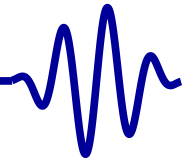
AXI Formal Verification

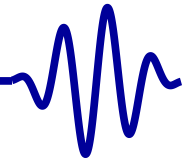
Gisselquist Technology, LLC

Daniel E. Gisselquist, Ph.D.

Overview

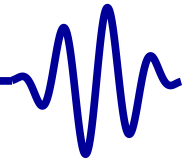


- Basic AXI formal rules
  - Bugs found
- Interconnect rules
  - Backpressure
  - IDs
- Exclusive Access
  - Surprising requirements
- Beat sizes
  - Generating unaligned requests
- Fixing simulation



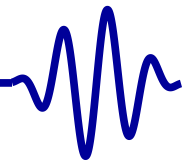
Who am I?

- 2015 – Built the ZipCPU
- 2017, Oct – First experiences with formal methods
- 2018, Dec – First AXI-lite proof
- 2019, May – First AXI(full) proof
  - Started with Vendor examples
    Found bugs in Xilinx's and Intel's examples
  - Then my own
  - Any open source I could find
  - Then Vendor IP
- 2021 – ZipCPU AXI bus interface
  - Big endian, full exclusive access support

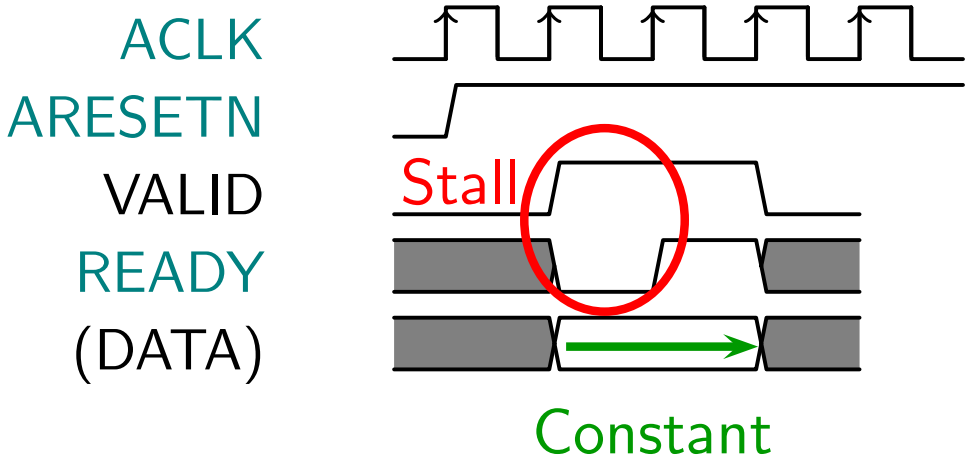


Basic AXI

Handshaking



1. Reset clears all requests
2. If a channel is stalled, nothing should change



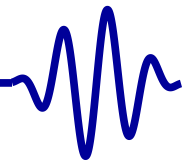

```
assert property (@(posedge ACLK)
    !ARESETN |=> !VALID);

assert property (@(posedge ACLK)
    disable iff (!ARESETN)
    VALID && !READY |=> VALID && $stable(DATA));
```

Logic form #1

All master logic needs to (roughly) follow the form:

```
if (!ARESETN)
begin
        VALID <= 0;
        (DATA <= defaults;)
end else if (!VALID || READY)
begin
        // VALID and data can change
        if (valid_will_be_zero)
                // Data can be reset
end
```

Logic form #2



I've also seen this be successful

```
if (!ARESETN)
begin
        VALID <= 0;
        (DATA <= defaults;)
end else if (VALID && READY)
begin
        // Move on to next beat
        // or end the transaction
end else if (!VALID)
begin
        // Consider starting a transaction
end
```

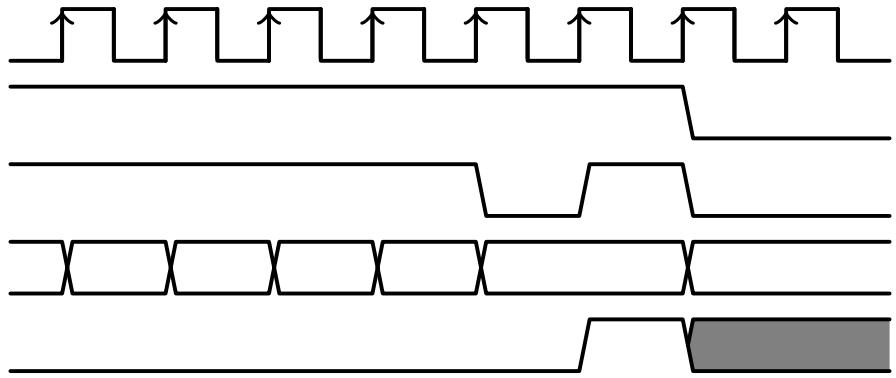
Stream Demo

This logic from Vivado's AXI stream template didn't work

```
assign  tlast = (read_ptr == NWORDS-1);

always @(posedge ACLK)
if (!ARESETN)
        tlast_delay <= 0;
else
        tlast_delay <= tlast;
```

Stream Master

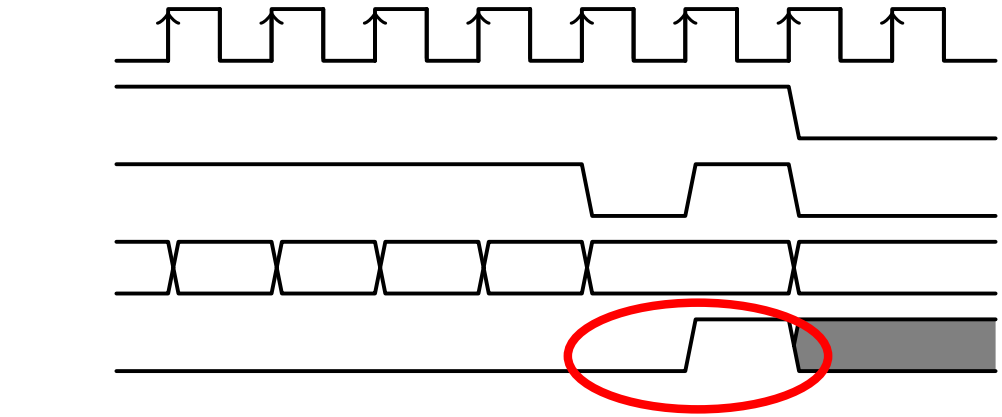
SymbiYosys produced a trace containing this feature:



-
-
- Time to bug?

Stream Master

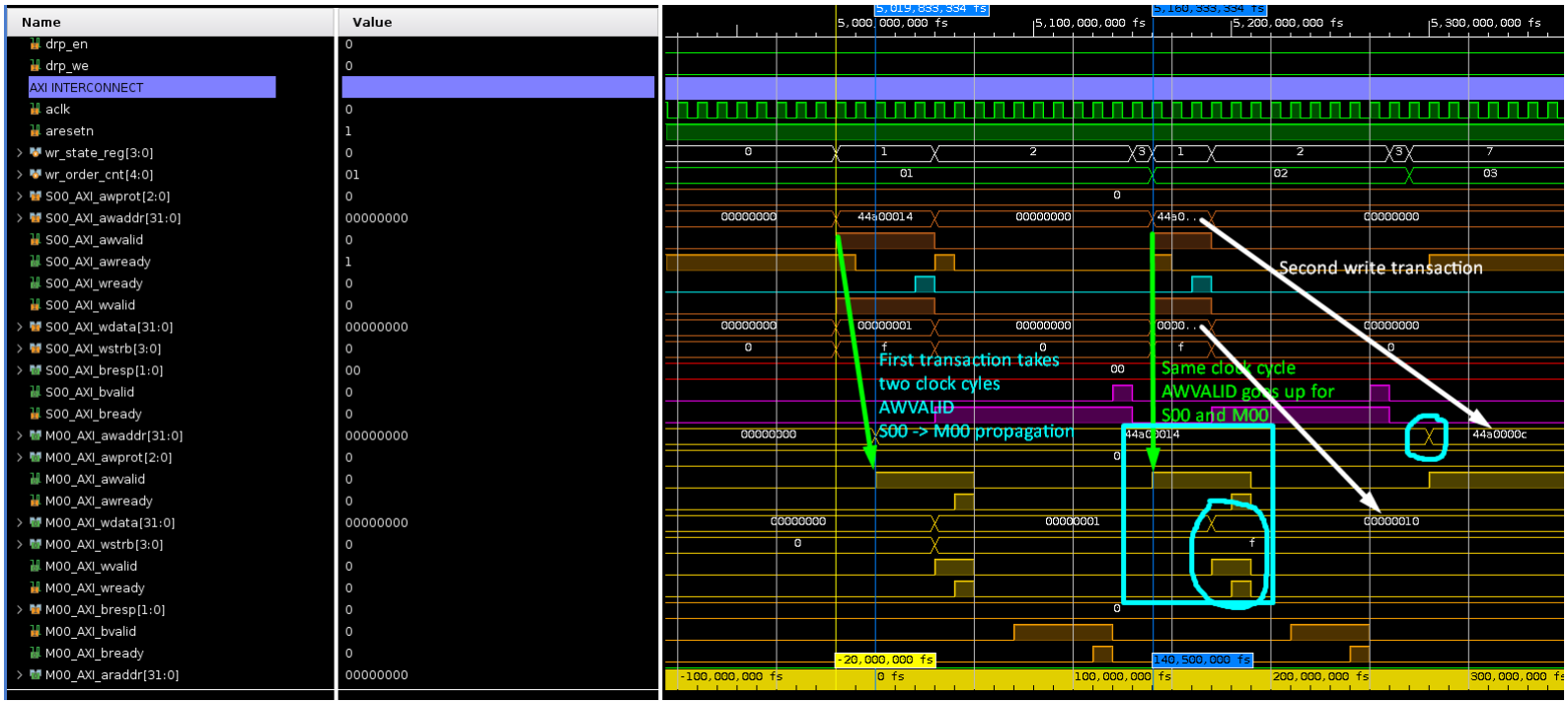
SymbiYosys produced a trace containing this feature:



- `TLAST` violates the handshaking rules
- Last beat of the packet would be ignored
- Time to bug? 3 seconds

Wrong Address

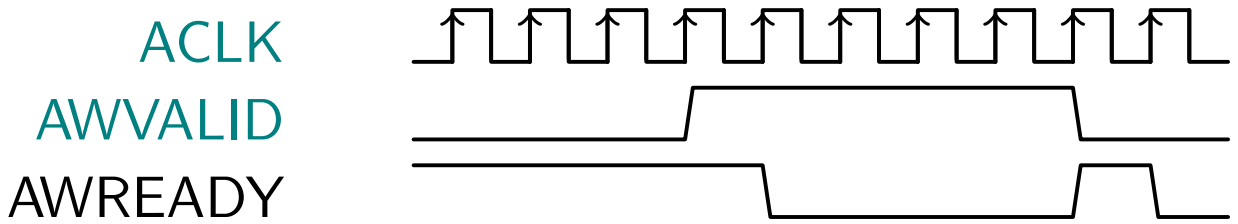
User: Xilinx's IP keeps writing my data to the wrong address



Where would you start?

Wrong Address

User: Xilinx's IP keeps writing my data to the wrong address

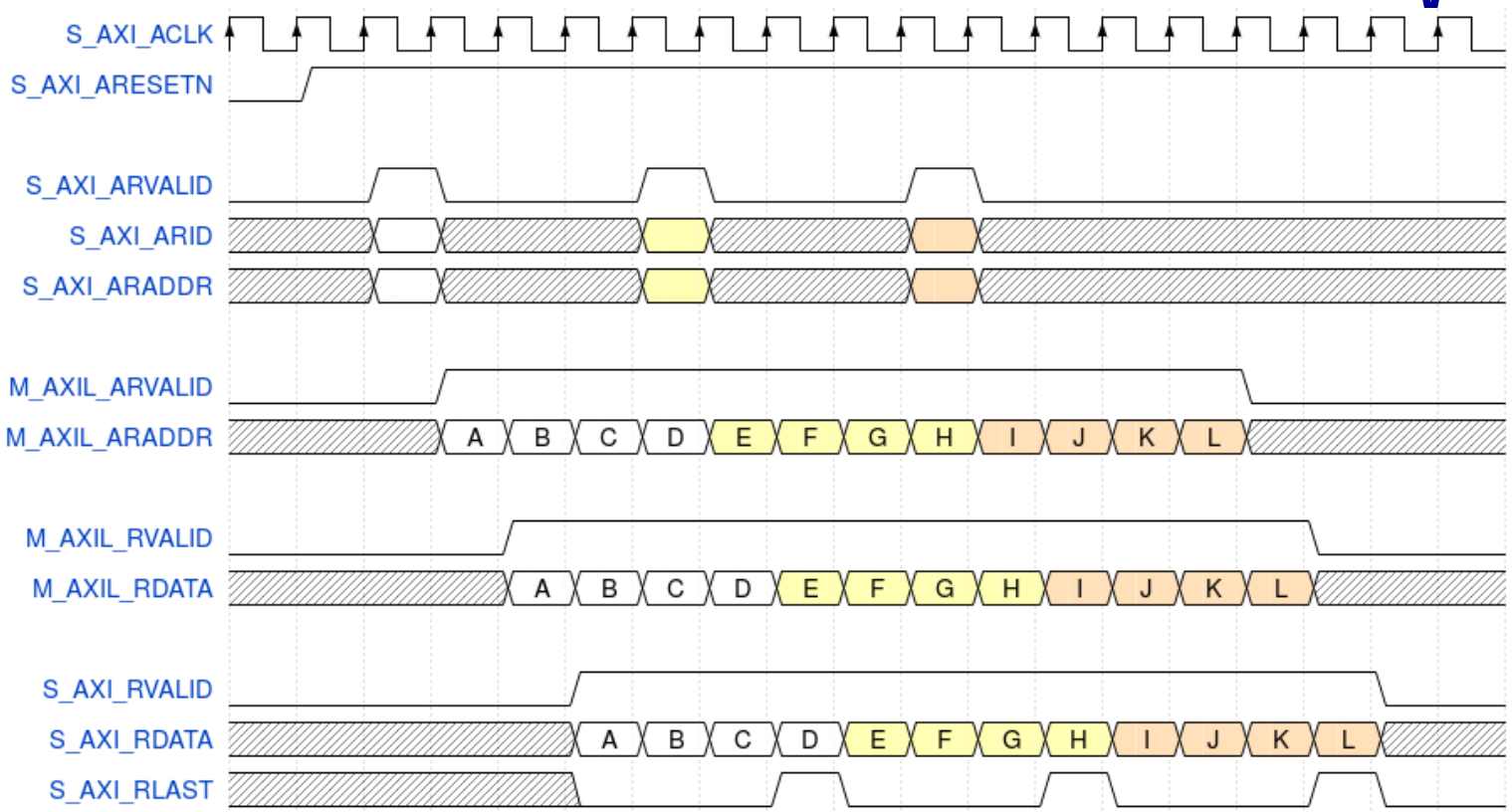


It took me all day to find this

- The interconnect was buffering an extra write address request

A formal check would've found this in seconds

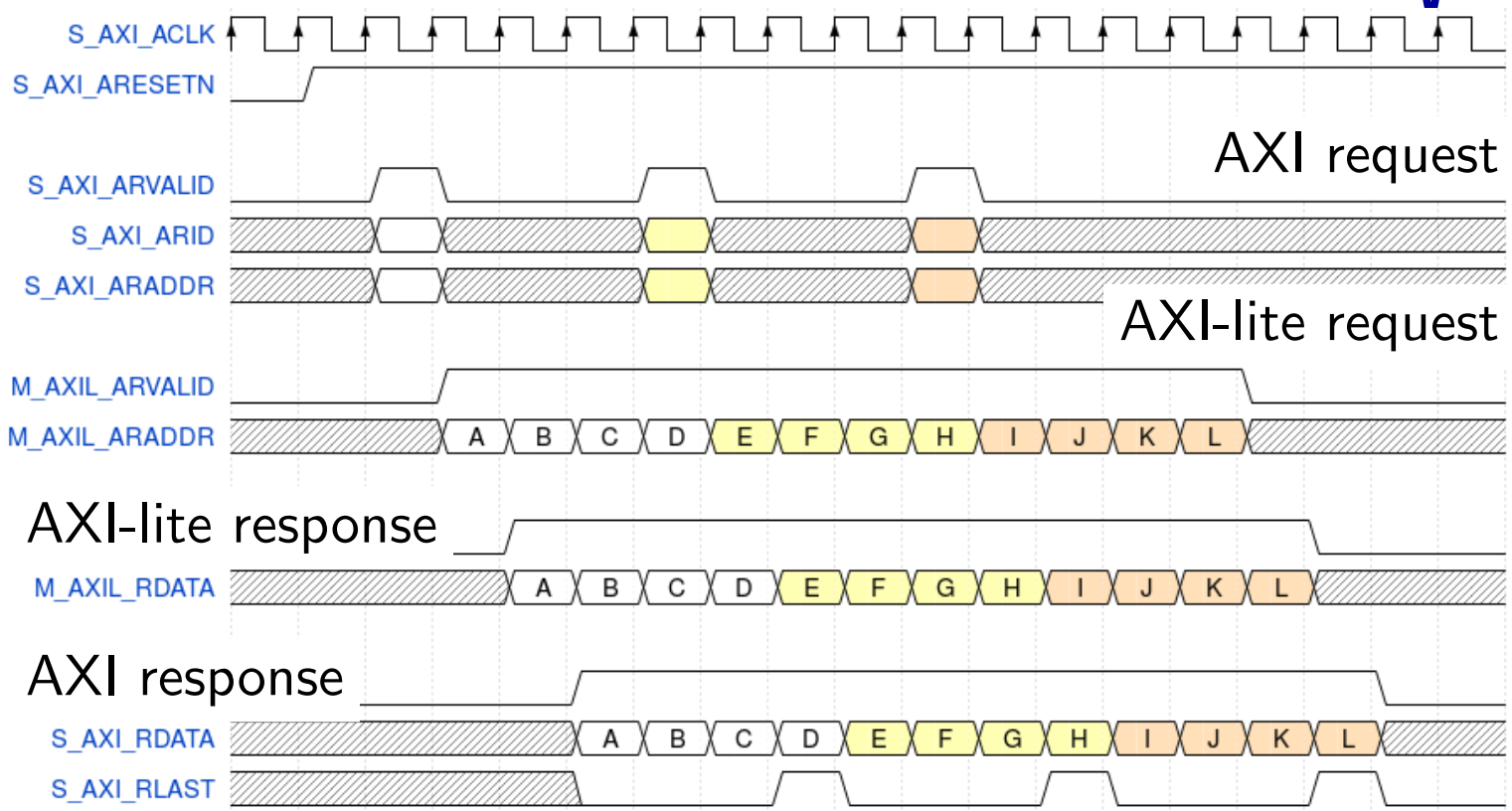
AXI to AXI-lite



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

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

AXI to AXI-lite

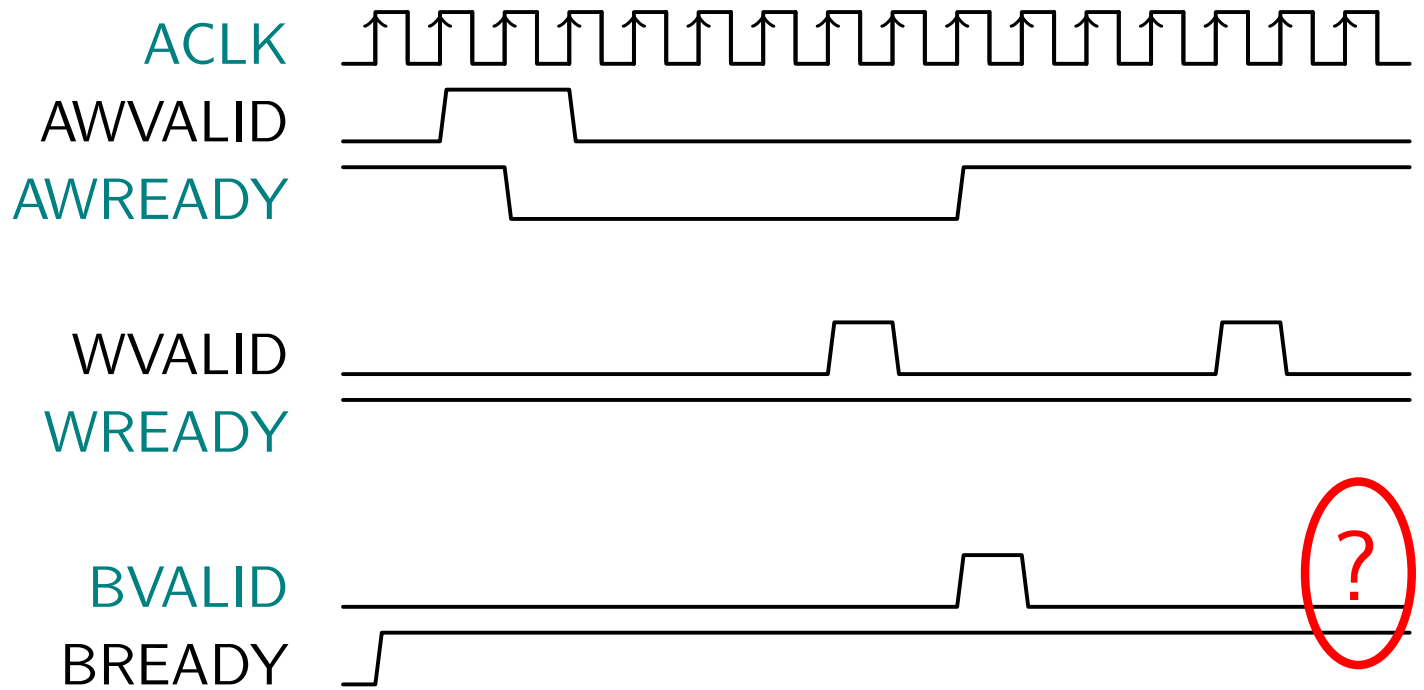


That's how an AXI to AXI-lite bridge should work

- No wasted clock cycles

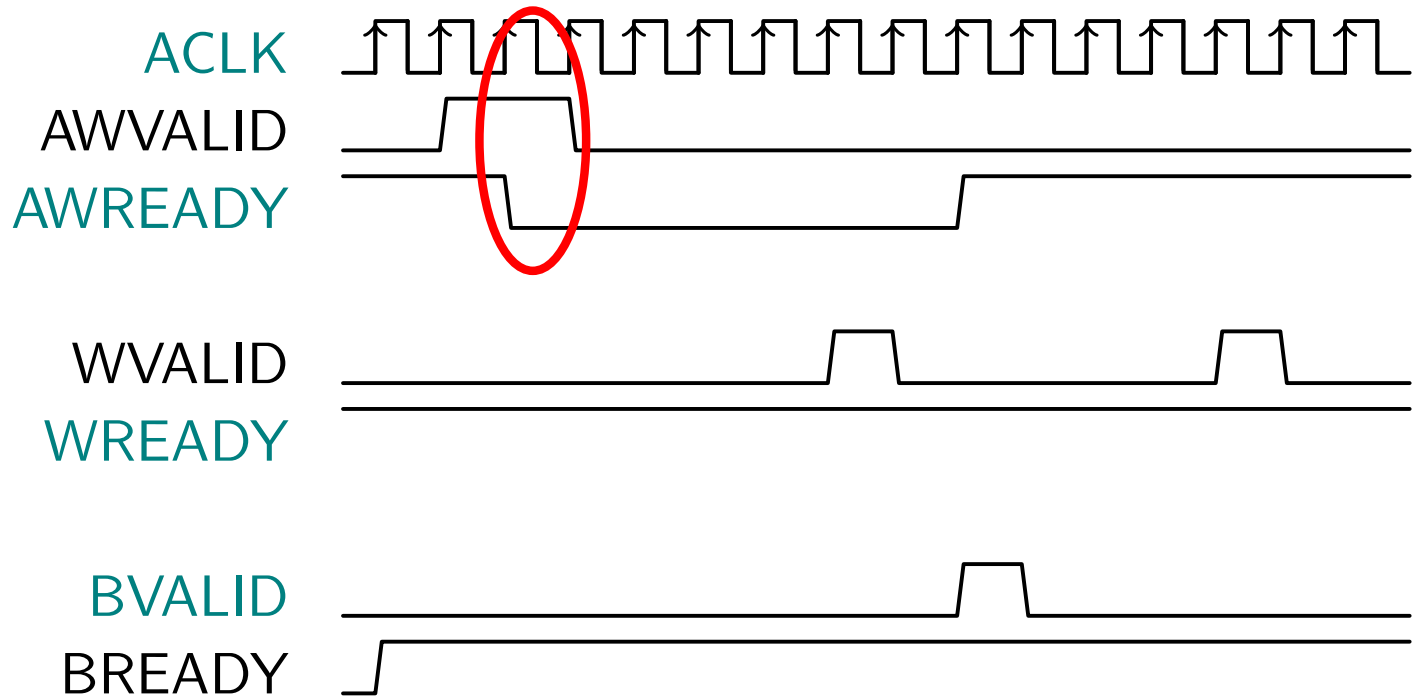
Bug?

User: Your AXI to AXI-lite bridge drops a `BVALID`

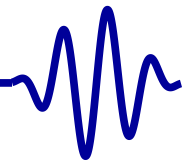


Bug?

User: Your AXI to AXI-lite bridge drops a `BVALID`



The cause



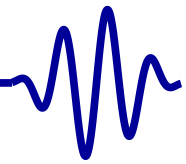
User: Your AXI to AXI-lite bridge drops a `BVALID`

- This user was trying to delay an AXI interaction
- He delayed all the AXI signals, independent of the `xREADY` signals

```
always @(posedge ACLK)
if (!ARESETN)
        sreg <= 0;
else
        sreg[N-1:0] <= { sreg[N-2:0], S_VALID };

assign  M_VALID = sreg[N-1];
```

If you need a delay, this isn't how you implement it

Phantom Returns

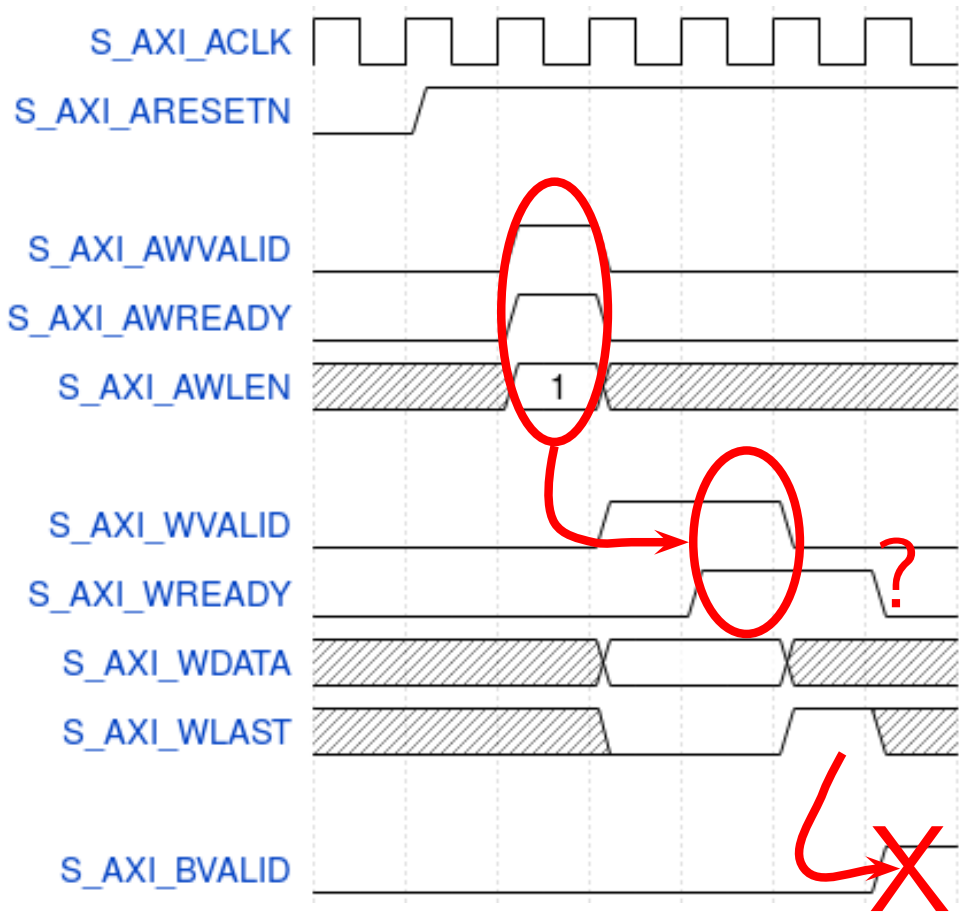


Rule: No responses without prior requests

1. Count requests
   - Count `AWVALID && AWREADY` minus `BVALID && BREADY`
   - Count `WVALID && WREADY` (and possibly `WLAST`) minus `BVALID && BREADY`
   - Count `ARVALID && ARREADY` minus `RVALID && RREADY` (and possibly `RLAST`)
2. Rules
   - No `BVALID` unless the both write counters are greater than zero
   - No `RVALID` unless the read counter is greater than zero

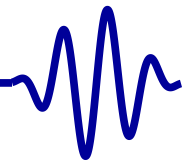
Intel Bugs

Intel's AXI3 demo



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

Bug

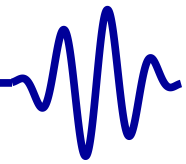


This was found in an ASIC IP vendor's AXI3 design:

1. If `WVALID && WID != current_AWID`, you have an out of order AXI3 write
2. The design couldn't handle out of order writes, so ...
3. It set `BVALID && BID = WID && BRESP = SLVERR`
4. This bug was not revealed in simulation
   - Couldn't be reproduced in simulation at all
   - The vendor VIP had no ability to issue out of order writes

Thankfully, no one encountered this bug

Responsiveness

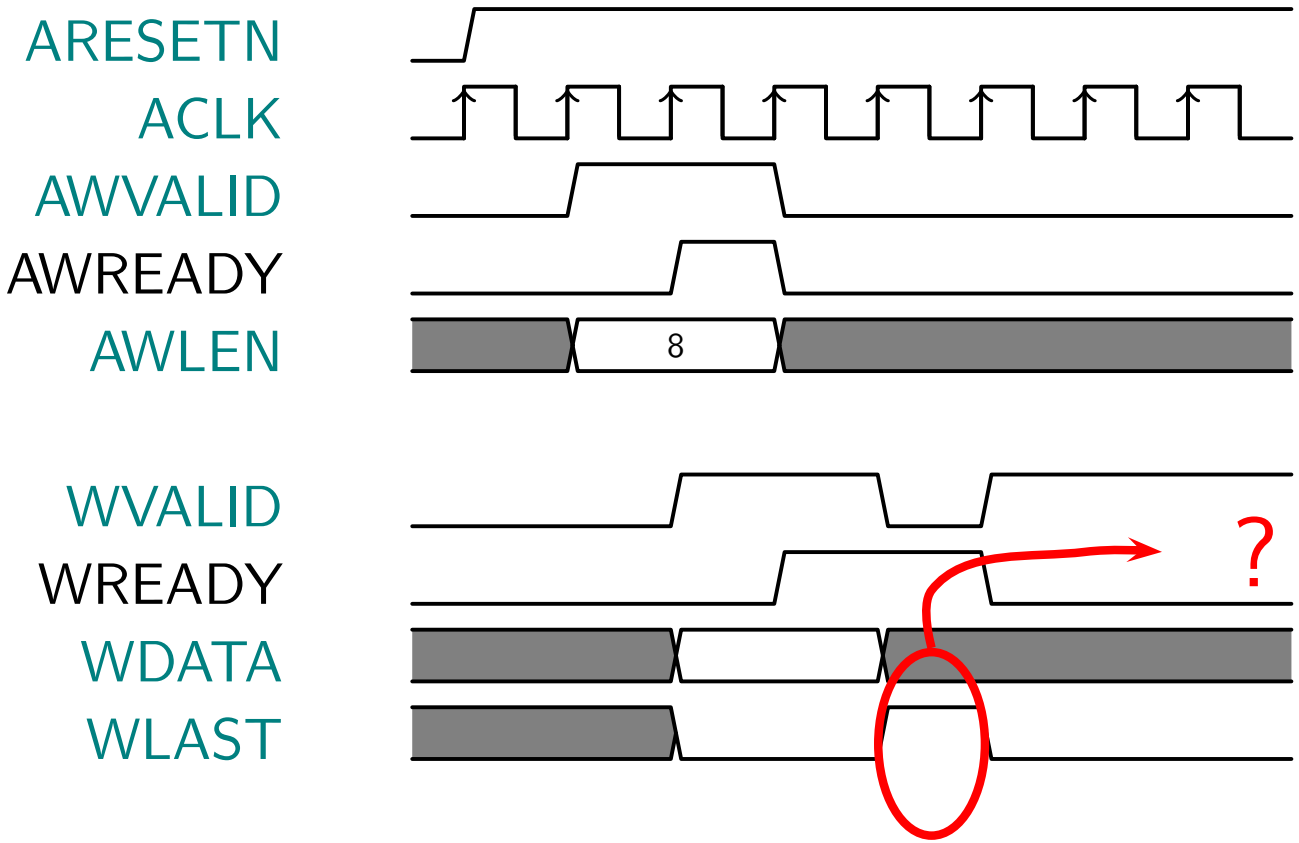


Can't stall indefinitely

1. Count stalls
   - Count times when slave is idle and `AWVALID && !AWREADY`
   - or `WVALID && !WREADY`
   - or `ARVALID && !ARREADY`
   - or, from the master, if the `AWVALID` counter indicates an outstanding request for which `!WVALID`
2. Rules
   - Force these counters less than a maximum value

Xilinx bug

This bug can be found in Vivado's AXI templates

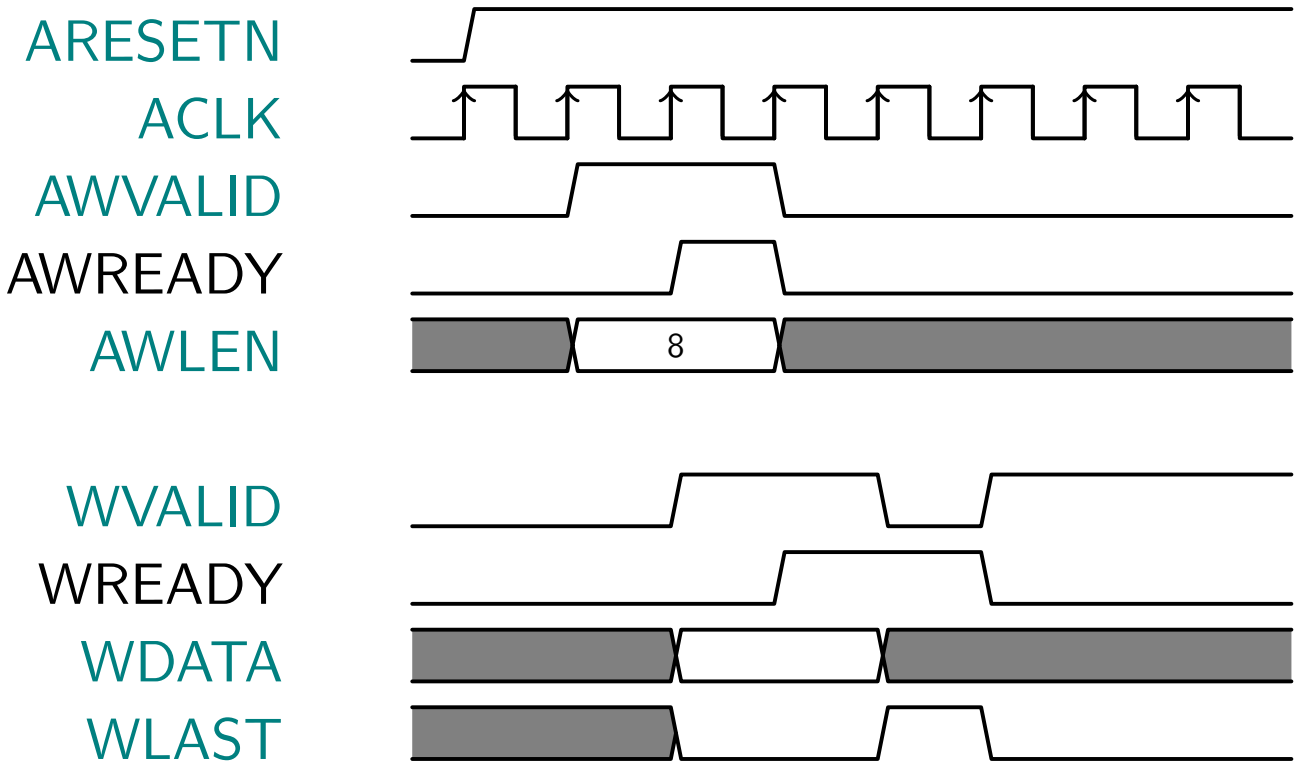


`WLAST` *should've* been ignored if `!WREADY`

- Time to bug?

Xilinx bug

This bug can be found in Vivado's AXI templates



`WLAST` *should've* been ignored if `!WREADY`

- Time to bug? < 1 second

AXI Ethernet-Lite

Here's another Xilinx bug, this time from their Ethernet–Lite IP

```
AXI_READ_VALID_P: process (S_AXI_ACLK) is
begin
  if (S_AXI_ACLK'event and S_AXI_ACLK = '1') then
      if (S_AXI_ARESETN=RST_ACTIVE) then
          S_AXI_RVALID <= '0';
      elsif S_AXI_RREADY = '1' then
          S_AXI_RVALID <= rvalid;
      end if;
  end if;
end process AXI_READ_VALID_P;
```

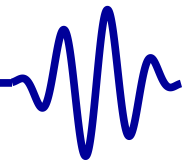
AXI Ethernet-Lite

Here's another Xilinx bug, this time from their Ethernet–Lite IP

```
AXI_READ_VALID_P: process (S_AXI_ACLK) is
begin
  if (S_AXI_ACLK'event and S_AXI_ACLK = '1') then
      if (S_AXI_ARESETN=RST_ACTIVE) then
          S_AXI_RVALID <= '0';
      elsif S_AXI_RREADY = '1' then
          S_AXI_RVALID <= rvalid;
      end if;
  end if;
end process AXI_READ_VALID_P;
```

The formal tool was just ornery enough to hold `S_AXI_RREADY` low long enough to trigger it

AXI Ethernet-Lite



Knowing there was a bug, Xilinx "fixed" their IP.

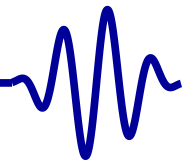
- After the fix, this was left behind:

```
AXI_READ_OUTPUT_P: process (S_AXI_ACLK) is
begin
  if (S_AXI_ACLK'event and S_AXI_ACLK = '1') then
      if (S_AXI_ARESETN=RST_ACTIVE) then
          S_AXI_RDATA  <= (others =>'0');
      elsif S_AXI_RREADY = '1' then
          S_AXI_RDATA   <= IP2Bus_Data;
      end if;
  end if;
end process AXI_READ_OUTPUT_P;
```

Beware, the first read result might be invalid.

- This bug still passed all their testing
- They couldn't see it

No Lockups

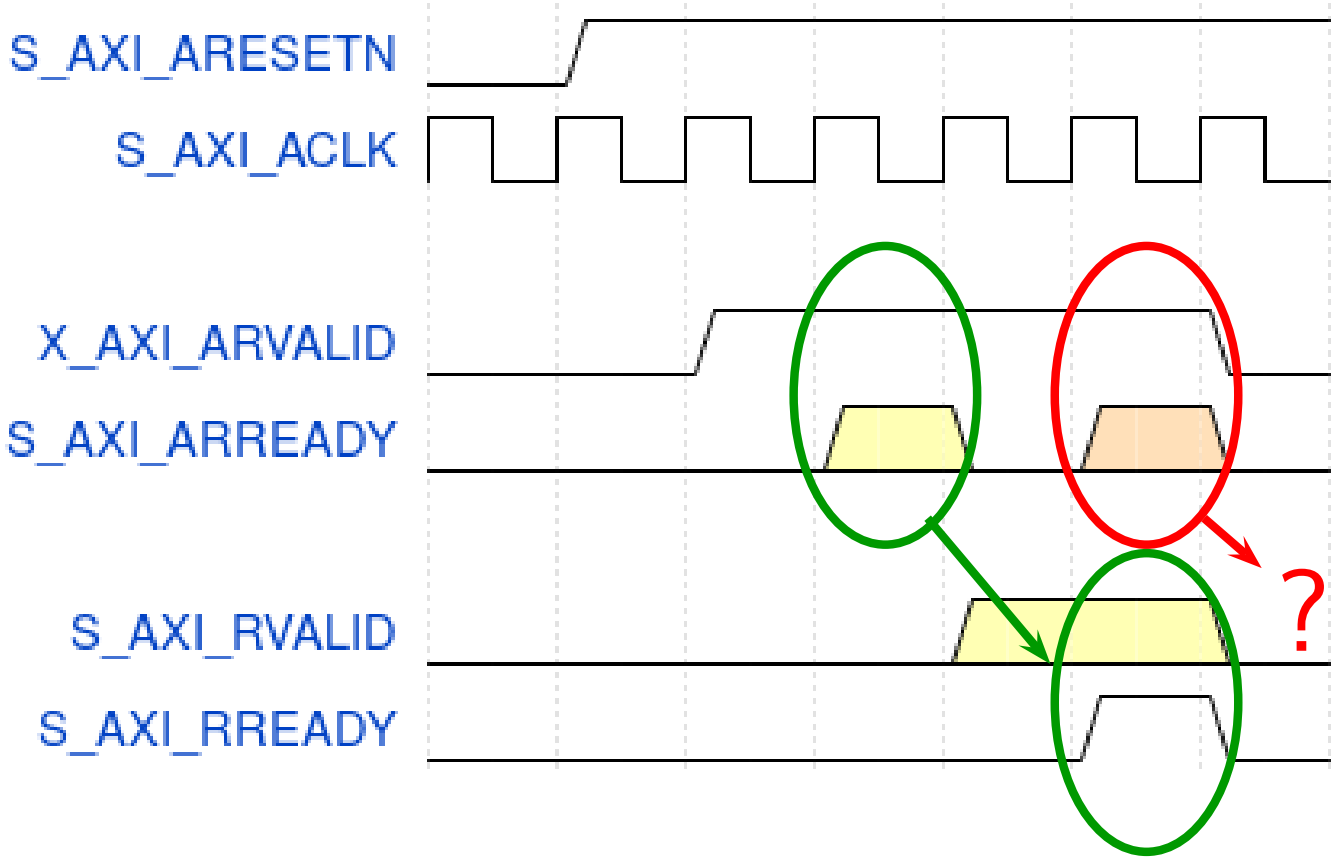


Rule: Every request must get a response

1. Count wait cycles
   - Count clock cycles between requests and responses
2. Rules
   - If at least one request is outstanding, a response should be returned in a minimum amount of time
   - The actual response time is slave dependent
3. Challenges
   - Some slaves join read and write channels together

Bugs Found

Vivado 2016.1–2021.1, AXI-lite READ

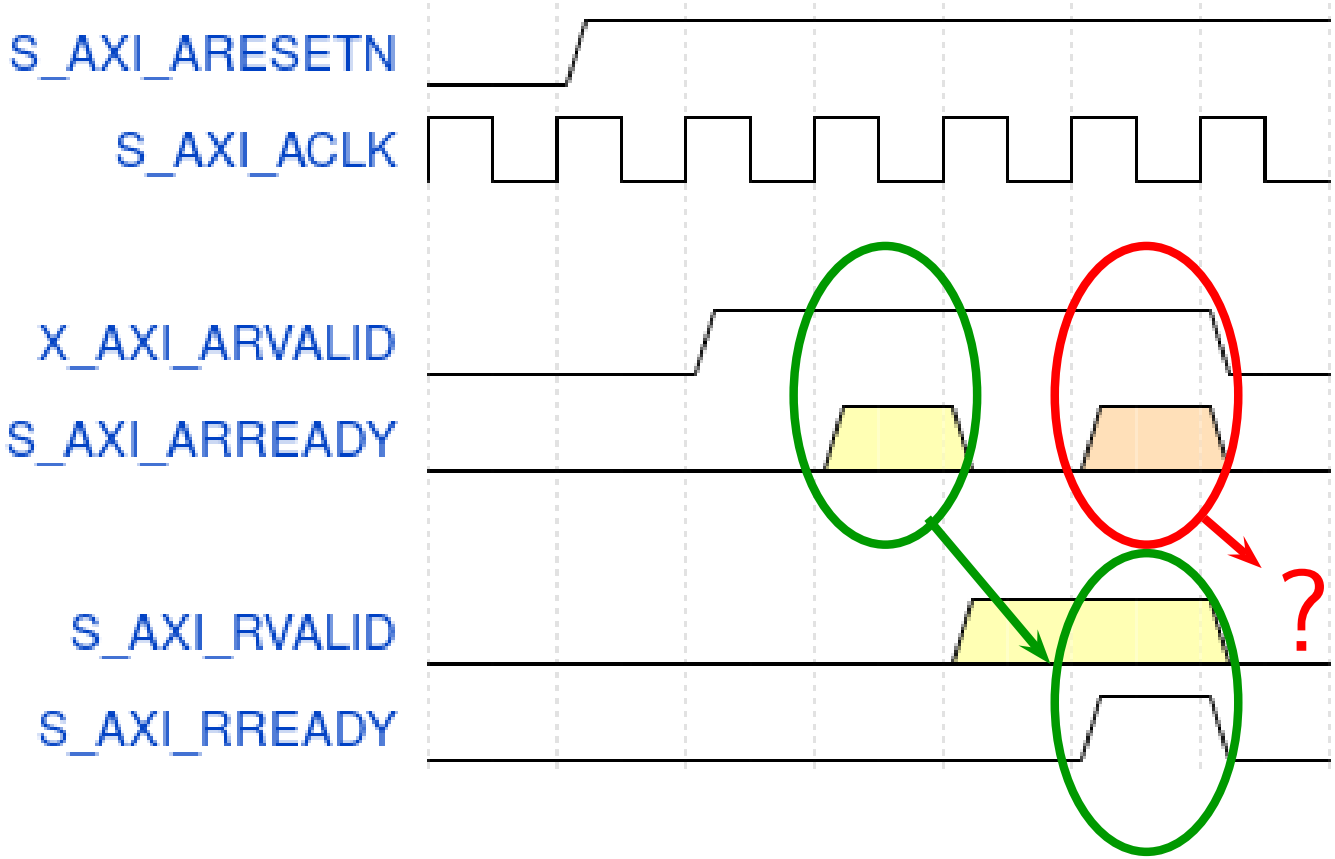


Two requests, one response: design will hang

- Time to bug?

Bugs Found

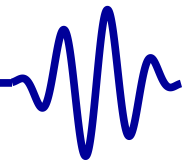
Vivado 2016.1–2021.1, AXI-lite READ



Two requests, one response: design will hang

- Time to bug? 2 seconds

TinyTPU



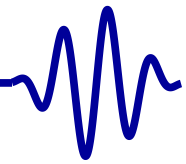
This example is drawn from the TinyTPU:

```
if (!ARESETN)
        // ...
else case(state)
IDLE:
        case({AWVALID, ARVALID })
        2'b10:
                state <= WRITE_ADDRESS;
        2'b01:
                state <= READ_ADDRESS;
        default:
        endcase
// ...
```

This design is now broken.

- Few simulation environments could trigger this bug
- Time to bug?

TinyTPU



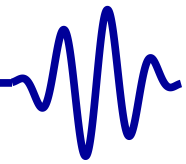
This example is drawn from the TinyTPU:

```
if (!ARESETN)
        // ...
else case(state)
IDLE:
        case({AWVALID, ARVALID })
        2'b10:
                state <= WRITE_ADDRESS;
        2'b01:
                state <= READ_ADDRESS;
        default:
        endcase
// ...
```

This design is now broken.

- Few simulation environments could trigger this bug
- Time to bug? 1 second

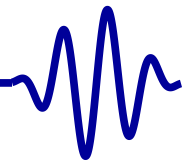
AXI-Lite



The above rules apply to AXI-lite

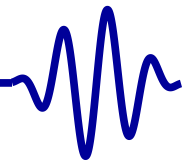
1. Sufficient for verifying an AXI-lite interface
2. Haven't yet met an AXI-lite design that couldn't be verified
3. If you can verify it, you can build it
   - Post: Building an AXI-Lite slave the easy way
   - I've now re-used that design over and over again

AXI is not so easy



Interconnects

Routing

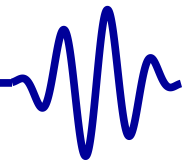


Routing from master to slave may be done by ID

- On request
  - Interconnect appends master's ID and forwards request to the slave
- On return
  - Interconnect routes the return back to the requesting master based upon the ID
  - The extra ID bits are then stripped
- Returns for different IDs can come back in any order

The process appears simple. Stateless.

Backpressure



If two requests are outstanding, . . .

- for two different IDs,
- for two or more different slaves,
- and both slaves for both return at the same time

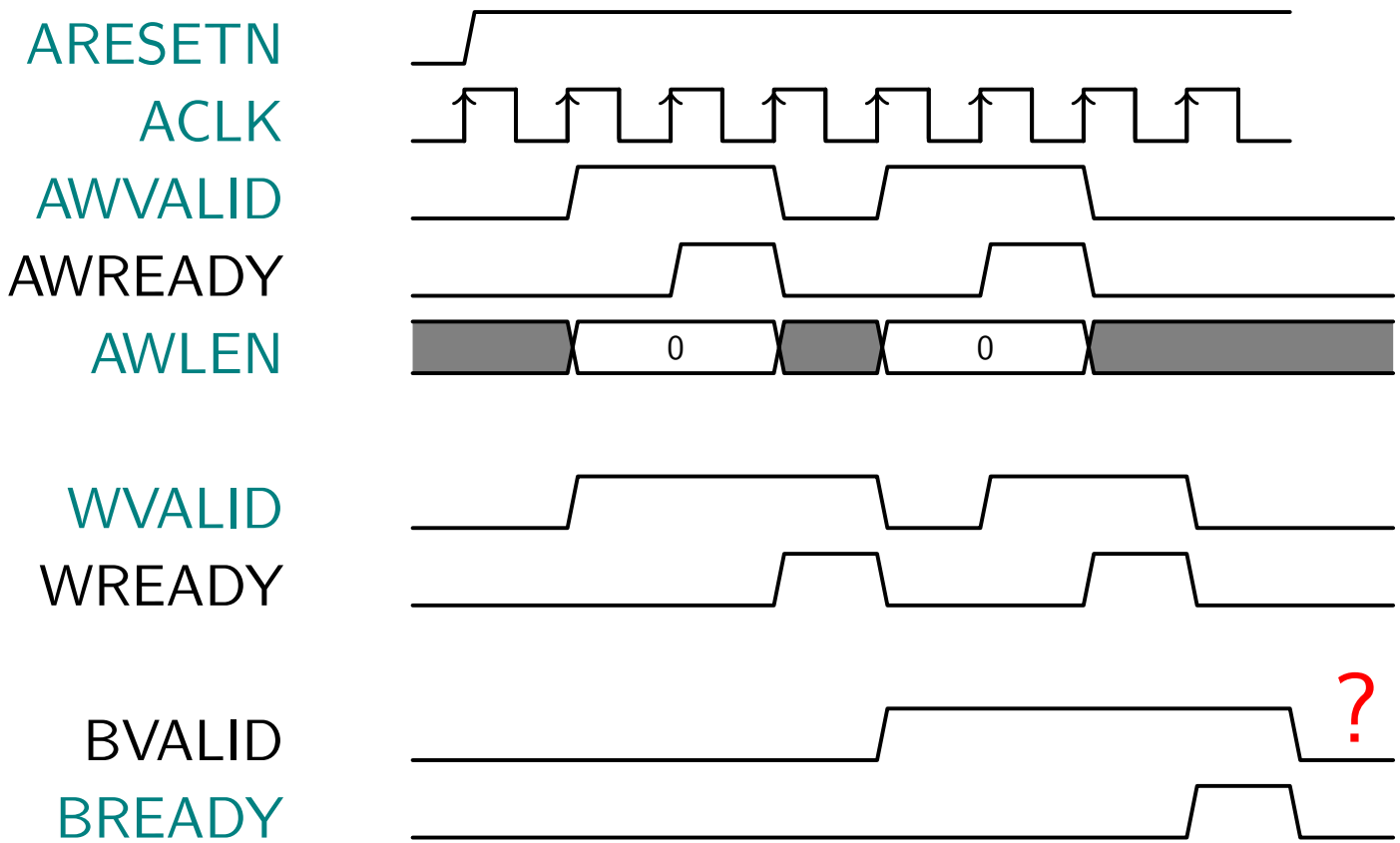
then the interconnect must stall one of the slaves.
This forces arbitration on the return

- This is called *backpressure*

Many designs will fail in the presence of backpressure

Backpressure

This bug can be found in Vivado's AXI slave template



Given sufficient backpressure, this demo slave will drop returns

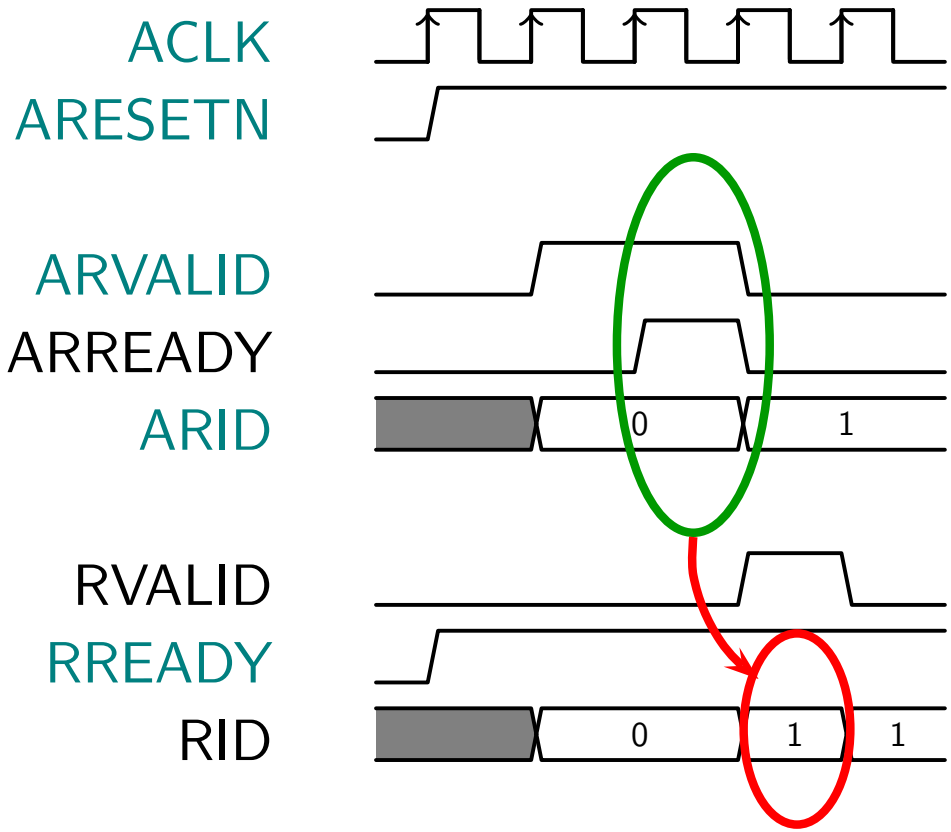
ID Requirement

Rule: Interconnect must guarantee that requests from the same ID are returned in order of issue

- Requires a counter in the interconnect
  - Must count outstanding items for a given ID
  - When the number of outstanding items is zero, can switch slaves
- (Unsaid) The interconnect may need to stall the channel to keep the counter from overflowing
- (Unsaid) You cannot reset a master or slave while anything is outstanding without also resetting the rest of the bus

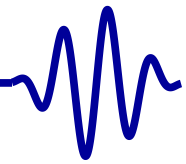
ID Error

Another bug from Vivado's AXI slave template



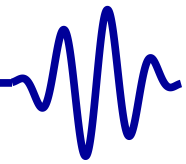
Problem: The ID *must* be registered with the return

Ultrascale+



The ARM on the Ultrascale+ supports 64k (16b) IDs

- Interconnect must either support 64k counters, or . . .
- It may reduce the number of IDs in flight at any given time



Exclusive Access

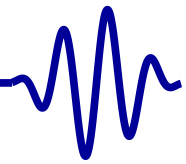
Atomic Instructions

Multiprocess coordination requires special atomic instructions

- Example instructions include:
  - Test and set
  - Atomic increment/decrement
- Semaphore implementation requires such instructions

RTL examples of this are hard to find

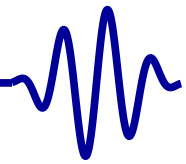
Legacy Atomics



Prior to AXI4, atomic access required locking the bus:

1. A lock request would be received
2. All other transactions would be flushed
   - No new requests would be accepted
   - Ongoing operations would complete
3. Lock is granted
   - The requesting master is then given access to the bus
   - All other requests are still stalled
   - The master would read a value, and
   - Operate on it, and
   - (Likely) Write a value back
4. The lock would be released
   - Bus returns to normal

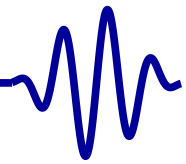
AXI4 Atomics



No bus locking any more

1. A master requests an exclusive access read
2. If the slave can support exclusive access,
   - It copies the details of the read request
   - Processes the read as it would normally
   - Returns `EXOKAY` status
   - If the slave doesn't support exclusive access
     - It returns `OKAY`

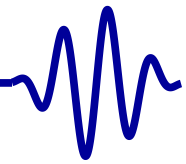
AXI Atomic



No bus locking any more

1. Exclusive access read return is `EXOKAY`
2. The master then requests an exclusive access write
   - If the write request matches the prior read request, and
     - nothing else has written to the given address since the prior read request
   - Then the write is accomplished, and
     - the slave returns `EXOKAY`
   - Else the write is ignored, and the slave returns `OKAY`

Assembly



The following is an example of an atomic increment with exclusive access

ARM	ZipCPU
`DMB ish`	
loop:	
	`LOCK`
`LDREX R2,[R3]`	`LW (R3),R2`
`ADD R0,R2,R1`	`ADD R1,R2`
`STREX R12,R0,[R3]`	`SW R2,(R3)`
`CMP R12,#0`	*(Implied loop)*
`BNE loop`	
`DMB ish`	

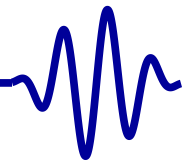
As of GCC 10.3, Microblaze has no support for atomic increment

RTL Examples

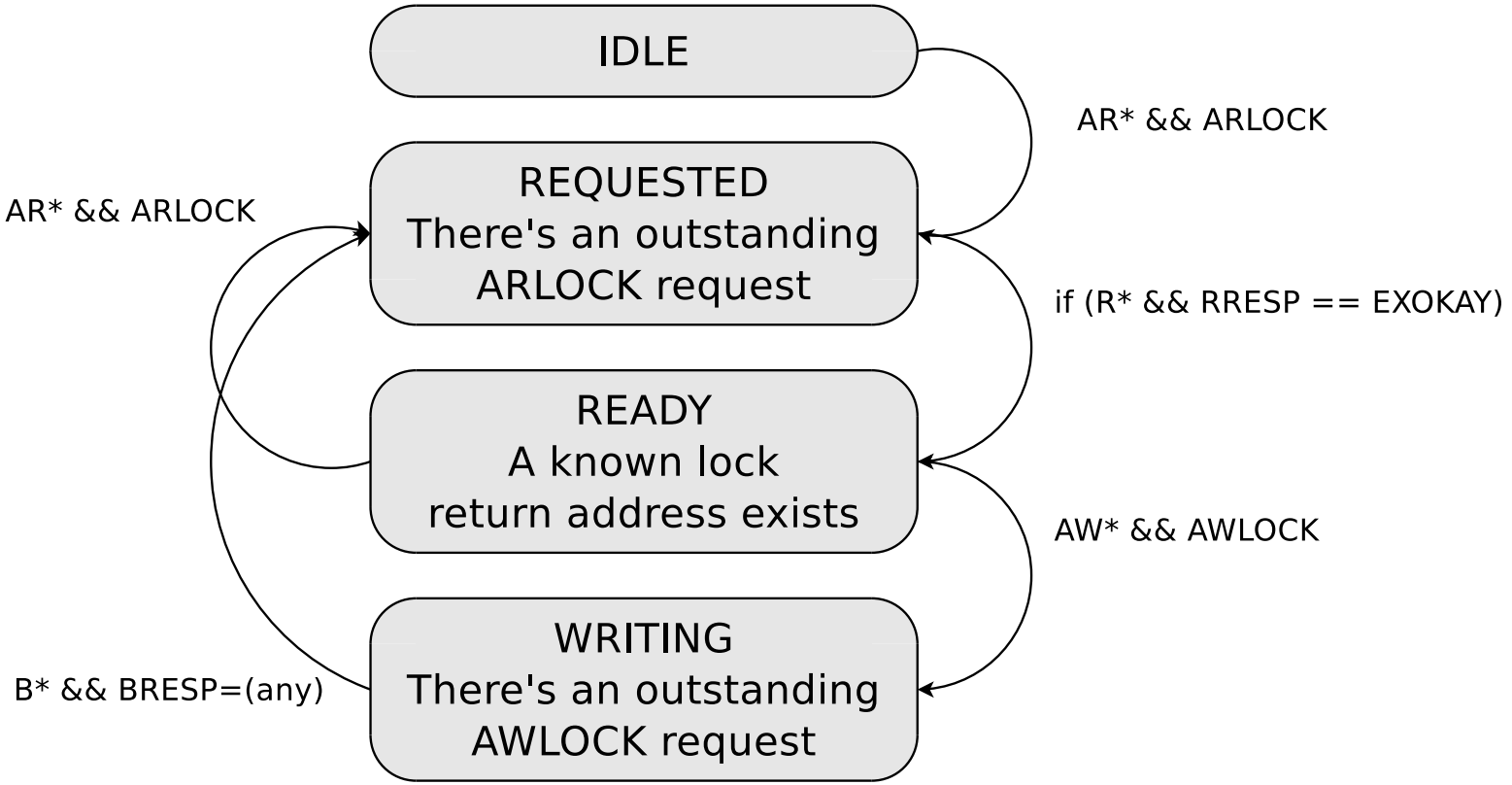
Neither Intel nor Xilinx offer any examples

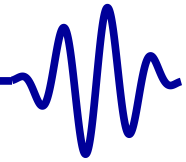
1. AXI slave examples doesn't support exclusive access
2. Xilinx's MIG doesn't support it
3. Xilinx's Block RAM controller doesn't support it

s_axi_awlock [2:0]	S_AXI (AW)	I		AXI write address lock signal: Provides information about atomic operation transfers and barrier transactions. Unused at this time, but listed here for future implementation and support.
s_axi_arlock [2:0]	S_AXI (AR)	I		AXI read address lock signal. Provides information about atomic operation transfers. Unused at this time, but listed here for future implementation and support.

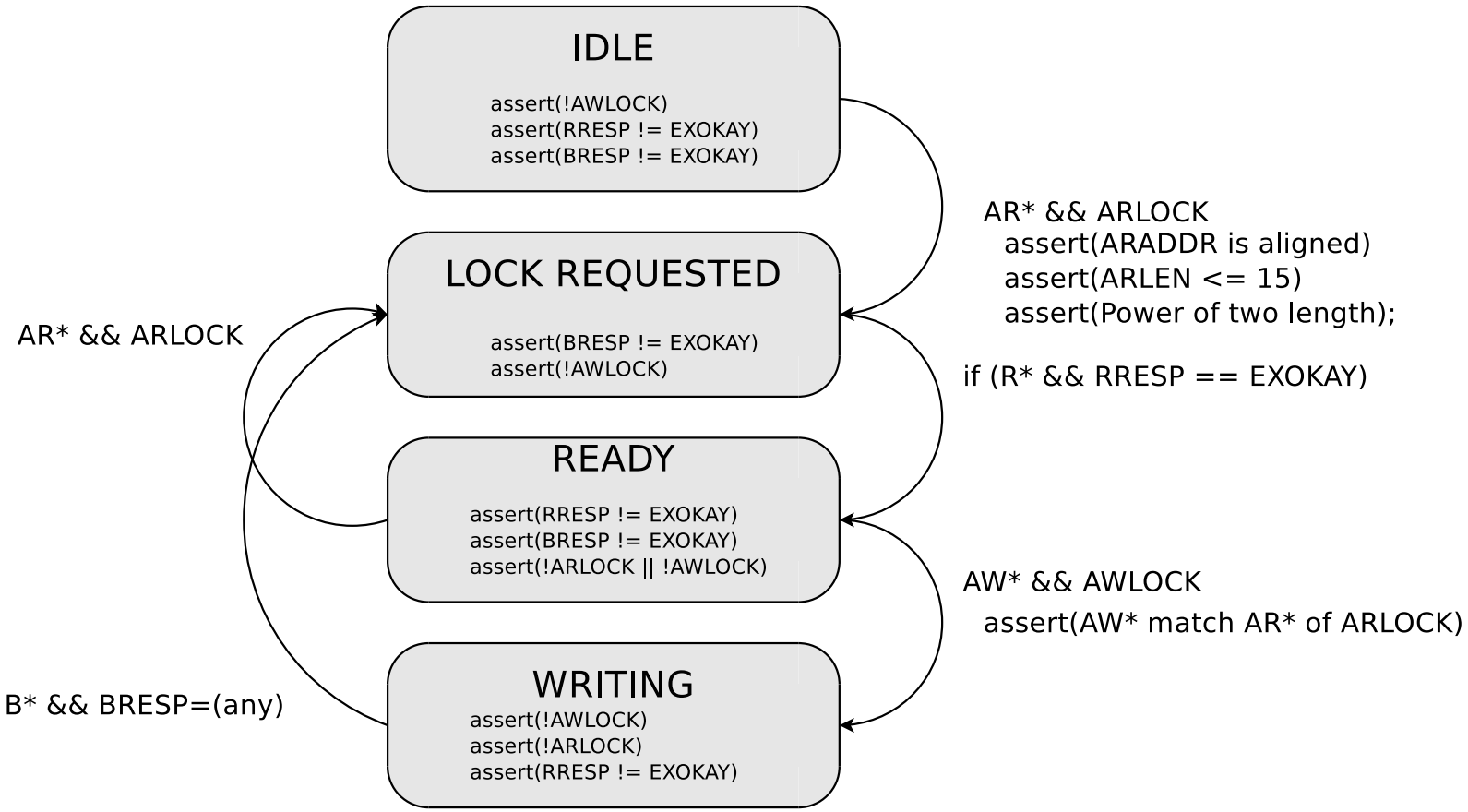


Exclusive Access Formal Verification follows an FSM



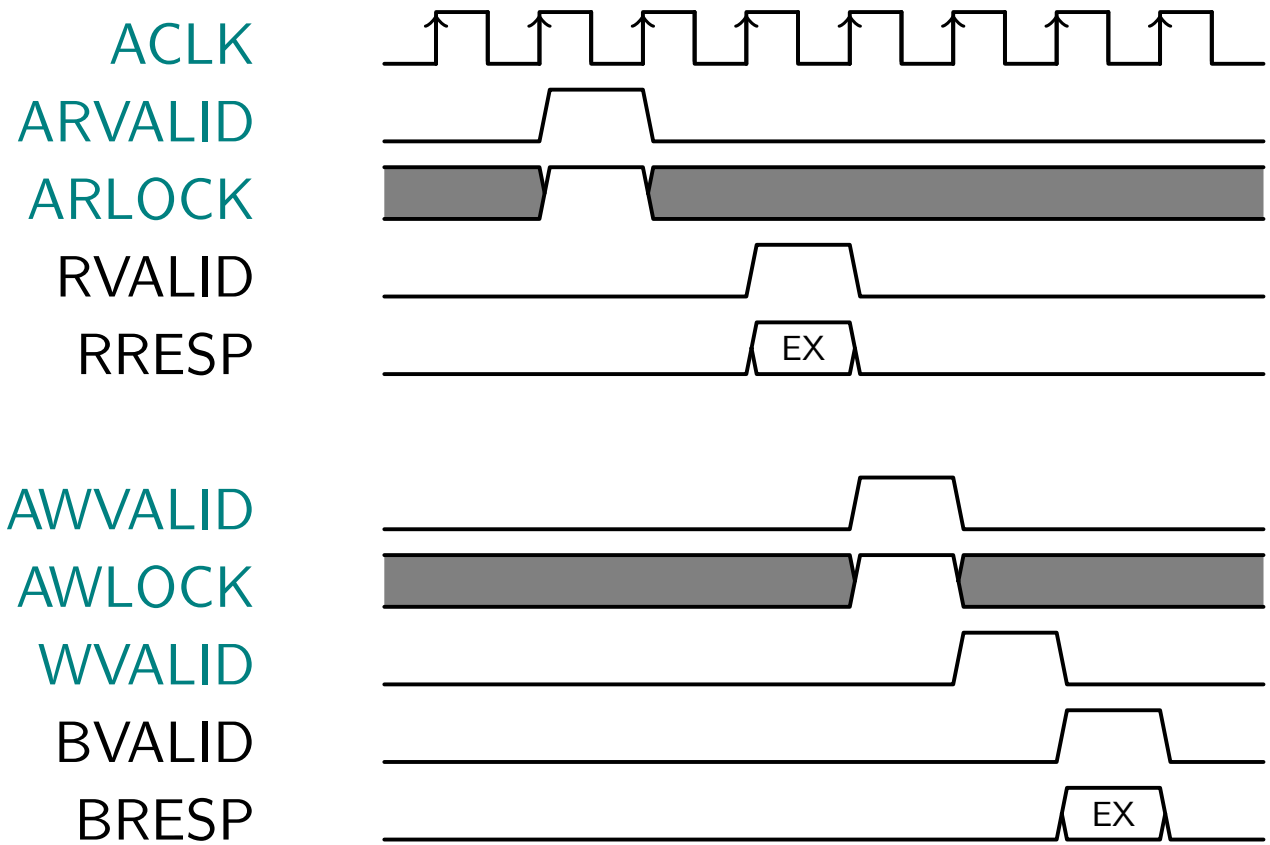


Assertions can then be tied to states



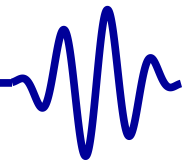
Example

Can now build and verify an example:



The ZipCPU now supports exclusive access

Surprise

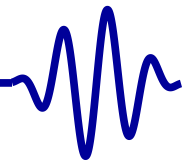


What happens on ID remapping?

1. Master #1 performs an exclusive access read, as ID #1
   - Interconnect remaps all IDs to ID #0
2. Master #2 now writes that address, as ID #2
3. Master #2 now requests an exclusive read
   - Slave grants exclusive request to ID #0
4. Master #1 performs an exclusive access write, as ID #1
   - Interconnect remaps this to ID #0
   - Slave thinks it's a request from the same master as before
   - *Protocol failure!*

New interconnect requirement: Prevent this from happening!

Surprise #2



The specification allows a slave to return `OKAY` on a read

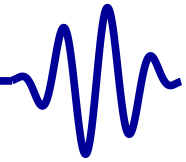
- The master *may* treat this as an error
  "The master can treat the `OKAY` response as an error indicating that the exclusive access is not supported."

If the master doesn't treat this as an error:

- The master (CPU) will be stuck in an endless loop

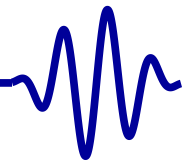
If a slave supports exclusive access:

- It *must* return `EXOKAY` on any exclusive access read request
- Even if it knows the subsequent write will fail



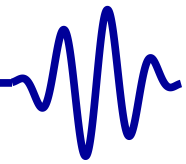
Beat Size

Rules



- `AWSIZE` rule:
  - Bounded by the bus size
  - Determines alignment
    - **if** `AWSIZE` $==$ 3'h0, always aligned
    - **if** `AWSIZE` $==$ 3'h1, aligned **if** `AWADDR`[0] $==$ 0
    - **if** `AWSIZE` $==$ 3'h2, aligned **if** `AWADDR`[1:0] $==$ 0
    - etc.
  - Provides the upper bound in WSTRB
- `AWADDR` rule:
  - The first word in any transfer need not be aligned
  - Any subsequent word must be aligned
  - Determines the minimum bit in WSTRB

Example



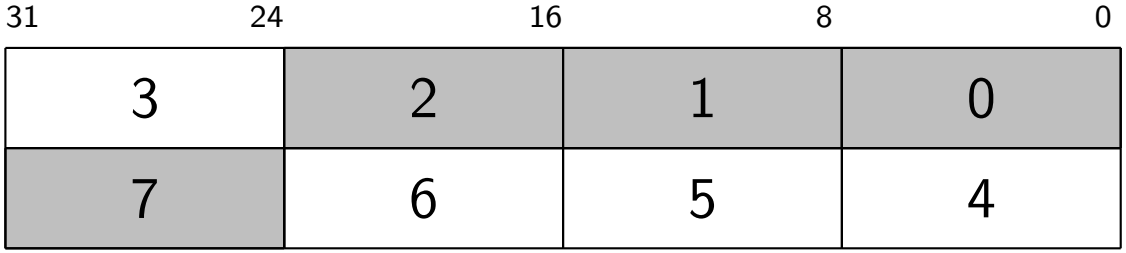
If the address is aligned, things are simple:

```
case(cpu_op) // Assuming a 32b bus
BYTE: begin
        AWSIZE<= 3'b0;
        WDATA <= cpu_data[7:0]<< 8*cpu_addr[1:0];
        WSTRB <= 4'b1 << cpu_addr[1:0];
        end
HALFWORD: begin
        AWSIZE<= 3'b1;
        WDATA <= cpu_data[15:0]<< 8*cpu_addr[1:0];
        WSTRB <= 4'b3 << cpu_addr[1:0];
        end
WORD: begin
        AWSIZE<= 3'b2;
        WDATA <= cpu_data[31:0];
        WSTRB <= 4'bf;
        end
endcase
```

Unaligned Word

What if the write data is split across two words?

- `AWADDR[1:0] == 2'b11`
- Write size is 32'bits



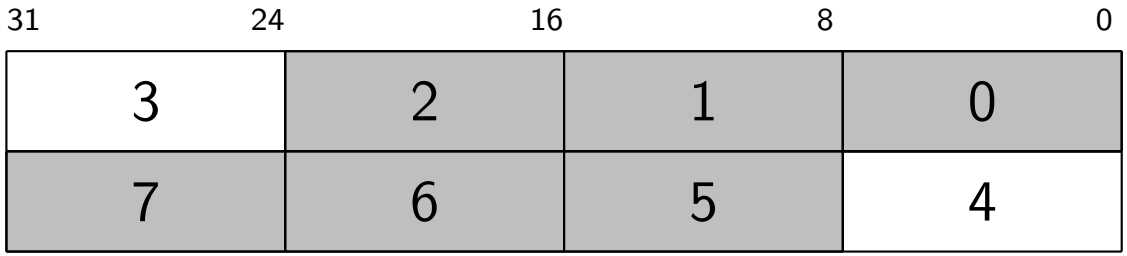
In this case,

- `WSIZE == 3'h2` (32b word), and you must take two beats

Unaligned Halfword

The logic works for halfwords as well (sometimes)

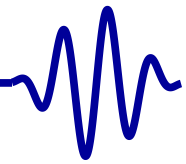
- `AWADDR[1:0] == 2'b11`
- Write size is 16'bits



In this case, either

- `WSIZE == 3'h1` (16b word), and you must take two beats, or
- `WSIZE == 3'h2` (32b word), and you still need two beats

Unaligned Halfword



For smaller word sizes, it's not so simple

```
case(cpu_op) // Assuming a 32b bus
HALFWORD: begin
        AWSIZE <= 3'b1;
        { nextd, WDATA } <= cpu_data[15:0]
                          << 8 * cpu_addr[1:0];
        { nexts, WSTRB } <= 4'b3 << cpu_addr[1:0];
        end
        // etc.
endcase
```

The obvious answer (above) doesn't work

Surprise

What if the write data is split across two half-words?

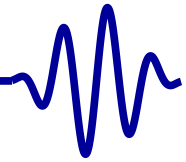
- `AWADDR[1:0] == 2'b01`
- Write size is 16'bits

31	24	16	8	0
3	2	1	0	
3	2	1	0	

In this case, either

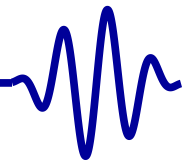
- `WSIZE == 3'h1` (halfword), and you take two beats, or
- `WSIZE == 3'h2` (32b word), and you take one beat
- `WSIZE == 3'h1` cannot have `WSTRB == 4'b0110`

Without the formal rules, I would've never noticed.



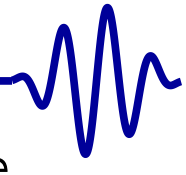
Simulation

Simulation



Why aren't these bugs caught in simulation?

AXI3 VIP



Customer: Verify your design. Here's an AXI3 VIP to use

- Supports issuing simulated read/write AXI commands
  - Like software: only one thread
  - One command at a time
  - One direction at a time

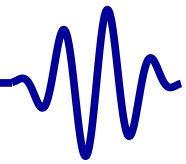
  Would never trigger bug in Xilinx's AXI Quad SPI

AXI3 VIP (Continued)

Customer: Verify your design. Here's an AXI3 VIP to use

- Doesn't support unaligned access
  - Automatically aligns requests given to it
- Won't issue `FIXED` or `WRAP` requests
  - A good MM2S or S2MM DMA will issue `FIXED` requests
  - I have nothing to test issue `WRAP` requests
- Doesn't support out of order write data
- Only supports randomized backpressure
  - 50% Probability of backpressure isn't going to be enough to trigger a bug in a design with a 13–deep pipeline
- With only one thread, it can't check exclusive access

Fixing Simulation



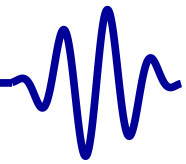
Need a simulation environment that:

1. Issues requests for multiple IDs
2. Issues both `WRAP` and `INCR` requests
3. Can generate sufficient backpressure

Note that we're still not testing exclusive access

- Exclusive access checking requires a simulation environment with ...
  1. 2+ CPUs
  2. Exclusive access capable memory
  3. Exclusive access capable interconnect

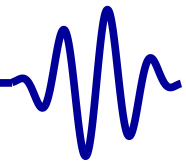
Constipated DMA



1. Issues many requests
2. Holds RREADY/BREADY low (whichever is under test)
3. Once ARREADY/AWREADY/WREADY become low, ...
   - for some (user defined number of cycles), ...
   - then release RREADY/BREADY
4. This verifies the operation
   - Verify the right number of returns
   - Verify the correct return values

This is a partial solution to the AXI simulation problem

Still Missing

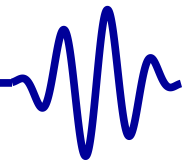


Still need a simulation environment that:

1. Issues requests for multiple IDs
2. Issues `WRAP` requests
3. Issues simultaneous read/write requests
4. Verifies exclusive access

Until then, these can easily be checked via formal methods

Conclusion



You have a design with . . .

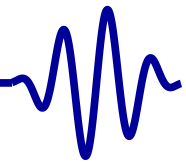
- Multiple address regions
- Each region has a separate pipeline depth

You need to verify it

Q: How do you guarantee compliance?

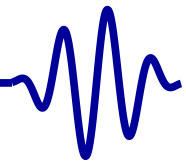
  - If you only ever issue requests of one region at a time?
  - If you never force sufficient backpressure?

A: You can't get there without formal methods



Questions?

References



	URL References
ZipCPU	https://github.com/ZipCPU/zipcpu/
Easy AXI-lite demo	https://zipcpu.com/blog/2020/03/08/easyaxil.html
Xilinx's AXI-lite	https://zipcpu.com/formal/2018/12/axilite.html
Xilinx's AXI full	https://zipcpu.com/formal/2019/05/13/axifull.html
AXI2AXILITE bridge	https://github.com/ZipCPU/wb2axip/blob/master/rtl/axi2axilite.v
Demo AXI full	https://github.com/ZipCPU/wb2axip/blob/master/rtl/demofull.v
MM2S DMA	https://github.com/ZipCPU/wb2axip/blob/master/rtl/aximm2s.v
S2MM DMA	https://github.com/ZipCPU/wb2axip/blob/master/rtl/axis2mm.v