

6.S965

Digital Systems Laboratory II

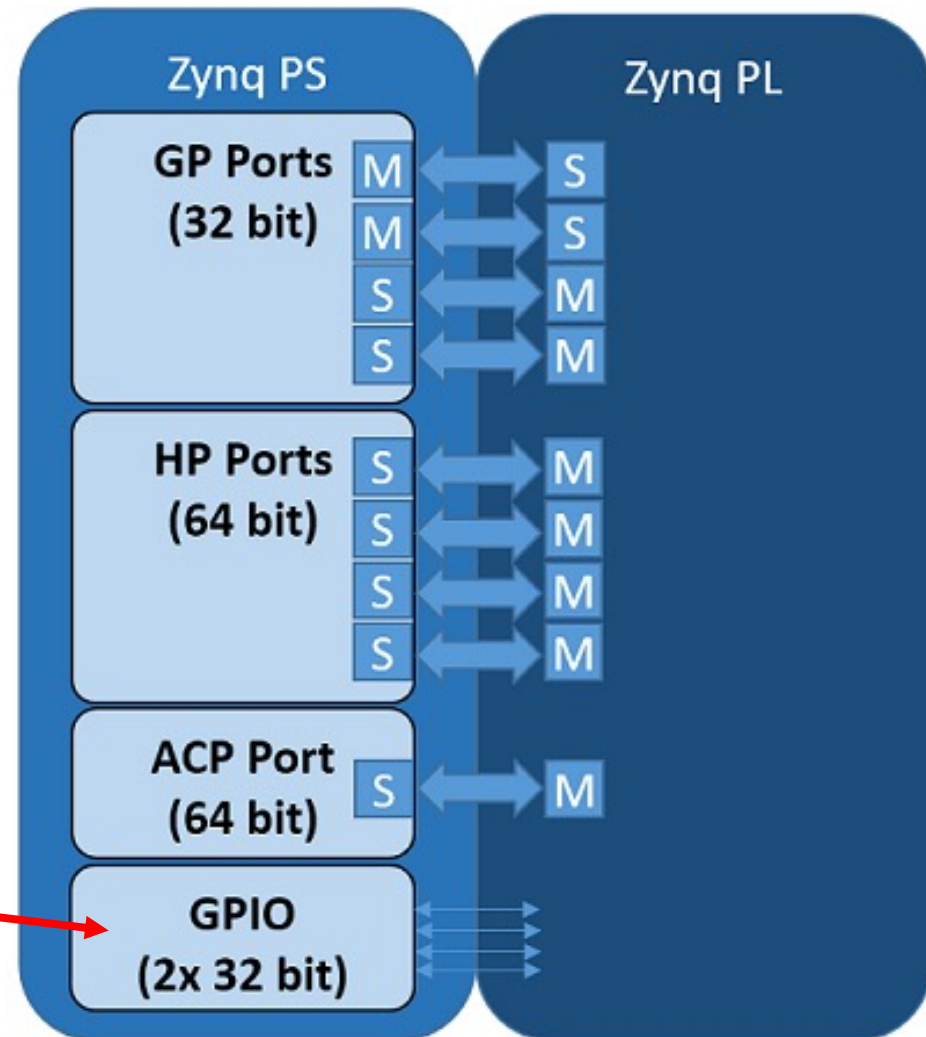
Lecture 6:

AXI

Interface Between PS and PL

- Four Ways to Transfer Data from the PS to the PL
 - 64 bits of GPIO
 - 4 GP AXI Ports
 - 4 HP AXI Ports
 - 1 ACP Port

Just talked about this



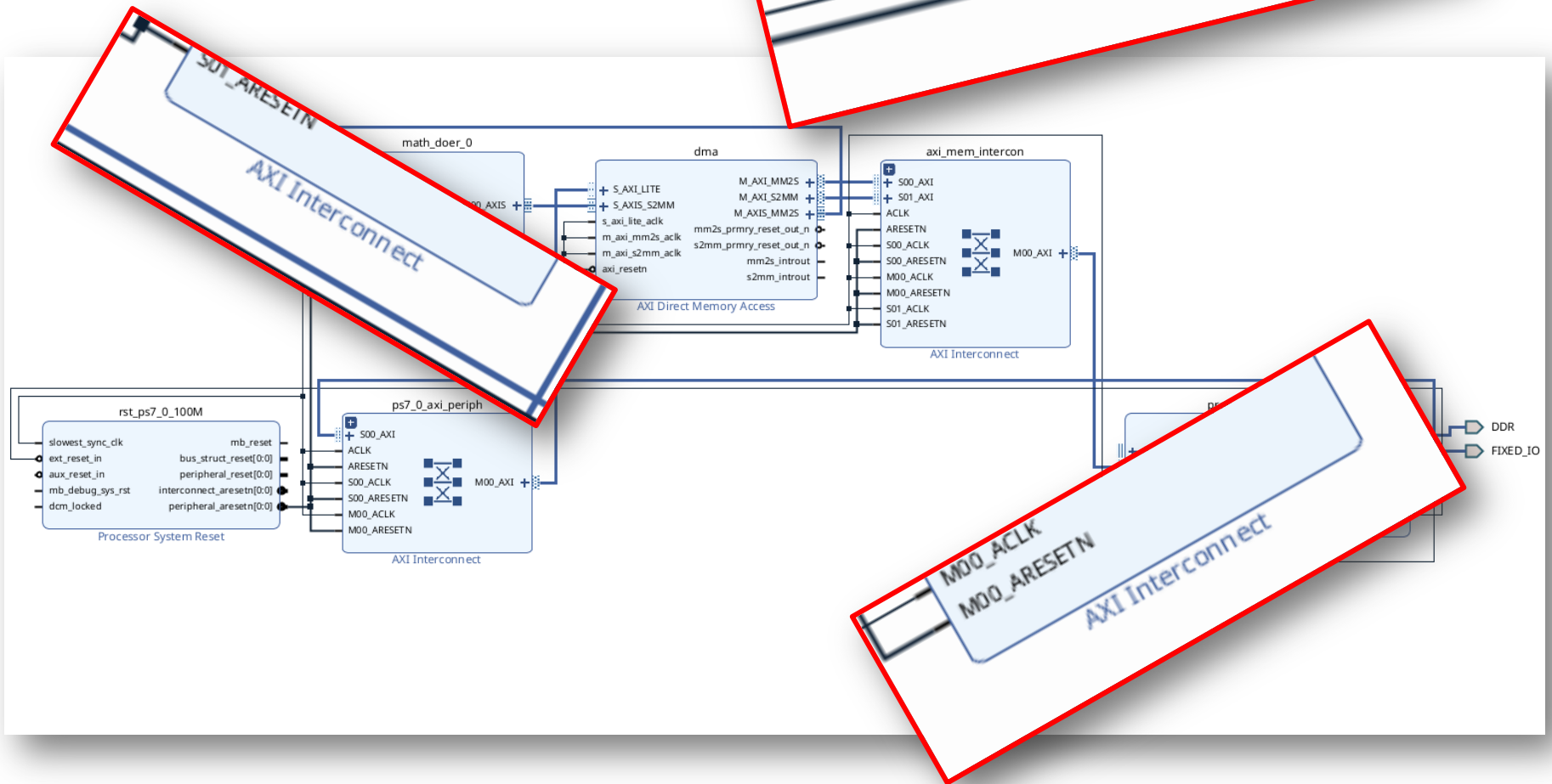
https://pynq.readthedocs.io/en/v2.3/overlay_design_methodology/pspl_interface.html

AXI

AXI Direct Memory Access

AXI Interconnect

M00_ACLK
M00_ARESETN
AXI Interconnect



- + S_AXI_LITE
- + S_AXIS_S2MM
- + S_AXIS_STS
- s_axi_lite_aclk
- m_axi_sg_aclk
- m_axi_mm2s_aclk
- m_axi_s2mm_aclk
- o axi_resetn

- M_AXI_MM2S +
- M_AXI_S2MM +
- M_AXIS_MM2S +

- M_AXI +
- M_AXIS_CNTRL +
- mm2s_prmry_reset_out_n o
- mm2s_cntrl_reset_out_n o
- s2mm_prmry_reset_out_n o
- s2mm_sts_reset_out_n o
- mm2s_introut -
- s2mm_introut -

ZYNQ7

Docu

Page

Zynq B

PS-PL C

Periphe

MIO Co

Clock C

DDR Co

SMC Ti

Interru



report

9/22/25

6S965 Fall 2025

cel

Search:

AXI4-Stream Data Width Converter

AXI4-Stream Interconnect

AXI4-Stream Protocol Checker

AXI4-Stream Register Slice

AXI4-Stream Subset Converter

AXI4-Stream Switch

AXI4-Stream to Video Out

AXI4-Stream Verification IP

AXI AHBLite Bridge

AXI APB Bridge

AXI BRAM Controller

ENTER to select, ESC to cancel, Ctrl+Q for IP details

Broken AXI-lite Design in



Report

M_AXIS_DATA —
is_data_tdata[31:0] ▶
m_axis_data_tlast ▶
m_axis_data_tready ◀
m_axis_data_tvalid ▶
event_frame_started
event_tlast_unexpected
event_tlast_missing
ent_status_channel_halt
nt_data_in_channel_halt
data_out_channel_halt

form

Advanced Microcontroller Bus Architecture (AMBA)

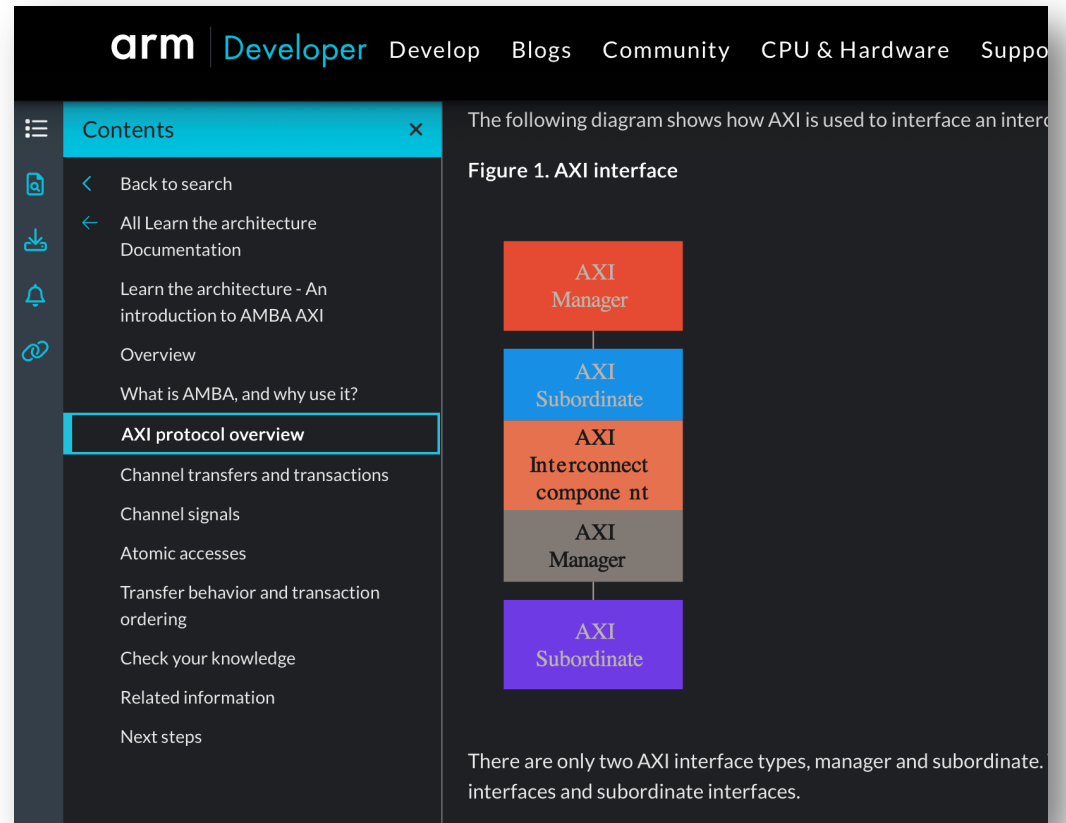
- Version 1 released in 1996 by ARM
- 2003 saw release of **Advanced eXtensible Interface (AXI3)**
- 2011 saw release of **AXI4**
- There are no royalties affiliated with AMBA/AXI so they're used a lot.
- It is a general, flexible, and relatively free* communication protocol for development of SOCs!!!

Master/Slave Terminology

- AXI historically used “Master/Slave” terminology to describe different parties in the data transactions
- I (and others) have been a big fan of moving away from this terminology.
- For SPI, for example, instead of MOSI/MISO, do COPI/CIPO (controller/peripheral), etc...
- The question is what to switch to? I used Main/Secondary for a while to support backwards compatibility

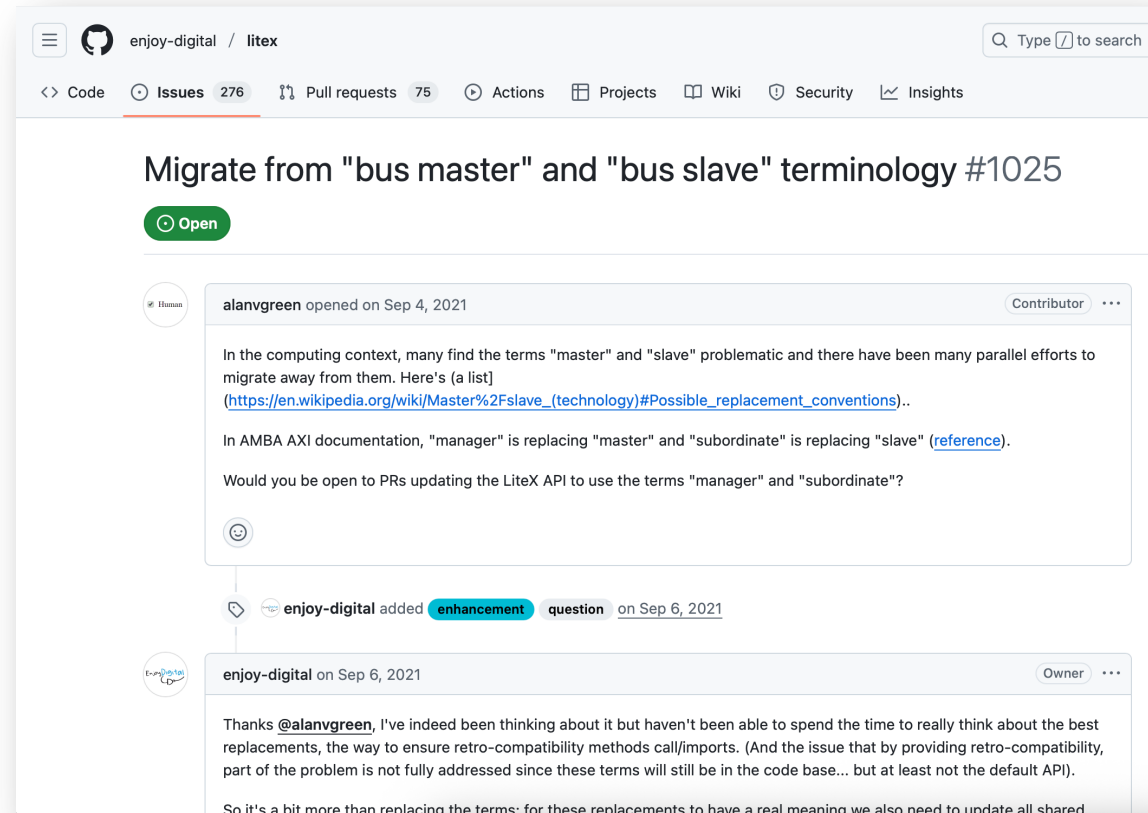
ARM (owner of AXI) updated in 2021/2-ish?

- Slowly rolled out
- Docs for the most part are now updated to use **Manager** and **Subordinate** terminology
- Probably good one to go with, still get to keep the M and S.



A lot of frameworks updated

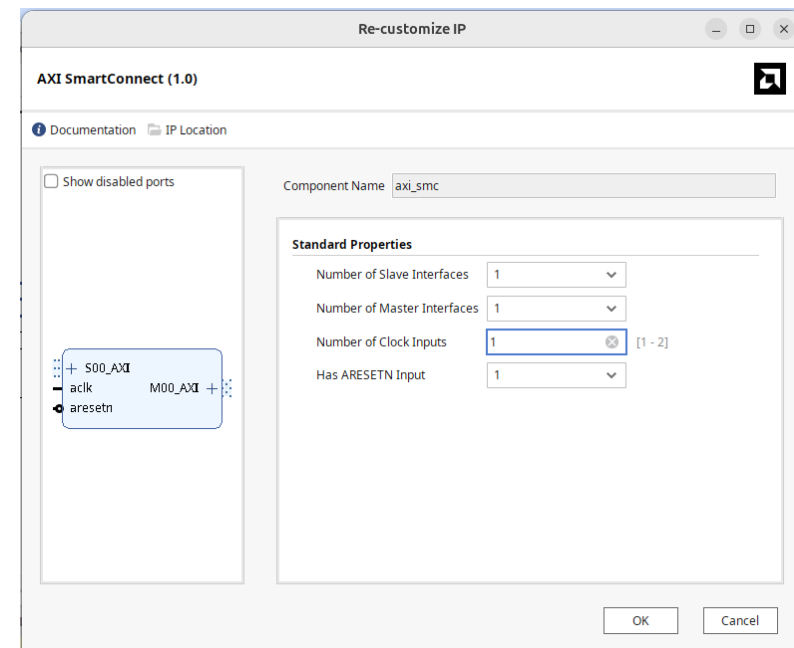
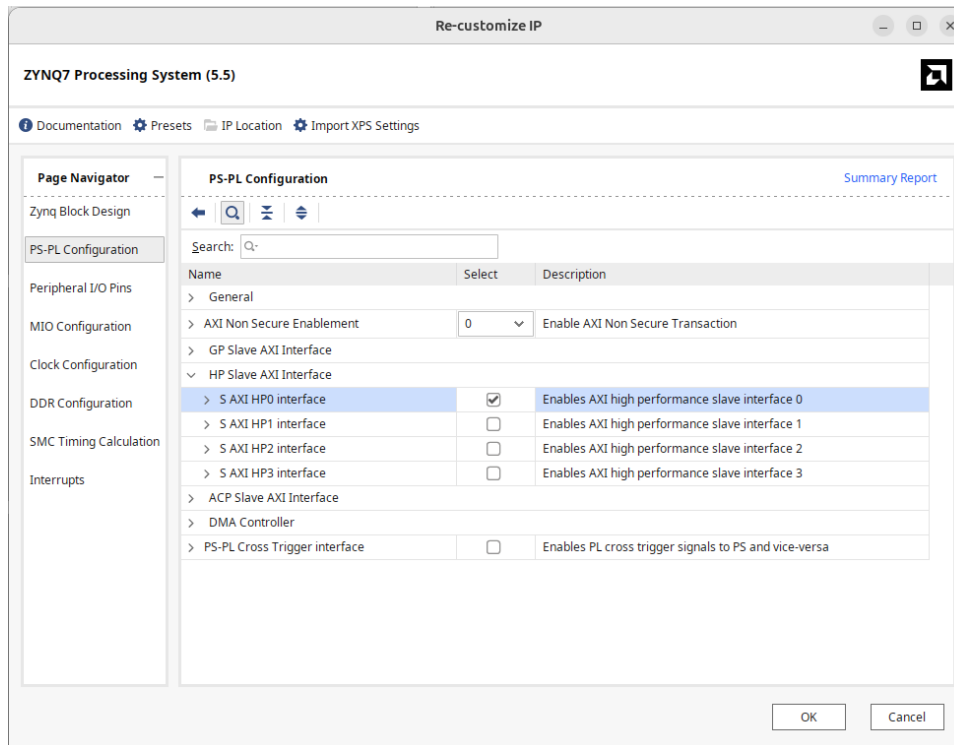
- Civil Discussion in the litex thread
- Others followed the ARM update



<https://github.com/enjoy-digital/litex>

Vivado hasn't updated

- Maybe we're asking too much of the toolchain for right now



9/22/25

From Vivado 2025.1 with all the updates

63965 Fall 2025

11

Master/Slave Terminology

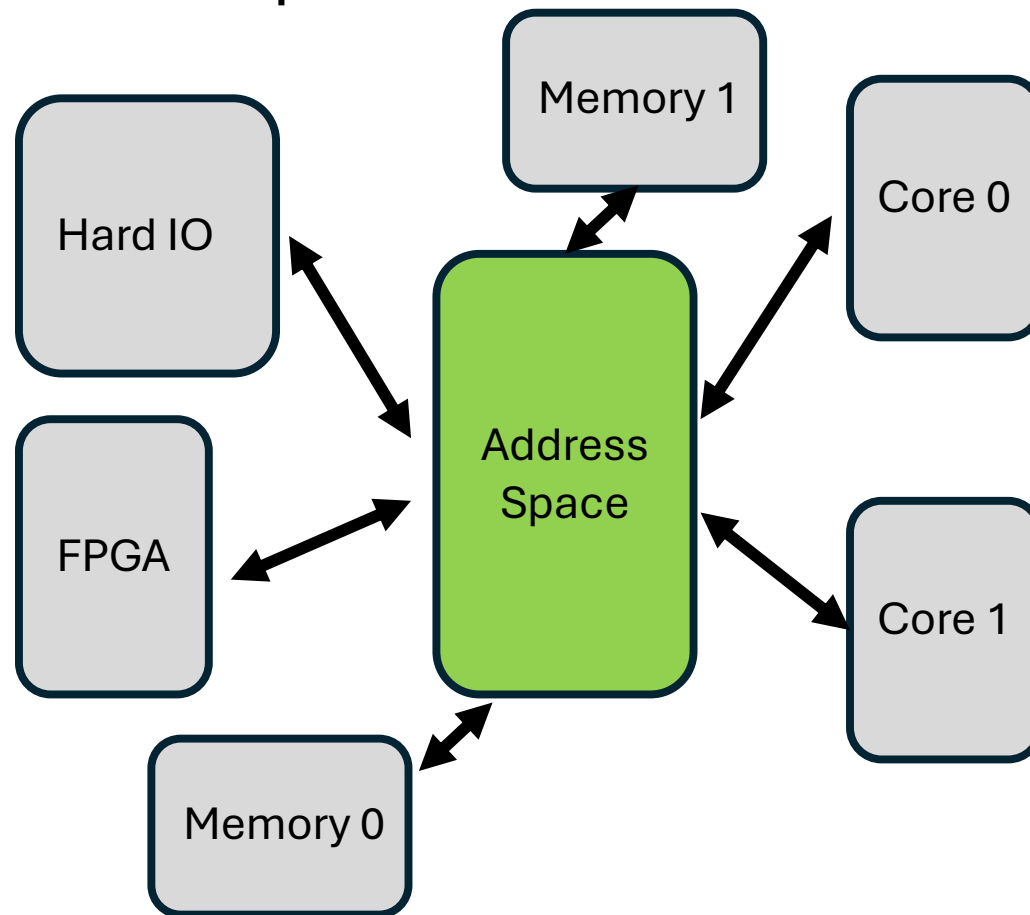
- **all** of the AMD/Xilinx, use Master/Slave and **everything** has that M's and S's prepended, appended, etc..
- I'm going to just use their nomenclature so we don't have to constantly be mapping between alternate names and sometimes also **manager-subordinate** when possible or maybe just "**M**" and "**S**" when possible.

Advanced Microcontroller Bus Architecture (AMBA)

- Version 1 released in 1996 by ARM
- 2003 saw release of **Advanced eXtensible Interface (AXI3)**
- 2011 saw release of **AXI4**
- There are no royalties affiliated with AMBA/AXI so they're used a lot.
- It is a general, flexible, and relatively free* communication protocol for development of SOCs!!!

Memory

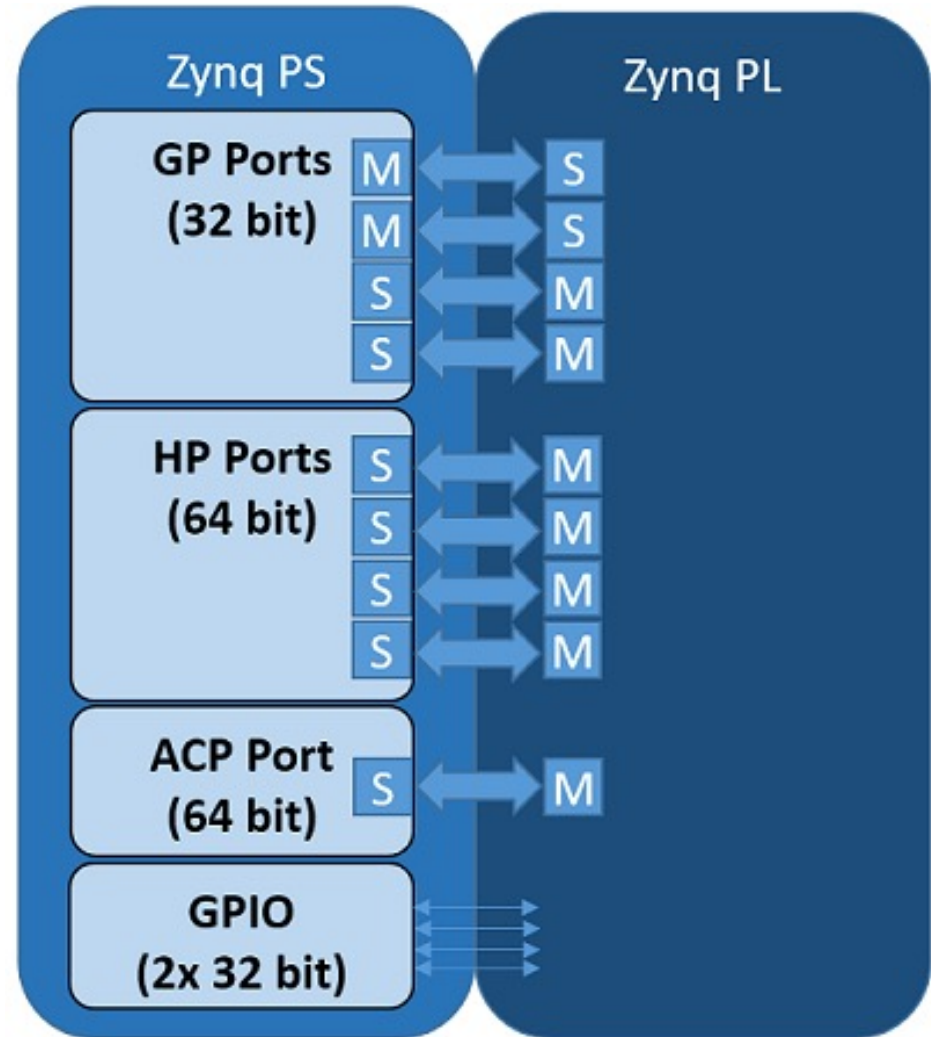
- Critical in maintaining the illusion of unified memory/address space



*Not really memory per say...

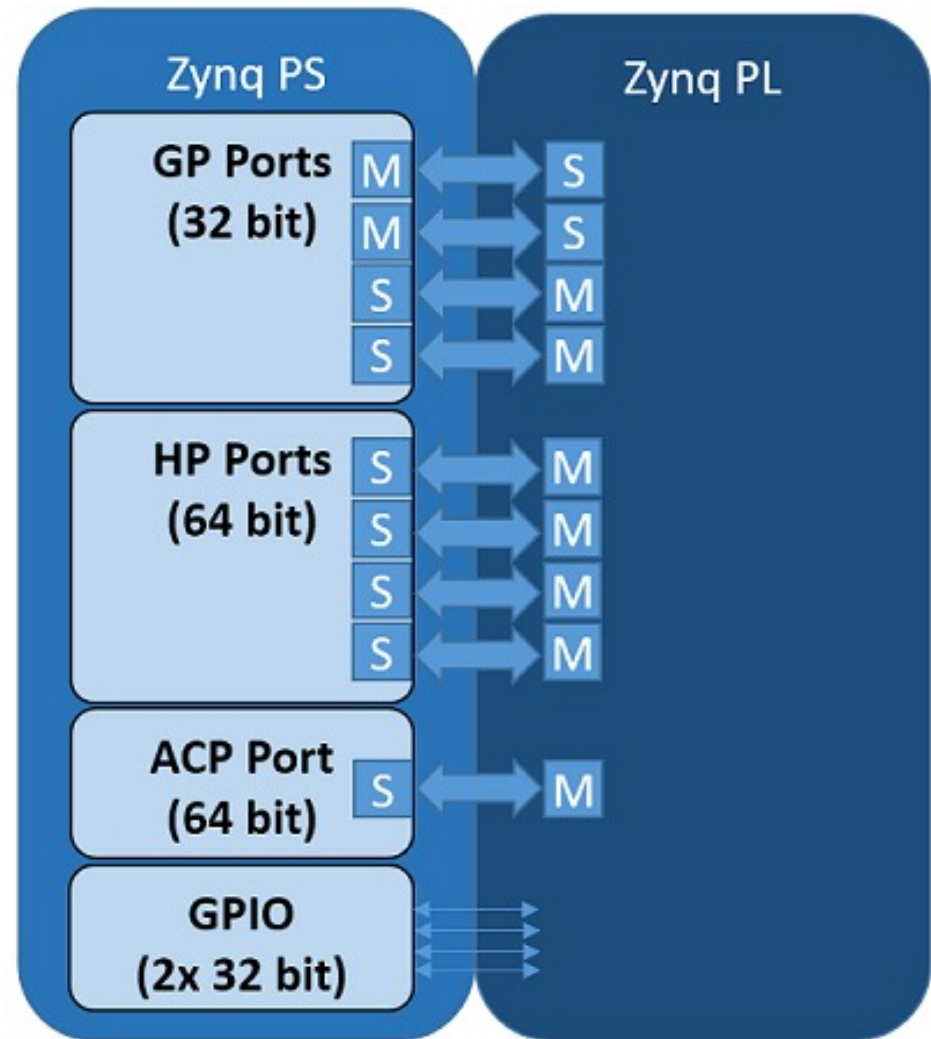
AXI Ports

- Parallel Busses of two different flavors that allow us to pretty quickly transfer data between the Processing System and the FPGA section using shared registers and some other stuff

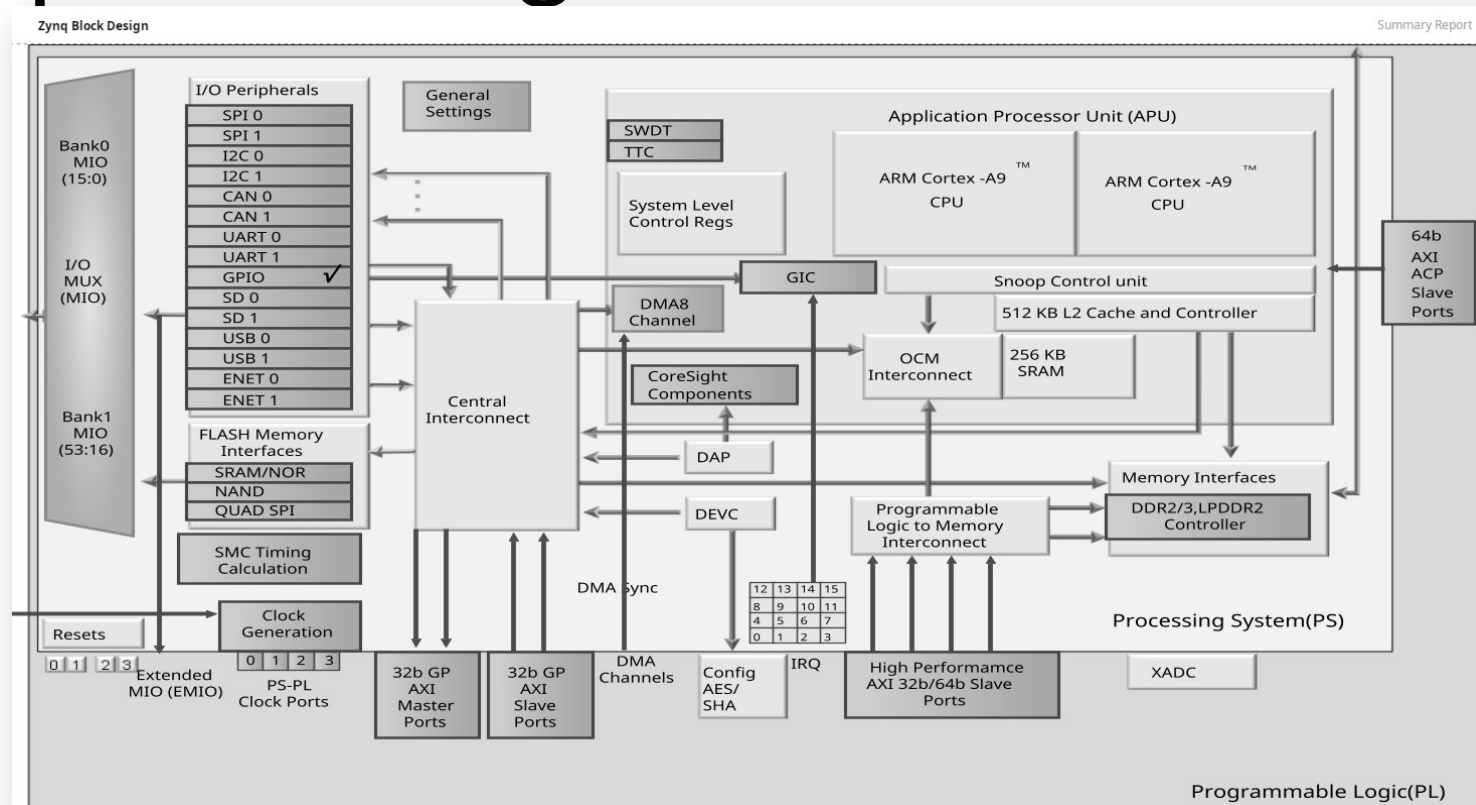


ACP Port

- Accelerator Coherency Port
- 64-bit wide bus that can transfer data from very quickly from PL fabric



Zynq Block Diagram



- A large fraction of the arrows in this diagram represent AMBA/AXI4 specification flows of information!

Three General Flavors of AXI4

- **AXI4 (Full AXI):** For memory-mapped links. Provides highest performance.
 1. Address is supplied
 2. Then a data burst transfer of up to 256 data words
- **AXI4 Lite:** A memory-mapped simplified link supporting only one data transfer per connection (no bursts). (also restricted to 32 bit addr/data)
 1. Address is supplied
 2. One data transfer
- **AXI4 Stream:** Meant high-speed streaming data
 - Can do burst transfers of unrestricted size
 - *No addressing*
 - Meant to stream data from one device to another quickly on its own direct connection

From the Zynq Book

Memory Map...

- Memory mapped means an address is specified within the transaction by the master (read or write). This corresponds to an address in the system memory space.
- For **AXI4-Lite**, which supports a single data transfer per transaction, data is then written to, or read from, the specified address
- For **Full-AXI4** sending a burst, the address specified is for the first data word to be transferred, and the slave must then calculate the addresses for the data words that follow.
- **AXI-Stream** has no addressing so no memory mapping

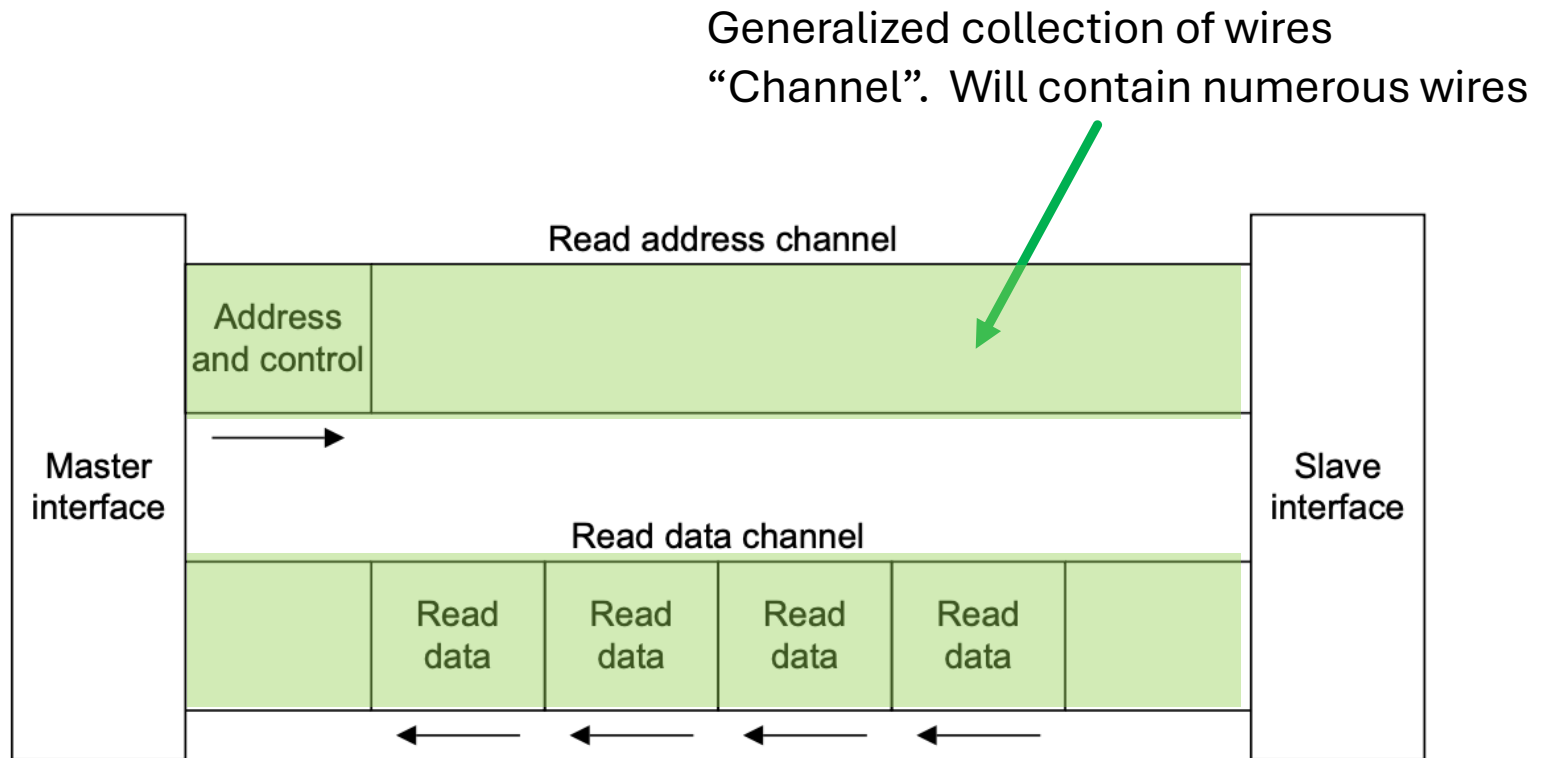
Burst vs. Single?

- A burst of data means you specify starting conditions and then send a bunch of ordered data:
 - Benefit is better overhead!
 - Downside is usually more complicated infrastructure and transactional rules
- A single data transfer is how we give data to our SPI module. One set of configurations, one transfer

AXI Idea

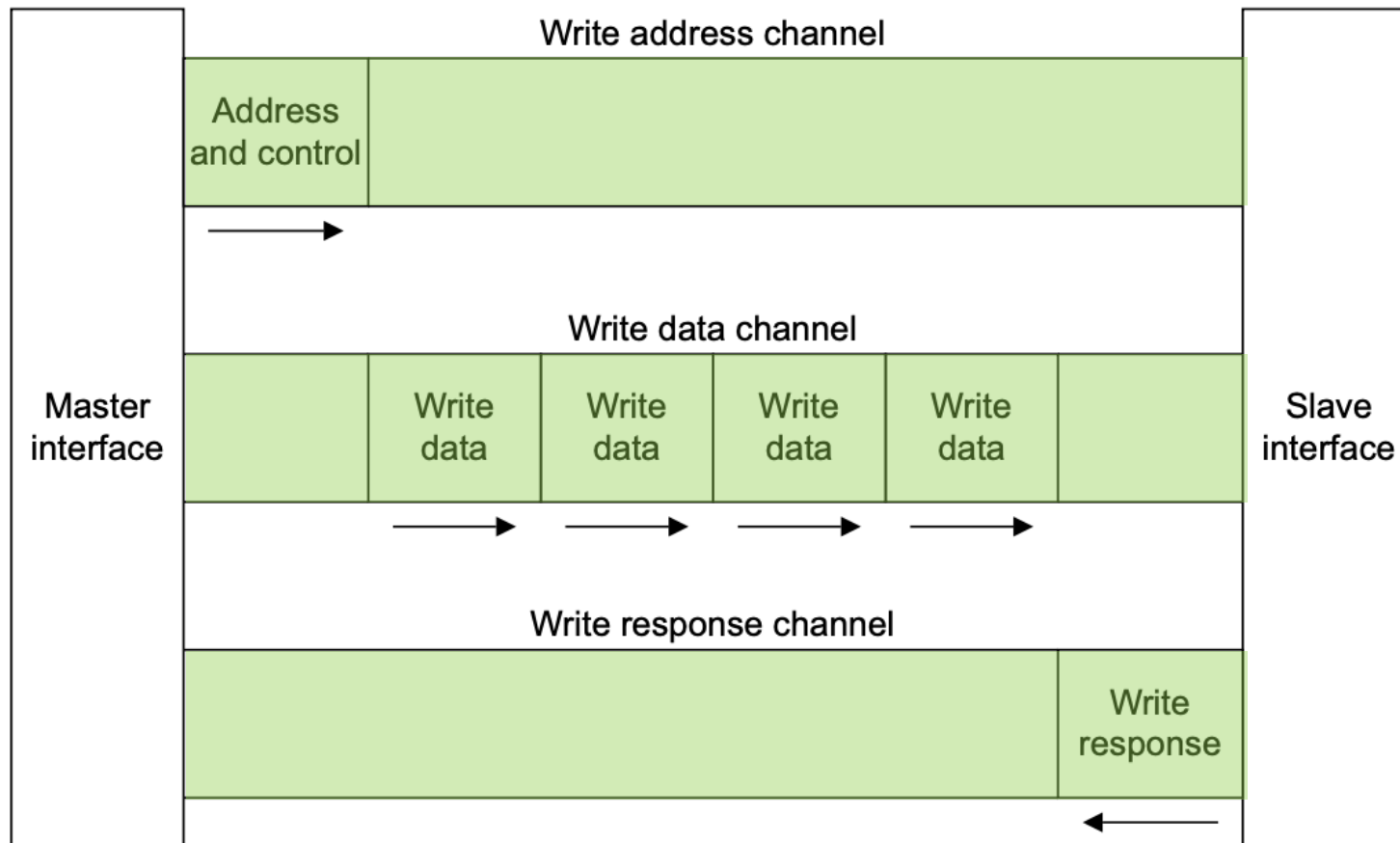
- Communication between two devices (Master and Slave, Manager and Subordinate) is carried out over multiple assigned “channels”
- Each channel has its own collection of wires which convey data, signals, etc.
- The channels can work somewhat independently, however in practice what one channel does is often the result of what a different one did previously
- Five Types of Channels (may have all or a subset):
 - Read Address: “AR” channel
 - Read Data: “R” channel
 - Write Address: “AW” channel
 - Write Data: “W” channel
 - Write Response: “B” channel

Read Wiring



Master initiates communication, Slave responds

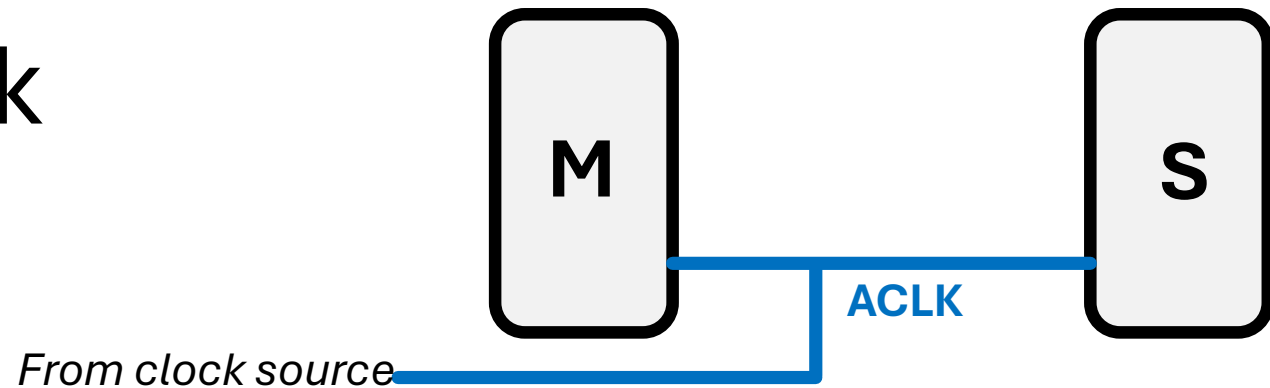
Write Wiring



Within Each Channel are wires:

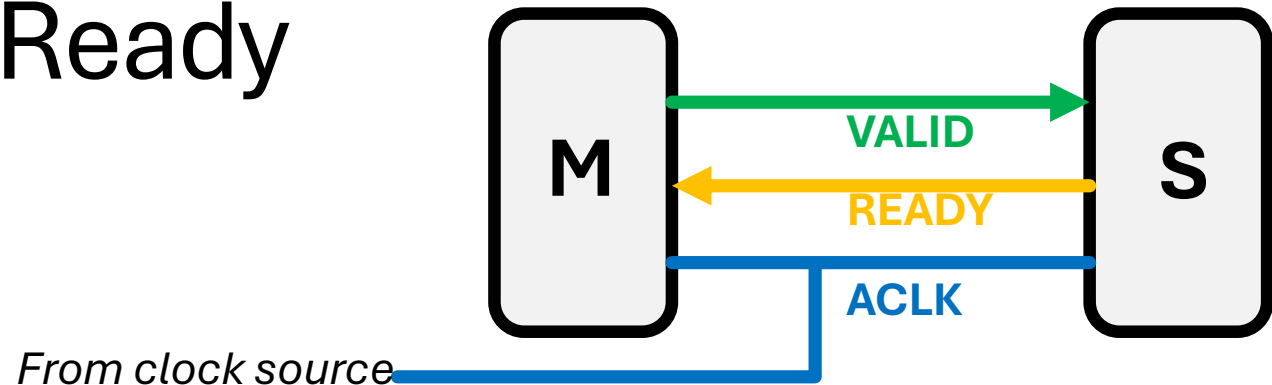
- These wires serve specific purposes.
- Some are universal to all channels, and others are specific

AXI Clock



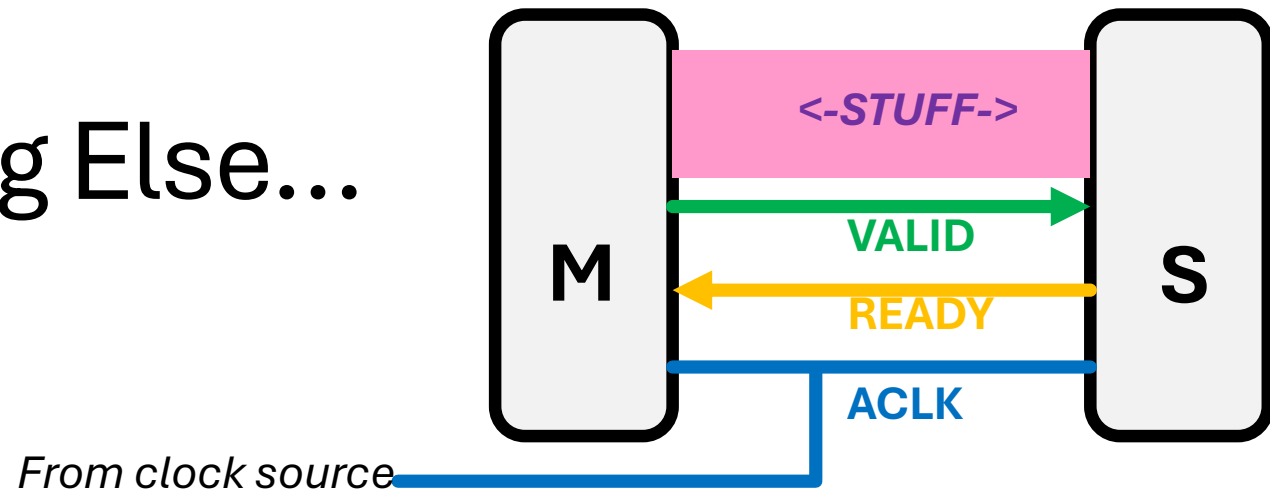
- Everything in system will run off of AXI clock usually called **ACLK** in documentation
- No combinatorial paths between inputs and outputs. Everything must be registered.
- All signals are sampled **on rising edge**
- AXI modules should also have Reset pins. AXI work ACTIVE LOW so the Reset pin is usually called **ARSTn** or **ARESETn**

Valid and Ready



- All of AXI uses the same handshake procedure:
- The source of a data generates a **VALID** signal
- The destination generates a **READY** signal
- Transfer of data only occurs when both are high on a rising edge of ACLK
- Both Master and Slave Devices can therefore control the flow of their data as needed

Everything Else...



- Everything else is information and depends on what is needed in situation. Could be:
 - Address
 - Data
 - Other specialized wires like:
 - STRB (used to specify which bytes in current data step are valid, sent by Master along with data payload to Slave)
 - RESP (sort of like a status)
 - LAST (sent to indicate the final data clock cycle of data in a burst)

Each channel has its own subset of “everything else” that goes along with those core signals shared by all

For example, the Write Data Channel (“W” channel)

Payload

Signal	Source	Description
WID	Master	Write ID tag. This signal is the ID tag of the write data transfer. Supported only in AXI3. See Transaction ID on page A5-77 .
WDATA	Master	Write data.
WSTRB	Master	Write strobes. This signal indicates which byte lanes hold valid data. There is one write strobe bit for each eight bits of the write data bus. See Write strobes on page A3-49 .
WLAST	Master	Write last. This signal indicates the last transfer in a write burst. See Write data channel on page A3-39 .
WUSER	Master	User signal. Optional User-defined signal in the write data channel. Supported only in AXI4. See User-defined signaling on page A8-100 .

CORE

WVALID	Master	Write valid. This signal indicates that valid write data and strobes are available. See Channel handshake signals on page A3-38 .
WREADY	Slave	Write ready. This signal indicates that the slave can accept the write data. See Channel handshake signals on page A3-38 .

Supplemental
Stuff

The Read Data Channel:

Table A2-6 Read data channel signals

Signal	Source	Description
RID	Slave	Read ID tag. This signal is the identification tag for the read data group of signals generated by the slave. See Transaction ID on page A5-77 .
RDATA	Slave	Read data.
RRESP	Slave	Read response. This signal indicates the status of the read transfer. See Read and write response structure on page A3-54 .
RLAST	Slave	Read last. This signal indicates the last transfer in a read burst. See Read data channel on page A3-39 .
RUSER	Slave	User signal. Optional User-defined signal in the read data channel. Supported only in AXI4. See User-defined signaling on page A8-100 .
RVALID	Slave	Read valid. This signal indicates that the channel is signaling the required read data. See Channel handshake signals on page A3-38 .
RREADY	Master	Read ready. This signal indicates that the master can accept the read data and response information. See Channel handshake signals on page A3-38 .

Payload

CORE

Supplemental Stuff

Read Address Chanel

Table A2-5 Read address channel signals

Payload

Signal	Source	Description
ARID	Master	Read address ID. This signal is the identification tag for the read address group of signals. See Transaction ID on page A5-77 .
ARADDR	Master	Read address. The read address gives the address of the first transfer in a read burst transaction. See Address structure on page A3-44 .
ARLEN	Master	Burst length. This signal indicates the exact number of transfers in a burst. This changes between AXI3 and AXI4. See Burst length on page A3-44 .
ARSIZE	Master	Burst size. This signal indicates the size of each transfer in the burst. See Burst size on page A3-45 .
ARBURST	Master	Burst type. The burst type and the size information determine how the address for each transfer within the burst is calculated. See Burst type on page A3-45 .
ARLOCK	Master	Lock type. This signal provides additional information about the atomic characteristics of the transfer. This changes between AXI3 and AXI4. See Locked accesses on page A7-95 .
ARCACHE	Master	Memory type. This signal indicates how transactions are required to progress through a system. See Memory types on page A4-65 .
ARPROT	Master	Protection type. This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access. See Access permissions on page A4-71 .
ARQOS	Master	<i>Quality of Service</i> , QoS. QoS identifier sent for each read transaction. Implemented only in AXI4. See QoS signaling on page A8-98 .
ARREGION	Master	Region identifier. Permits a single physical interface on a slave to be used for multiple logical interfaces. Implemented only in AXI4. See Multiple region signaling on page A8-99 .
ARUSER	Master	User signal. Optional User-defined signal in the read address channel. Supported only in AXI4. See User defined signaling on page A8-100 .
ARVALID	Master	Read address valid. This signal indicates that the channel is signaling valid read address and control information. See Channel handshake signals on page A3-38 .
ARREADY	Slave	Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals. See Channel handshake signals on page A3-38 .

CORE

Write Response “B” channel

Table A2-4 Write response channel signals

	Signal	Source	Description
	BID	Slave	Response ID tag. This signal is the ID tag of the write response. See <i>Transaction ID</i> on page A5-77.
Payload	BRESP	Slave	Write response. This signal indicates the status of the write transaction. See <i>Read and write response structure</i> on page A3-54.
	BUSER	Slave	User signal. Optional User-defined signal in the write response channel. Supported only in AXI4. See <i>User-defined signaling</i> on page A8-100.
CORE	BVALID	Slave	Write response valid. This signal indicates that the channel is signaling a valid write response. See <i>Channel handshake signals</i> on page A3-38.
	BREADY	Master	Response ready. This signal indicates that the master can accept a write response. See <i>Channel handshake signals</i> on page A3-38.

Write Address Channel

Table A2-2 Write address channel signals

Payload

Signal	Source	Description
AWID	Master	Write address ID. This signal is the identification tag for the write address group of signals. See Transaction ID on page A5-77 .
AWADDR	Master	Write address. The write address gives the address of the first transfer in a write burst transaction. See Address structure on page A3-44 .
AWLEN	Master	Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address. This changes between AXI3 and AXI4. See Burst length on page A3-44 .
AWSIZE	Master	Burst size. This signal indicates the size of each transfer in the burst. See Burst size on page A3-45 .
AWBURST	Master	Burst type. The burst type and the size information, determine how the address for each transfer within the burst is calculated. See Burst type on page A3-45 .
AWLOCK	Master	Lock type. Provides additional information about the atomic characteristics of the transfer. This changes between AXI3 and AXI4. See Locked accesses on page A7-95 .
AWCACHE	Master	Memory type. This signal indicates how transactions are required to progress through a system. See Memory types on page A4-65 .
AWPROT	Master	Protection type. This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access. See Access permissions on page A4-71 .
AWQOS	Master	<i>Quality of Service</i> , QoS. The QoS identifier sent for each write transaction. Implemented only in AXI4. See QoS signaling on page A8-98 .
AWREGION	Master	Region identifier. Permits a single physical interface on a slave to be used for multiple logical interfaces. Implemented only in AXI4. See Multiple region signaling on page A8-99 .
AWUSER	Master	User signal. Optional User-defined signal in the write address channel. Supported only in AXI4. See User-defined signaling on page A8-100 .
AWVALID	Master	Write address valid. This signal indicates that the channel is signaling valid write address and control information. See Channel handshake signals on page A3-38 .
AWREADY	Slave	Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals. See Channel handshake signals on page A3-38 .

CORE

Generalized Transaction

- All Channel Interactions follow same high-level structure
- Data is handed off IF AND ONLY IF VALID and READY are high on the rising edge of the clock
- If that happens, both parties must realize that data transfer has happened

Keep in mind this
could be 64 parallel
wires of 1's and 0's of
info or 8 bytes for
example...
Or it could be
something else

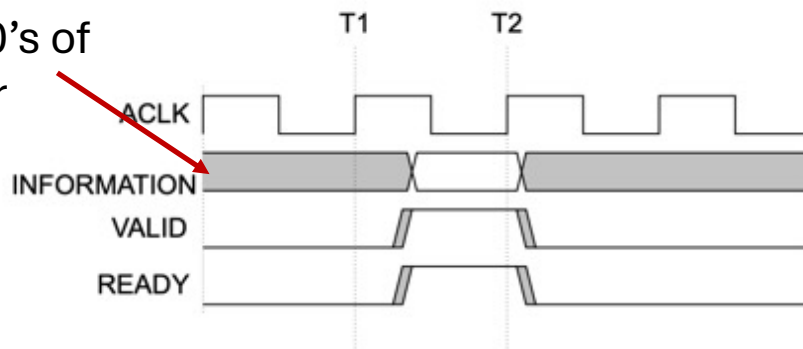


Figure A3-4 VALID with READY handshake

VALID then READY

- Valid can be high first
- Then ready can show up later
- Only when both are high is data exchanged

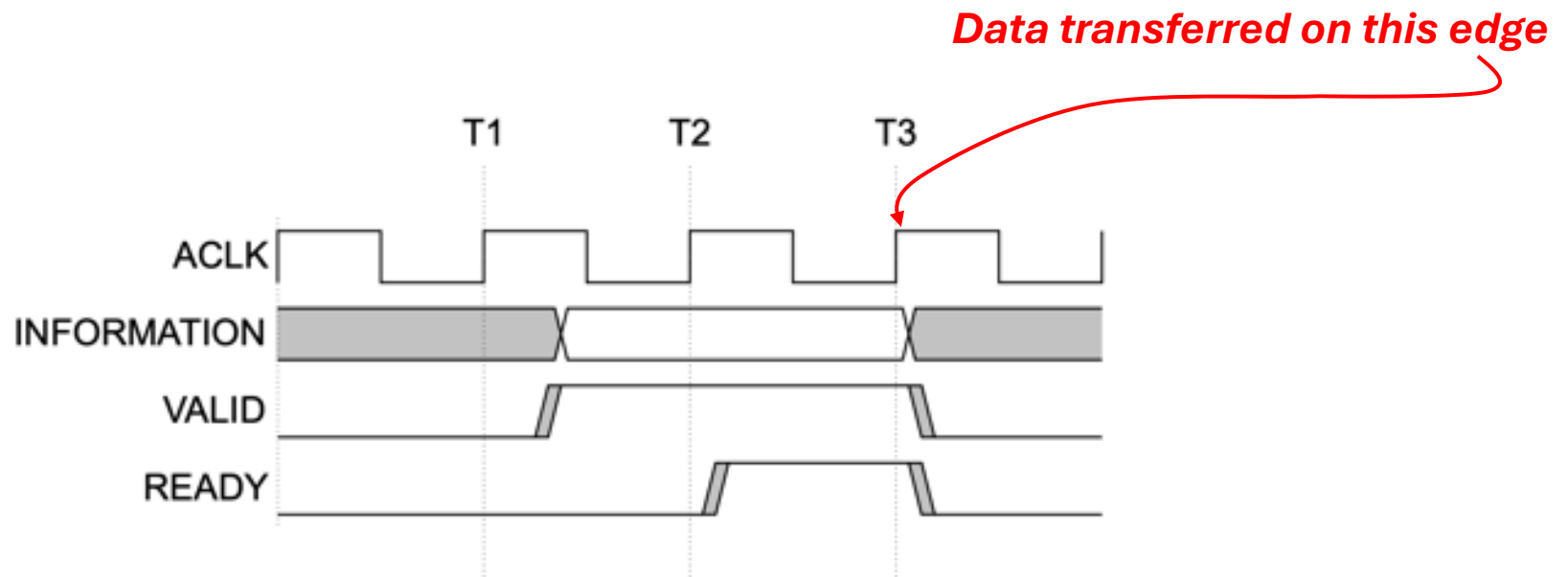


Figure A3-2 VALID before READY handshake

READY then VALID

- Ready can be high first
- Then Valid can show up later
- Only when both are high is data exchanged

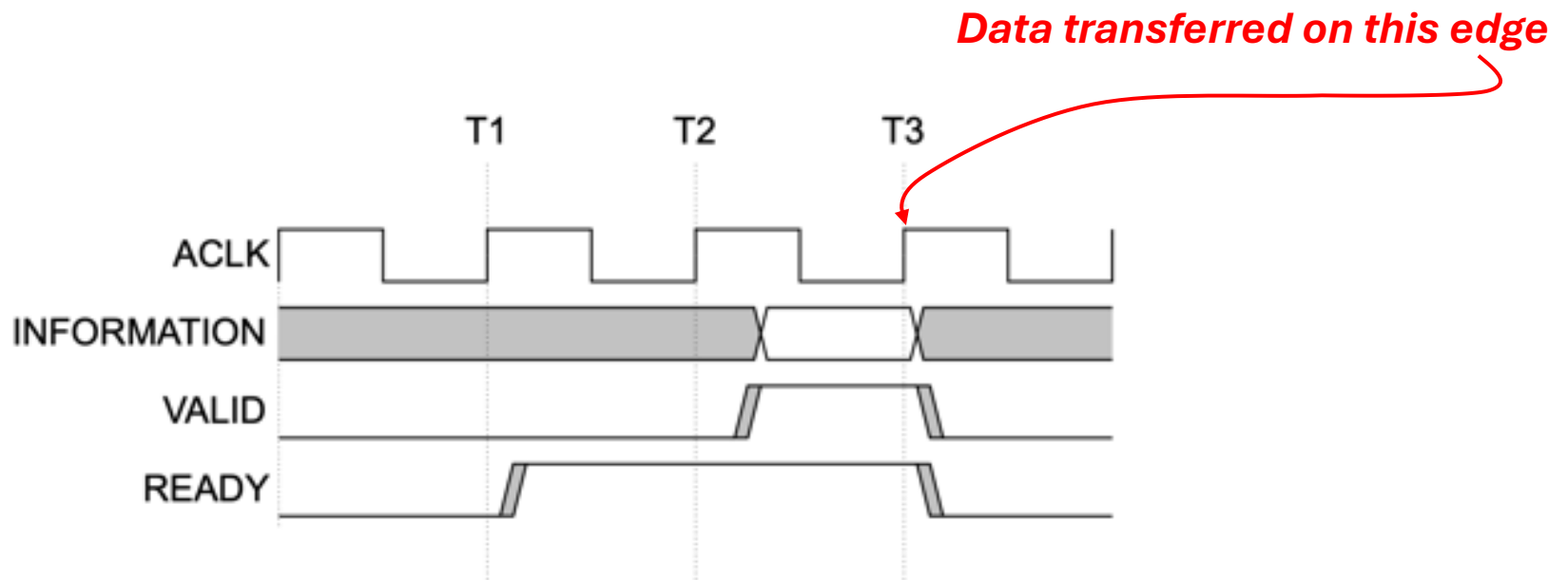


Figure A3-3 READY before VALID handshake

READY WITH VALID

- Ready and Valid come high at the same time
- Totally allowed
- Data is exchanged on that clock edge

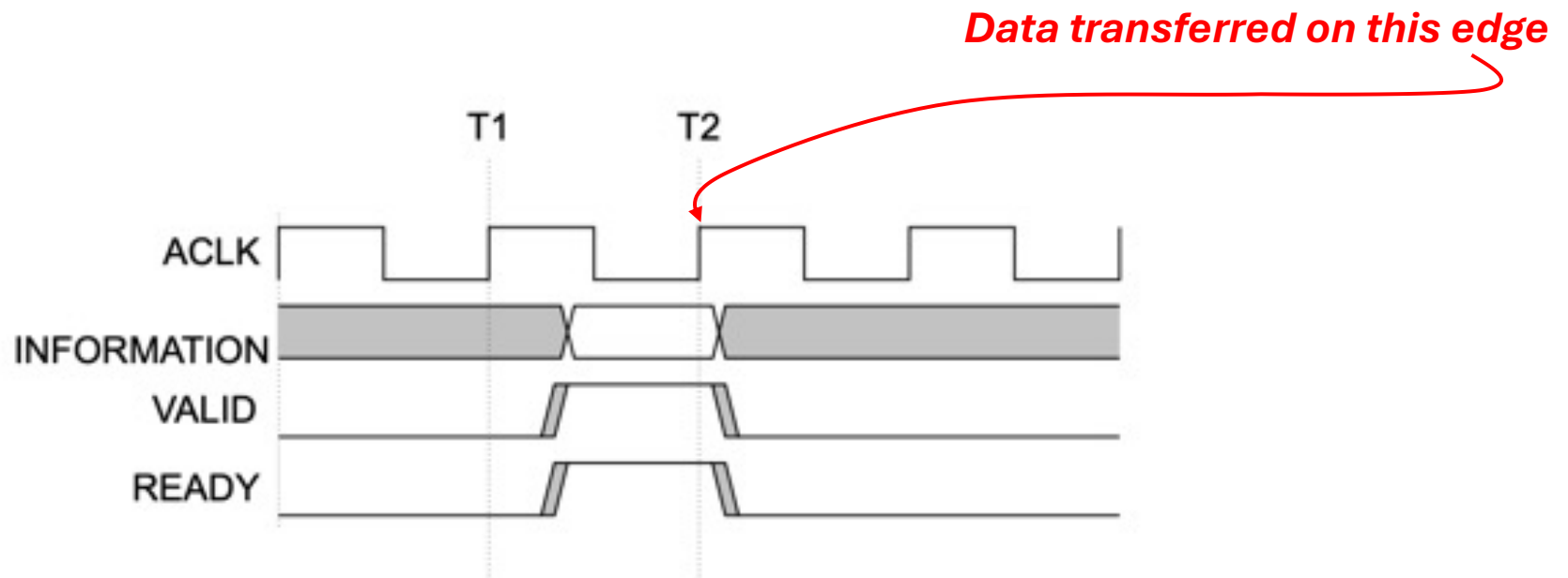


Figure A3-4 VALID with READY handshake

IMPORTANT

- the **VALID** signal of the AXI interface sending information *must not be dependent* on the **READY** signal of the AXI interface receiving that information
 - an AXI interface that is receiving information *may* wait until it detects a **VALID** signal before it asserts its corresponding **READY** signal.
 - In other words **READY** can depend on **VALID**, but not the other way around.
- Once **VALID** is asserted, it cannot be deasserted until **READY** has also been asserted for at least one cycle

Other Important Things to Keep in Mind!

- Both parties must be monitoring READY and VALID. A design should never “count” data without checking both of those!
- Errors in design happen when you assume a transfer happened only based off of VALID or READY’s value.

Up to All Five AXI channels can come from one device

- While operating independently at their individual transaction level, they can be dependent on one another at the higher module level to provide overall interfaces
- Example:
 - The slave device receives address on write channel address
 - The write data channel then becomes active and knows where to point incoming data
 - The response channel then opens and does its thing
 - And so on
- Hierarchy of Control/Design

Three General Flavors of AXI4

- **AXI4 (Full AXI):** For memory-mapped links. Provides highest performance.
 1. Address is supplied
 2. Then a data burst transfer of up to 256 data words
- **AXI4 Lite:** A memory-mapped simplified link supporting only one data transfer per connection (no bursts). (also restricted to 32 bit addr/data)
 1. Address is supplied
 2. One data transfer
- **AXI4 Stream:** Meant for high-speed streaming data
 - Can do burst transfers of unrestricted size
 - No addressing
 - Meant to stream data from one device to another quickly on its own direct connection

Complexity

- In terms of wires and options, Full-AXI is the most complex
- AXI-LITE has a lot less options (single data beat so all the supplemental stuff that specifies burst characteristics gets skipped)
- AXI-STREAM has even less...basically a high-speed write channel (Few options), but often needs that extra TLAST signal

Full-AXI4



AXI-LITE



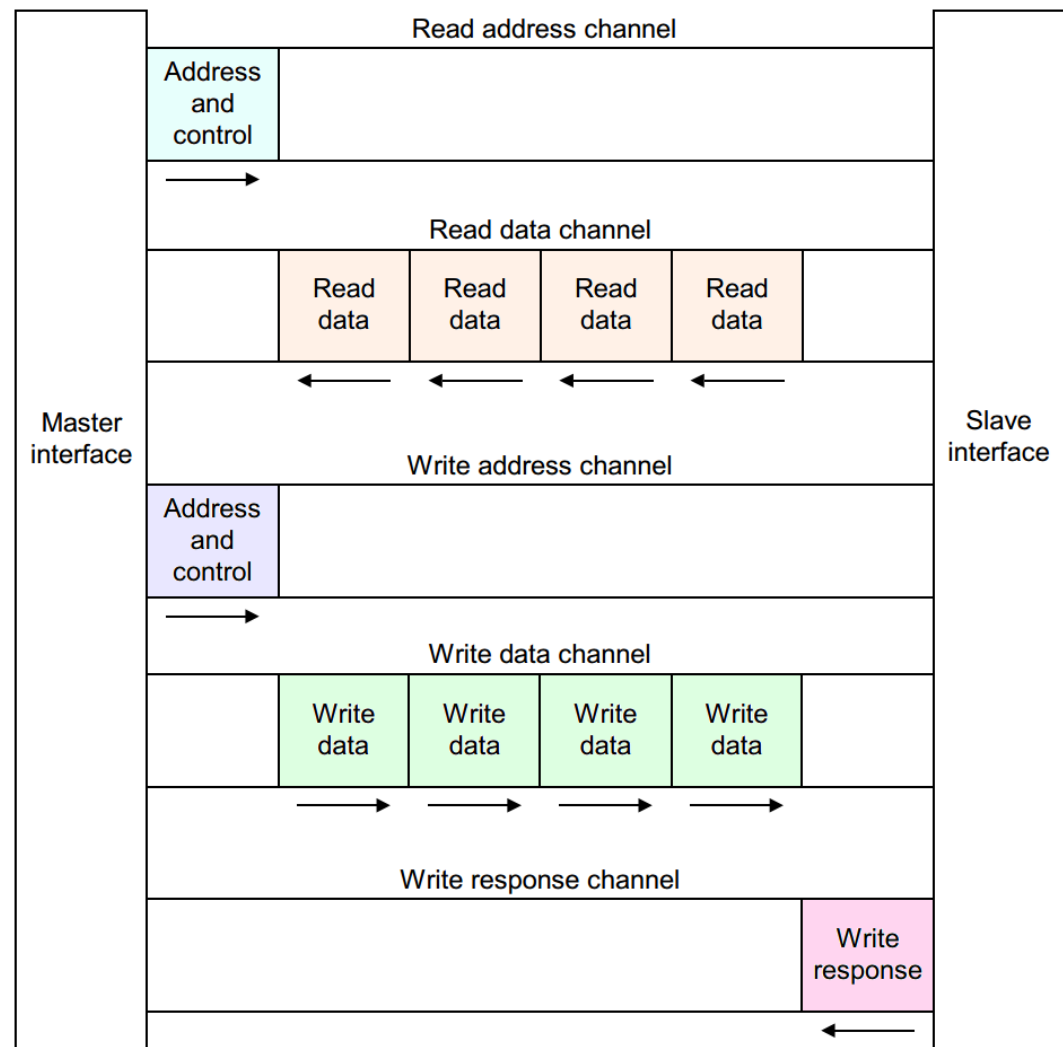
AXI-STREAM

Three General Flavors of AXI4

- **AXI4 (Full AXI):** For memory-mapped links. Provides highest performance.
 1. Address is supplied
 2. Then a data burst transfer of up to 256 data words
- **AXI4 Lite:** A memory-mapped simplified link supporting only one data transfer per connection (no bursts). (also restricted to 32 bit addr/data)
 1. Address is supplied
 2. One data transfer
- **AXI4 Stream:** Meant for high-speed streaming data
 - Can do burst transfers of unrestricted size
 - No addressing
 - Meant to stream data from one device to another quickly on its own direct connection

AXI4

- 2 channels for read, 3 for write, or all 5 for read/write
- Single address/control commands can be issued and then responses can be read back



AXI4 Read

- ARSIZE is 0 indexed so “0” has size of 1 lol
- ARBURST specifies burst type (fixed, in-place, wrap)

Read Address Channel Signals

ARID[3:0] Read address ID	Master
-------------------------------------	--------

ARADDR[31:0] Read address	Master
-------------------------------------	--------

ARLEN[3:0] Burst length	Master
-----------------------------------	--------

ARSIZE[2:0] Burst size	Master
----------------------------------	--------

ARBURST[1:0] Burst type	Master
-----------------------------------	--------

ARLOCK[1:0] Lock type	Master
---------------------------------	--------

ARCACHE[3:0] Cache type	Master
-----------------------------------	--------

ARPROT[2:0] Protection type	Master
---------------------------------------	--------

ARVALID Read address valid	Master
--------------------------------------	--------

ARREADY Read address ready	Slave
--------------------------------------	-------

Global Signals

ACLK Global clock signal	Clock source
------------------------------------	--------------

ARESETn Global reset signal	Reset source
---------------------------------------	--------------

Read Data Channel Signals

RID[3:0] Read ID tag	Slave
--------------------------------	-------

RDATA[31:0] Read data	Slave
---------------------------------	-------

RRESP[1:0] Read response	Slave
------------------------------------	-------

RLAST Read last	Slave
---------------------------	-------

RVALID Read valid	Slave
-----------------------------	-------

RREADY Read ready	Master
-----------------------------	--------

AXI4 Write

- Similar to Read but also has data channel

Write Address Channel Signals		Write Data Channel Signals	
AWID[3:0] Write address ID	Master	WID[3:0] Write ID tag	Master
AWADDR[31:0] Write address	Master	WDATA[31:0] Write data	Master
AWLEN[3:0] Burst length	Master	WSTRB[3:0] Write strobes	Master
AWSIZE[2:0] Burst size	Master	WLAST Write last	Master
AWBURST[1:0] Burst type	Master	WVALID Write valid	Master
AWLOCK[1:0] Lock type	Master	WREADY Write ready	Slave
AWCACHE[3:0] Cache type	Master	Write Response Channel Signals	
AWPROT[2:0] Protection type	Master	BID[3:0] Response ID	Slave
AWVALID Write address valid	Master	BRESP[1:0] Write response	Slave
AWREADY Write address ready	Slave	BVALID Write response valid	Slave
		BREADY Response ready	Master

AXI4 supports out-of-order reads but not writes

- The ID signals allow labels to accompany requests and then on the response those labels can be used to link up responses and requests if they are out-of-order

Read Address Channel Signals		Global Signals	
ARID[3:0] Read address ID	Master	ACLK Global clock signal	Clock source
ARADDR[31:0] Read address	Master	ARESETn Global reset signal	Reset source
ARLEN[3:0] Burst length	Master	Read Data Channel Signals	
ARSIZE[2:0] Burst size	Master	RID[3:0] Read ID tag	Slave
ARBURST[1:0] Burst type	Master	RDATA[31:0] Read data	Slave
ARLOCK[1:0] Lock type	Master	RRESP[1:0] Read response	Slave
ARCACHE[3:0] Cache type	Master	RLAST Read last	Slave
ARPROT[2:0] Protection type	Master	RVALID Read valid	Slave
ARVALID Read address valid	Master	RREADY Read ready	Master
ARREADY Read address ready	Slave		

Every Channel Supports AXI

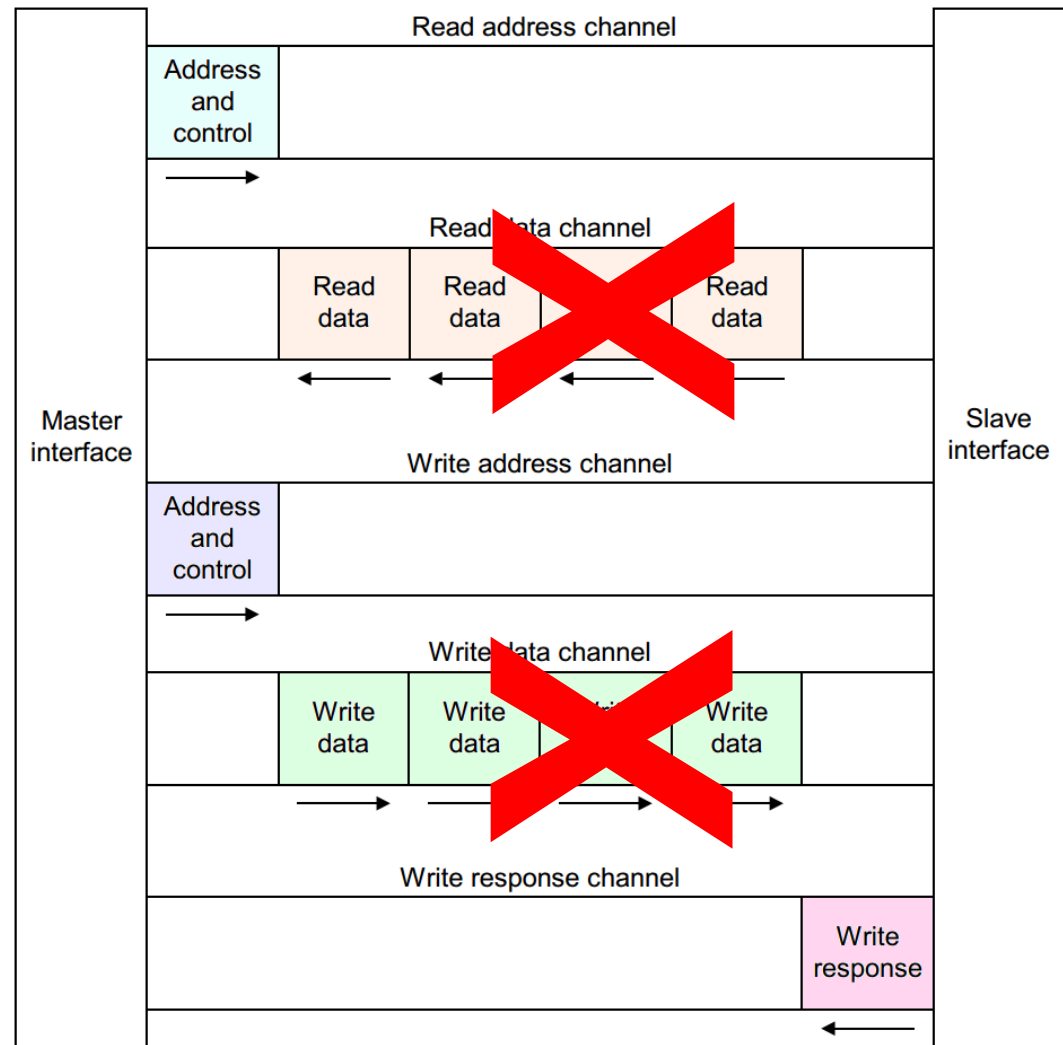
- In the middle of a burst of write-data, the slave/subordinate device is free to deassert its READY signal to pause transfers!
- The master/manager must watch for this and react appropriately.

Three General Flavors of AXI4

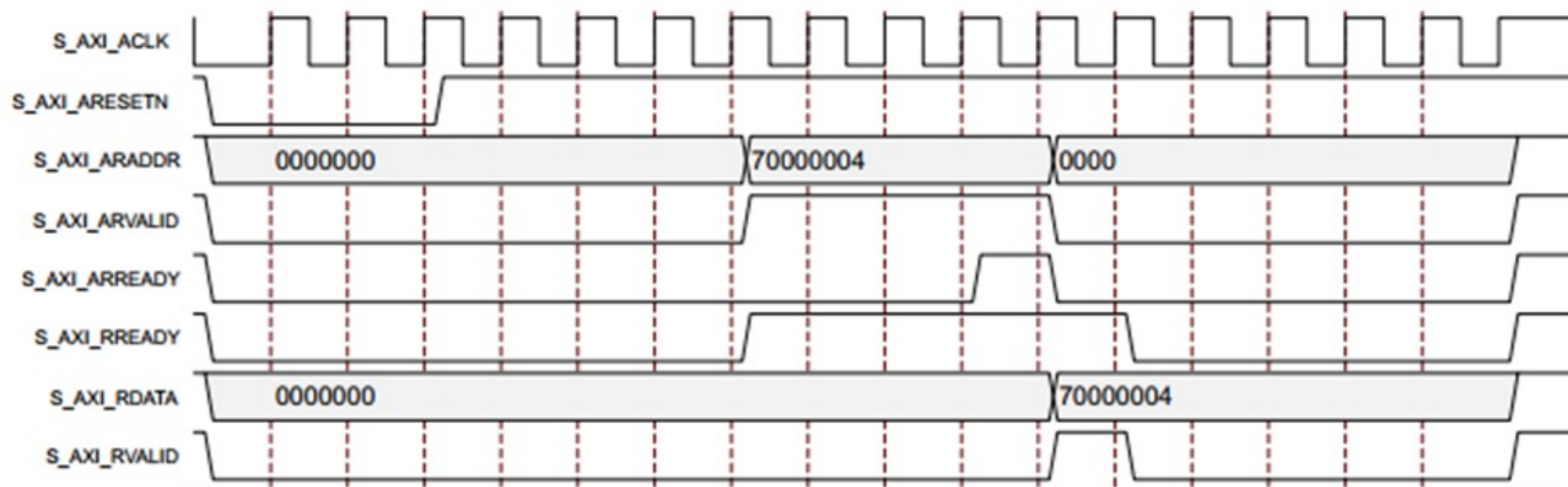
- **AXI4 (Full AXI):** For memory-mapped links. Provides highest performance.
 1. Address is supplied
 2. Then a data burst transfer of up to 256 data words
- **AXI4 Lite:** A memory-mapped simplified link supporting only one data transfer per connection (no bursts). (also restricted to 32 bit addr/data)
 1. Address is supplied
 2. One data transfer
- **AXI4 Stream:** Meant for high-speed streaming data
 - Can do burst transfers of unrestricted size
 - No addressing
 - Meant to stream data from one device to another quickly on its own direct connection

AXI4-Lite

- 2 channels for read, 3 for write, or all 5 for read/write
- Single address/control commands can be issued and then responses can be read back
- No burst

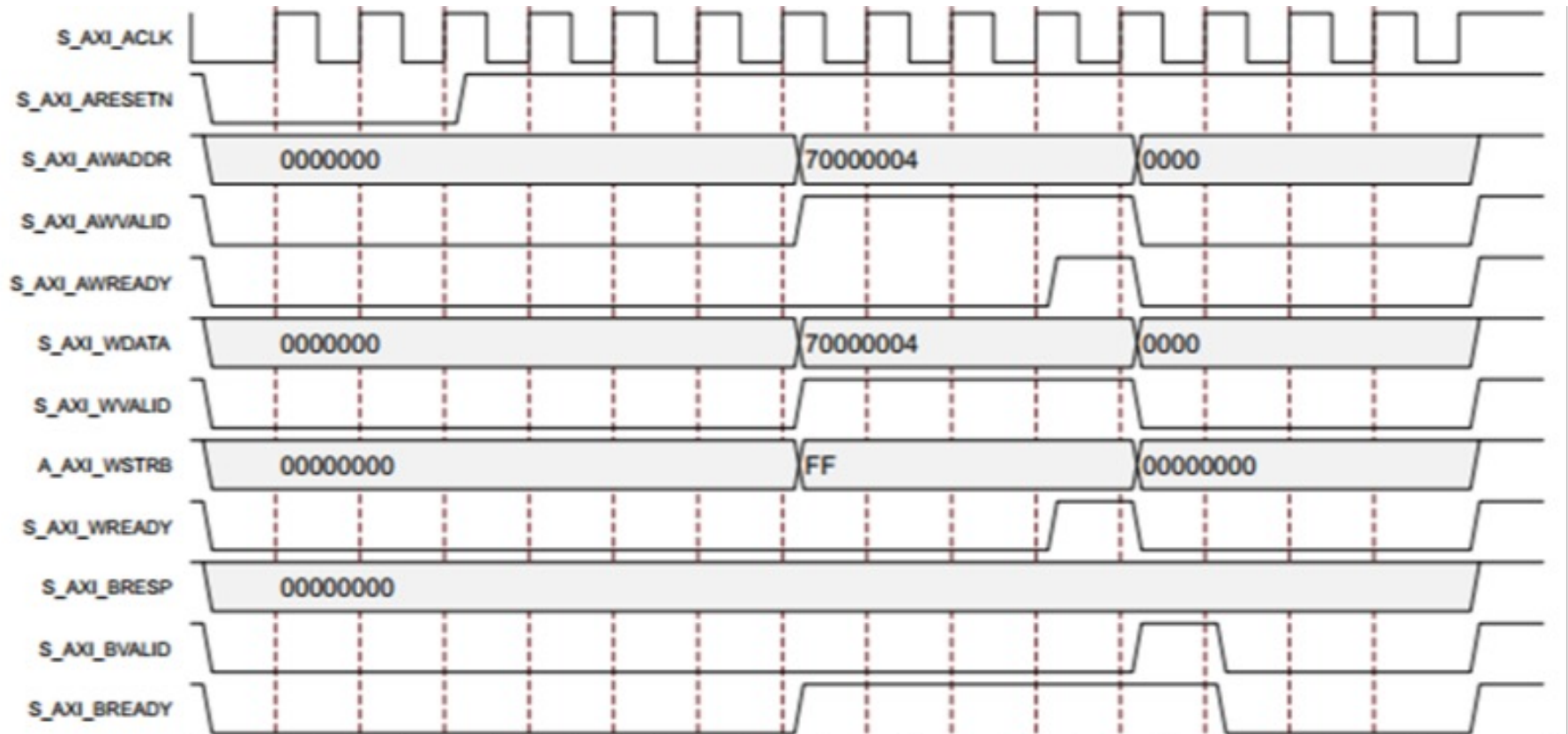


AXI4-LITE Read transaction



<https://www.realdigital.org/doc/a9fee931f7a172423e1ba73f66ca4081>

AXI4-LITE Write transaction



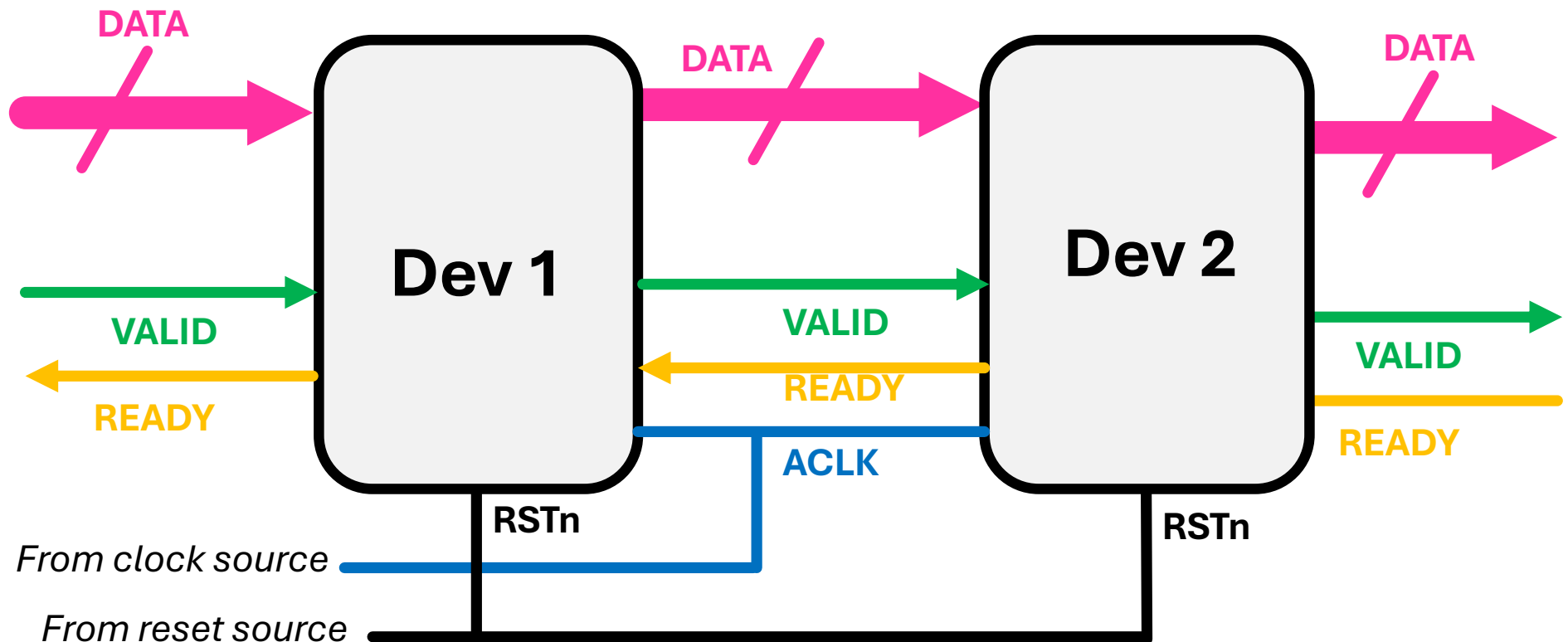
<https://www.realdigital.org/doc/a9fee931f7a172423e1ba73f66ca4081>

Three General Flavors of AXI4

- **AXI4 (Full AXI):** For memory-mapped links. Provides highest performance.
 1. Address is supplied
 2. Then a data burst transfer of up to 256 data words
- **AXI4 Lite:** A memory-mapped simplified link supporting only one data transfer per connection (no bursts). (also restricted to 32 bit addr/data)
 1. Address is supplied
 2. One data transfer
- **AXI4 Stream:** Meant for high-speed streaming data
 - Can do burst transfers of unrestricted size
 - No addressing
 - Meant to stream data from one device to another quickly on its own direct connection

Good News about AXI5

- It is the least complicated of the AXI protocols

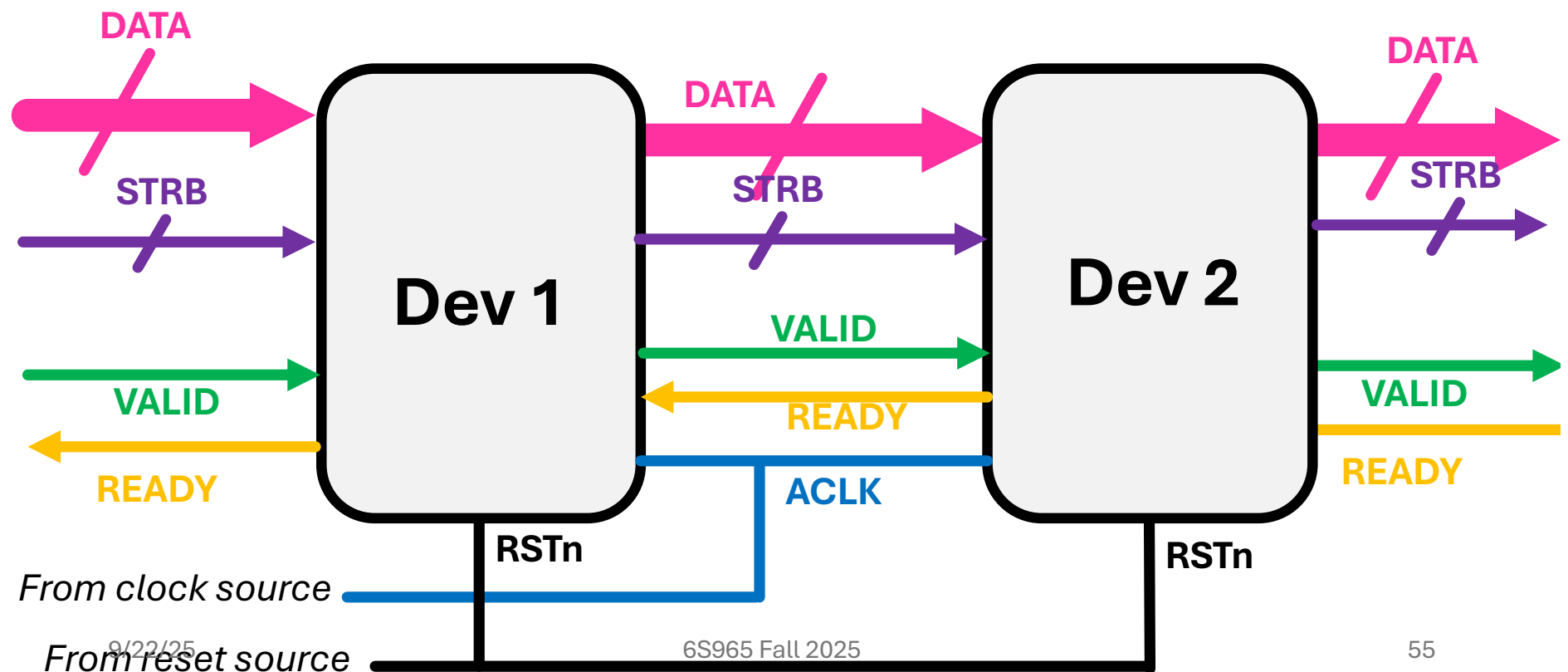


No Addressing!

- Data flows unidirectionally
- Data's "place" is where it is in the chain. It doesn't have an address it is supposed to live at
- For values that are independent of one another this is pretty much all that's needed
- Often a few other signals...

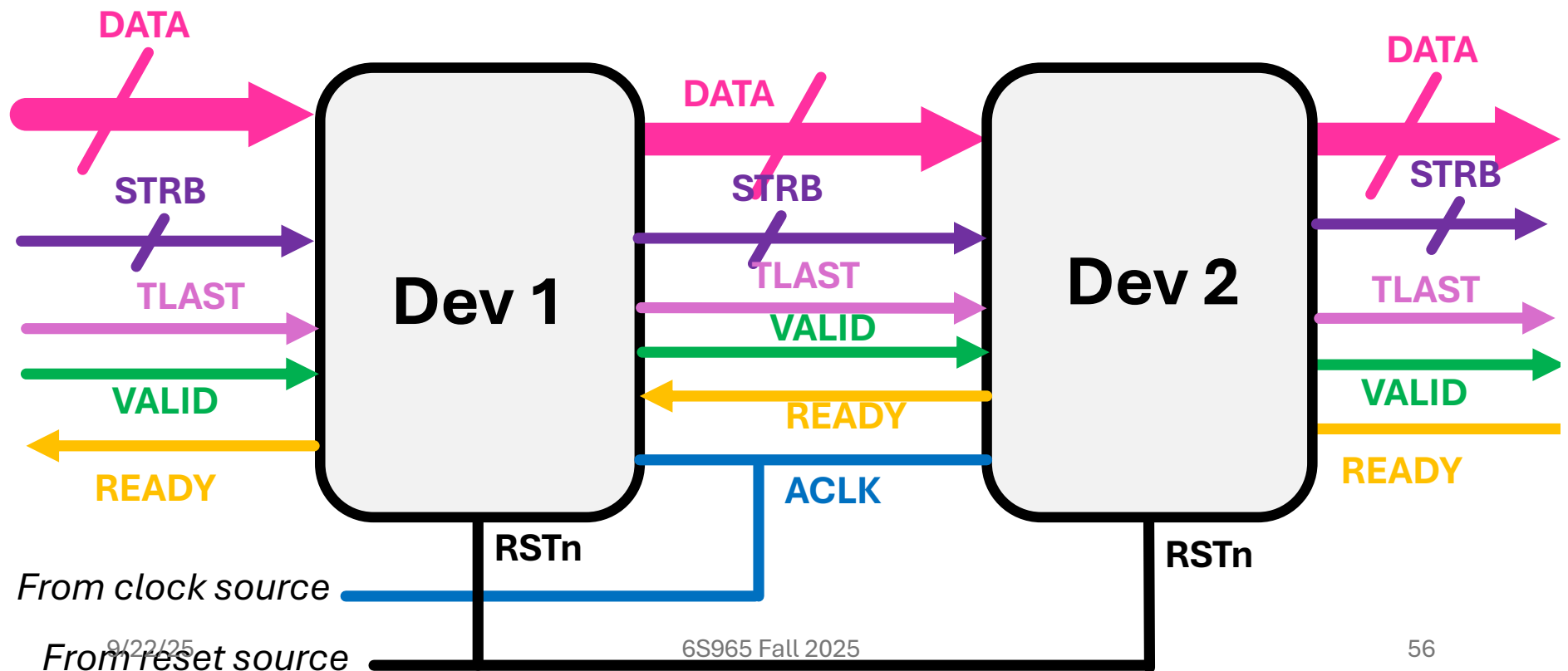
Strobe

- The strobe line will clarify which bytes in data are to be acted upon (default 0b1111 aka all)



TLAST

- For data sent in packets (such as samples of a signal where), a TLAST signal is asserted on the final sample to ensure



It is too much!

Read Address Channel Signals

ARID[3:0] Read address ID	Master
ARADDR[31:0] Read address	Master
ARLEN[3:0] Burst length	Master
ARSIZE[2:0] Burst size	Master
ARBURST[1:0] Burst type	Master
ARLOCK[1:0] Lock type	Master
ARCACHE[3:0] Cache type	Master
ARPROT[2:0] Protection type	Master
ARVALID Read address valid	Master
ARREADY Read address ready	Slave

Global Signals

ACLK Global clock signal	Clock source
ARESETn Global reset signal	Reset source

Read Data Channel Signals

RID[3:0] Read ID tag	Slave
RDATA[31:0] Read data	Slave
RRESP[1:0] Read response	Slave
RLAST Read last	Slave
RVALID Read valid	Slave
RREADY Read ready	Master

Read Address Channel Signals

ARID[3:0] Read address ID	Master
ARADDR[31:0] Read address	Master

Write Address Channel Signals

WRID[3:0] Write ID tag	Master
WRADDR[31:0] Write address	Master
WRLEN[3:0] Write burst length	Master
WRSIZE[2:0] Write burst size	Master
WRBURST[1:0] Write burst type	Master
WRLOCK[1:0] Write lock type	Master

Write Data Channel Signals

WID[3:0] Write ID tag	Master
WDATA[31:0] Write data	Master
WSTRB[3:0] Write strobes	Master
WLAST Write last	Master
WVALID Write valid	Master
WREADY Write ready	Slave

Write Response Channel Signals

BID[3:0] Response ID	Slave
BRESP[1:0] Write response	Slave
BVALID Write response valid	Slave
BREADY Response ready	Master

Global Signals

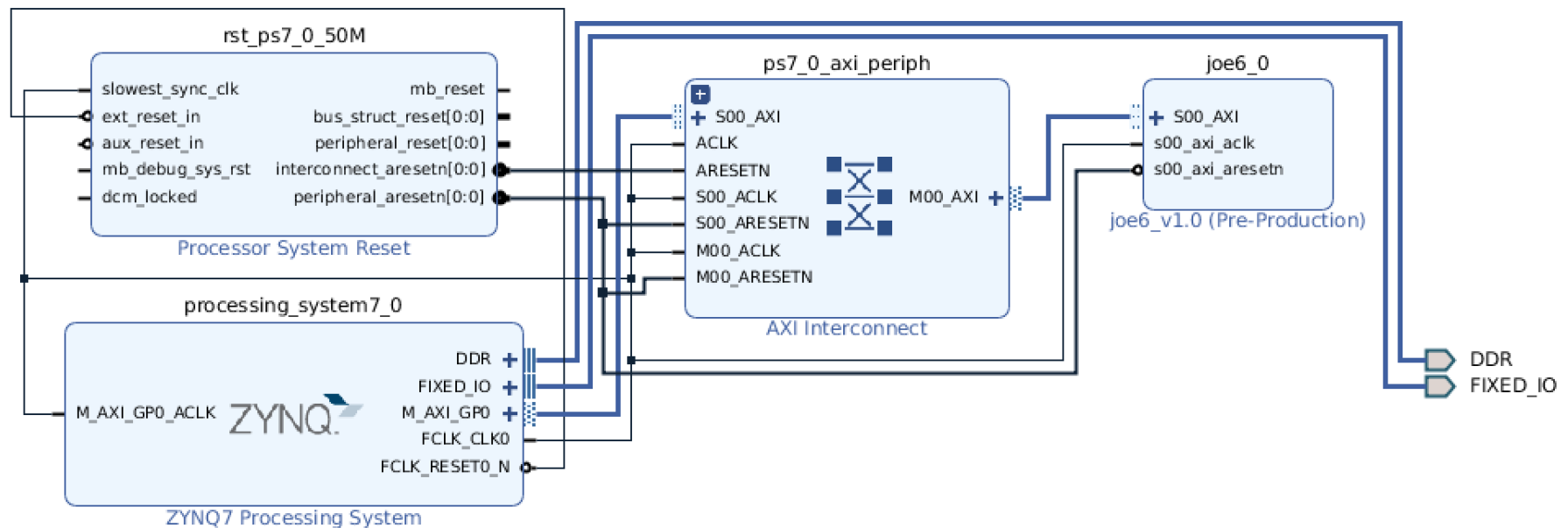
ACLK Global clock signal	Clock source
ARESETn Global reset signal	Reset source

Read Data Channel Signals

RID[3:0] Read ID tag	Slave
RDATA[31:0] Read data	Slave
RRESP[1:0] Read response	Slave
RLAST Read last	Slave
RVALID Read valid	Slave
RREADY Read ready	Master

GET ME OUT OF HERE

One of the Strengths of the Vivado Workflow is its encapsulating of all that (necessary) mess up into bold “wires” that you drag around



Block Diagram Makes:

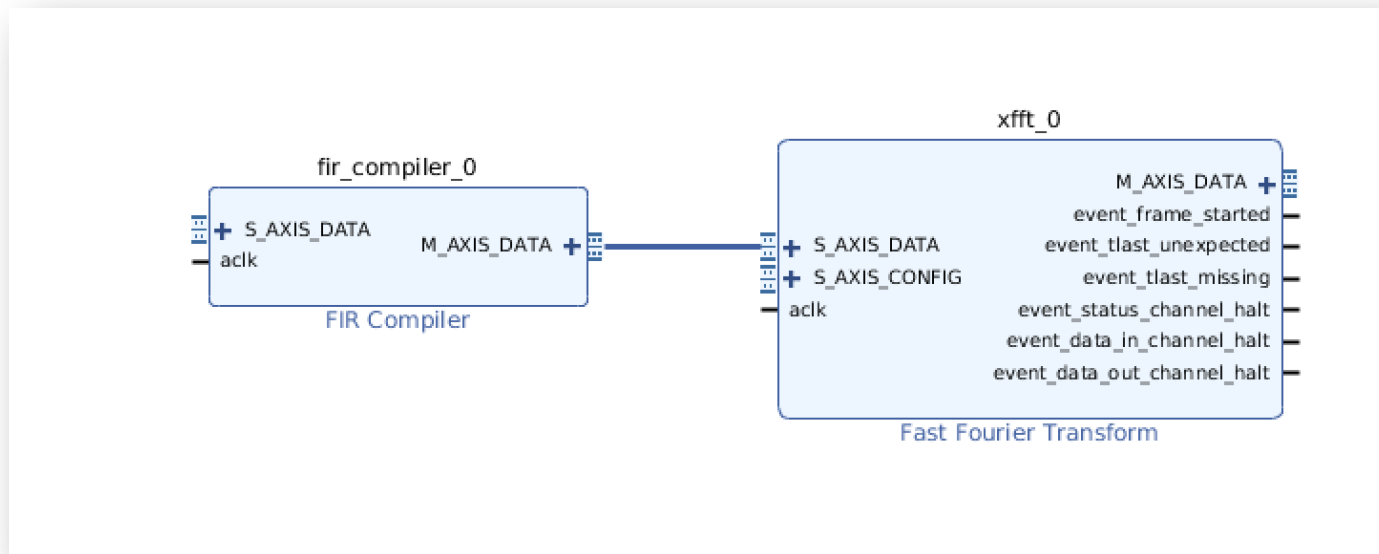
- The block diagram automates this portion of work.
- Or at least tries to

```
test_axi.py test_square_rooter.py //Copyright 1986-2022 Xilinx, Inc. All Rights Reserved

241     .trigger(disp_interface_0_trigger));
242     assign leds = disp_interface_0_command[3:0];
243     design_1_processing_system7_0_0 processing_system7_0
244     (
245         .DDR_Addr(DDR_addr),
246         .DDR_BankAddr(DDR_ba),
247         .DDR_CAS_n(DDR_cas_n),
248         .DDR_CKE(DDR_cke),
249         .DDR_CS_n(DDR_cs_n),
250         .DDR_Clk(DDR_ck_p),
251         .DDR_Clk_n(DDR_ck_n),
252         .DDR_DM(DDR_dm),
253         .DDR_DQ(DDR_dq),
254         .DDR_DQS(DDR_dqs_p),
255         .DDR_DQS_n(DDR_dqs_n),
256         .DDR_DRSTB(DDR_reset_n),
257         .DDR_ODT(DDR_odt),
258         .DDR_RAS_n(DDR_ras_n),
259         .DDR_VRN(FIXED_IO_ddr_vrn),
260         .DDR_VRP(FIXED_IO_ddr_vrp),
261         .DDR_WEB(DDR_we_n),
262         .FCLK_CLK0(processing_system7_0_FCLK_CLK0),
263         .FCLK_RESET0_N(processing_system7_0_FCLK_RESET0_N),
264         .MIO(FIXED_IO_mio),
265         .M_AXI_GP0_ACLK(processing_system7_0_FCLK_CLK0),
266         .M_AXI_GP0_ARADDR(processing_system7_0_M_AXI_GP0_ARADDR),
267         .M_AXI_GP0_ARBURST(processing_system7_0_M_AXI_GP0_ARBURST),
268         .M_AXI_GP0_ARCACHE(processing_system7_0_M_AXI_GP0_ARCACHE),
269         .M_AXI_GP0_ARID(processing_system7_0_M_AXI_GP0_ARID),
270         .M_AXI_GP0_ARLEN(processing_system7_0_M_AXI_GP0_ARLEN),
271         .M_AXI_GP0_ARLOCK(processing_system7_0_M_AXI_GP0_ARLOCK),
272         .M_AXI_GP0_ARPROT(processing_system7_0_M_AXI_GP0_ARPROT),
273         .M_AXI_GP0_ARQOS(processing_system7_0_M_AXI_GP0_ARQOS),
274         .M_AXI_GP0_ARREADY(processing_system7_0_M_AXI_GP0_ARREADY),
275         .M_AXI_GP0_ARSIZE(processing_system7_0_M_AXI_GP0_ARSIZE),
276         .M_AXI_GP0_ARVALID(processing_system7_0_M_AXI_GP0_ARVALID),
277         .M_AXI_GP0_AWADDR(processing_system7_0_M_AXI_GP0_AWADDR),
278         .M_AXI_GP0_AWBURST(processing_system7_0_M_AXI_GP0_AWBURST),
279         .M_AXI_GP0_AWCACHE(processing_system7_0_M_AXI_GP0_AWCACHE),
280         .M_AXI_GP0_AWID(processing_system7_0_M_AXI_GP0_AWID),
281         .M_AXI_GP0_AWLEN(processing_system7_0_M_AXI_GP0_AWLEN),
282         .M_AXI_GP0_AWLOCK(processing_system7_0_M_AXI_GP0_AWLOCK),
283         .M_AXI_GP0_AWPROT(processing_system7_0_M_AXI_GP0_AWPROT),
284         .M_AXI_GP0_AWQOS(processing_system7_0_M_AXI_GP0_AWQOS),
285         .M_AXI_GP0_AWREADY(processing_system7_0_M_AXI_GP0_AWREADY),
286         .M_AXI_GP0_AWSIZE(processing_system7_0_M_AXI_GP0_AWSIZE),
287         .M_AXI_GP0_AWVALID(processing_system7_0_M_AXI_GP0_AWVALID),
288         .M_AXI_GP0_BID(processing_system7_0_M_AXI_GP0_BID),
289         .M_AXI_GP0_BREADY(processing_system7_0_M_AXI_GP0_BREADY),
290         .M_AXI_GP0_BRESP(processing_system7_0_M_AXI_GP0_BRESP),
291         .M_AXI_GP0_BVALID(processing_system7_0_M_AXI_GP0_BVALID),
```

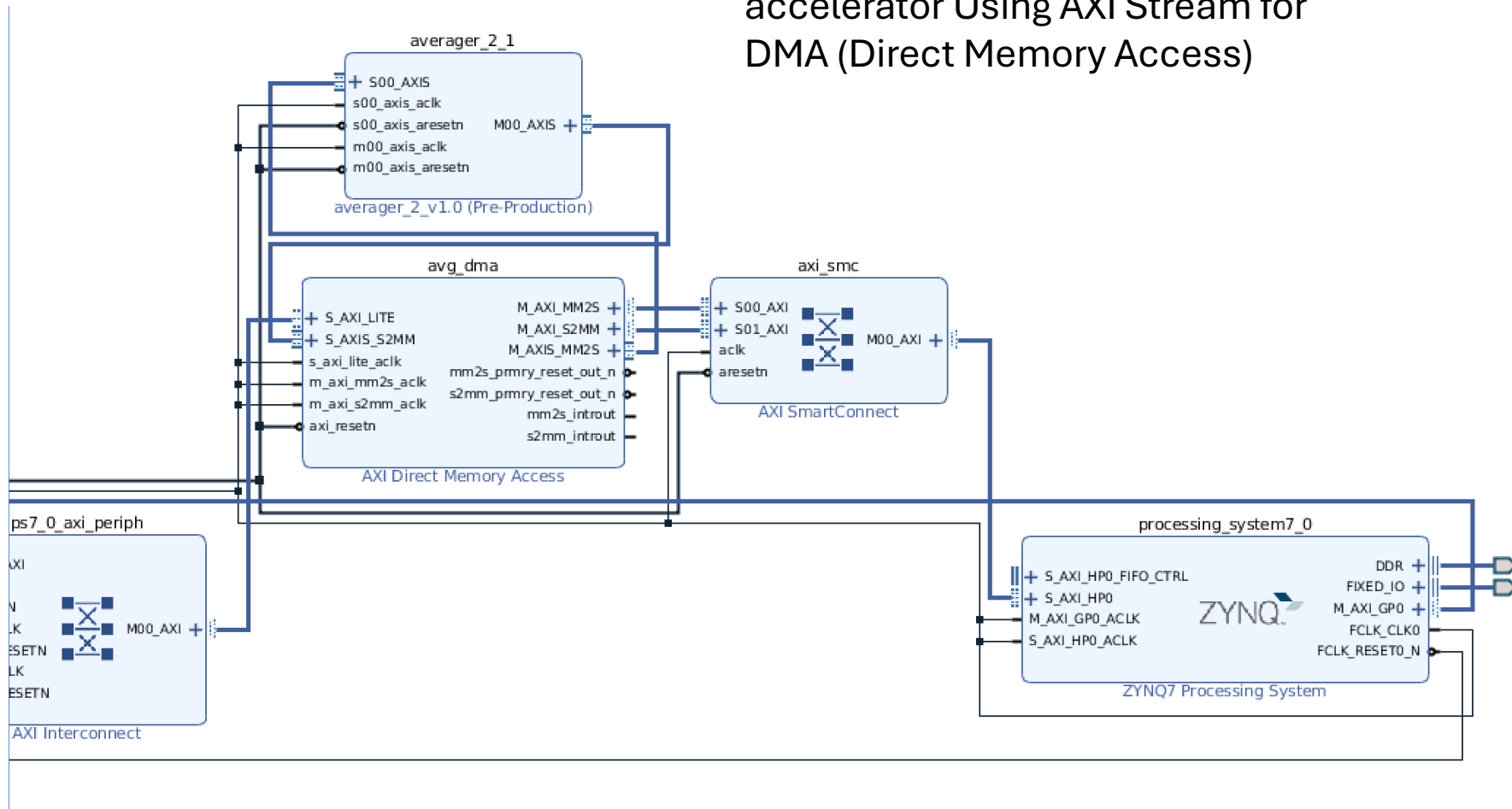
And you Can Use AXI to Interface with Tons of things!

- For data sent in packets (such as samples of a signal where), a TLAST signal is asserted on the final sample to ensure



And you Can Use AXI to Interface with Tons of things!

A running-average hardware accelerator Using AXI Stream for DMA (Direct Memory Access)



The AXI Interfaces on the Zynq Enable PS to PL communication effectively

Interface Name	Interface Description	Master	Slave
M_AXI_GP0	General Purpose (AXI_GP)	PS	PL
M_AXI_GP1		PS	PL
S_AXI_GP0	General Purpose (AXI_GP)	PL	PS
S_AXI_GP1		PL	PS
S_AXI_ACP	Accelerator Coherency Port (ACP), cache coherent transaction	PL	PS
S_AXI_HP0	High Performance Ports (AXI_HP) with read/write FIFOs. (Note that AXI_HP interfaces are sometimes referred to as AXI Fifo Interfaces, or AFIs).	PL	PS
S_AXI_HP1		PL	PS
S_AXI_HP2		PL	PS
S_AXI_HP3		PL	PS

Master/Slave refers to who controls/initiates comms on that bus that bus

From Zynq Book

General Purpose/Performance “GP” AXI Ports

- 32 bits in size
- Maximum flexibility
- Allow register access from:
 - PS to PL
 - PL to PS
- Routed through lower-priority interconnects

High Performance “HP” AXI Ports

- Can be 32 or 64 bits wide (or variable between, but avoid)
- Maximum bandwidth access to external memory and on-chip-memory (OCM)
- When use all four HP ports at 64 bits, you can outpace ability to write to DDR and OCM bandwidths!
 - HP Ports : $4 * 64 \text{ bits} * 150 \text{ MHz} * 2 = \mathbf{9.6 \text{ GByte/sec}}$
 - external DDR: $1 * 32 \text{ bits} * 1066 \text{ MHz} * 2 = \mathbf{4.3 \text{ GByte/sec}}$
 - OCM : $64 \text{ bits} * 222 \text{ MHz} * 2 = \mathbf{3.5 \text{ GByte/sec}}$
- Optimized for large burst lengths

Taken from ECE699 lec 6 notes gm.edu

But Also Be On the Lookout for Issues!

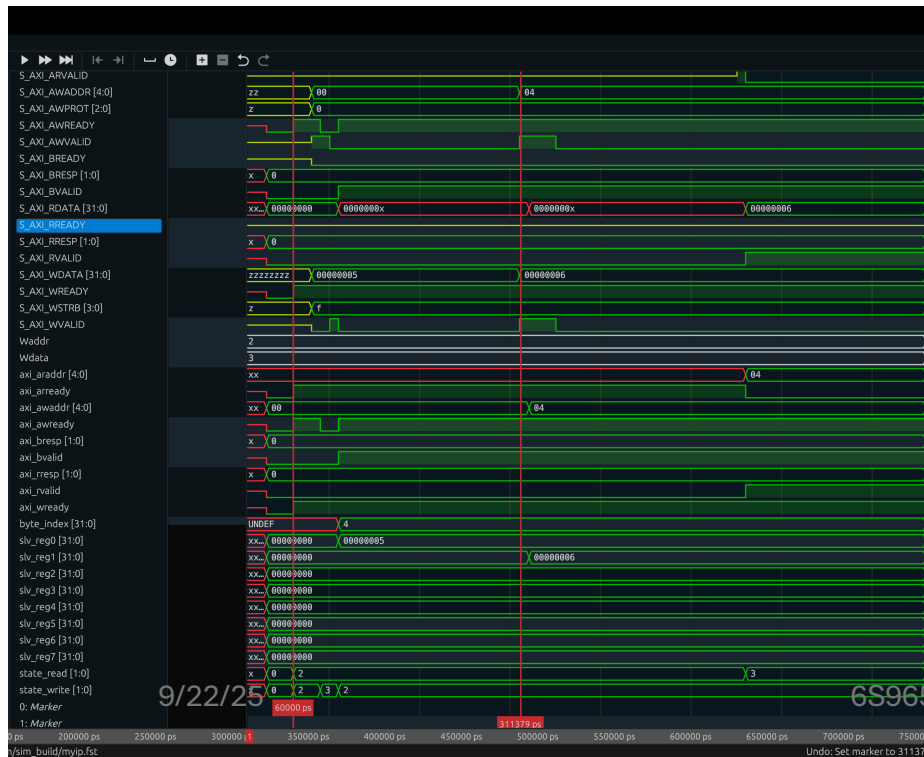
AXI-Lite Packager Broke in 2024.

- Still not sure *what* broke going from 2023.2 to 2024.1
- The new source for AXI Lite mentions burst mode...not sure if that's a typo or indicative of something else weird.
- Also incompletely specifies read logic compared to <2024.1

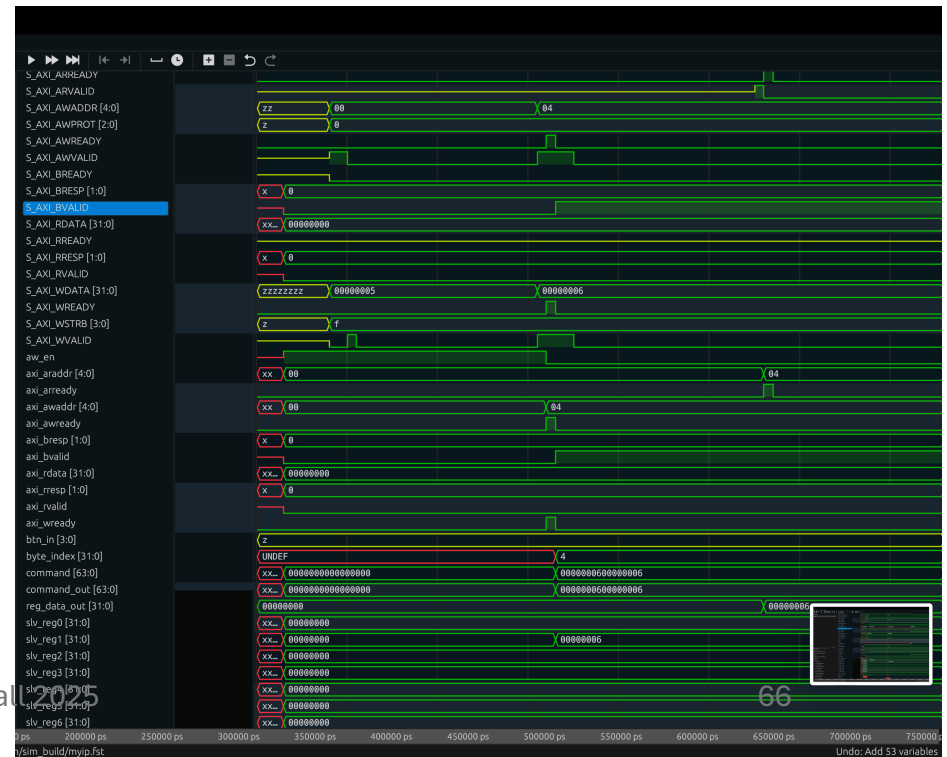
Variants

- I think there was a bug in their READY implementation

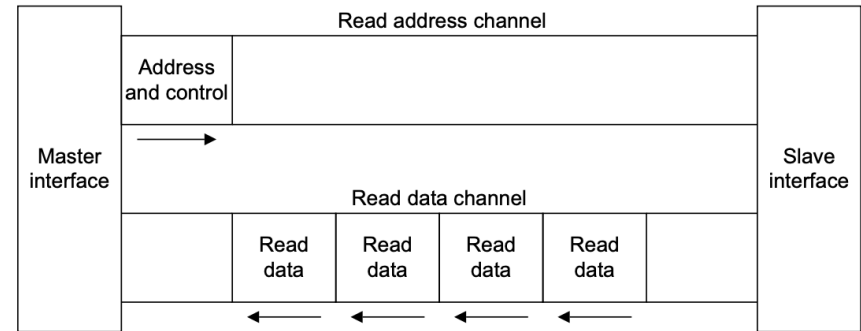
Not-working (2024.1)



Less Not-working (2023.2)

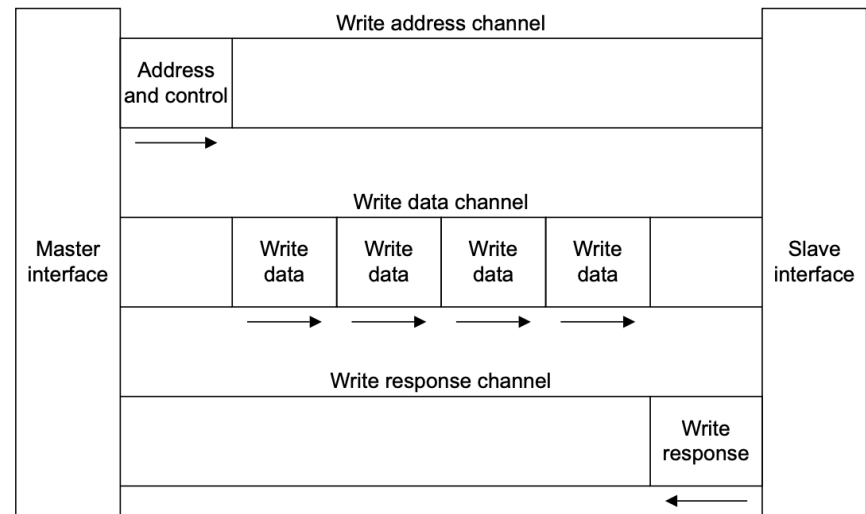


Reads Worked Fine



- The “IP wizard” does fail to create all the appropriate read logic by default, but for registers it does, things work
- And you can add in the logic to read the “forgotten” registers (>4) and things still work

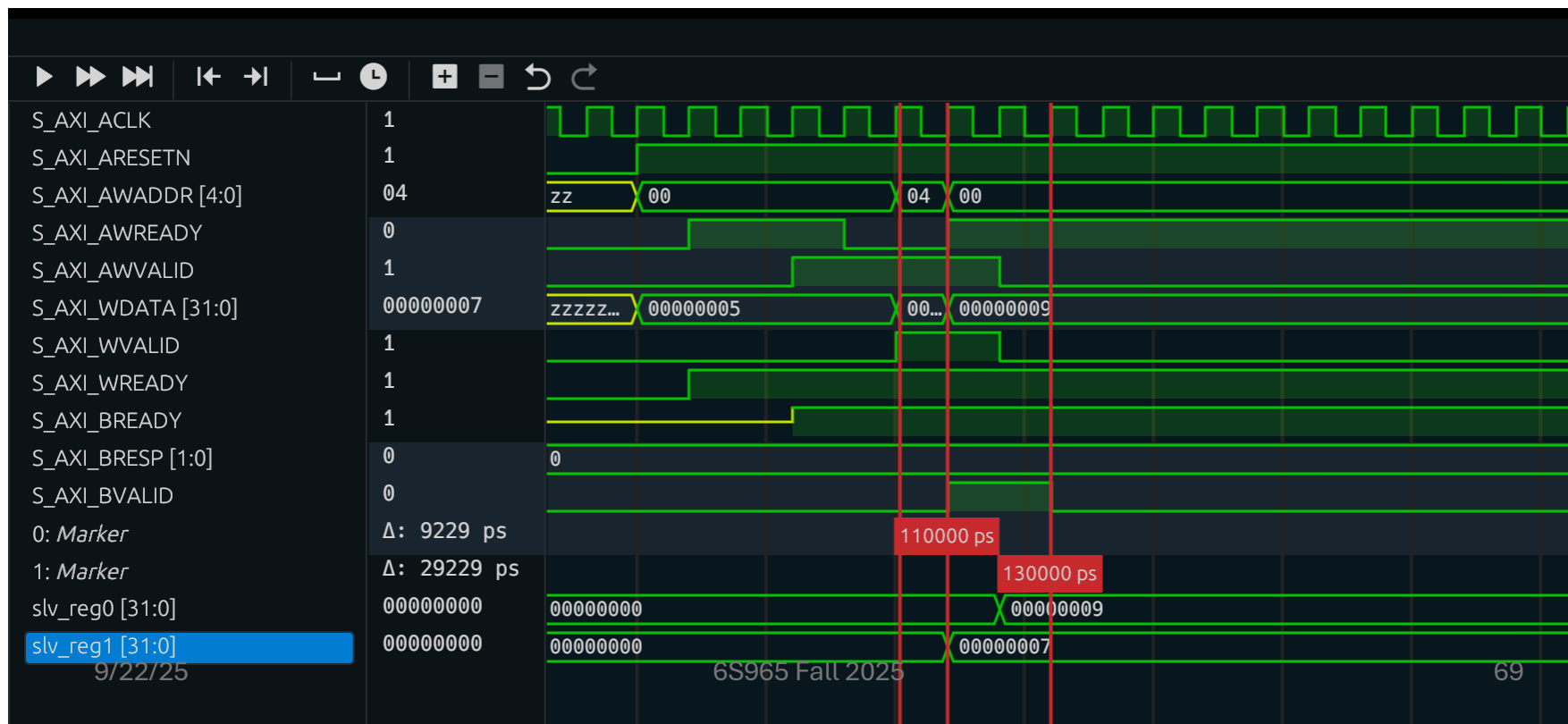
What was Broken?



- Hard Crash/Timeout when a Write is made to the AXI MMIO created
- My guess is it is related to the response channel logic
- An AXI write interface will have three channels:
 - Write Address (“AW”) (address to write data to)
 - Write Data (“W”) (data to write)
 - Response Data (“B”) A response

I think this is one issue:

- AXI_AWADDR getting used when AXI_AWREADY and AXI_AWVALID are both not asserted.



Line Letting Un-hand-shaken data through:

- Early in module there is this:

```
if (S_AXI_AWVALID && S_AXI_AWREADY)begin  
    axi_awaddr <= S_AXI_AWADDR;
```

- Elsewhere the write logic had this

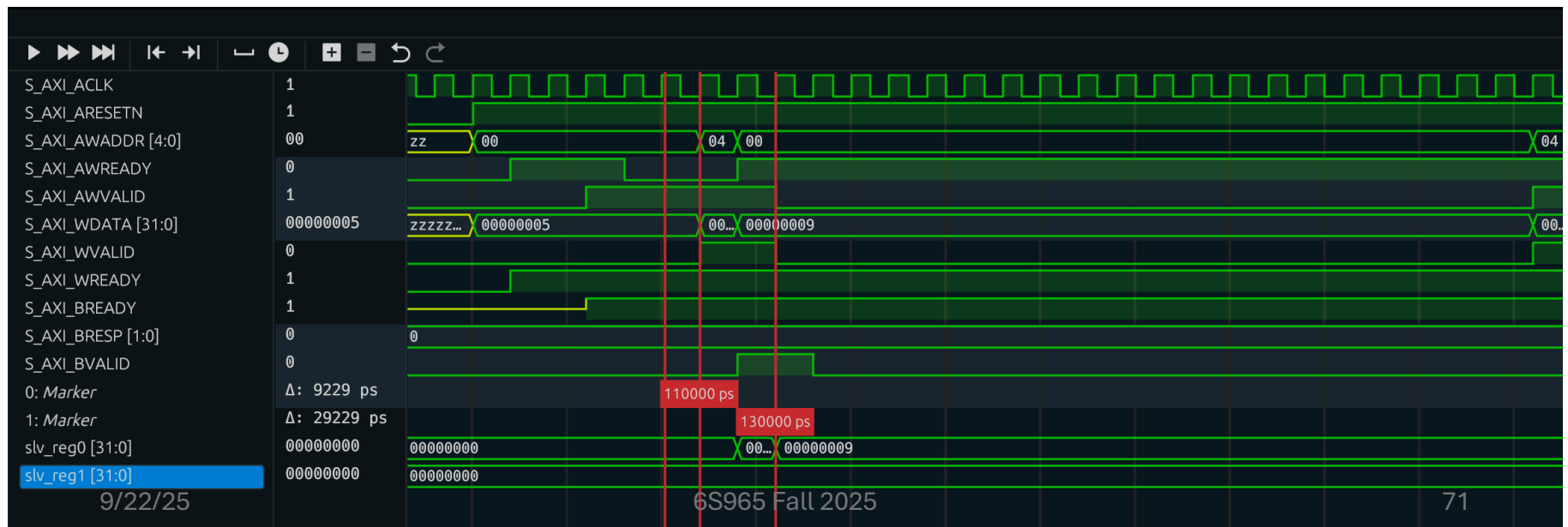
```
//suck:  
case ( (S_AXI_AWVALID) ? S_AXI_AWADDR[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB]  
      : axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
```

- Change to this:

```
//seems better:  
case ( (S_AXI_AWREADY && S_AXI_AWVALID) ? S_AXI_AWADDR[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB]  
      : axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
```

Update: mostly fixed problem

- There's a line that uses the raw address based only on AXI_AWVALID
- Change it

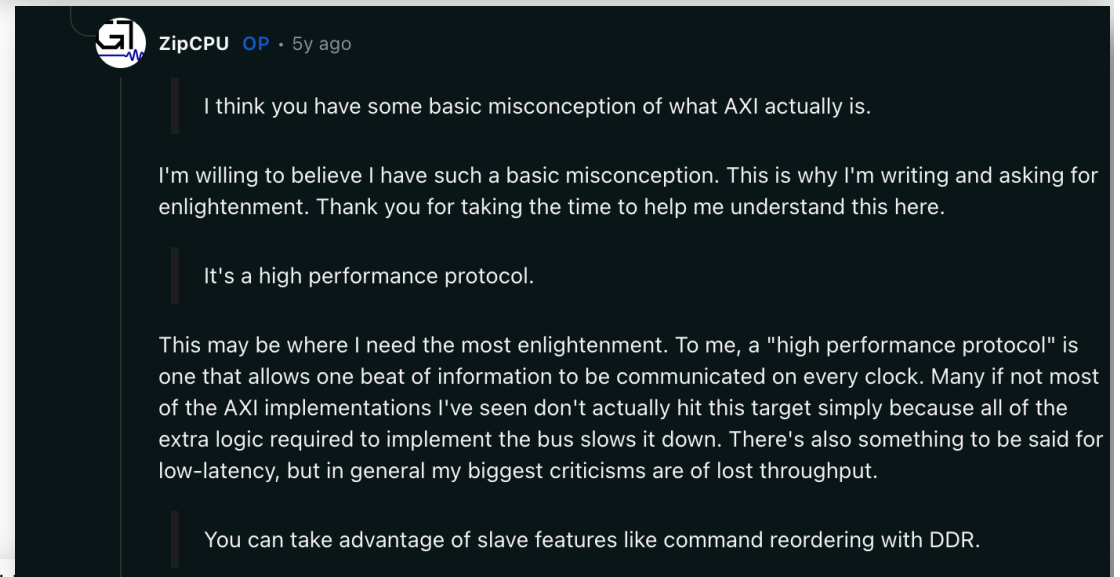


Still Not Exactly Sure

- So while Vivado is a powerful tool, we should always be aware of what it is making for us and how it is helping us.
- It has been known to make non-AXI-compliant stuff in the past.
- And I'm not the only one to speak about this.

AXI Culture

- This Gisselquist guy is anywhere anybody mentions AXI on the internet



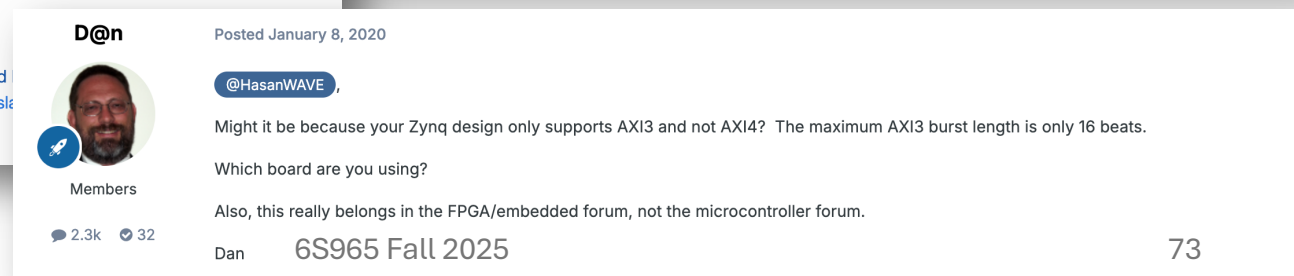
While I have filed bug reports in 2017 and 2018 on Xilinx's forums regarding these broken demonstration designs, Xilinx has yet to fix their designs as of Vivado 2020.2. [1], [2] Indeed, at this point, it's not clear if Xilinx will ever fix their demonstration designs. Perhaps I shouldn't complain—their designs simply make the services I offer and sell that much more valuable.

The most common AXI mistake

Apr 16, 2019

Some time ago, I posted a set of formal properties which could slave or master. I then applied these properties to the AXI-lite slave and found multiple errors within their core.

9/22/25



73

←  **r/FPGA** · 2 yr. ago
guyWithTheFaceTatto

An actual demonstration of ZipCpu on an FPGA Board

Advice / Help

I'm at the beginning of my journey on writing my own softcore CPU. I found the [ZipCpu](#) to be very interesting, but couldn't find a single project demo or a video where this CPU was loaded onto a board and a C program was run on it. I'm not sure if there is a lot of extra complexity involved in porting it to an actual fpga given that it's working fine on the [simulator](#).

It would help if someone could point me to any project demo and code where a CPU was built from scratch and an actual operating system was booted onto it. It would be a great encouragement for my upcoming project.

↑ 19 ↓ 💬 29 👤 ➦ Share

 **xfinity** · Official · Promoted

Level up your tech with Xfinity Mobile.

[Learn More](#)

[xfinity.com](#)



threespeedlogic · 2y ago · Edited 2y ago

Why ZipCPU, and not any of the RISC-V cores out there?

I suspect (and am 100% prepared to be mistaken!) that the ZipCPU has a developer community of [one](#) and a user community of [one](#). It's going to be challenging to manage tooling and ecosystem interfaces for any CPU with such a tiny community.

⊖ ↑ 21 ↓ 💬 Reply 👤 Award ➦ Share ...



Hairburt_Derhelle · 2y ago

Not only this. The developer is constantly promoting ZipCPU in forums when someone is asking questions about other CPUs

↑ 5 ↓ 💬 Reply 👤 Award ➦ Share ...

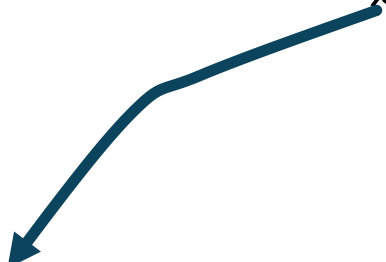
xfinity mobile

iPhone 17

PRO

Sources

This is the thing right here...the spec sheet/manual is surprisingly good!!



- **“AMBA® AXITM and ACETM Protocol Specification”, ARM 2011**
- **“The Zynq Book”, L.H. Crockett, R.A. Elliot, M.A. Enderwitz, and R.W. Stewart, University of Glasgow**
- **“Building Zynq Accelerators with Vivado High Level Synthesis”
Xilinx Technical Note**
- **Some material from ECE699 Spring 2016**
https://ece.gmu.edu/coursewebpages/ECE/ECE699_SW_HW/S16/

Crack open the AXI spec sheet with a few data sheets for some Xilinx IP cores (like the CORDIC, FFT, etc...) and you should be able to start making sense of it.