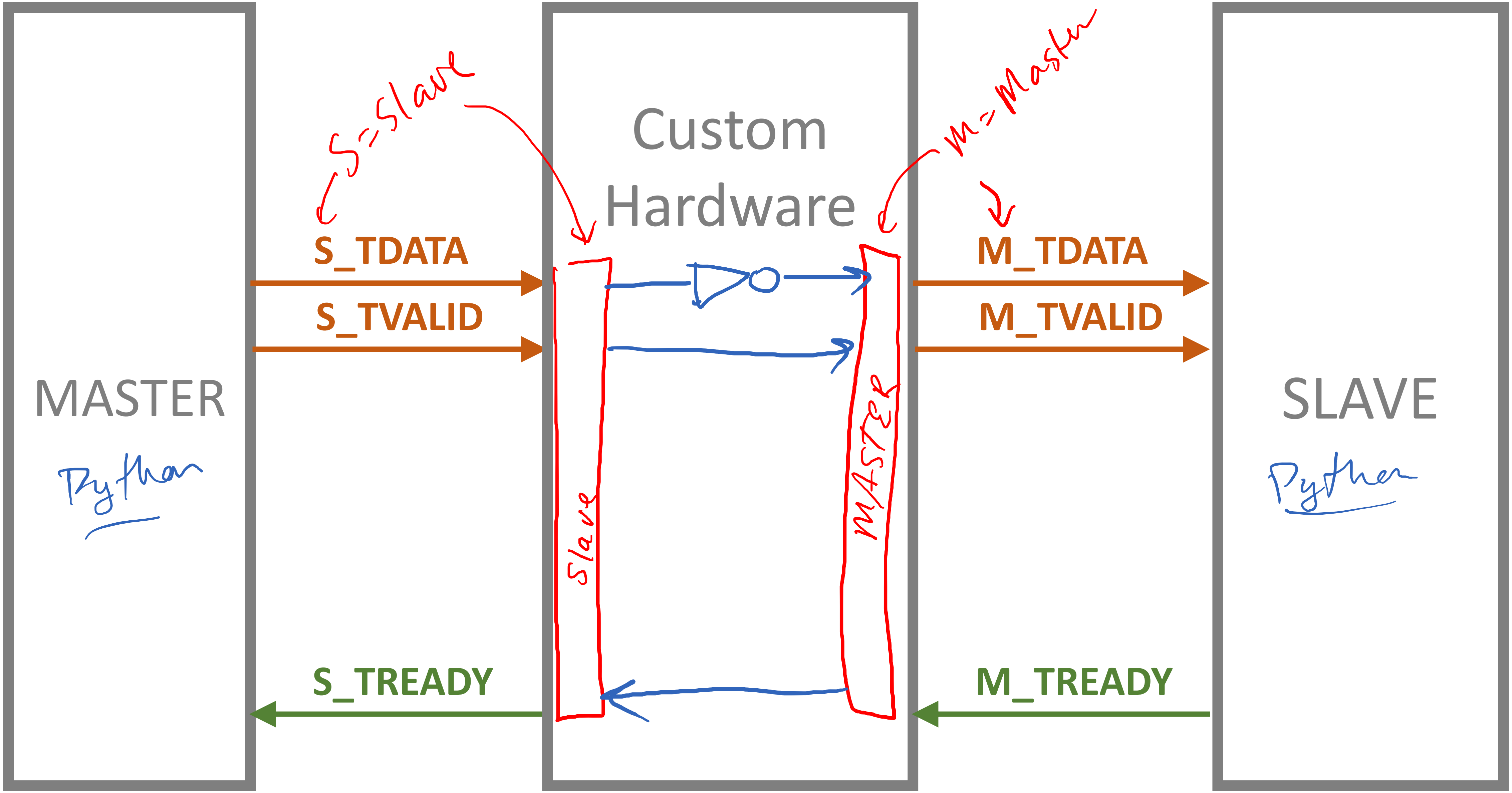
Custom Hardware
(EMA)

M_TDATA

M_TVALID

M_TREADY

SLAVE
(CPU)



MASTER

S_TDATA

S_TVALID

S_TREADY

Custom Hardware

M_TDATA

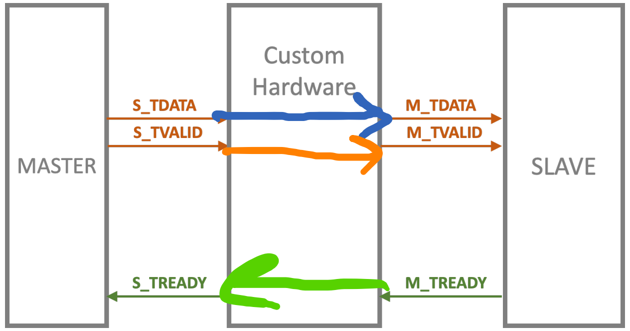
M_TVALID

M_TREADY

SLAVE

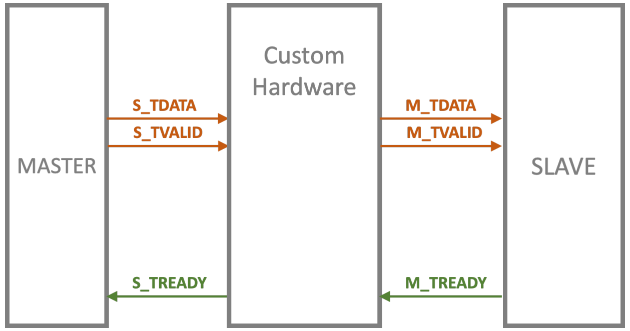
Let’s build a custom block that does nothing!

```
module custom_hw (
        input          ACLK,
        input          ARESET,
        input  [31:0]  S_TDATA,
        input          S_TVALID,
        output         S_TREADY,
        output  [31:0] M_TDATA,
        output         M_TVALID,
        input          M_TREADY
);
      assign    S_TREADY  = M_TREADY;
    → assign    M_TDATA   = S_TDATA;
      assign    M_TVALID  = S_TVALID;

endmodule
```



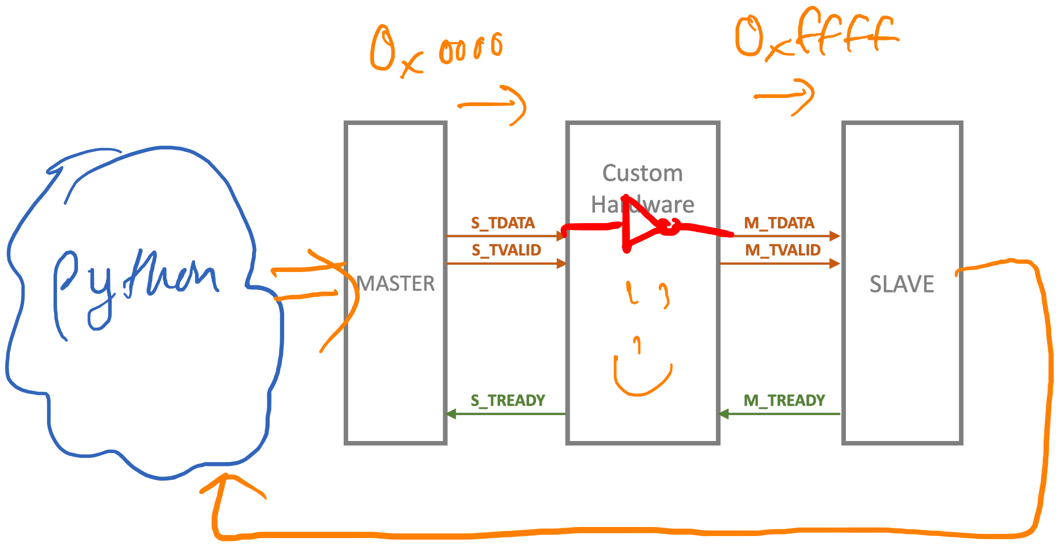
Let’s build a custom block that does nothing!

```
module custom_hw (
       input        ACLK,
       input        ARESET,
       input [31:0] S_TDATA,
       input        S_TVALID,
       output       S_TREADY,
       output [31:0]M_TDATA,
       output       M_TVALID,
       input        M_TREADY
);

assign M_TDATA = S_TDATA;
assign M_TVALID = S_TVALID;
assign S_TREADY = M_TREADY;

endmodule
```



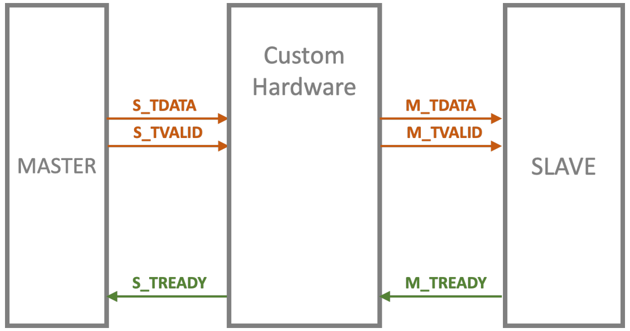
How would I flip all the bits of TDATA?

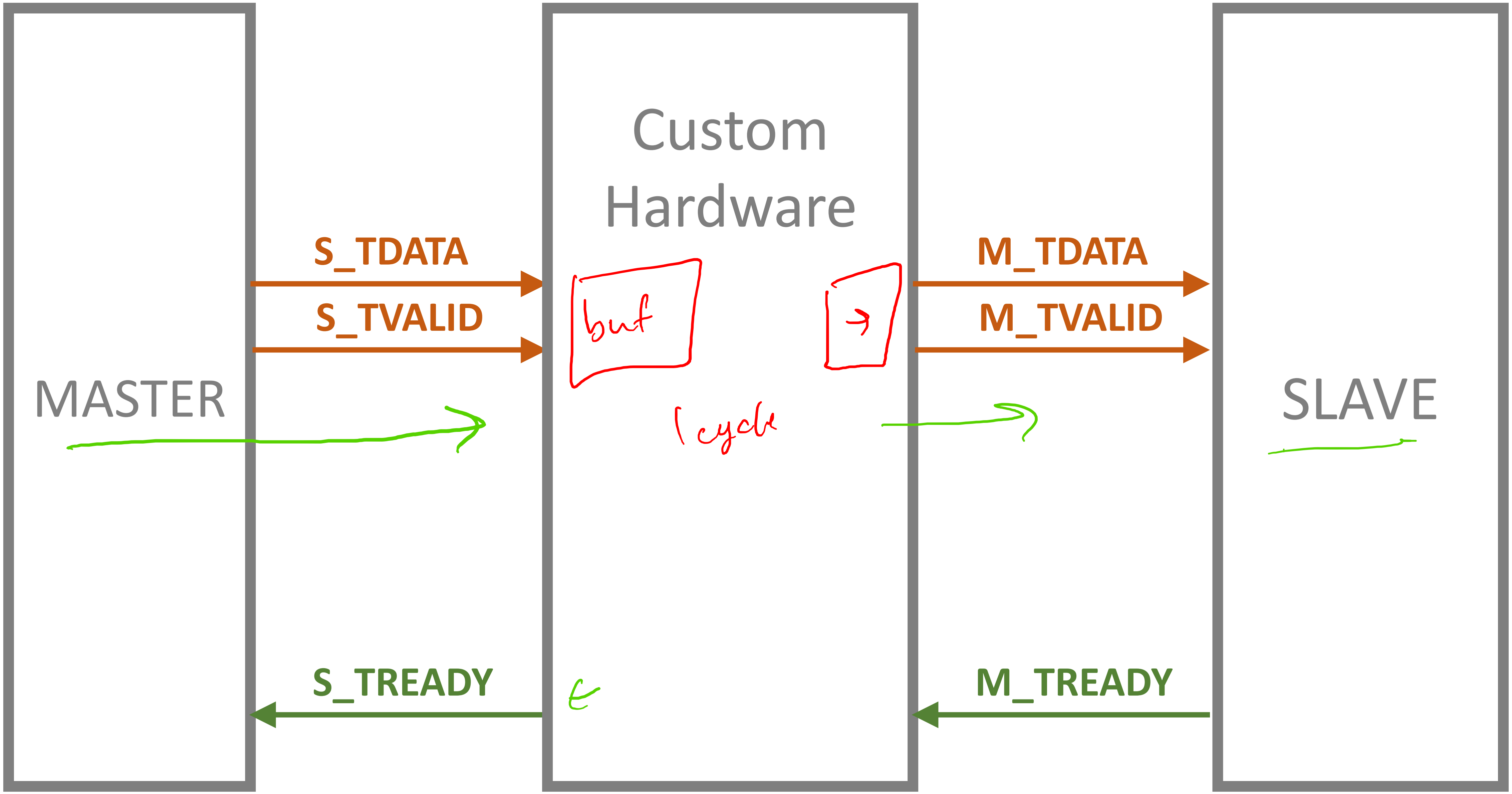
```
module custom_hw (
       input          ACLK,
       input          ARESET,
       input [31:0]   S_TDATA,
       input          S_TVALID,
       output         S_TREADY,
       output [31:0]  M_TDATA,
       output         M_TVALID,
       input          M_TREADY
);

assign M_TDATA = ~S_TDATA;
assign M_TVALID = S_TVALID;
assign S_TREADY = M_TREADY;

endmodule
```



How would I flip all the bits of TDATA?

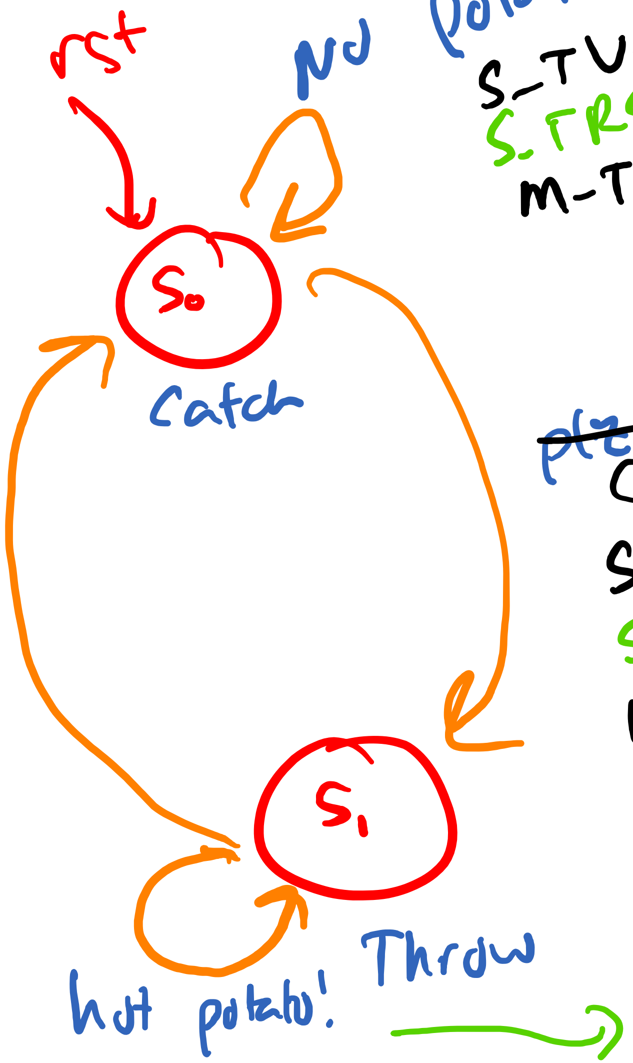
```
module custom_hw (
       input          ACLK,
       input          ARESET,
       input [31:0]   S_TDATA,
       input          S_TVALID,
       output         S_TREADY,
       output  [31:0] M_TDATA,
       output         M_TVALID,
       input          M_TREADY
);

assign M_TDATA = ~S_TDATA;
assign M_TVALID = S_TVALID;
assign S_TREADY = M_TREADY;

endmodule
```




MASTER
Custom Hardware
SLAVE
S_TDATA
S_TVALID
S_TREADY
M_TDATA
M_TVALID
M_TREADY
buf
1 cycle

~~Pop~~ potatoes

Let’s build a buffer state machine.

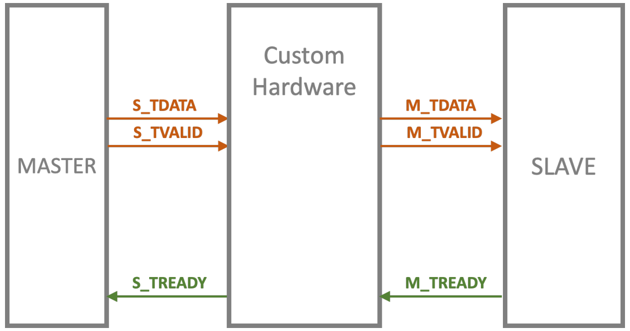


S_TVALID==0/
S_TREADY=0
M_TVALID=0

M_TREADY=1/
M_TVALID=1
S_TREADY=0

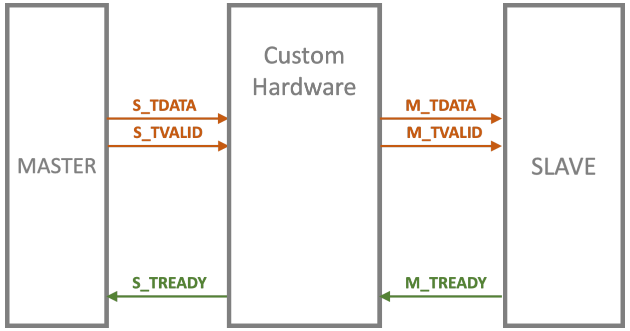
~~plz throw~~
Catch potato
S_TVALID==1/
S_TREADY=1
M_TVALID=0

M_TREADY==0/
M_TVALID=1
S_TREADY=0

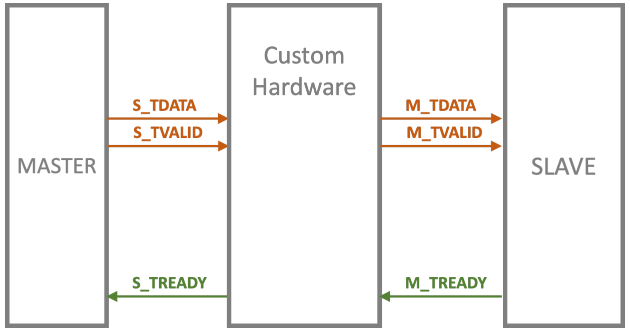


Let’s build a buffer state machine.

```
module custom_hw_buf (
        input           ACLK,
        input           ARESET,
        input [31:0]    S_TDATA,
        input           S_TVALID,
        output          S_TREADY,
        output [31:0]   M_TDATA,
        output          M_TVALID,
        input           M_TREADY
);






endmodule
```



Let’s build a buffer state machine.



```
module custom_hw_buf (
       input             ACLK,
       input             ARESETN,
       input [31:0]      S_TDATA,
       input             S_TVALID,
       output            S_TREADY,
       output [31:0]     M_TDATA,
       output            M_TVALID,
       input             M_TREADY
);

enum {S0, S1} state, nextState;
reg [31:0] nextVal;

always_ff @(posedge ACLK) begin
    if (ARESETN=0 begin
       state <= S0;
       M_TDATA <= 32’h0
    end else begin
       state <= nextState;
       M_TDATA <= nextVal;
    end
end
```

```
always_comb begin
    S_TREADY = ‘h1;
    M_TVALID = ‘h0;
    nextState = state;
    nextVal = M_TDATA;
    case(state)
        S0:  begin
            if (S_TVALID) begin
                nextState = S1;
                nextVal = S_TDATA;
            end
        end
        S1: begin
            S_TREADY = ‘h0;
            M_TVALID = ‘h1;
            if (M_TREADY) begin
                nextState = S0;
                M_TDATA = 32'h0;
        end
    endcase
end

endmodule
```

Next Time

- Memory-Mapped I/O
- Memory-Mapped Buses

References

- Zynq Book, Chapter 19 “AXI Interfacing”
- Practical Introduction to Hardware/Software Codesign
  - Chapter 10
- AMBA AXI Protocol v1.0
  - http://mazsola.iit.uni-miskolc.hu/~drdani/docs_arm/AMBAaxi.pdf
- https://lauri.võsandi.com/hdl/zynq/axi-stream.html

04: Bus Interfaces

Engr 315: Hardware / Software Codesign

Andrew Lukefahr

Indiana University