Gisselquist Technology, LLC

A Demonstration of Formal Methods

Daniel E. Gisselquist, Ph.D.
October, 2019

My Background

My Background

Retired from USAF in 2013

- Background was in Signal Processing
- Set out on my own
  Ye are bought with a price; be not ye the servants of men. (1 Cor 7:23)
- Kept my integrity
  Buy the truth, and sell it not; also wisdom, and instruction, and understanding (Prov 23:23)
  A just weight and balance are the LORD's: all the weights of the bag are his work (Prov 16:11)

Digital Design

I wanted to start out new, in a new field

- Decided to pick up digital design
  - Used open source to advertise my work
  - Build and maintain an on-line portfolio
  - Started blogging
- Still trying to pick up work, evaluated formal methods
  - Goal was to blog about how wonderful my designs were
  - Found bugs in the most simple of my designs
  - Started methodically going from design to design

Digital Design

I wanted to start out new, in a new field

- Decided to pick up digital design
  - Used open source to advertise my work
  - Build and maintain an on-line portfolio
  - Started blogging
- Still trying to pick up work, evaluated formal methods
  - Goal was to blog about how wonderful my designs were
  - Found bugs in the most simple of my designs
  - Started methodically going from design to design
    - I found bugs in every one of my designs

Broken Portfolio

How bad were the bugs?

- Dropped data
- Memory controller that returned the wrong data
  - This had already passed *a full week* of HITL testing
- An I-Cache returning the wrong instruction
- Compression algorithm using an illegal codeword
- A CPU operating on the wrong data

Chance to learn

As owner of my own business, I was in a unique situation

- I could discuss my own bugs

This is normally taboo in this industry

- Corporations own all the code, all the change requests
- Even as a sub-contractor, my work gets owned by whoever I contract to
  - Only just got permission to discuss my Ri5cy work
- Any bugs I find are kept between me and whoever hires me

Bugs are only rumored, and hushed up

- Example: Xilinx has been deleting forum comments I've made about their broken designs

Formal Advocate

This experience naturally made me an advocate for formal methods

- Because I had formally verified my own code . . .
  - I had a lot of relatively rare experience
  - Blogging brought attention to my work
- Became a formal advocate for SymbioticEDA
  - Created and taught formal verification course for them
    - ▷ My first contract in my new business
  - Verified the Ri5cy RISC-V processor
  - Built and verified an FPGA accelerator
  - Lots of more blog articles

Signals Engineer

I'm still a Signals Engineer at heart

- Also doing SONAR contracts for US Navy
  - My customer has been amazed at how rock solid my designs have been

Formal methods are an integral part of anything I do

Formal Verification

Formal Verification

Formal Verification is a means of exhaustively searching for bugs

- Search over *all* possible design inputs

SymbiYosys supports three basic modes

1. Bounded Model Checks: Search the first N timesteps
   - Success is declared when the first fault is found
   - **Black-Box**, no internal design knowledge required
2. Induction Checks: Search N timesteps, starting anywhere
   - Success is declared when no fault can be found
   - **White-Box** testing, requires detailed internal knowledge
3. Cover
   - Success is finding a trace to match each goal
   - The tool will find the first trace that matches each

Basic Counter

Let's start by examining a basic counter

```
parameter [15:0] START_COUNT = 16'h3ff;
reg       [15:0]  counter;

initial counter = START_COUNT;
always @(posedge i_clk)
        counter <= counter - 1;

assert property (@(posedge i_clk)
        counter <= START_COUNT);
```

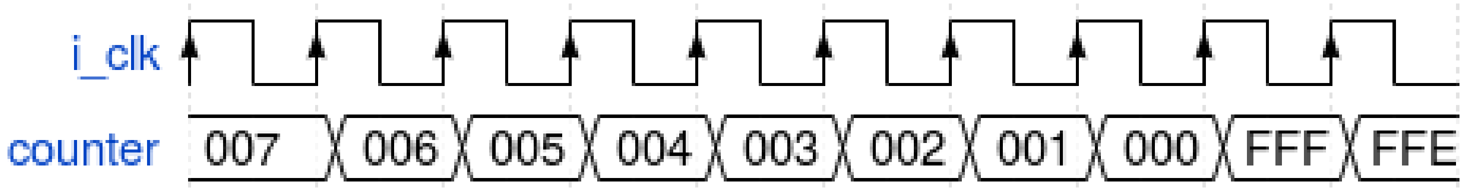
Basic Counter

We'll need a SymbiYosys configuration file

```
[tasks]
prf

[options]
prf: mode prove

[engines]
smtbmc

[script]
read -formal counter.v
prep -top counter

[files]
counter.v
```

Counter Example

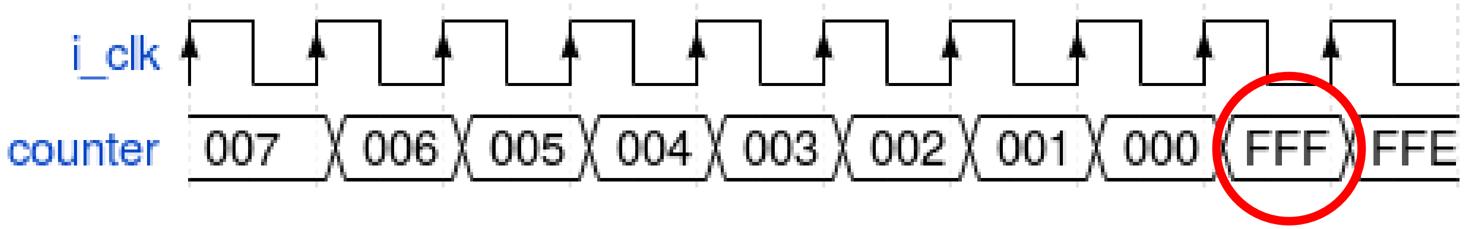
The formal tool returns this trace



- ☐
- ☐
- ☐

Counter Example

The formal tool returns this trace

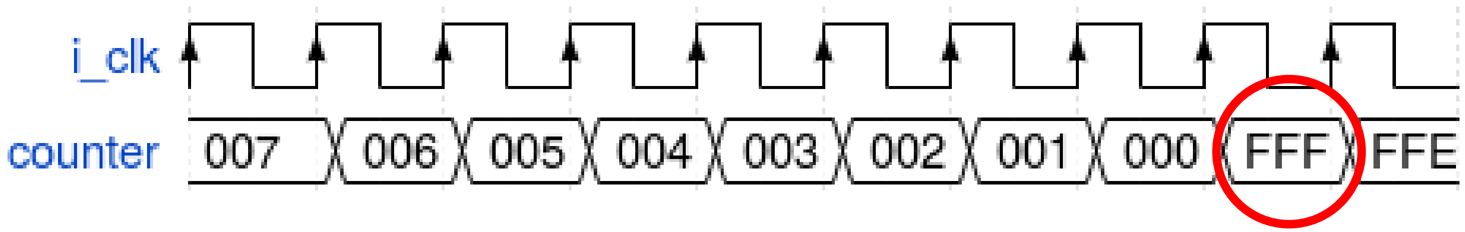


We forgot to check for wrap at zero

- Formal solver returns the line number of the failing assertion
-
-

Counter Example

The formal tool returns this trace



We forgot to check for wrap at zero

- Formal solver returns the line number of the failing assertion
- Note that the error is just prior to the last positive edge
- There's no question of what caused the bug

Counter Reset

Let's try adding in a reset

```
always @(posedge i_clk)
if (i_reset)
        counter <= START_COUNT;
else if (counter > 0)
        counter <= counter - 1;
```

On a reset, the counter should go back to `START_COUNT`

```
assert property (@(posedge i_clk)
        i_reset |=> (counter == START_COUNT));
```

Counter Reset

Let's try adding in a reset

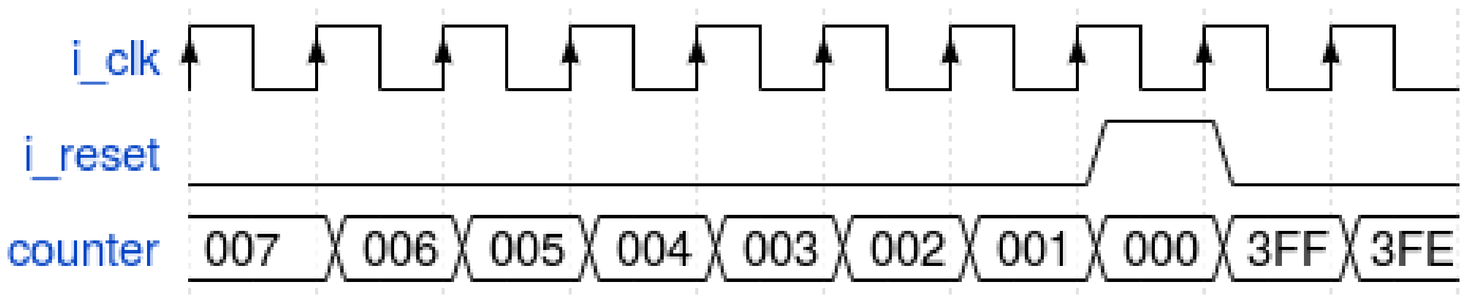
```
always @(posedge i_clk)
if (i_reset)
        counter <= START_COUNT;
else if (counter > 0)
        counter <= counter - 1;
```

Once the counter hits zero, it should stay there

```
assert property (@(posedge i_clk)
        (counter == 0)
        |=> (counter == 0));
```

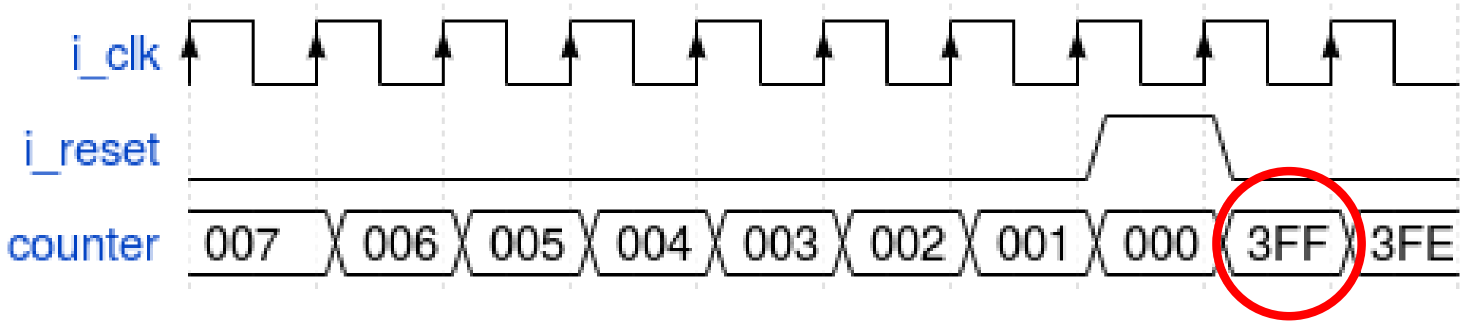
Counter Example

The formal tool returns this trace



Counter Example

The formal tool returns this trace



We overconstrained the design

- The design might not stay at zero if it is reset
- In this case, the design "works", the properties have a bug

Counter Reset

What did we do wrong?

```
always @(posedge i_clk)
if (i_reset)
        counter <= START_COUNT;
else if (counter > 0)
        counter <= counter - 1;
```

We forgot to make an exception for `i_reset`

```
assert property (@(posedge i_clk)
        disable iff (i_reset)
        (counter == 0)
        |=> (counter == 0));
```

Counter Reset

What did we do wrong?

```
always @(posedge i_clk)
if (i_reset)
        counter <= START_COUNT;
else if (counter > 0)
        counter <= counter - 1;
```

We forgot to make an exception for `i_reset`

```
assert property (@(posedge i_clk)
        disable iff (i_reset)
        (counter == 0)
        |=> (counter == 0));
```

This is neat and all, but . . .

- How practical is all this?

Request Counting

Consider a bus protocol

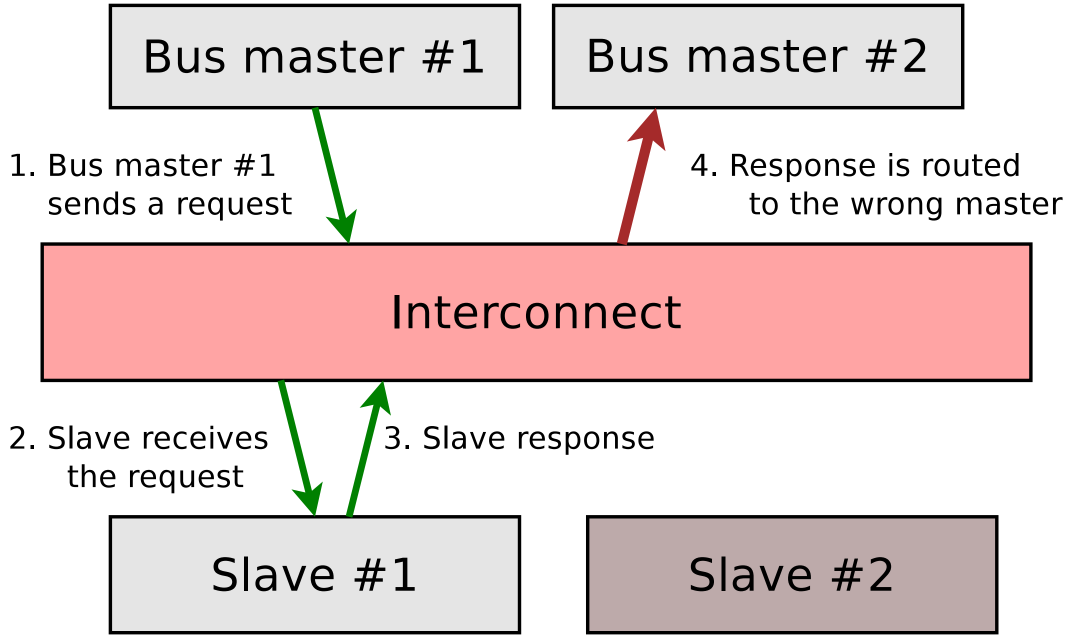
- Without a request, you can't have any acknowledgments

```
// Count requests
always @(posedge i_clk)
if (i_reset)
        outstanding <= 0;
else case({ request && !stall, ack })
2'b10: outstanding <= outstanding + 1;
2'b01: outstanding <= outstanding - 1;
default: begin end
endcase
// Check acknowledgments
always @(*)
if (outstanding == 0)
        assert(!ack);
```

Interconnect

From a "correct-by-construction" design

- Built with a higher level design language

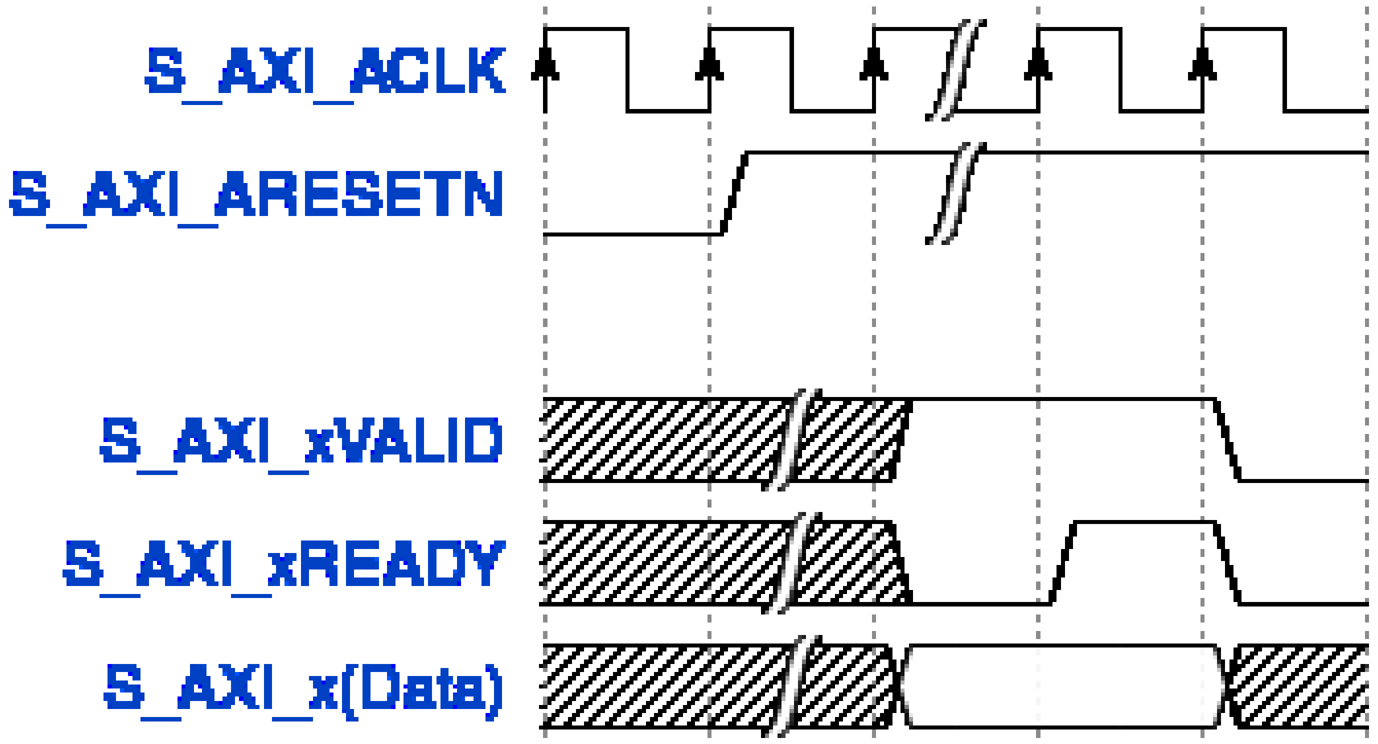


Bus master #1
Bus master #2
1. Bus master #1 sends a request
4. Response is routed to the wrong master
Interconnect
2. Slave receives the request
3. Slave response
Slave #1
Slave #2

All found within 20 timesteps

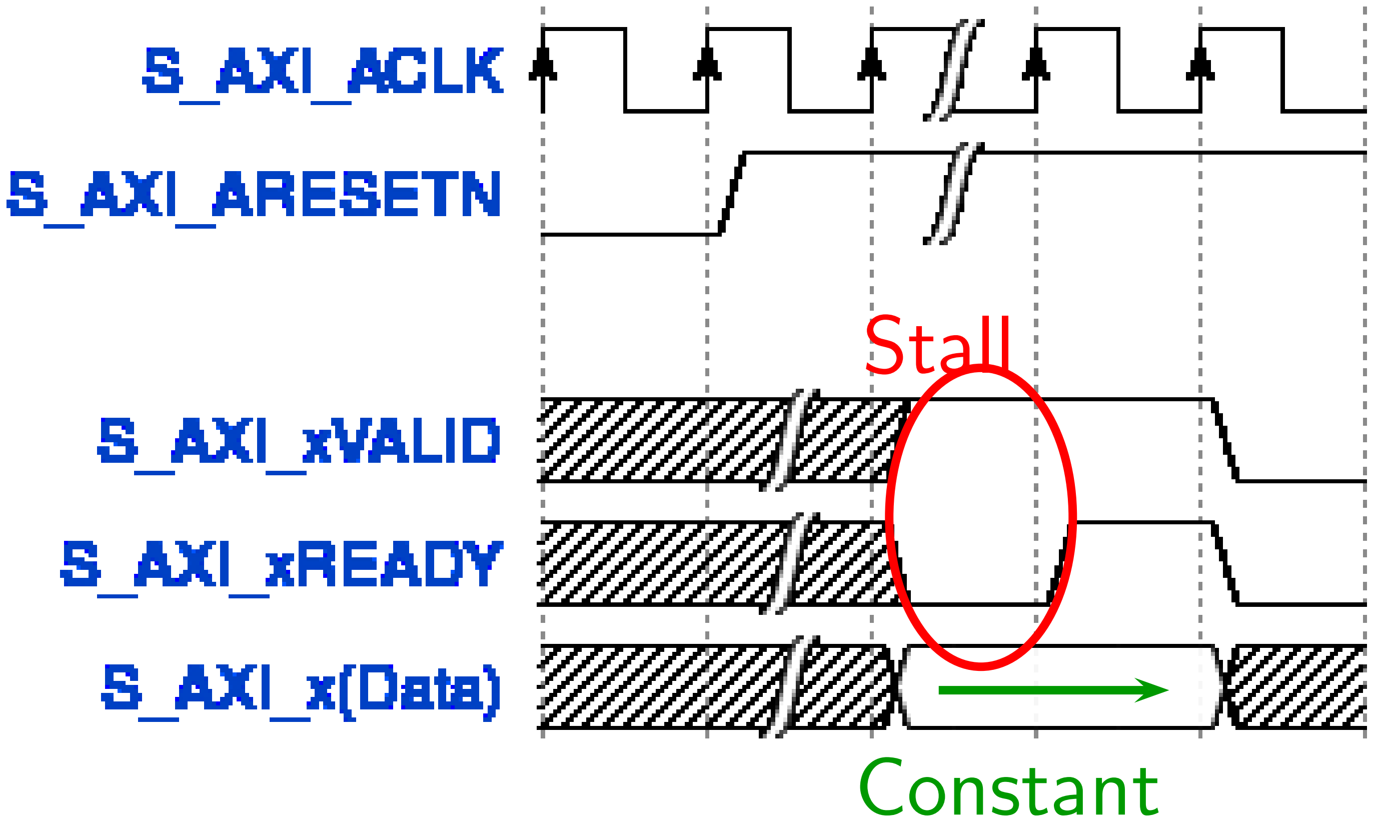
Stalls

If a channel is stalled, nothing should change



Stalls

If a channel is stalled, nothing should change



Stalls

If a channel is stalled, nothing should change

```
assert  property (@(posedge i_clk)
        request && stall
        |=> request && $stable(data));
```

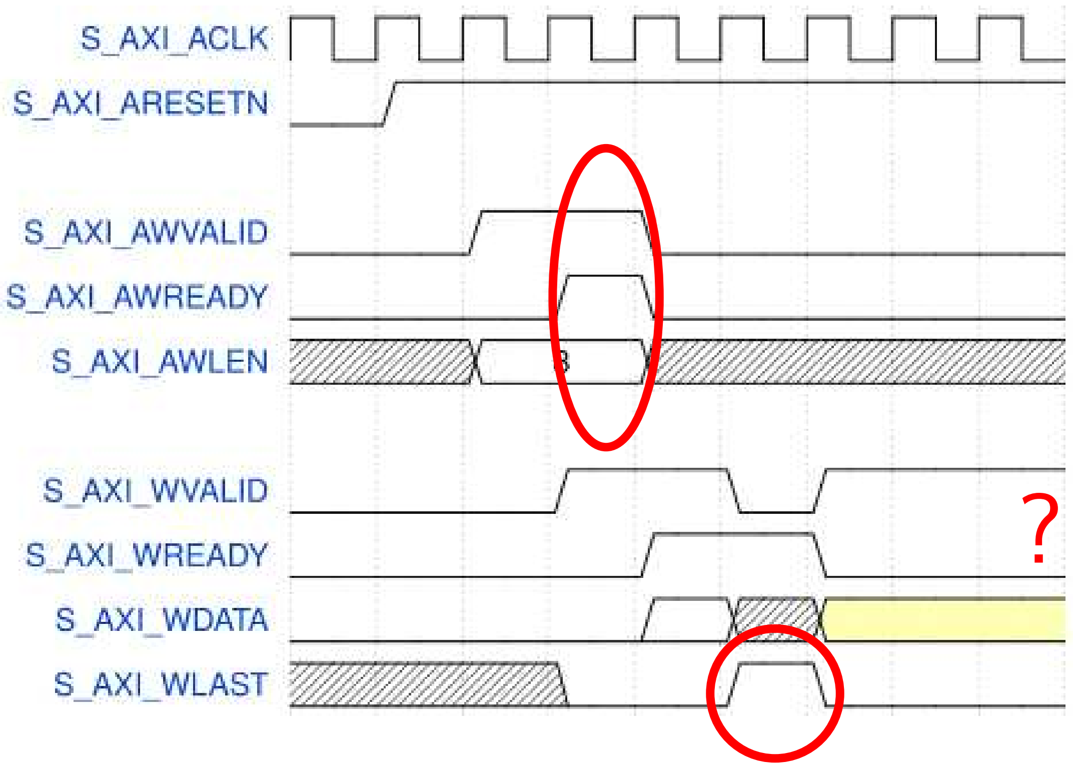
Stall Counting

To ensure all responses get accepted, let's force a maximum number of stalls

```
always @(posedge i_clk)
if (i_reset || !request || !stall)
        stall_count <= 0;
else // if (request && stall)
        stall_count <= stall_count + 1;

always @(*)
        assert(stall_count < MAX_STALL_COUNT);
```

Bugs Found

Vivado's 2019.1 AXI (full) demo



Xilinx's core stops accepting data if ever `WLAST`

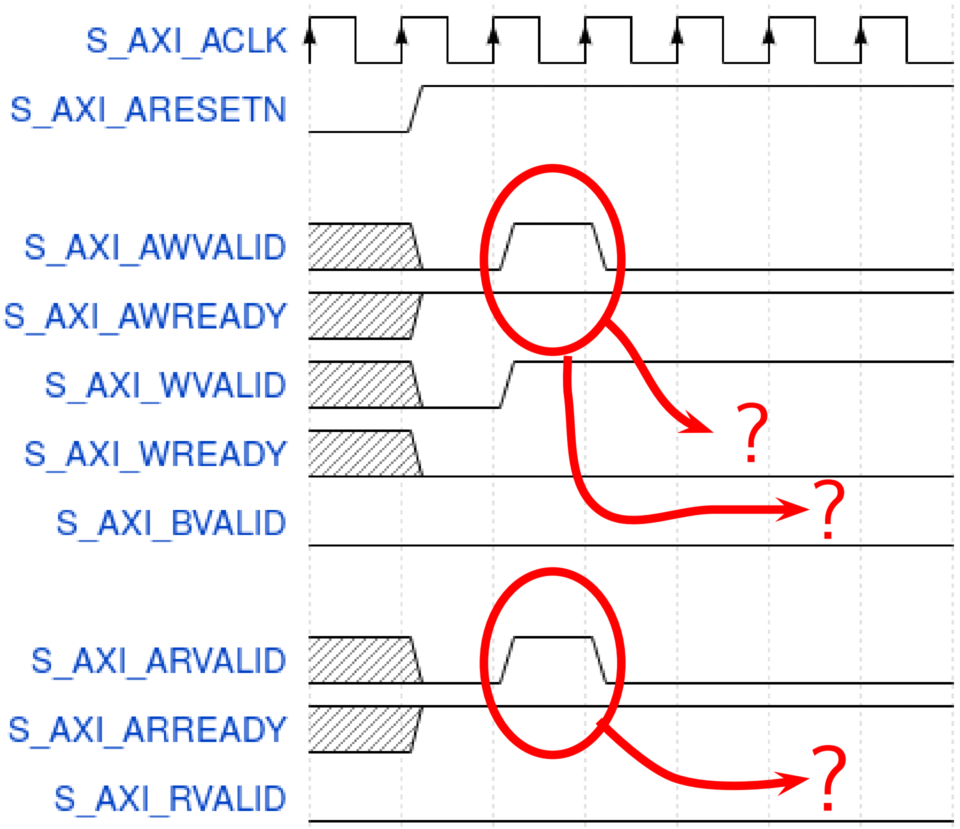
Responses

Rule: All requests must get responses

```
always @(posedge i_clk)
if (i_reset || outstanding == 0 || ack)
        ack_delay <= 0;
else // if (outstanding > 0)
        ack_delay <= ack_delay + 1;

always @(*)
        assert(ack_delay < MAX_ACK_DELAY);
```

TinyTPU

From the TinyTPU:

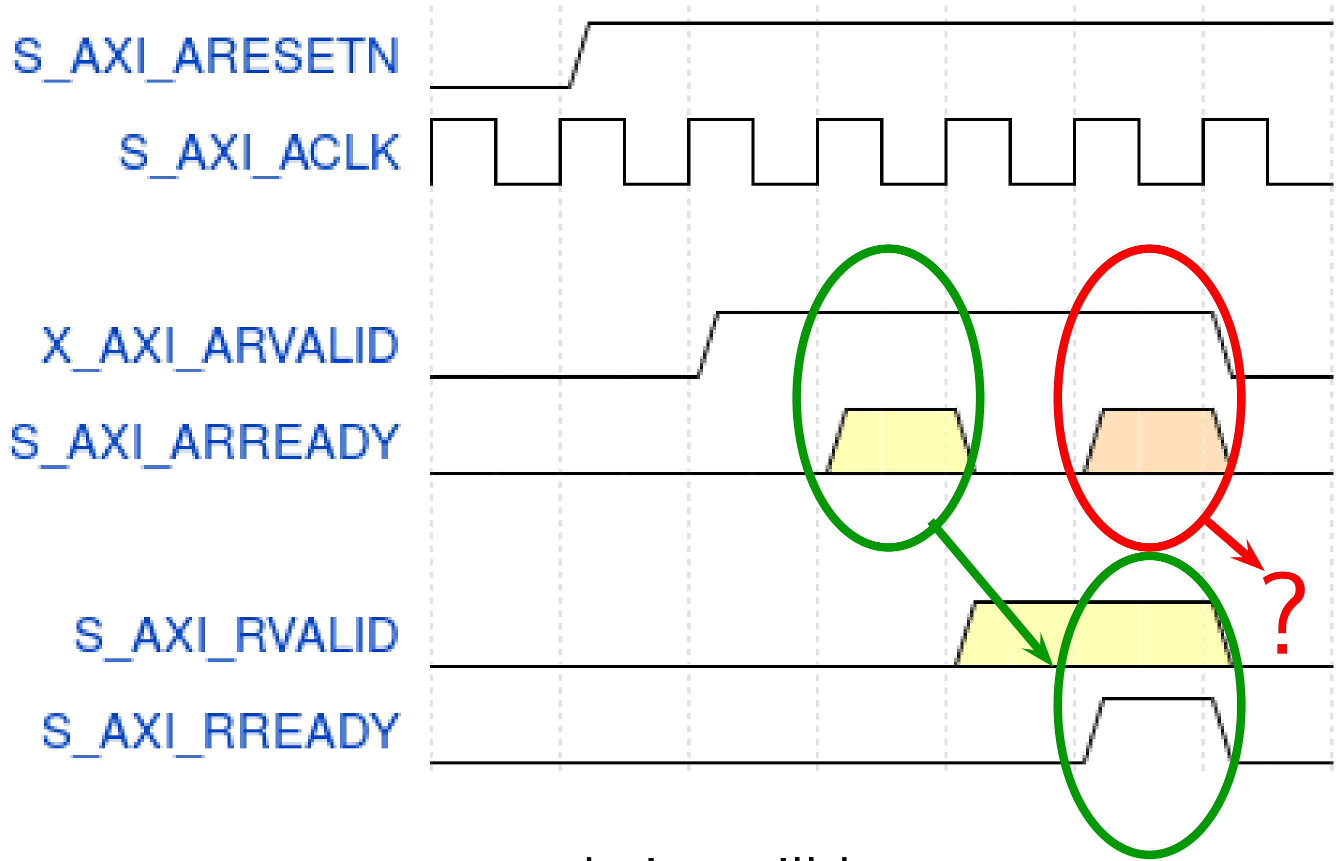


If ever `AWVALID` and `ARVALID` are both true at once . . .

- The state machine remains in idle

Xilinx's Demo

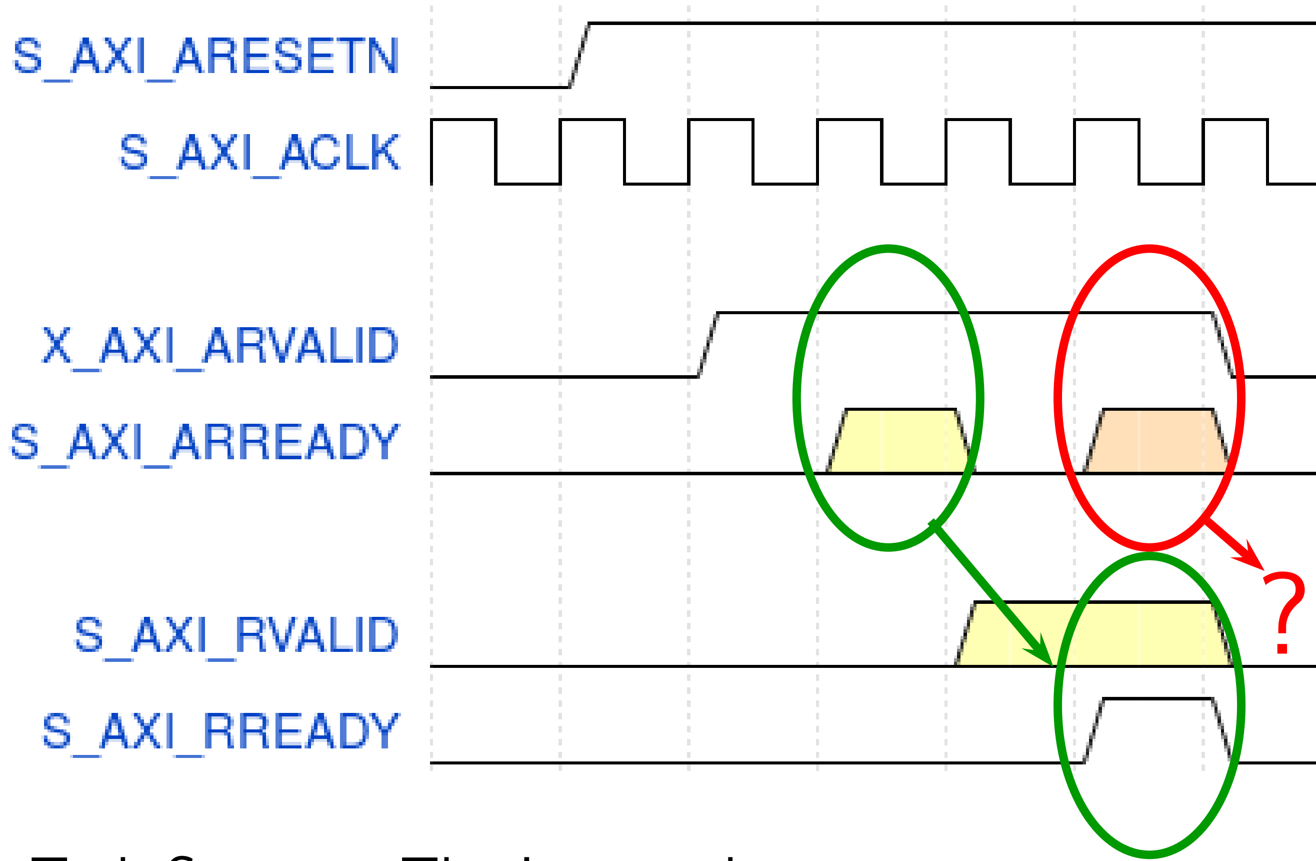
Vivado 2016.1, AXI-lite READ



Two requests, one response: design will hang

Xilinx's Demo

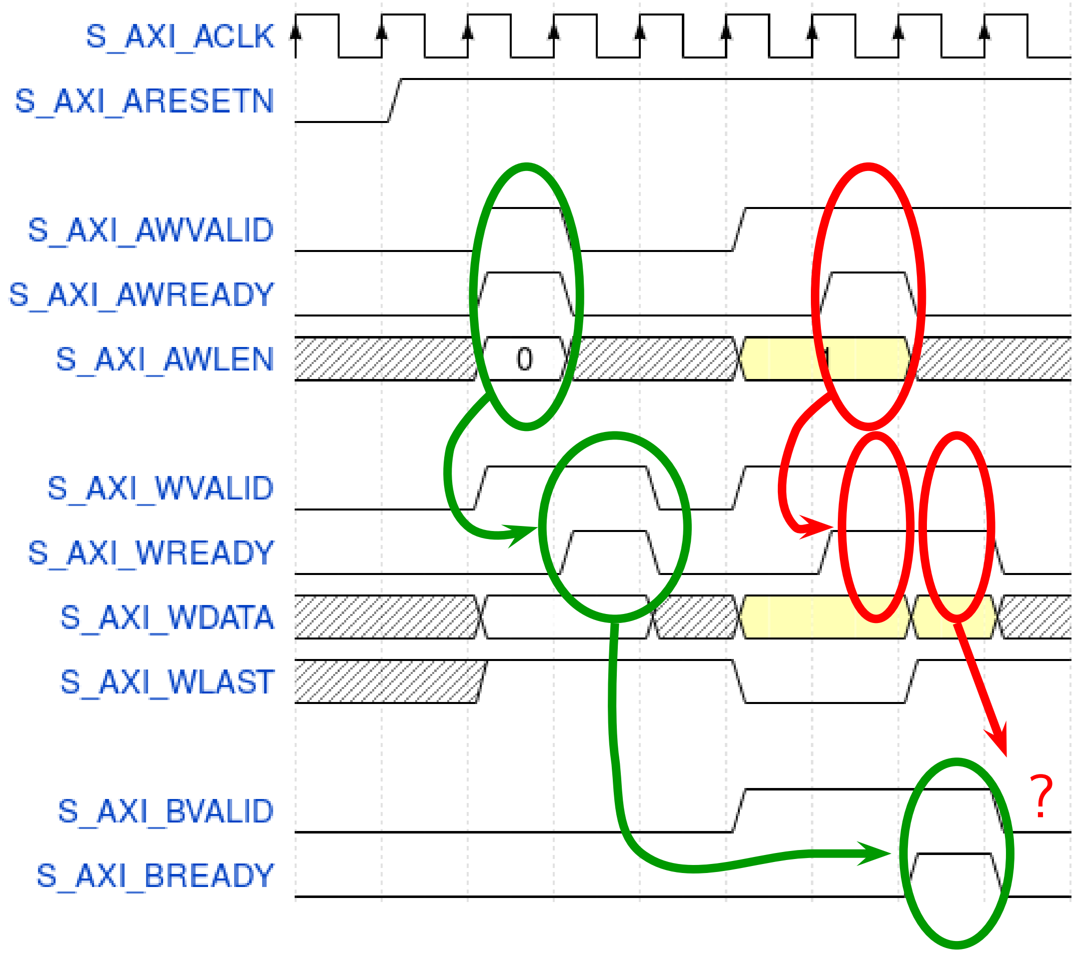
Vivado 2016.1, AXI-lite READ



Xilinx Tech Support: That's not a bug

Intel's Demo

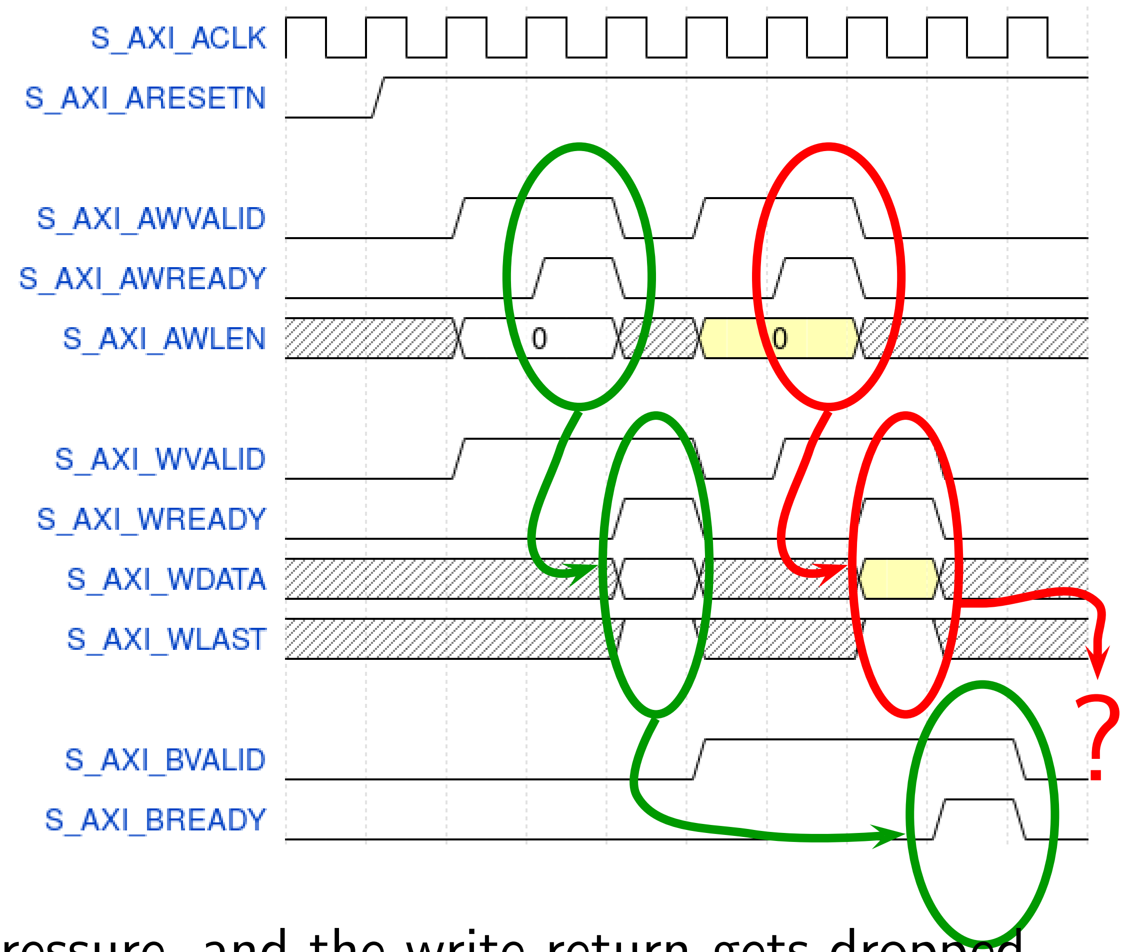
Intel's AXI3 demo



Any backpressure, and the write return gets dropped

Xilinx Bugs

Vivado's 2019.1 AXI (full) demo



Any backpressure, and the write return gets dropped

Cover

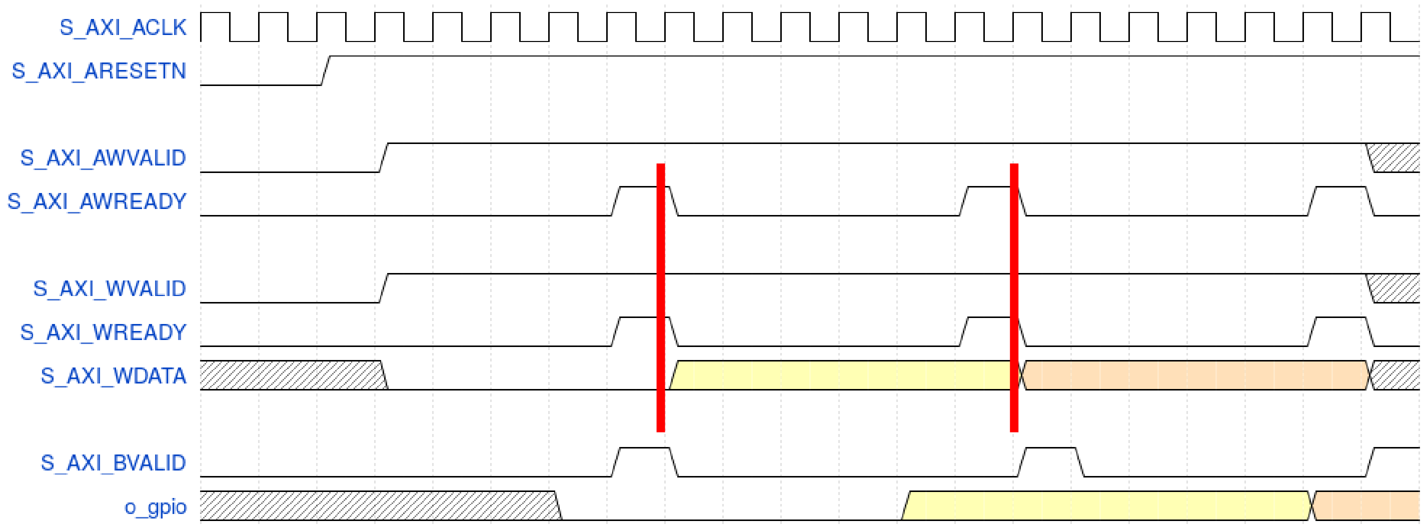
cover() will return the first trace that fulfills the request

```
// Count acknowledgments
initial ack_count = 0;
always @(posedge i_clk)
if (i_reset)
        ack_count <= 0;
else if (ack)
        ack_count <= ack_count + 1;

// Find the first way to return the third
// acknowledgment
always @(*)
        cover(ack_count == 3);
```

Xilinx's AXI GPIO

cover() can be used to determine a core's throughput

- `cover(third_write_return);`

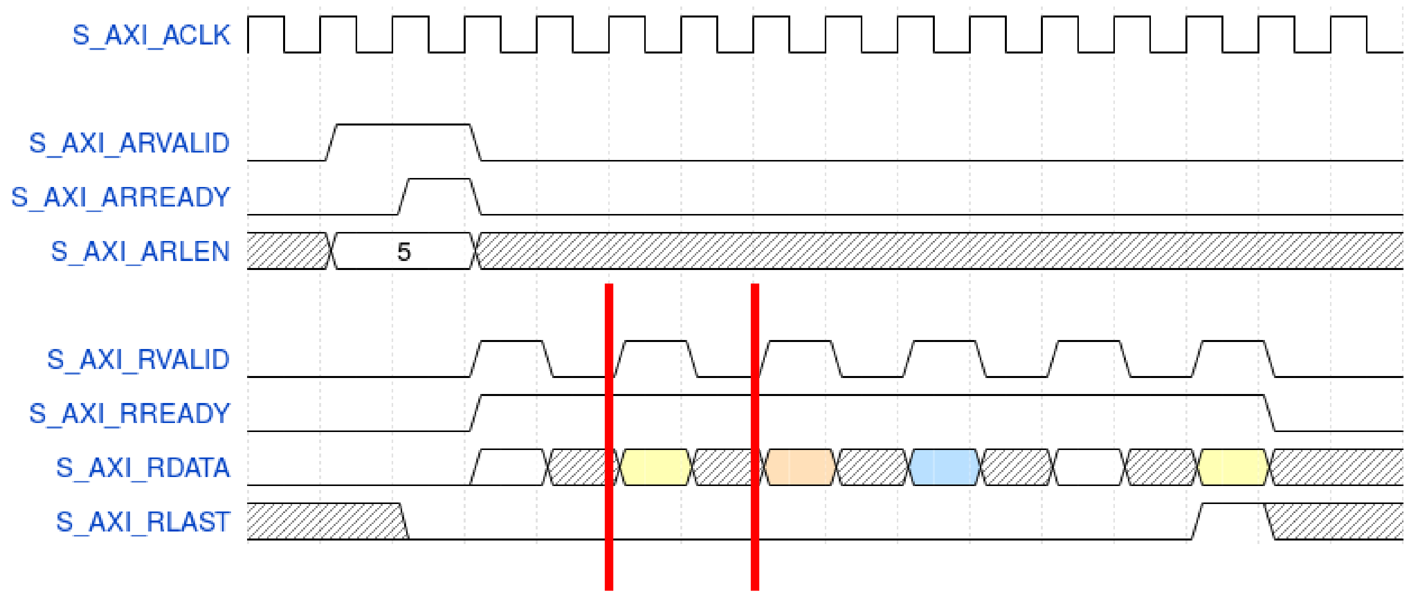


Max write throughput: One transaction every six clocks

- This would contribute to why so many folk complain of blinky being slow when using either ARM or MicroBlaze

Xilinx's AXI demo

Vivado's 2019.1 AXI (full) demo burst read performance

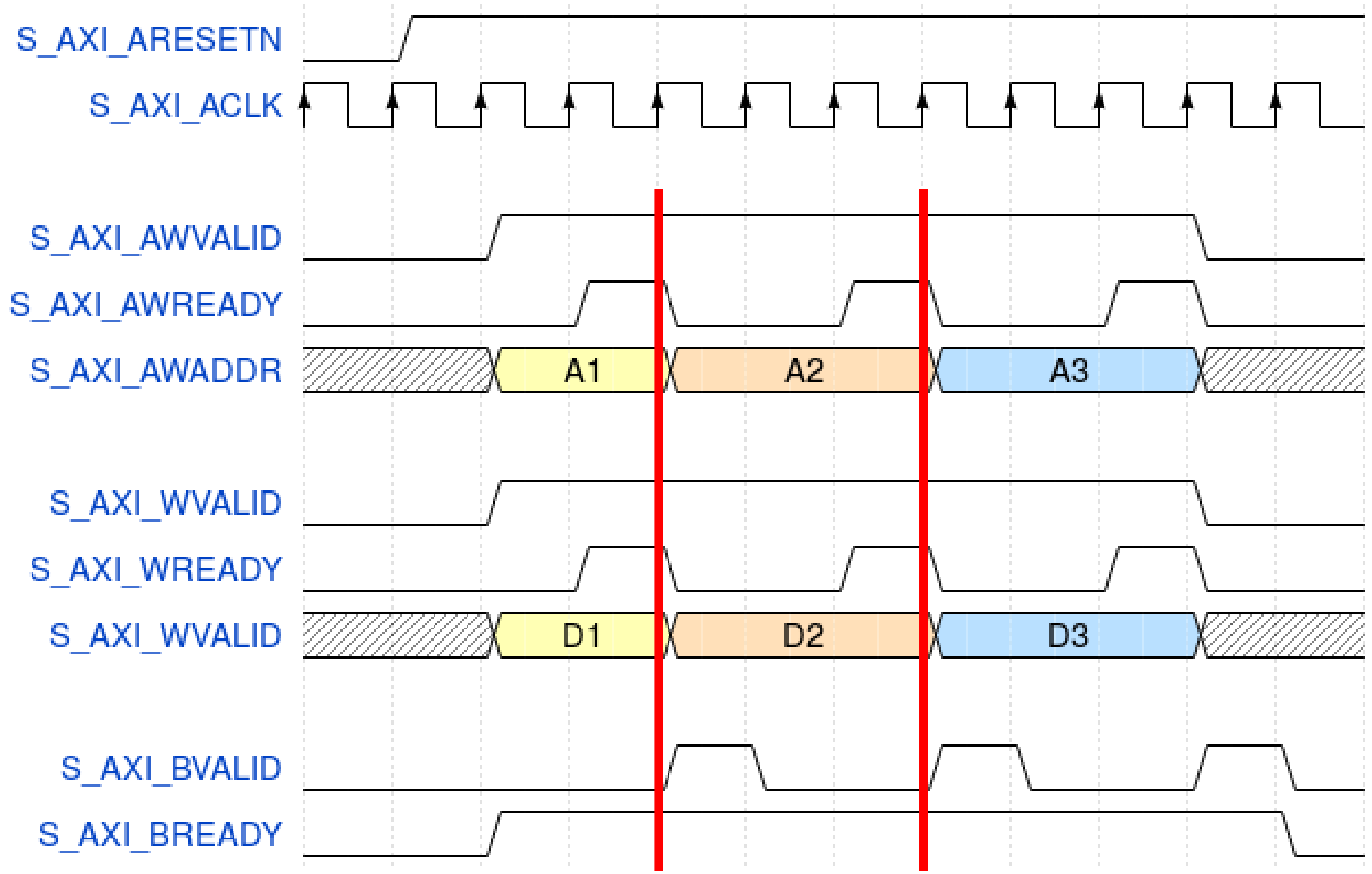


50% throughput??

- Build your design from this demo, and . . .
- You'll cripple your FPGAs data processing throughput

Xilinx's AXI demo

Vivado's 2019.1 AXI-lite demo write performance



33% throughput??

Get Serious

Both Xilinx and Intel offer these broken designs to new users

- Anyone who uses them accepts a ticking time-bug into their designs
  - Xilinx's demos have been broken since at least 2016
  - Unsolved support requests date back years
  - I've found these bugs in many support requests
    - *Even if the customer used Xilinx's AXI VIP!*
- Not safe for military (or any other) use

Not even software changes are safe

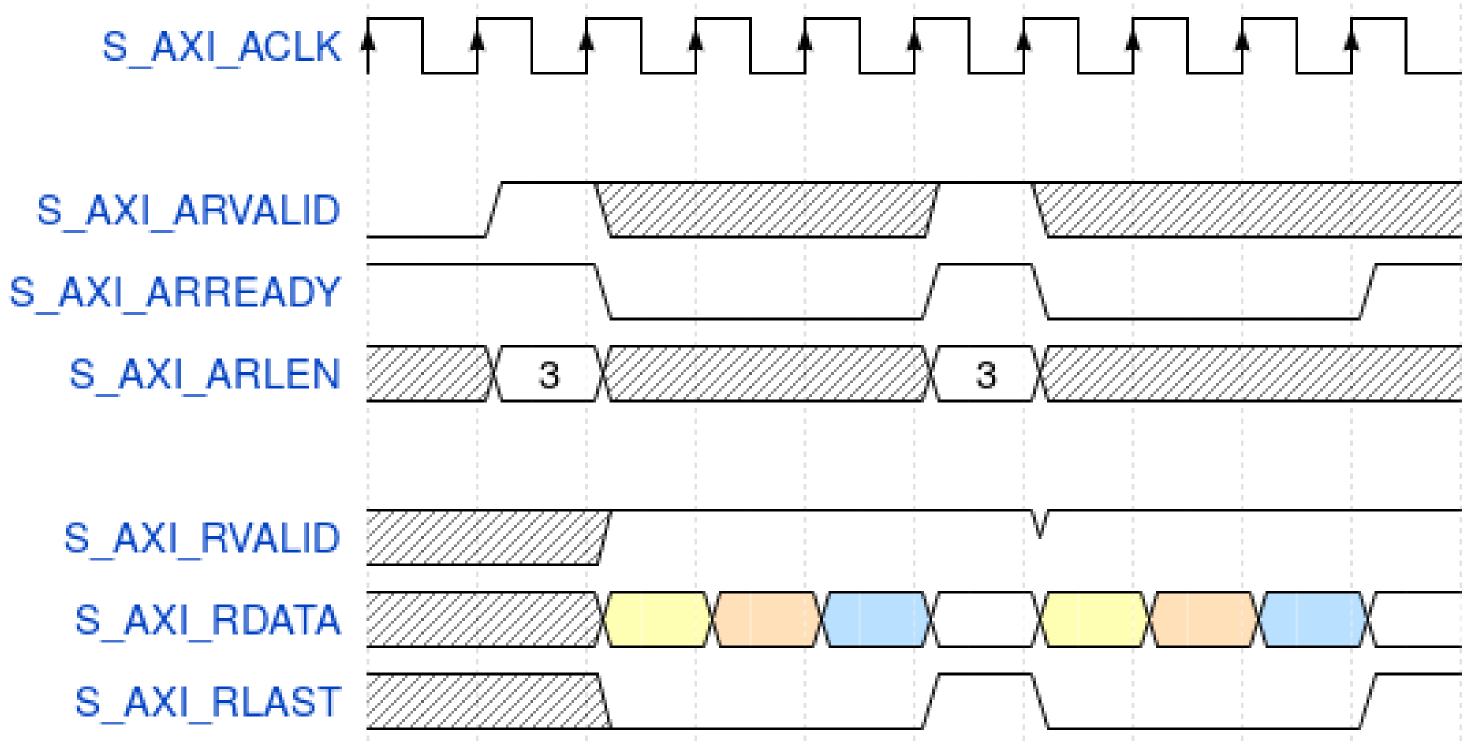
Formal in Design

What if you started your design with formal?

```
module(/* ... */);
        //
        // Design logic to be written
        //

`ifdef  FORMAL
        // Start with a formal bus property
        // set drawn from a library file
        faxi_slave
        faxi(i_clk, i_reset, // ...
                );
`endif
endmodule
```

AXI demo

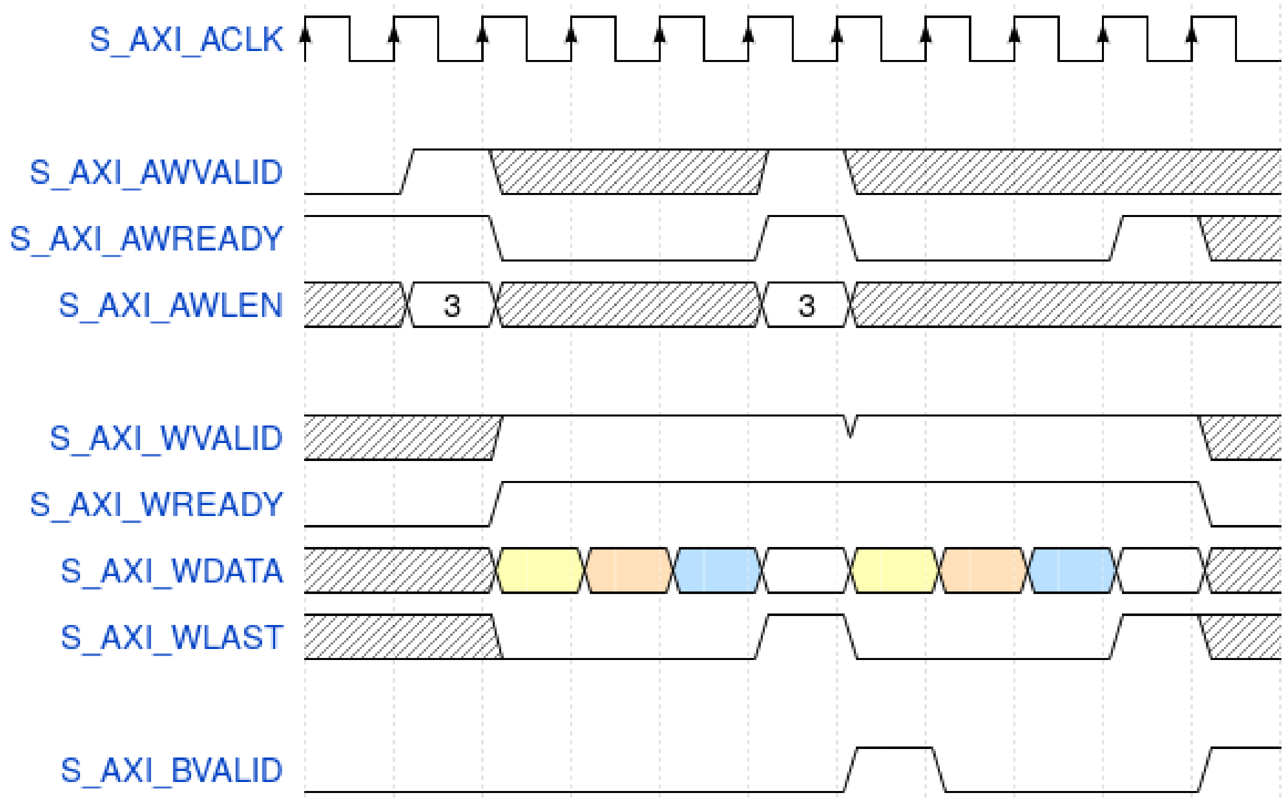
Better read performance



Link: Open Source AXI (full) demonstration design

Better AXI demo

Better write performance



Link: Open Source AXI (full) demonstration design

Personal Examples

My own designs may be found on Github

- AXI-lite demonstration design
- (Full) AXI demonstration design
- AXI-lite to AXI bridge
  - Unlike Xilinx's design, costs no logic, and no clocks
- AXI to AXI-lite bridge
  - This is really a high performance, high throughput design
  - Keeps up with the pace of AXI, no lost beats
- Crossbars: AXI, AXI-lite (Wishbone pipeline)
- AXI Stream to Memory Mapped (S2MM)
- AXI Memory Mapped to Stream (MM2S)

Yes, these are all Open Source

Personal Examples

My own designs may be found on Github

- AXI-lite demonstration design
- (Full) AXI demonstration design
- AXI-lite to AXI bridge
  - Unlike Xilinx's design, costs no logic, and no clocks
- AXI to AXI-lite bridge
  - This is really a high performance, high throughput design
  - Keeps up with the pace of AXI, no lost beats
- Crossbars: AXI, AXI-lite (Wishbone pipeline)
- AXI Stream to Memory Mapped (S2MM)
- AXI Memory Mapped to Stream (MM2S)

Yes, these are all Open Source

- *The full AXI DMA remains dreamware*

Formal & DSP

Can formal methods be used in DSP?

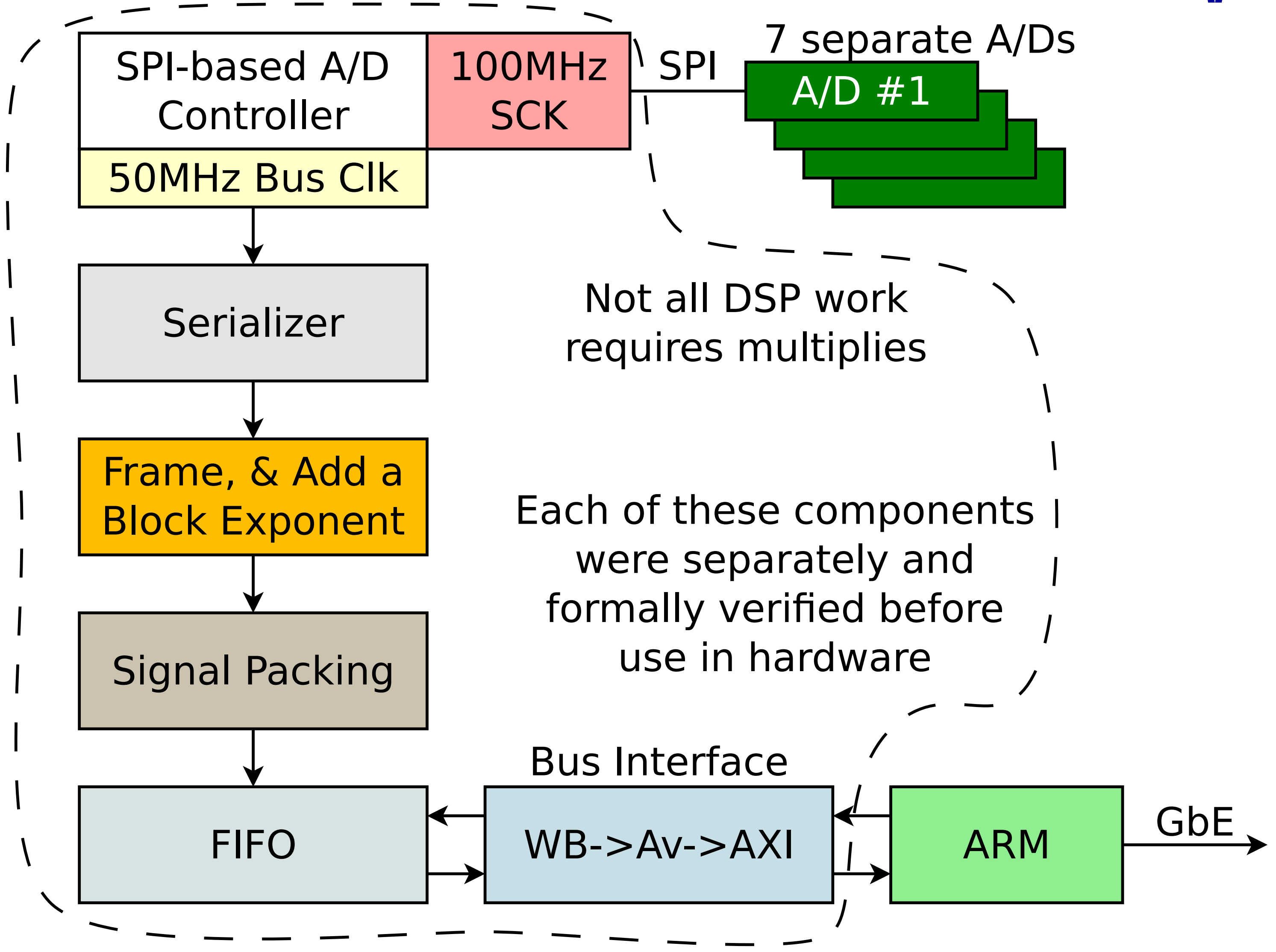
Problem: Formal tools have a problem with multiplies

- They're just too complex for most formal verification tools
- Other alternatives are required to verify DSP code in spite of the multiplies

Let's look at two examples

- SONAR receiver
- An FFT

SONAR Receiver



Not all DSP work requires multiplies

Each of these components were separately and formally verified before use in hardware

FFT Design

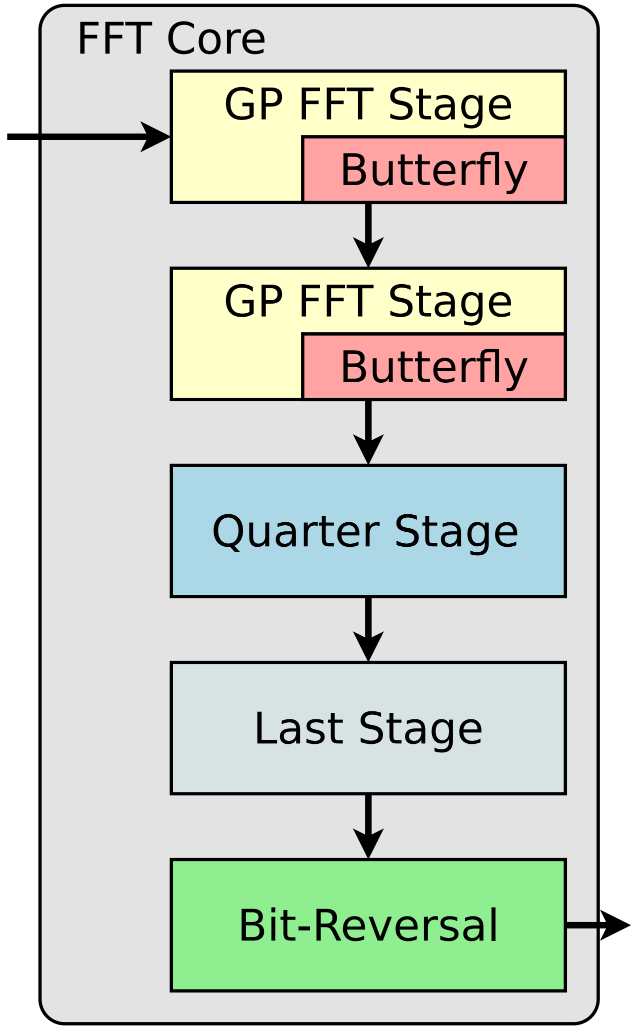
Rewrote my FFT for general purpose processing

- Original bench tests
  - Adjusted, and all passed
- Full test required checking impulses of different amplitudes, and sine and cosine waves of different frequencies
- 75 total tests

$$\begin{tikzcd}\boxed{\text{Calculate FFT}} \arrow[r] & \boxed{\text{Export to File}} \arrow[r] & \boxed{\text{Visual Inspection}}\end{tikzcd}$$

Finding a bug was like finding a needle in a hay stack

FFT Design

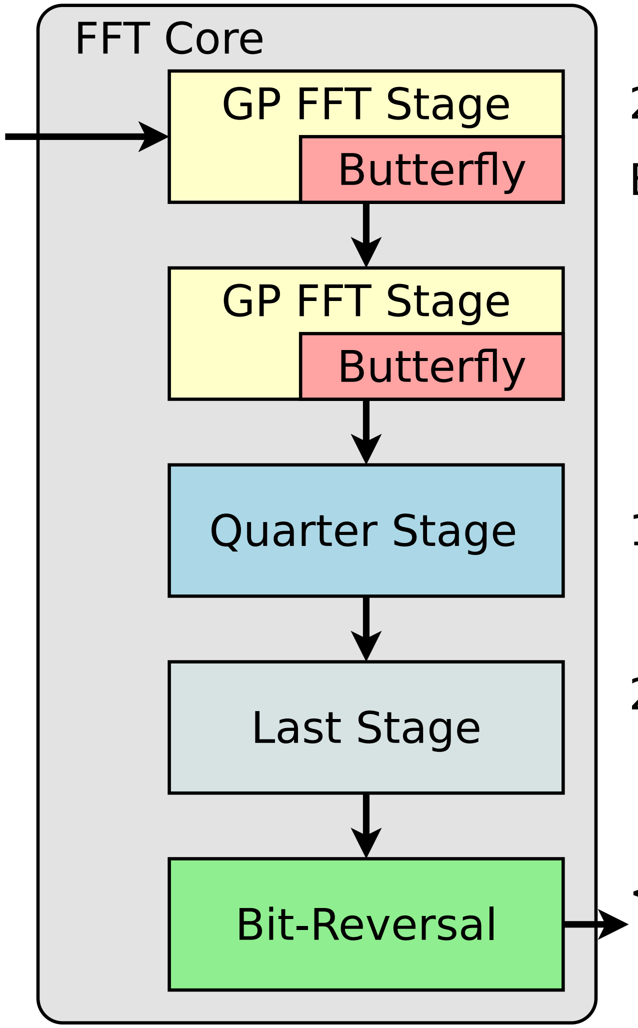


Take your pick:

- Search 5+GB in order to find a bug somewhere
- Use the formal tool to
  - Place the bug at the last timestep
  - Point you to the problem
  - Keep your traces less than 100kB

Formal was the fastest way to find bugs

FFT Design



2s for the data handling

Butterfly:
  Abstract MPY: 6 m
  Soft MPY: 2hrs

11s

2s

< 1s

Design time:

- ▫ Design
- ▫ Design test
- ▫ Run/sim test
- ▫ Examine failure
- ▫ Fix bug
- ▫ Repeat

"OODA loop" with formal is tighter

How Long?

How long does it take to verify a design?

- Most of the work is really in the setup (1hr +)
- Here are some tool times

AXI-lite		
Wishbone to AXI-lite bridge	2	sec
AXI-lite to Wishbone bridge	13	sec
Basic AXI-lite slave	40	sec
Basic AXI-lite crossbar	2.1	mins
AXI		
Basic AXI (full) slave	10	sec
Wishbone to AXI bridge	40	sec
Bus Fault Isolator	4	min
Basic 4x8 AXI crossbar	25	min

- Finding a bug is faster than proving there are none

The Problem

Why were so many cores broken?

- Most cores are tested via simulation and verification IP
- Simulation is dependent upon the creativity of the tester
  - Not checking what happens when xREADY low
  - Not checking what happens with back-to-back transactions
  - Not anticipating AWVALID & ARVALID at the same time
  - Only ever checking for one bus error type, never both
- The vendor verification materials are incomplete
- The vendor training examples have bugs!
- Xilinx has been deleting unflattering forum posts
  - This keeps users blind to the extent of the problem

Formal methods will check for bugs you aren't expecting

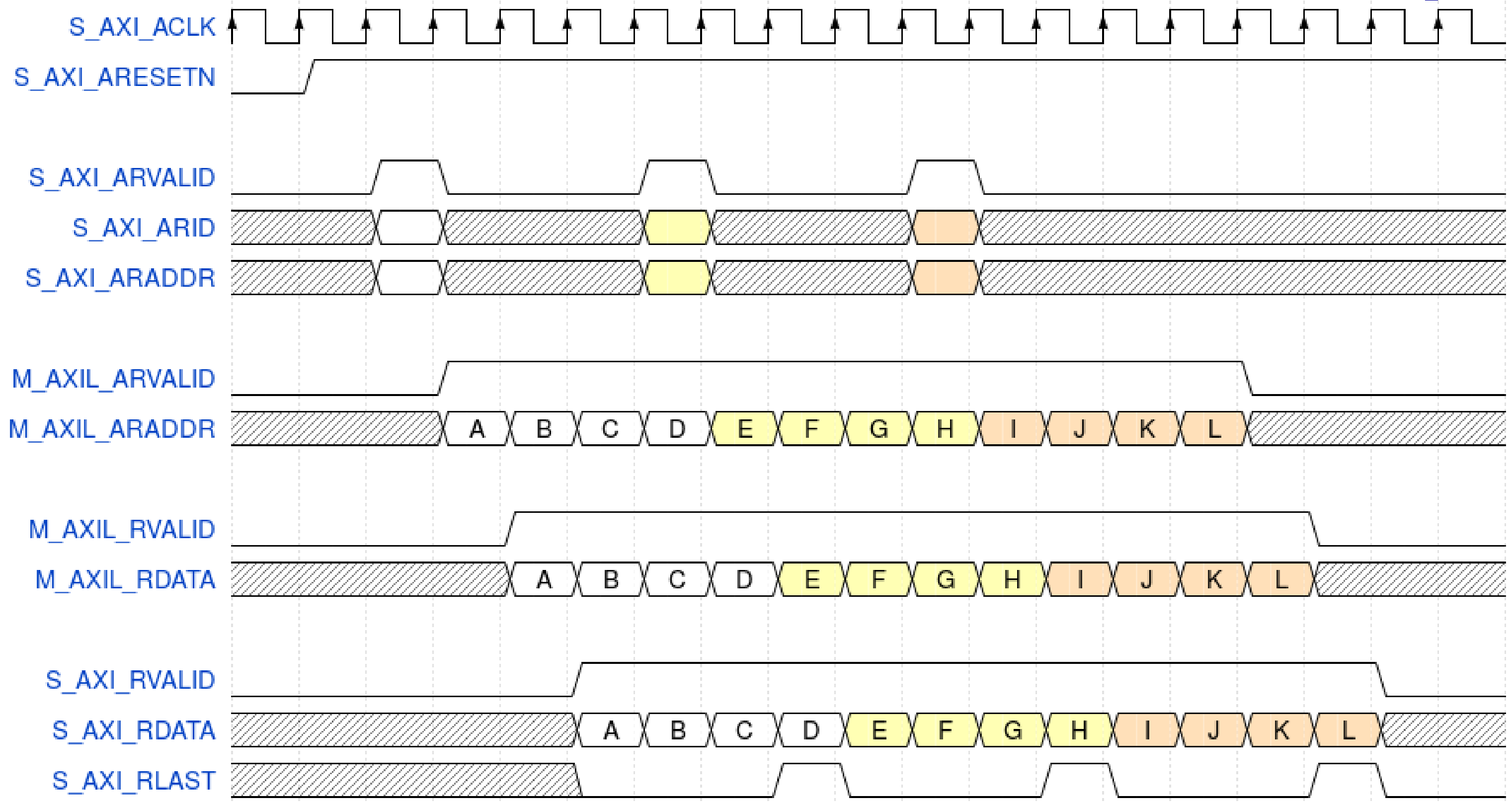
Conclusions

JE	(Junior Engr): My AXI-lite IP isn't working. Could Xilinx's code be broken?
Me:	Have you formally verified it?
JE:	(Ignores me, switches to Xilinx's forums) My IP isn't working. Your interconnect must be broken
Me:	Have you formally verified your design?
JE:	Here's the trace proving the problem
Me:	In your trace, one bus request is producing two responses.

Formal Verification would've caught this

Backups

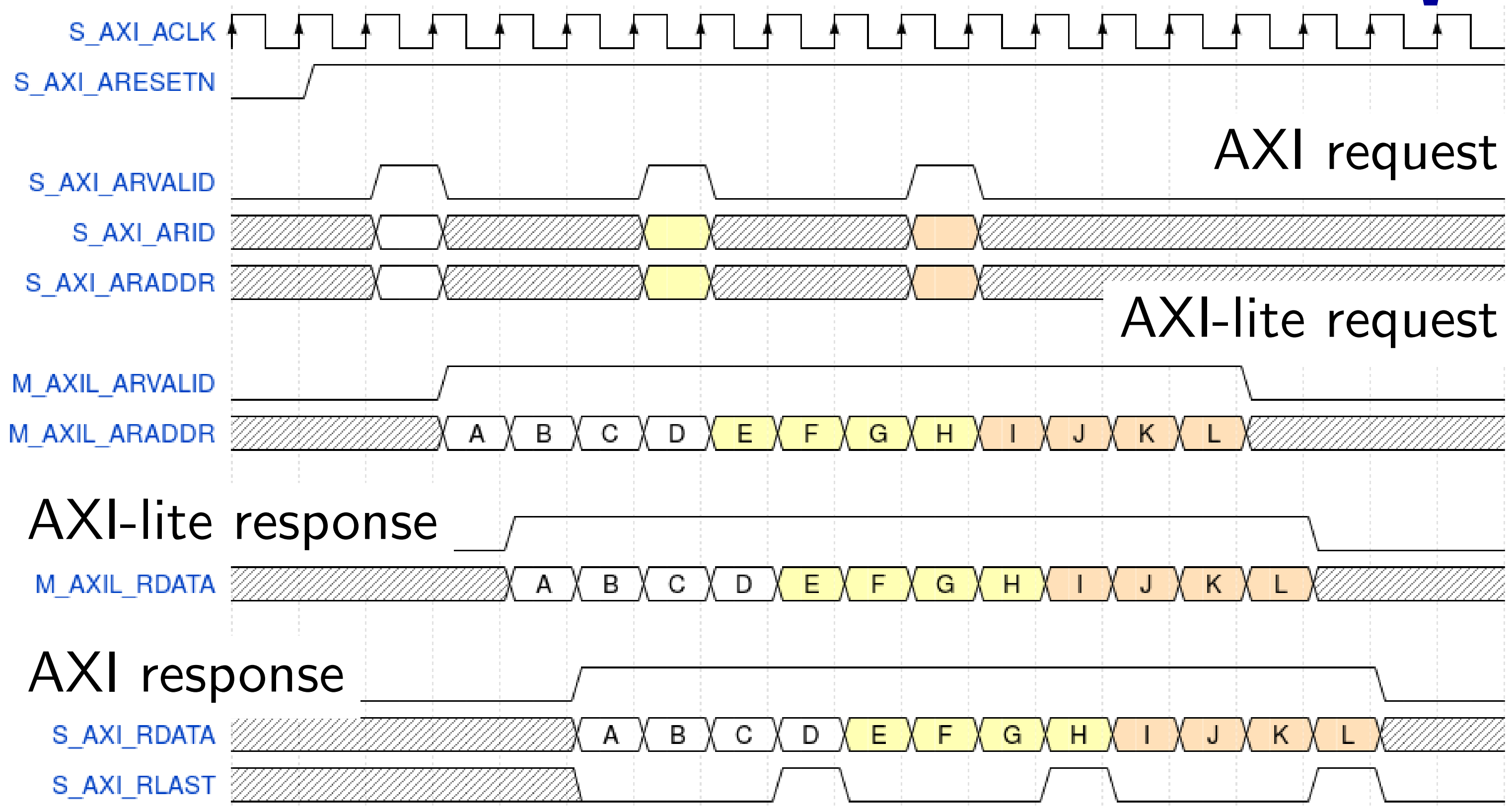
AXI-lite is powerful



That's how an AXI to AXI-lite bridge should work

- AXI protocol forces a minimum of one clock lost per bridge

AXI-lite is powerful

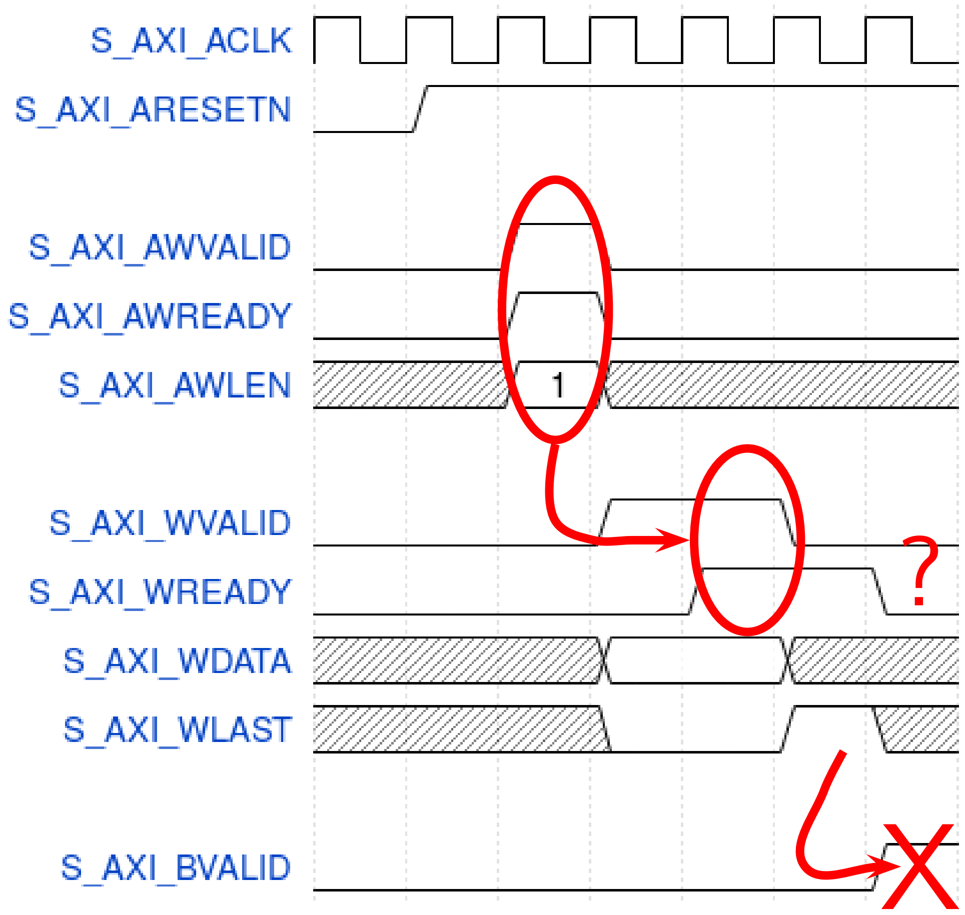


That's how an AXI to AXI-lite bridge should work

- Would be very difficult to do without formal

Intel Bugs

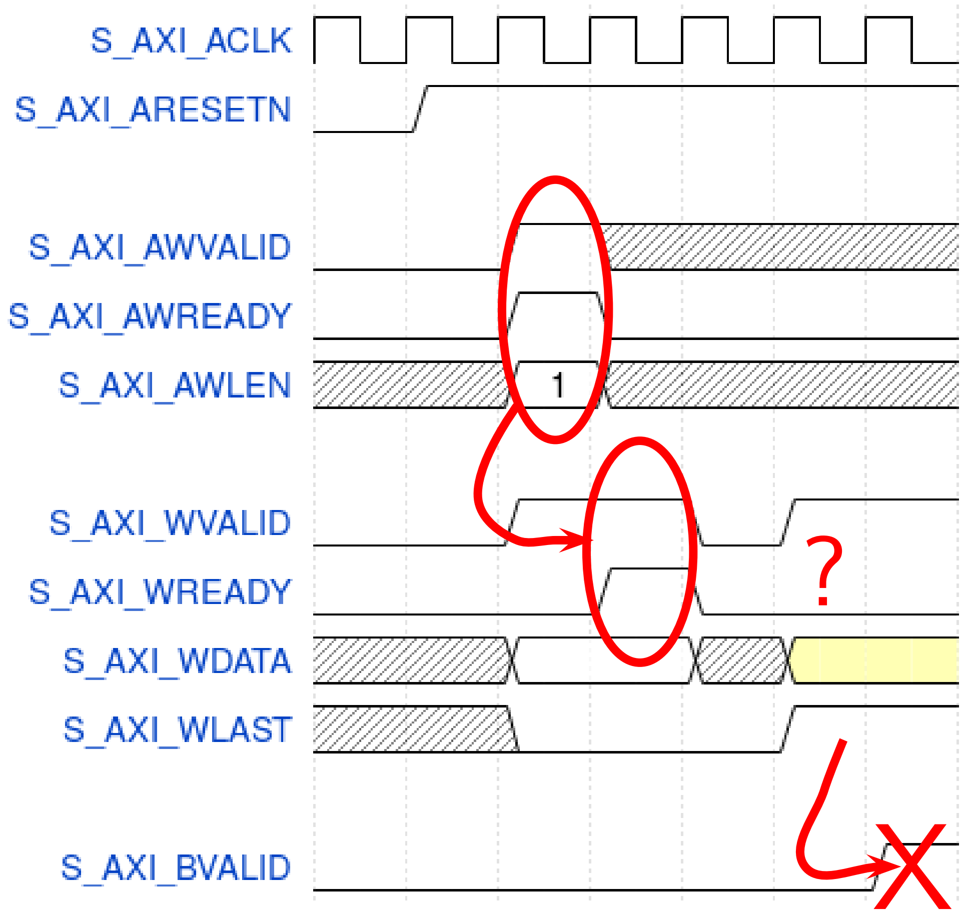
Intel's AXI3 demo



`WLAST` isn't ignored when `!WVALID` like it should be

Intel Bugs

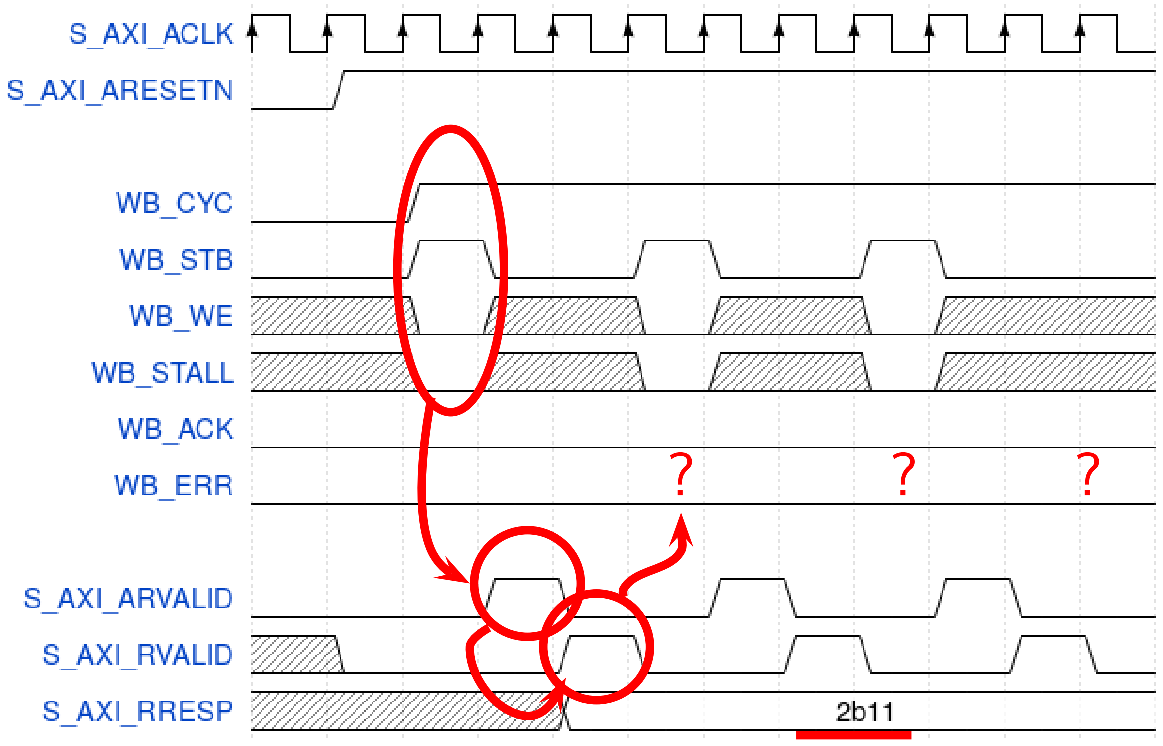
Intel's AXI3 demo



`WLAST` isn't ignored when `!WREADY`

WishboneAXI

From a WishboneAXI bridge project – since fixed

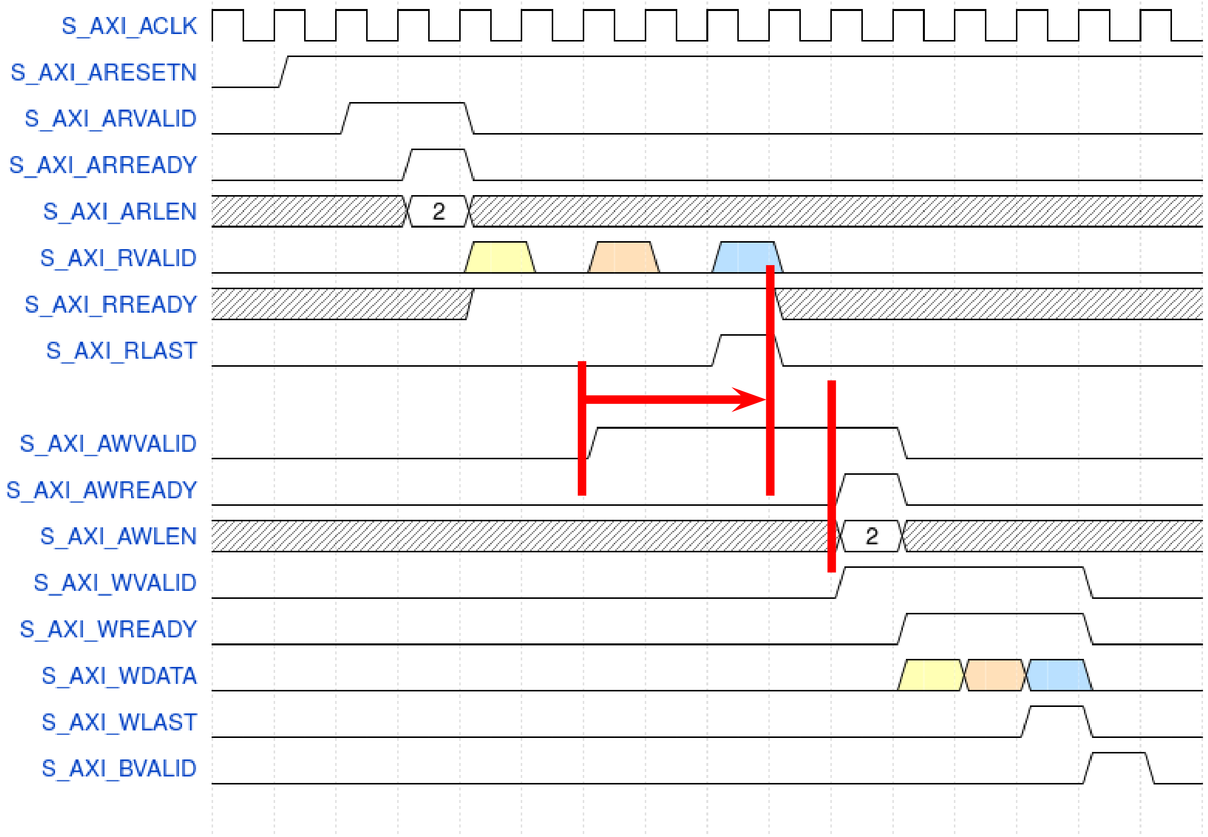


Unchecked return codes

- 2'b01: `EXOKAY` (Illegal in AXI-lite)
- 2'b11: `DECERR` (*Return was dropped*)

Xilinx's AXI demo

Vivado's 2019.1 AXI (full) demo burst write after read

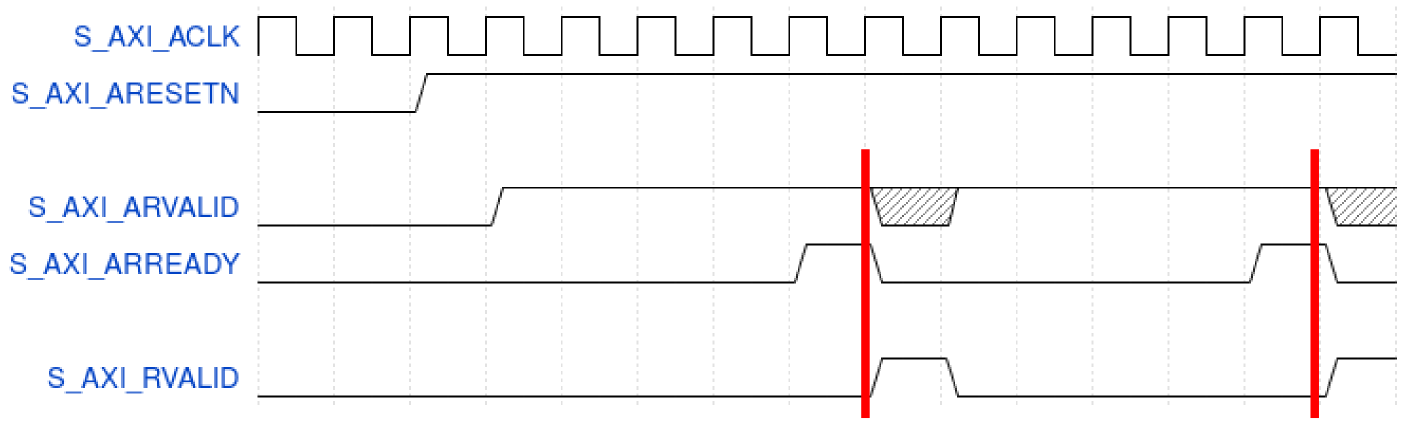


Read transactions cause any pending write transactions to wait

Xilinx's AXI GPIO

cover() can be used to determine a core's performance

- `cover(second_read_return);`

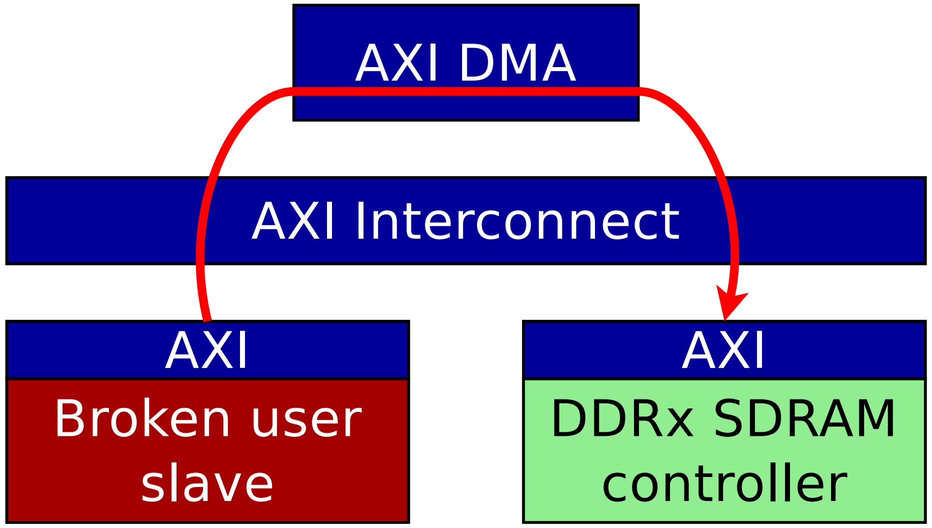


Max read throughput: One transaction every six clocks

Frozen DMA

Customer wants to reset the DMA

- Copying data to/from his (AXI) core and SDRAM
- DMA freezes. How should it be reset?



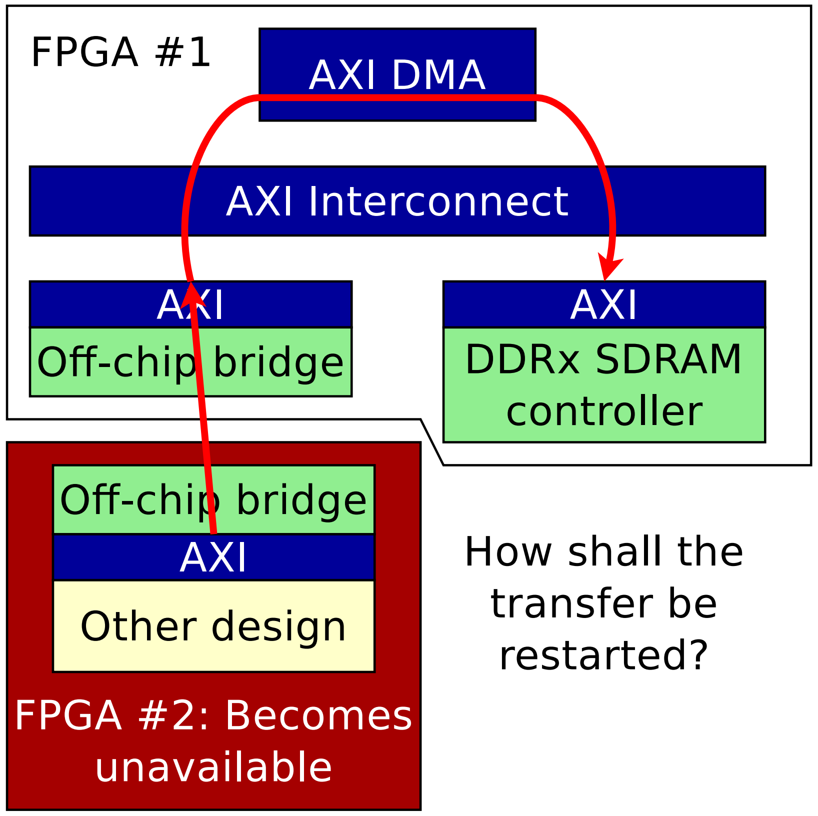
This is consistent with the bugs in Xilinx's designs

- AXI has no means of aborting a transaction
- If a transaction gets dropped, the design is dead until a reset

AXI DMA

Second customer wants to reset the DMA

- Copying data to/from an AXI-to-external bridge
- External device becomes unavailable, so the DMA freezes



How shall the transfer be restarted?

AXI DMA

Second customer wants to reset the DMA

- Copying data to/from an AXI-to-external bridge
- External device becomes unavailable, so the DMA freezes
- How should the transfer be restarted?

Xilinx's recommendation? Reset the DMA

- This would break any interconnect or other intermediate core that depends upon an outstanding transaction counter

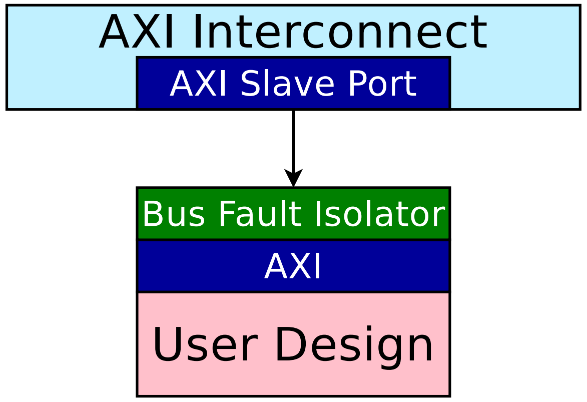
Bus Fault Isolator

Problem: One AXI mistake will hang a design

- AXI is complicated
- Neither Intel, nor Xilinx with their AXI VIP, could get it right

Solution: The bus fault isolator

- Sits between AXI infrastructure and slave design

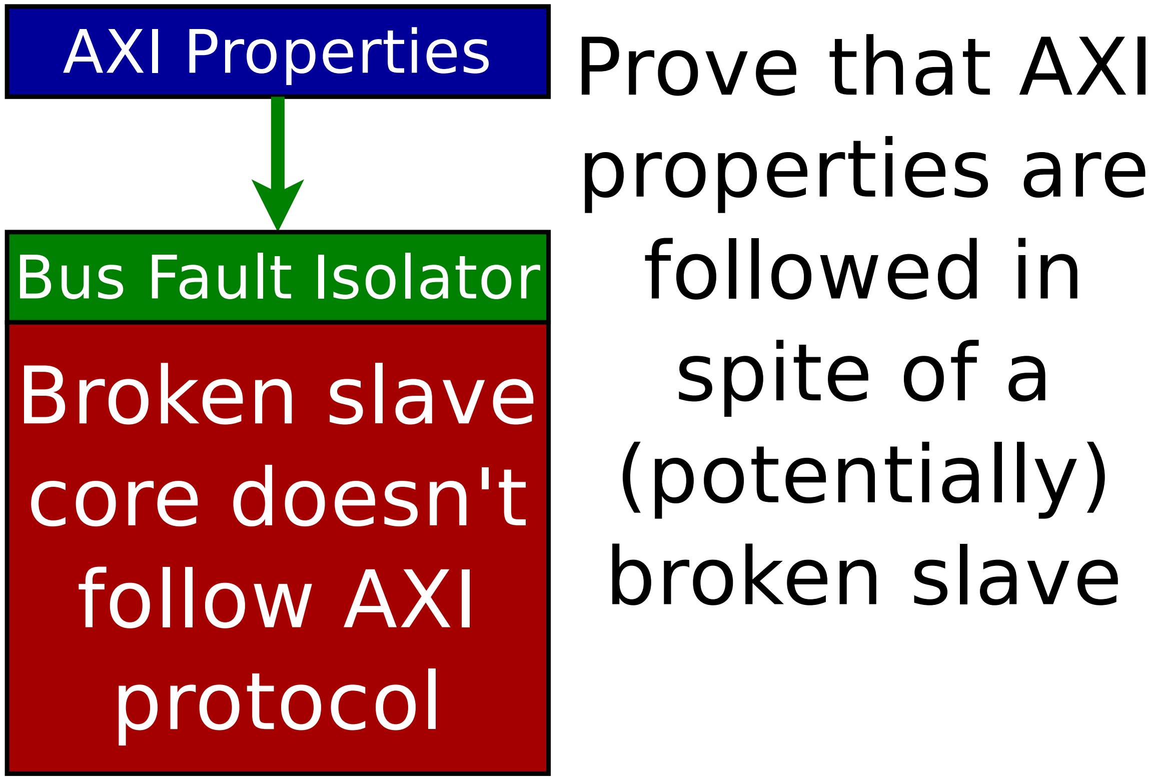


- Guarantees the slave design behaves
- Returns a valid AXI response (`SLVERR`) on any fault
- Sets flags to (potentially) trigger an internal logic analyzer
- Can be configured to reset the slave on error

Bus Fault Isolator

Two proofs:

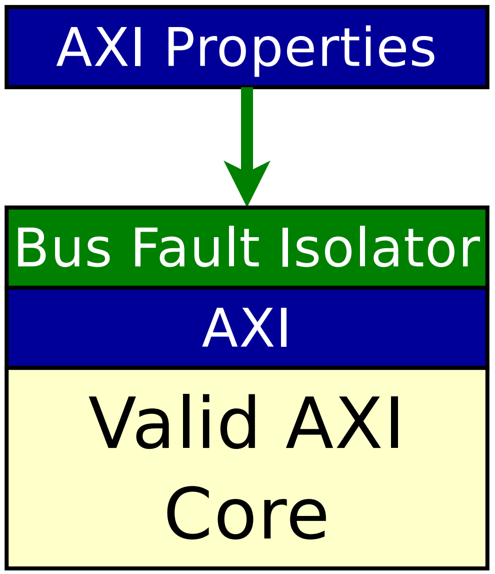
1. Prove upstream will always be valid
   - Independent of whatever the slave might do


AXI Properties
Bus Fault Isolator
Broken slave core doesn't follow AXI protocol
Prove that AXI properties are followed in spite of a (potentially) broken slave

Bus Fault Isolator

Two proofs:

1. Prove upstream will always be valid
   - Independent of whatever the slave might do
2. Assume downstream is valid
   - Prove no fault will be detected



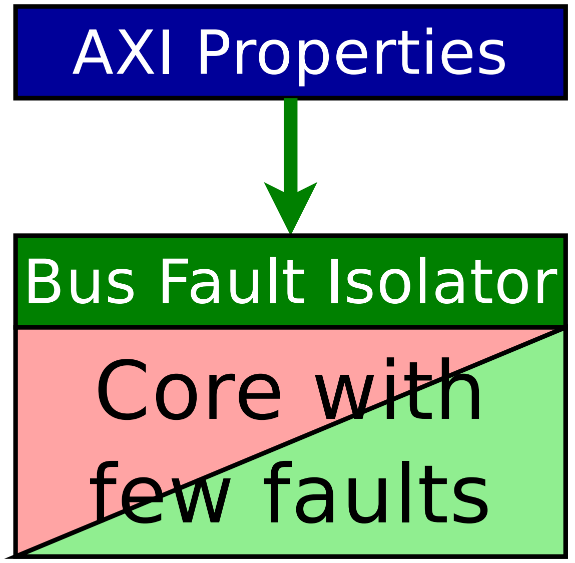
Prove that no faults are ever triggered as long as the slave core follows the AXI protocol

Bus Fault Isolator

Two proofs:

1. Prove upstream will always be valid
2. Prove valid AXI components never fault

Optional feature: Reset downstream core on any fault



Issue a reset upon a fault, repeat two proofs above

Link: axisafety.v

MicroBlaze Story

FPGA Engineer left

- The design "worked" when he left

Software engineer then modified their MicroBlaze code

- Now issued two `strh` instructions in a row
- Design stopped working

This is consistent with the bugs in Xilinx's demonstration designs
Link: (Until Xilinx deletes it)