Cloud FPGA

EENG 428
ENAS 968

bit.ly/cloudfpga

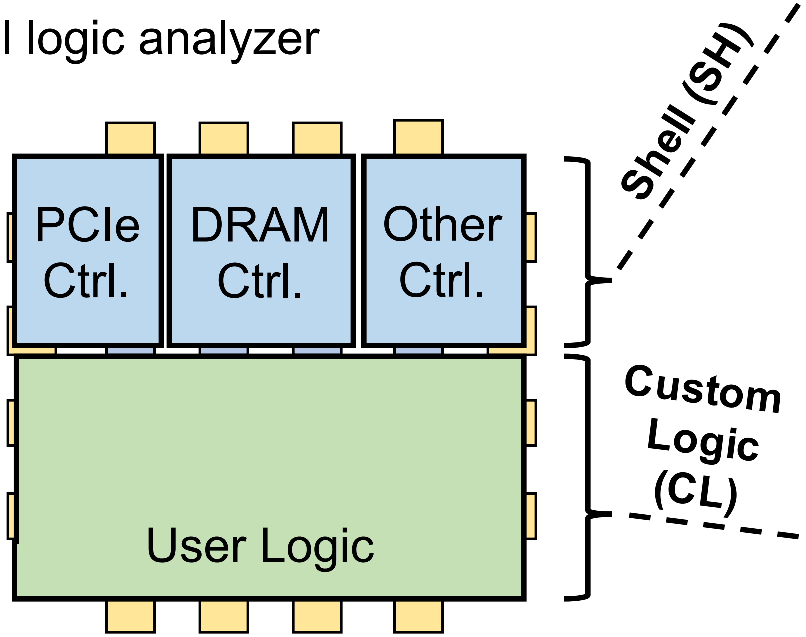
Lecture: Advanced eXtensible Interface (AXI)

Prof. Jakub Szefer
Dept. of Electrical Engineering, Yale University

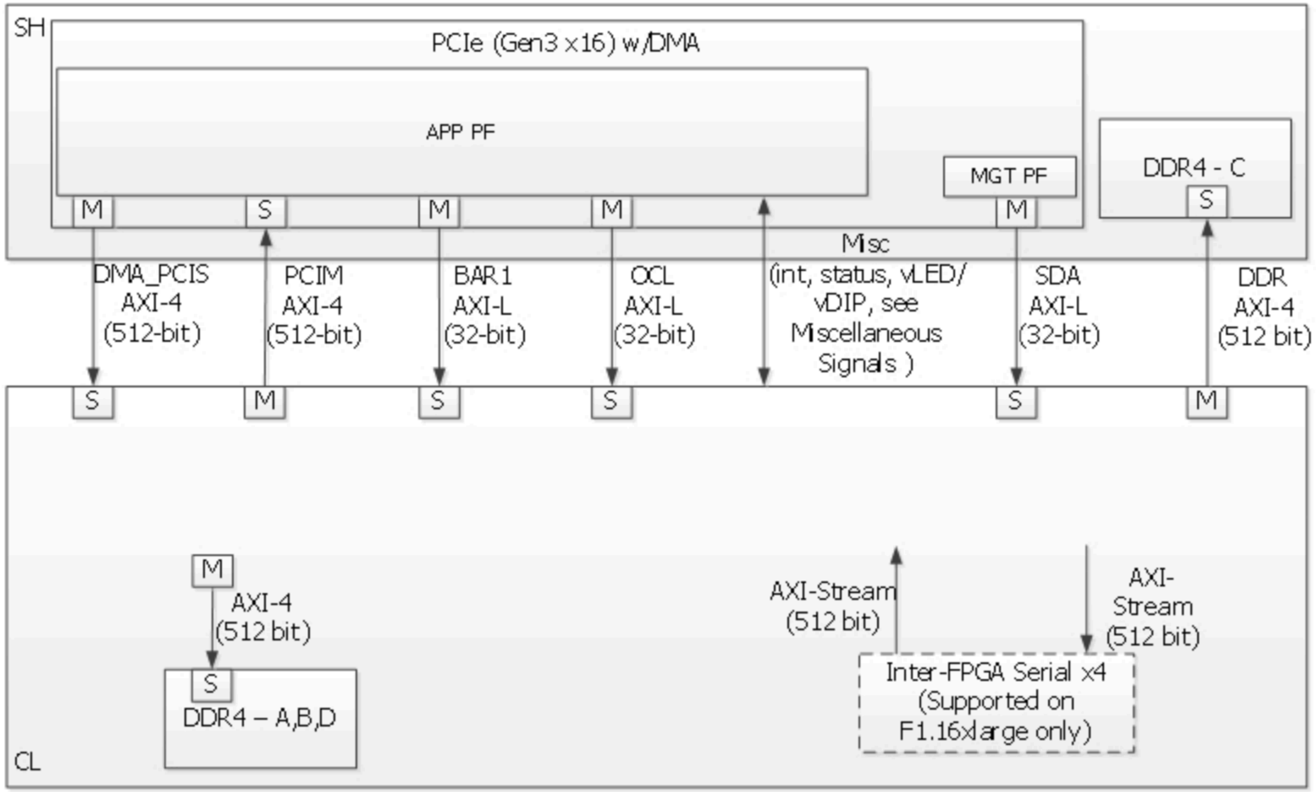
EENG 428 / ENAS 968
Cloud FPGA

FPGA Shell Review

- Recall that Amazon's Cloud FPGAs contain a Shell (SH) which contains a number of modules usable by the user's Custom Logic (CL)
    - PCIe controller to communicate with the server
    - DRAM controller to use DRAM modules
    - AXI bus interfaces
    - QSFP interfaces
    - Virtual logic analyzer
    - …



FPGA chip

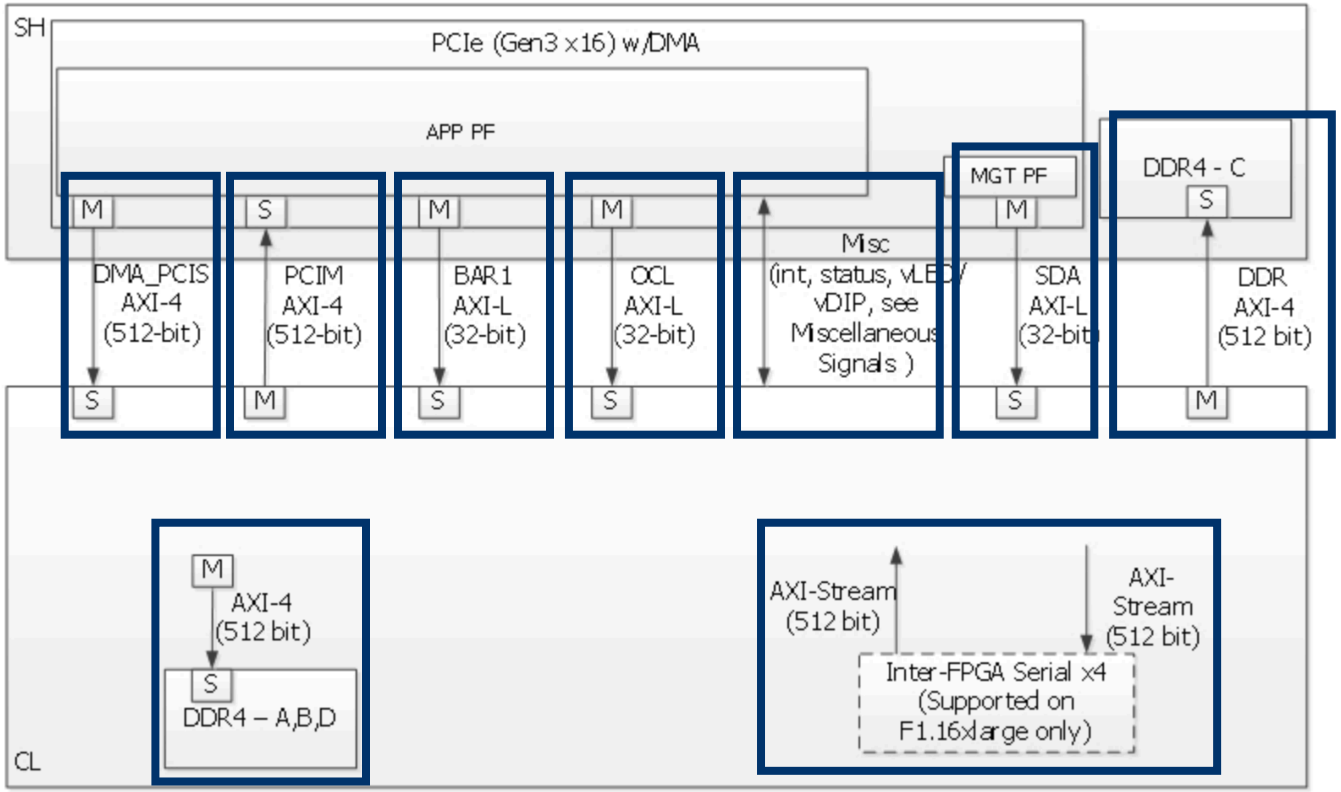


Block diagram from [2]

Use of AXI in the Shell and Custom Logic

Most of the communication between Shell and the Custom Logic is done through AXI buses

- Different variants of AXI are used
  - AXI4 512-bit
  - AXI4-Lite 32-bit
  - AXI4-Stream 512-bit
- Most custom logic (CL) modules require use of AXI
  - Except if only virtual LEDs and DIP switches are used
- Modules developed with AXI can be used outside of Cloud FPGAs, in any design using AXI



Block diagram from [2]

Advanced eXtensible Interface (AXI)

AXI Background

- **Advanced eXtensible Interface** (**AXI**) is a communication interface that is
  - parallel
  - high-performance
  - synchronous
  - high-frequency
  - multi-master and multi-slave

Multiple bits are sent in parallel, typically 32 but can have other sizes

Contains features such as streaming, to move data more quickly than word by word

Many devices can be on the same bus, but typically only worried about one master (controller) and slave (module doing computation)

- AXI targets on-chip communication in System-on-Chip (SoC) designs
- AXI is available royalty-free and its specification is freely available from ARM
- Latest version is 4:
  - AXI4
  - AXI4-Lite
  - AXI4-Stream

- All variants are utilized by Cloud FPGAs in Amazon
- Most designs will need at least AXI4-Lite for basic communication with the server

AXI information from [3]

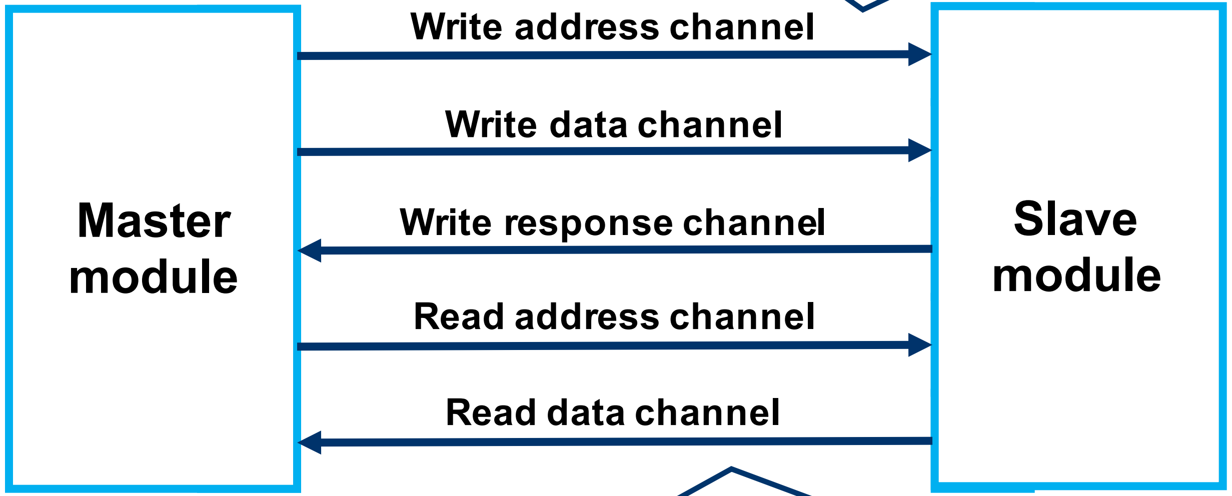
Basic AXI Connections between Master and Slave

In a most basic configuration, AXI is used to connect two modules:

- Master module initiates communication and data read/write requests
- Slave module responds to the requests

Communication is achieved over ‘channels’, which contain many wires each

Each channel only has one direction



Data channels are for sending actual data, while others are to control the data sending process

All communication is with respect to addresses, each address and it’s purpose is module specific

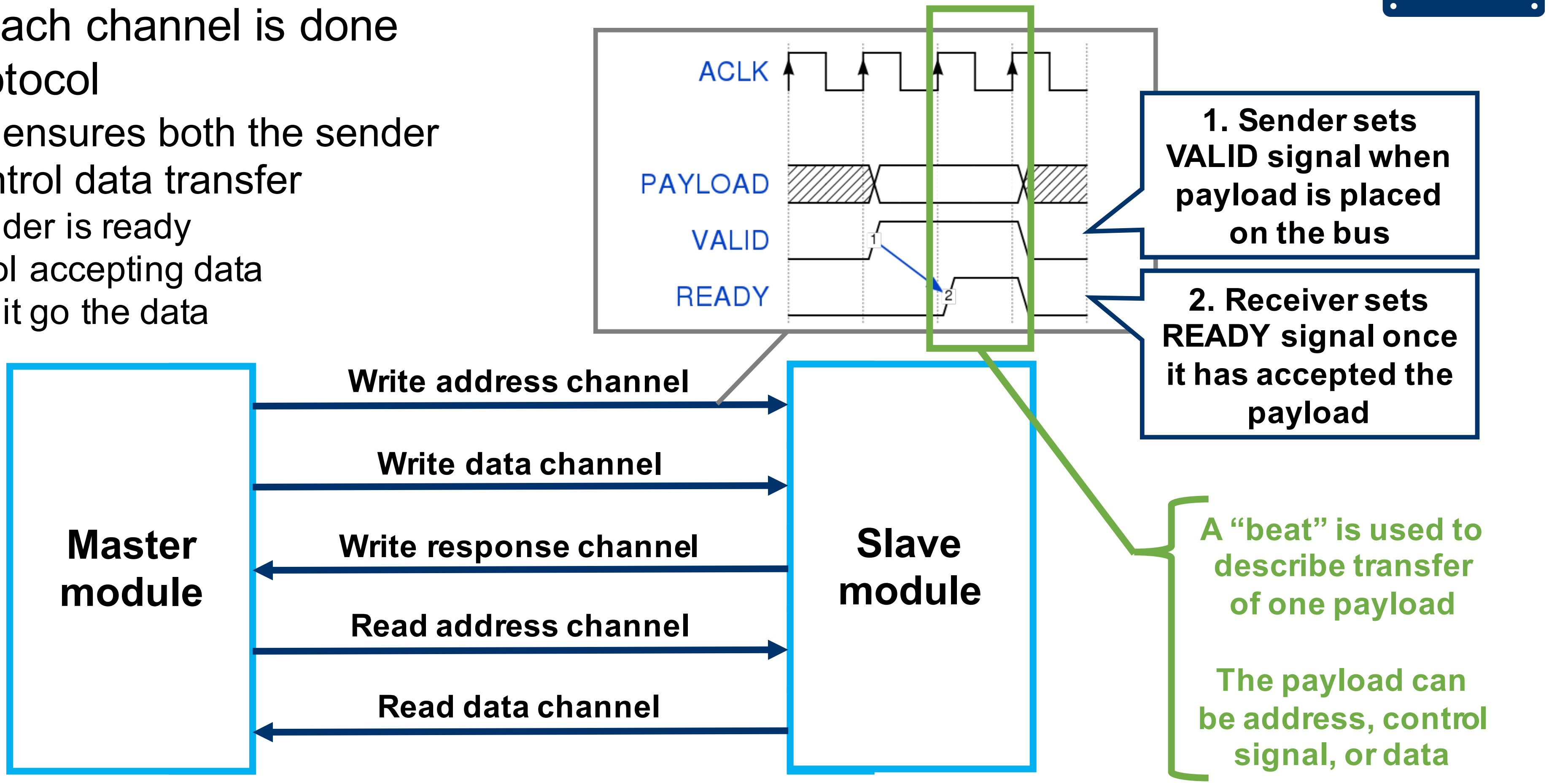
Register	Purpose
0x1000	Reg1
0x2000	Reg2
0x3000	Config
…	…

Note, can have a protocol where ‘commands’ are sent on the data bus, so it’s not just pure data

Share: bit.ly/cloudfpga

AXI Handshake

- Communication over each channel is done using a handshake protocol
  - Handshake protocol ensures both the sender and receiver can control data transfer
    - Indicate when sender is ready
    - Let receiver control accepting data and acknowledge it go the data



1. Sender sets VALID signal when payload is placed on the bus

2. Receiver sets READY signal once it has accepted the payload

A "beat" is used to describe transfer of one payload

The payload can be address, control signal, or data

Handshake protocol image from: https://commons.wikimedia.org/wiki/File:AMBA_AXI_Handshake.svg

AXI Channels

Sizes (widths) of the addresses can be design specific, 32 or 64 bit

- **Write Address channel (AW)**
  - Mainly provide address at which data should be written
  - Can optionally (depending on AXI type) specify burst size, beats per burst, etc.
  - AWVALID (master to slave) and AWREADY (slave to master)
- **Write Data channel (W)**
  - Actual data to that is sent
  - Can optionally specify data id, beat identifier, etc.
  - WVALID (master to slave) and WREADY (slave to master)

Sizes (widths) of the data can be design specific, e.g., 32, 64, 512

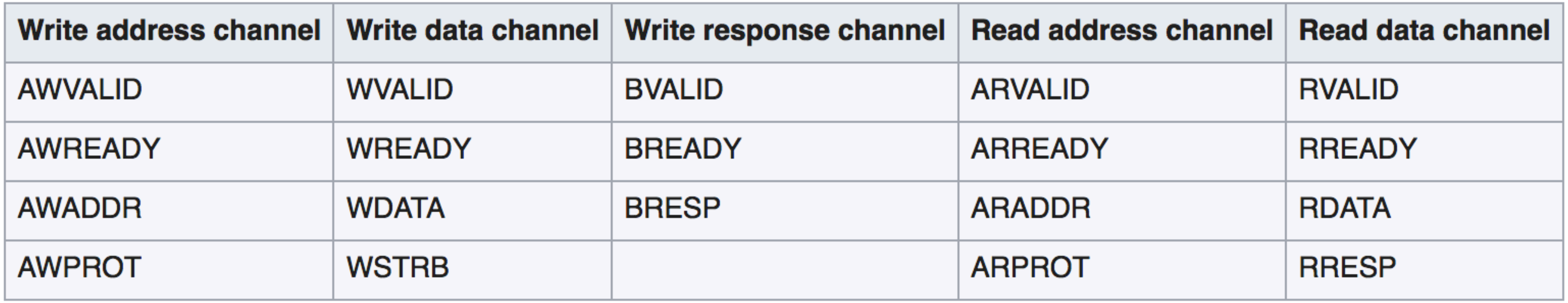
- **Write Response channel (B)**
  - Mainly to specify burst status
  - BVALID (slave to master) and BREADY (master to slave)
- **Read Address channel (AR)**
  - Mainly provide address from which to read data
  - Optional burst size, etc.
  - ARVALID (master to slave) and ARREADY (slave to master)
- **Read Data channel (R)**
  - Actual data sent back, plus optional data id, etc.
  - RVALID (slave to master) and RREAD (master to slave)

AXI4-Lite

AXI4-Lite is a subset of the AXI4 protocol, with only basic features

- No bursts, only send one piece of data (beat) at a time
- All data accesses use the full data bus width, which can be either 32 or 64 bits
- AXI4-Lite removes many of the AXI4 signals but follows the AXI4 specification for the rest
  - AXI4-Lite transactions are fully compatible with AXI4 devices
  - AXI4-Lite masters can be used with AXI4 slaves
  - AXI4 masters can work with AXI4-Lite slaves, if none of the extra features are triggered

AXI4-Lite signals:

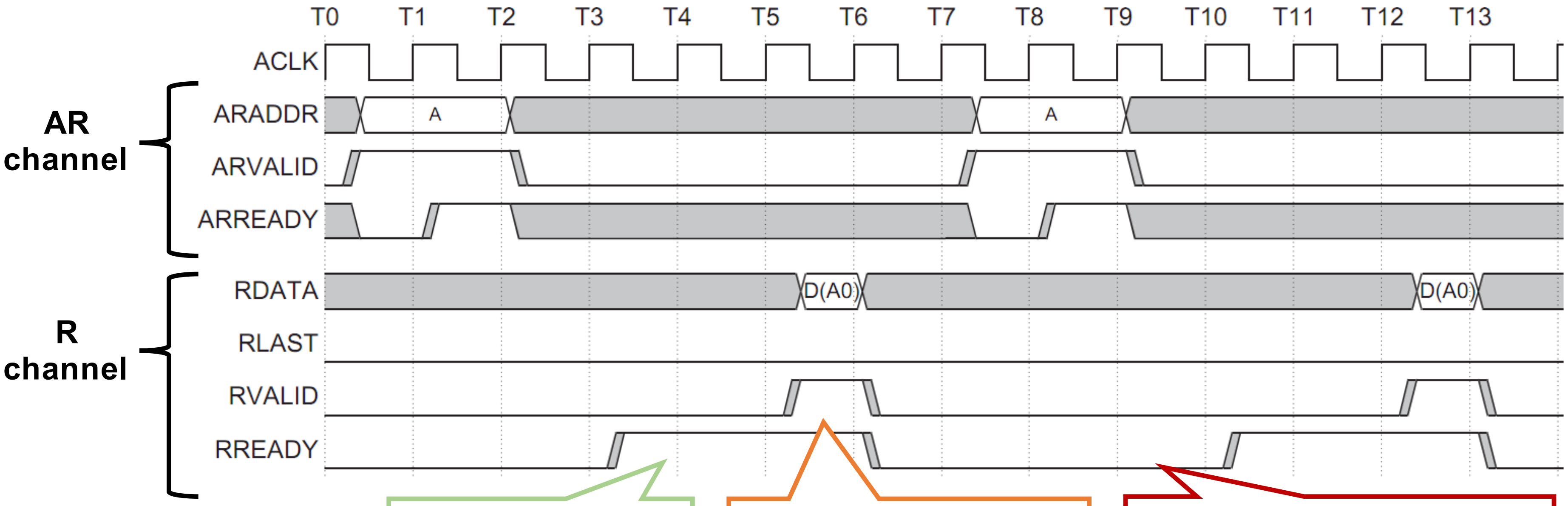


Write address channel	Write data channel	Write response channel	Read address channel	Read data channel
AWVALID	WVALID	BVALID	ARVALID	RVALID
AWREADY	WREADY	BREADY	ARREADY	RREADY
AWADDR	WDATA	BRESP	ARADDR	RDATA
AWPROT	WSTRB		ARPROT	RRESP

AXI4-Lite table and information from [3]

AXI4-Lite Read Example

Example of AXI4-Lite read, need to specify address for each data transfer



Ready to receive data

Data is valid, and received

Not ready to receive more data

Share: bit.ly/cloudfpga

Modified waveform from Xilinx manuals

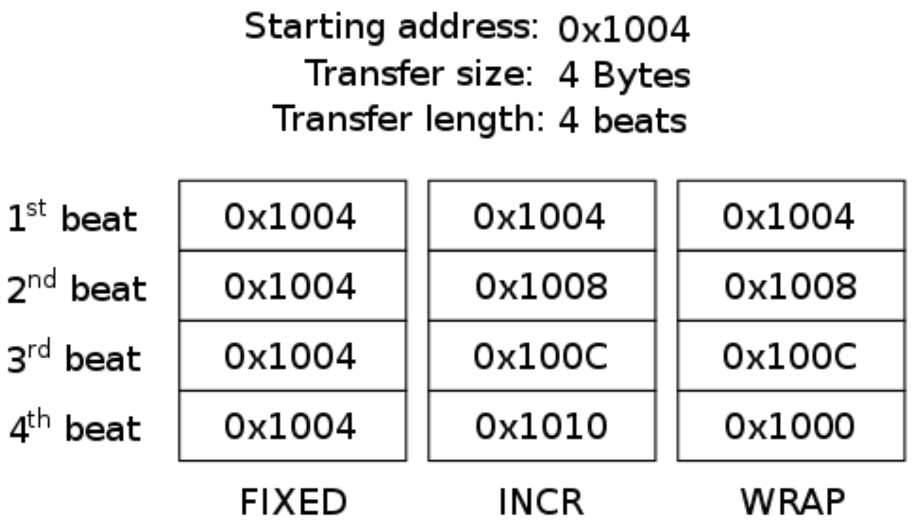
AXI4 (Regular)

Main advantage of AXI4 over AXI4-Lite is that it supports bursts

- Allows multiple data transfers per single request
- Save on addressing overhead, better bandwidth

- Three burst types are supported
  - FIXED
  - INCR
  - WRAP

- Burst addressing specifies where each read/write should go to



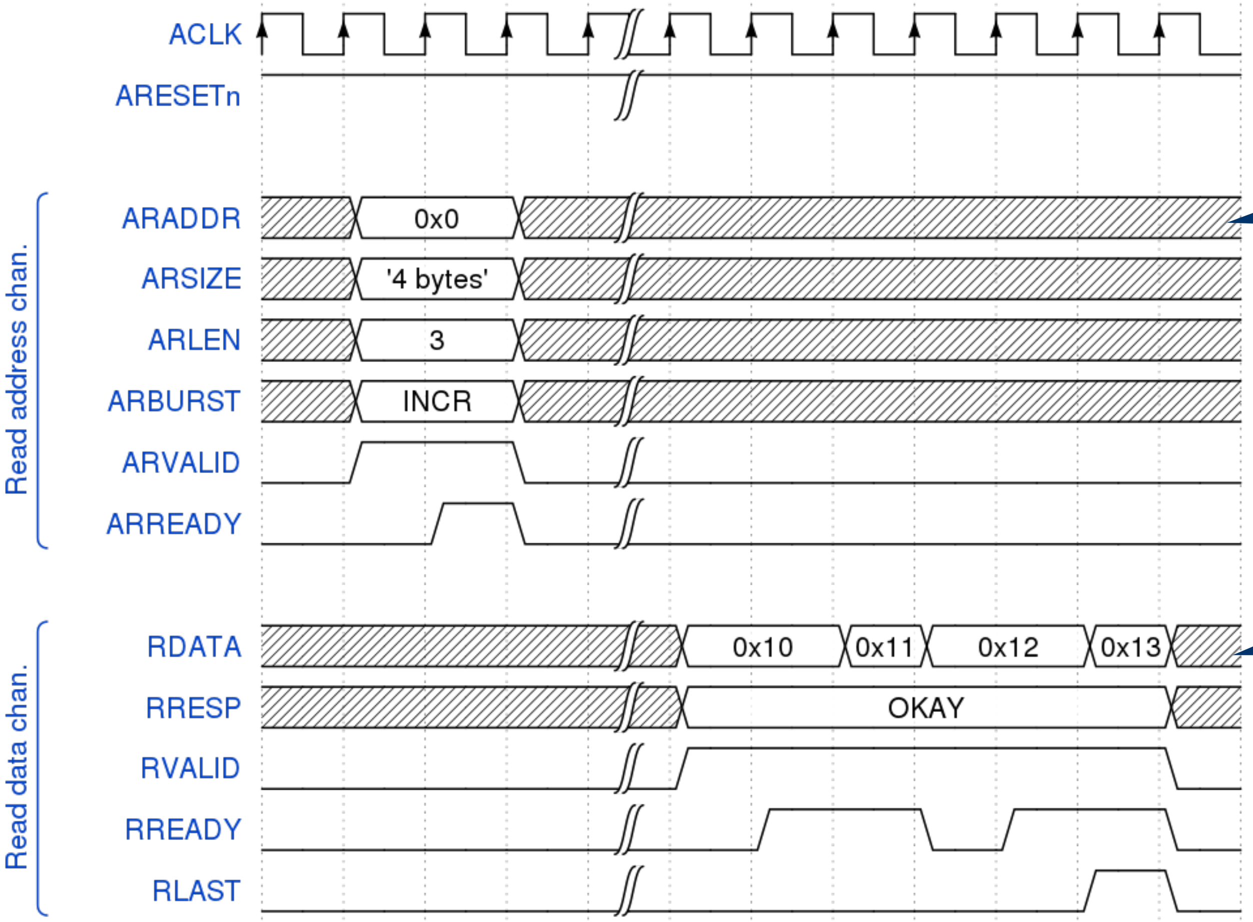
Starting address: 0x1004
Transfer size: 4 Bytes
Transfer length: 4 beats

	FIXED	INCR	WRAP
1st beat	0x1004	0x1004	0x1004
2nd beat	0x1004	0x1008	0x1008
3rd beat	0x1004	0x100C	0x100C
4th beat	0x1004	0x1010	0x1000

Burst image from: https://en.wikipedia.org/wiki/Advanced_eXtensible_Interface#/media/File:AXI_Bursts.svg

AXI4 Read Example

Example read:



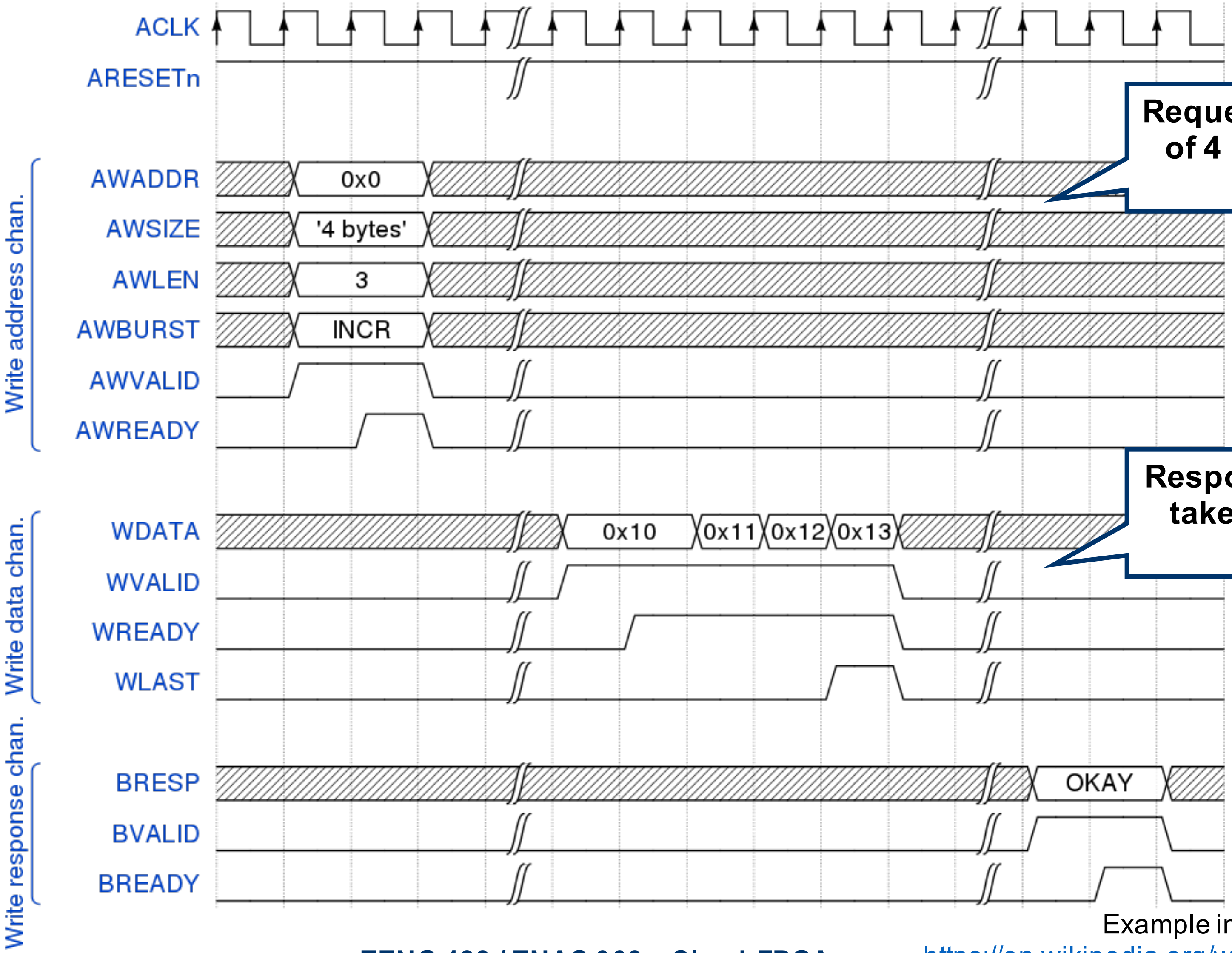
Request 4 transfers (ARLEN + 1) of 4 bytes (32 bits) each from address 0x0

Respond with 4 transfers, some take longer as the receiver is not ready

Example image from: https://en.wikipedia.org/wiki/Advanced_eXtensible_Interface#/media/File:AXI_read_transaction.svg

AXI4 Write Example

Example write:



Request 4 transfers (ARLEN + 1) of 4 bytes (32 bits) each from address 0x0

Respond with 4 transfers, some take longer as the receiver is not ready

Example image from: https://en.wikipedia.org/wiki/Advanced_eXtensible_Interface#/media/File:AXI_write_transaction.svg

AXI4 Stream

- **AXI4** is for memory mapped interfaces and allows burst of up to 256 data transfer cycles with just a single address phase

- **AXI4-Lite** is a light-weight, single transaction memory mapped interface. It has a small logic footprint and is a simple interface to work with both in design and usage

- **AXI4-Stream** removes the requirement for an address phase altogether and allows unlimited data burst size
  - AXI4-Stream interfaces and transfers do not have address phases and are therefore not considered to be memory-mapped.

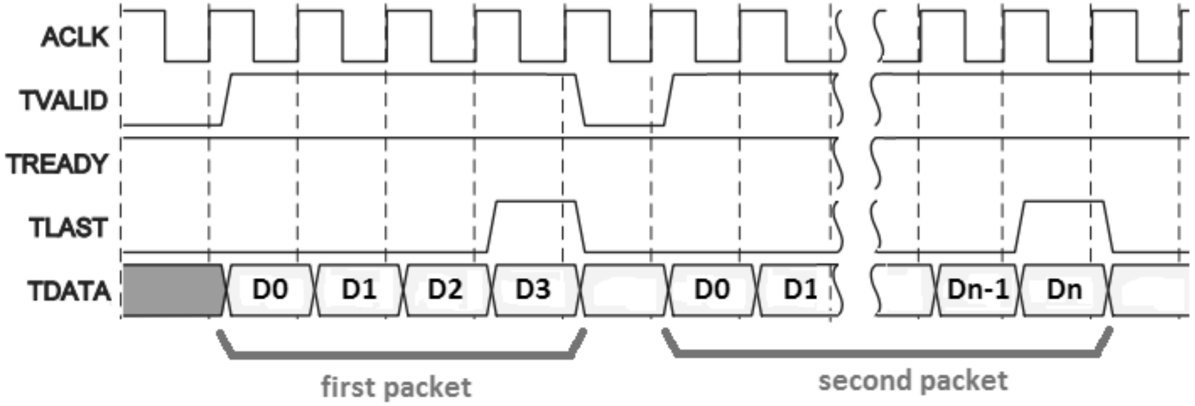
  - The AXI4-Stream protocol is used for applications that typically focus on a data-centric and data-flow paradigm where the concept of an address is not present or not required. Each AXI4-Stream acts as a single unidirectional channel for a handshake data flow

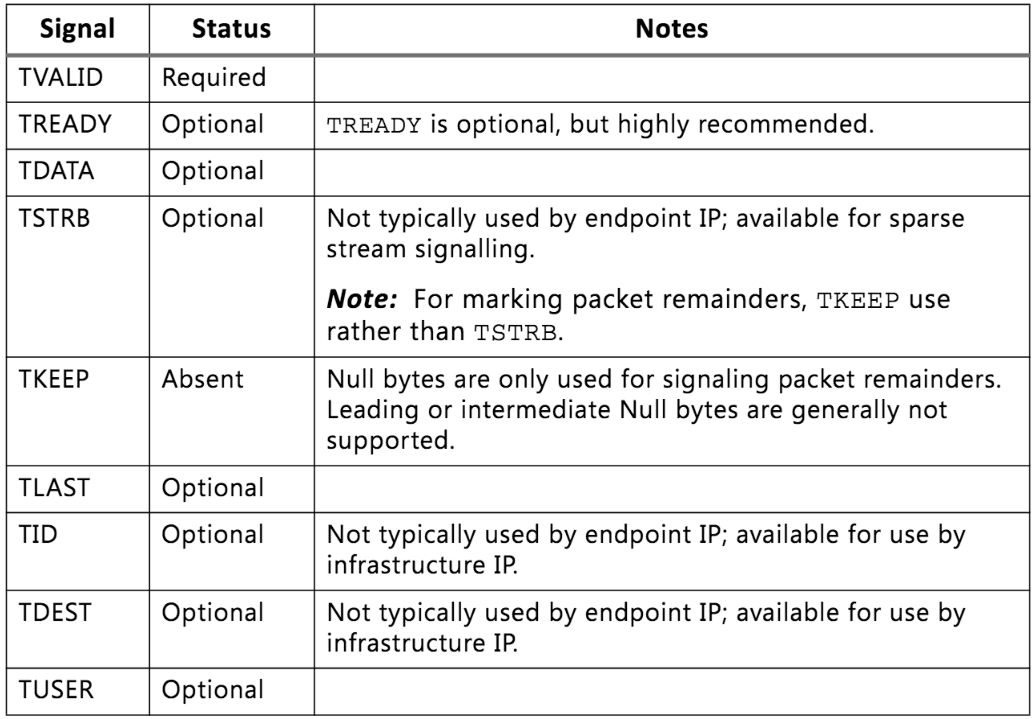
AXI4 information from [1]

AXI4 Stream Signals

The stream protocol minimizes overhead by removing need for addressing

- Bus signals indicate when data is available, TVALID
- Receiver can optionally specify ready, TREADY
- Data is sent using TDATA
- Signal end of packet of data with TLAST





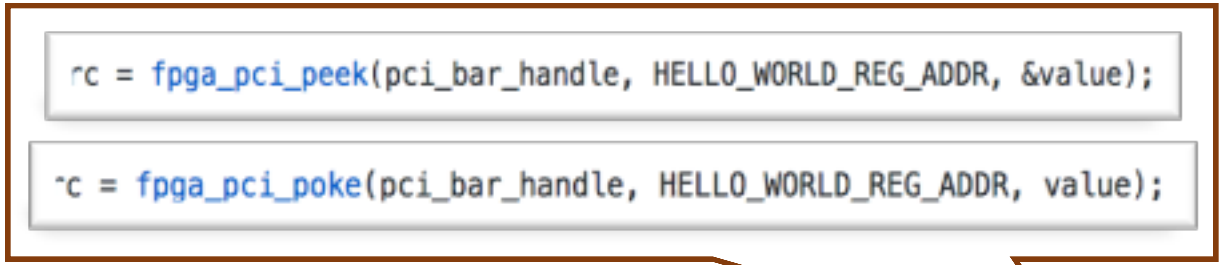
Signal	Status	Notes
TVALID	Required	
TREADY	Optional	`TREADY` is optional, but highly recommended.
TDATA	Optional	
TSTRB	Optional	Not typically used by endpoint IP; available for sparse stream signalling. ***Note:*** For marking packet remainders, `TKEEP` use rather than `TSTRB`.
TKEEP	Absent	Null bytes are only used for signaling packet remainders. Leading or intermediate Null bytes are generally not supported.
TLAST	Optional	
TID	Optional	Not typically used by endpoint IP; available for use by infrastructure IP.
TDEST	Optional	Not typically used by endpoint IP; available for use by infrastructure IP.
TUSER	Optional	

AXI4 Stream signals table from [1],
Left image from:
https://fpgasite.blogspot.com/2017/07/xilinx-axi-stream-tutorial-part-1.html

Communicating with Cloud FPGA Servers via AXI

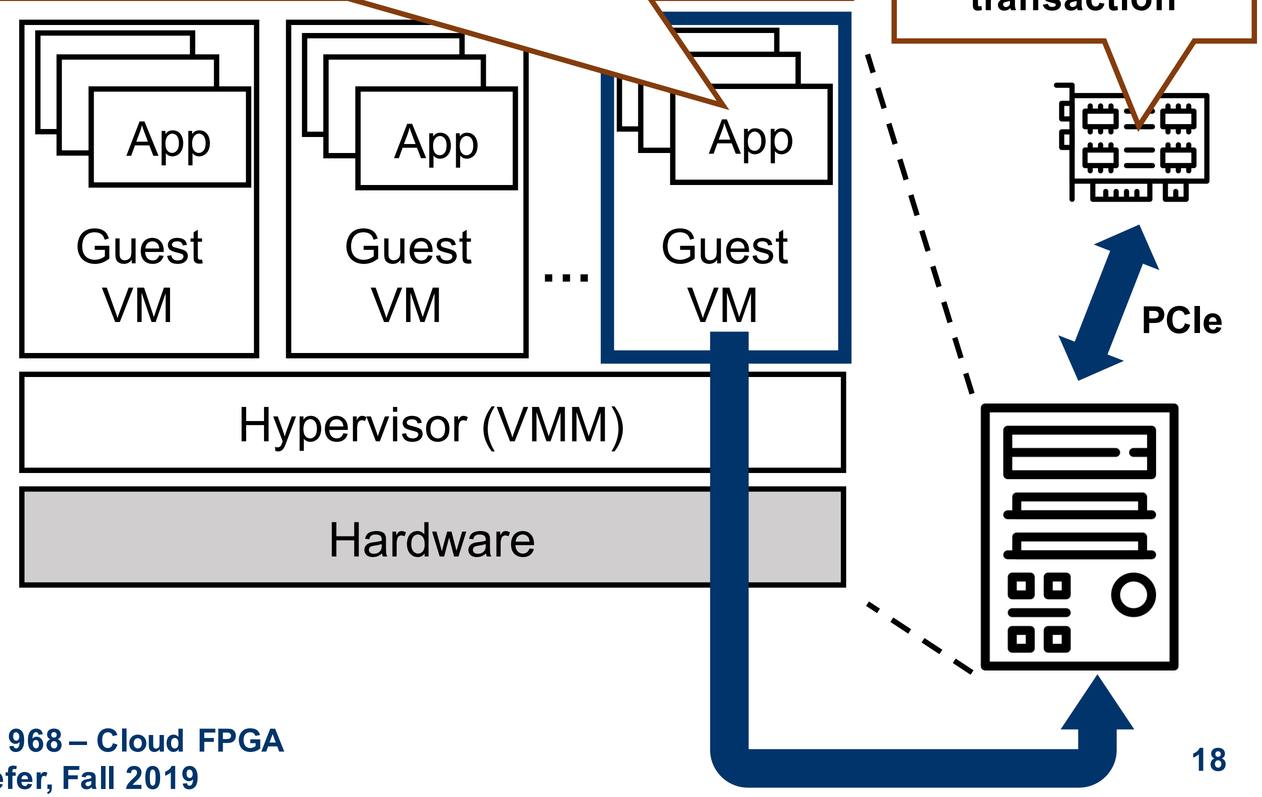
peek(), poke() and AXI4-Lite

- The software libraries (SDK) provides means to read and write data to the FPGA
- **peek()** and **poke()** are most basic ways to read or write data
- They are triggered by software and result in AXI read or write request to show up



```
rc = fpga_pci_peek(pci_bar_handle, HELLO_WORLD_REG_ADDR, &value);
```

```
c = fpga_pci_poke(pci_bar_handle, HELLO_WORLD_REG_ADDR, value);
```



Other communication ways:

- DMA, uses AXI4
- FPGA-to-FPGA, uses AXI4-Stream

Share: bit.ly/cloudfpga

References

1. “AXI Reference Guide, UG761 (v14.3) November 15, 2012” Available at: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf
2. “AWS Shell Interface Specification, v1.4.5” Available at: https://github.com/aws/aws-fpga/blob/master/hdk/docs/AWS_Shell_Interface_Specification.md
3. “Advanced eXtensible Interface” Wikipedia, The Free Encyclopedia. Available at: https://en.wikipedia.org/wiki/Advanced_eXtensible_Interface