# ESE5320: System-on-a-Chip Architecture

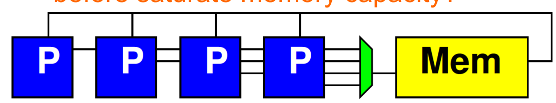
Day 11: October 9, 2023
Data Movement
(Interconnect, DMA)



1

## Preclass 1

- N processors
- Each: 1 read, 10 cycle compute, 1 write
- Memory: 1 read or write per cycle
- How many processors can support before saturate memory capacity?






2

## Schedule Memory Port

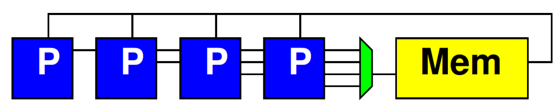
| 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P1.1 write | P1.2 read | P2.1 write | P2.2 read | P3.1 write | P3.2 read | P4.1 write | P4.2 read | P5.1 write | P5.2 read | P6.1 write | P6.2 read | P1.2 write | P1.3 read |
| | | P1 compute f on 2$^{nd}$ iteration | | | | | | | | | | | |
| | | | | P2 compute f on 2$^{nd}$ iteration | | | | | | | | | |



3

## Bottleneck

- Sequential access to a common memory can become the bottleneck






4

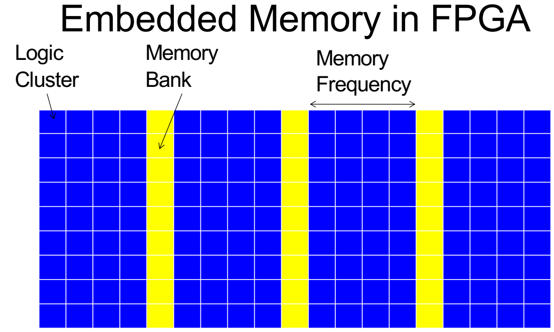
## Previously

- Want data in small memories
  - Low latency, high bandwidth
- FPGA has many memories all over fabric



5

## Embedded Memory in FPGA




ZU3EG (Ultra96) has 216 36Kb BRAMs
VU9P (Amazon F1) has 2,160



6

# Previously

- Want data in small memories
  - Low latency, high bandwidth
- FPGA has many memories all over fabric
- Want C arrays in small memories
  - Partitioned so can perform enough reads (writes) in a cycle to avoid memory bottleneck



7

# Today

- Interconnect Infrastructure (Part 1)
- Peripherals (Part 2)
- Data Movement Threads (Part 3)
- DMA -- Direct Memory Access (Part 4)



8

# Message

- Need to move data
- Often use shared interconnect to make physical connections
- Useful to move data as separate thread of control
  - Dedicating a processor is inefficient
  - Useful to have dedicated data-movement hardware: Direct Memory Access (DMA)



9

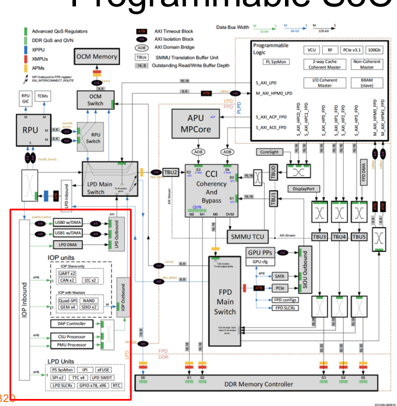
# Term: Peripheral

- "On the edge (or perhiphery) of something"
- Peripheral device – device used to put information onto or get information off of a computer
  - E.g.
    - Keyboard, mouse, modem, USB flash drive, …



10

# Programmable SoC




UG1085 Xilinx UltraScale Zynq TRM (p27)



11

# Memory and I/O Organization

- Architecture contains
  - Large memories
    - For density, necessary sharing
  - Small memories local to compute
    - For high bandwidth, low latency, low energy
  - **Peripherals** for I/O
- Need to move data
  - Among memories and I/O
    - Large to small and back
    - Among small
    - From Inputs, To Outputs



12

## How move data?

- Abstractly, using stream links.
- Connect stream between producer and consumer.
- Ideally: dedicated wires



13

## Dedicated Wires?

- What might prevent us from having dedicated wires between all communicating units?



14

## Making Connections

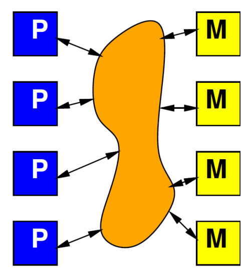
- Cannot always be dedicated wires
  - Programmable
  - Wires take up area
  - Don't always have enough traffic to consume the bandwidth of point-to-point wire
  - May need to serialize use of resource
    - E.g. one memory read per cycle
  - Source or destination may be sequentialized on hardware



15

## Model

- Programmable, possibly shared interconnect



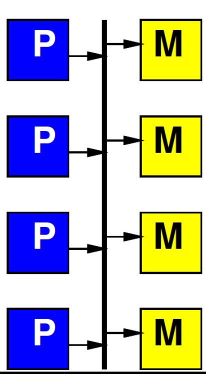



16

## Simple Realization

Shared Bus

- Write to bus with address of destination
- When address match, take value off bus
- Pros?
- Cons?

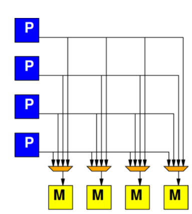





17

## Alternate: Crossbar

- Provide programmable connection between all sources and destinations
- Any destination can be connected to any single source



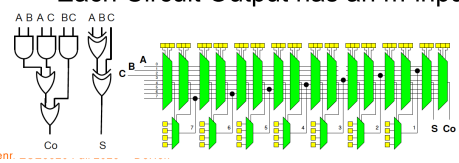



18

## Simplistic FPGA (illustrate possibility)

Day 8

- Every LUT input has a mux
- Every such mux has m=(N+I) inputs
  - An input for each LUT output (N 2-LUTs)
  - An input for each Circuit Input (I Circuit inputs)
- Each Circuit Output has an m-input mux

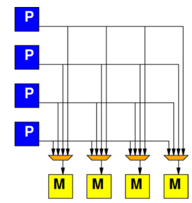





19

## Alternate: Crossbar

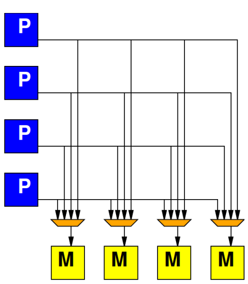
- Provide programmable connection between all sources and destinations
- Any destination can be connected to any single source

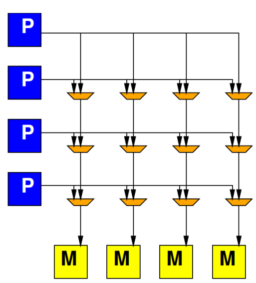





20

## Crossbar



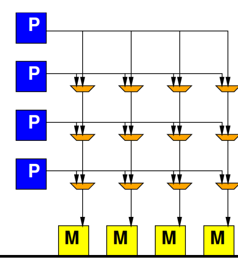






21

## Preclass 2

- K-input, O-output Crossbar
- How many 2-input muxes?

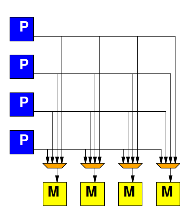





22

## Crossbar

- Provides high bandwidth
  - Minimal blocking
- Costs large amounts of area
  - Grows fast with inputs, outputs






23

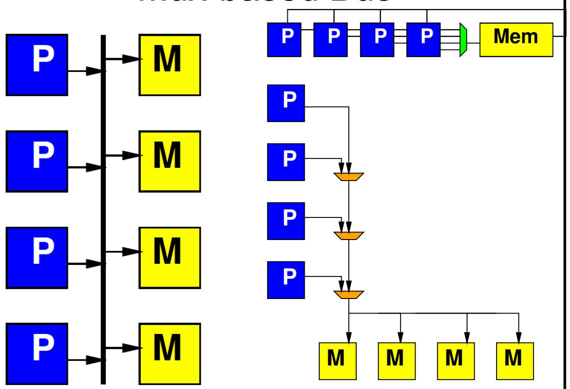
## General Interconnect

- Generally, want to be able to parameterize designs
- Here: tune area-bandwidth
  - Control how much bandwidth provide



24

## Mux-based Bus

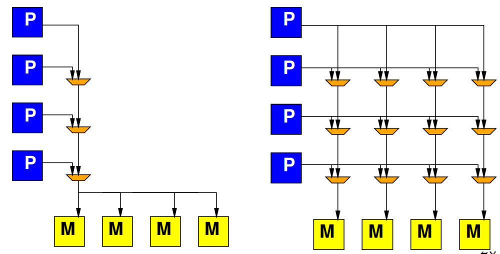





25

## Multiple Busses

- Think of crossbar as one bus per output
- Simple bus is one bus total

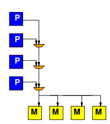


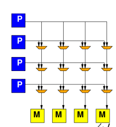



26

## Interconnect

- How might get design points between bus and crossbar?
- How could reduce number
  - Inputs to crossbar?
  - Outputs from crossbar?



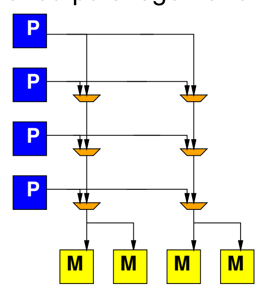




27

## Share Crossbar Outputs

- Group set of outputs together on a bus

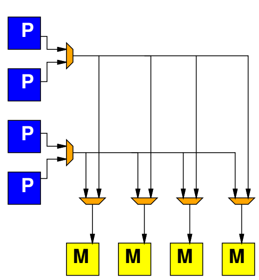





28

## Share Crossbar Inputs

- Group number of inputs together on an input port to crossbar

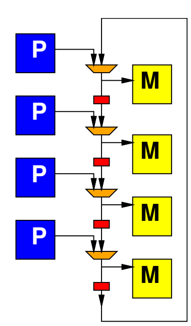





29

## Delay

- Delay proportional to distance
- Pipeline bus to keep cycle time down
  - Take many cycles to travel long distance
  - …but fewer cycles when distance small
- Sometimes call this a ``Ring''

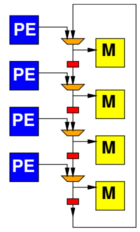


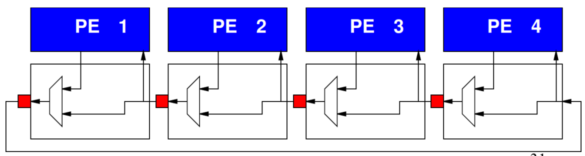



30

# Local Interconnect

- How many cycles from:
  - PE3 to PE2
  - PE3 to PE1
  - PE3 to PE4

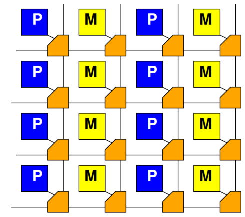


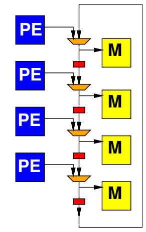



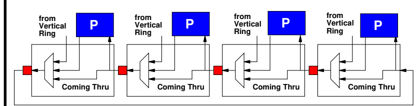



31

# Mesh



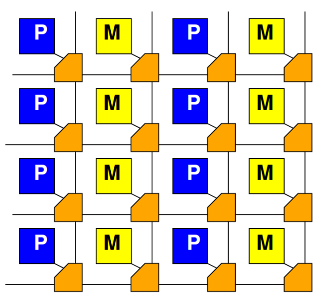




Mesh is a set
Of Horizontal
And Vertical
Rings with
Option to switch
H to V



32

# Mesh

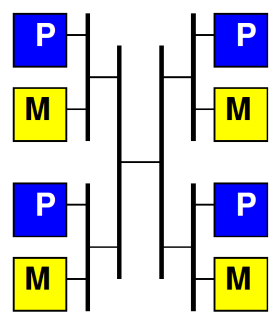



- Delay Proportional to distance in 2D



33

# Hierarchical Busses

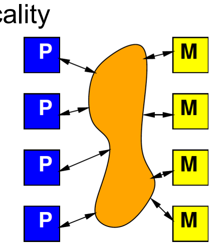





34

# Interconnect

- Will need an infrastructure for programmable connections
- Rich design space to tune area-bandwidth-locality






35

# Peripherals

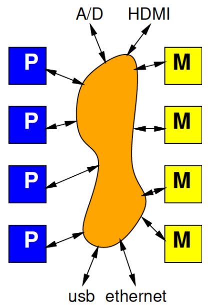
Part 2



36

## Input and Output

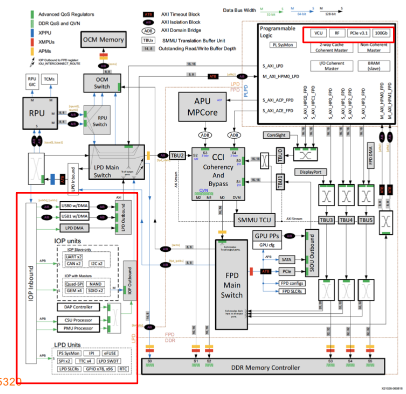
- Typical SoC has I/O with external world
  - Sensors
  - Actuators
  - Keyboard/mouse, display
  - Communications
- Also accessible from interconnect








## Programmable SoC



UG1085 Xilinx UltraScale Zynq TRM (p27)







## High Speed I/O

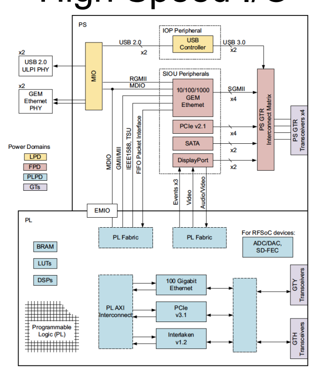


*Figure 1-3:* **High-Speed Serial I/O Block Diagram**

UG1085 Xilinx UltraScale Zynq TRM (p27)







## Masters and Slaves

- Two kinds of entities on interconnect
- Master – can initiate requests
  - E.g. **processor** that can perform a read or write
- Slaves – can only respond to requests
  - E.g. **memory** that can return the read data from a read request

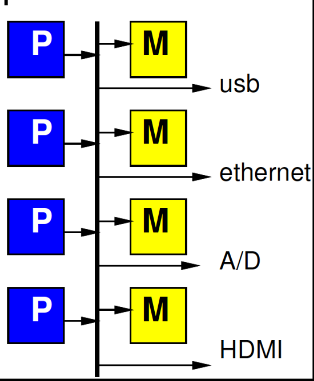






## Simple Peripheral Model

- Peripherals are slave devices
  - Masters can read input data
  - Masters can write output data
  - To move data, master (e.g. processor) initiates

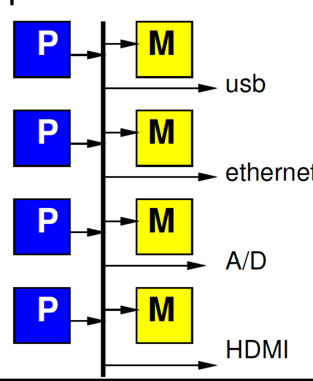







## Simple Peripheral Model

- Peripherals are slave devices
  - Masters can read input data
  - Masters can write output data
  - To move data, master (e.g. processor) initiates
- Demanding processor touch every data item has some negative consequences



# Timing Demands

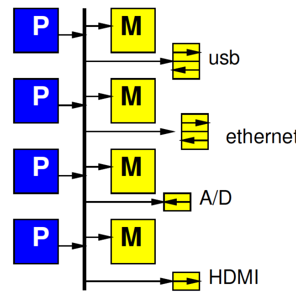
- Must read each input before overwritten
- Must write each output within real-time window
- Must guarantee processor scheduled to service each I/O at appropriate frequency
- How many cycles between 32b input words for 1Gb/s network and 32b, 1GHz processor?
  - Consider input data shifted into register 1b per ns
  - Must read out 32b register before overwritten

Penn ESE5320 Fall 2023 -- DeHon 43

43

# Refine Model

- Give each peripheral local FIFO
- Processor must still move data
- For same input data rate, how does this change requirements and impact?
  - How many cycles can processor run between servicing inputs?




Penn ESE5320 Fall 2023 -- DeHon 44

44

# Long Latency Memory Operations

Part 3

Penn ESE5320 Fall 2023 -- DeHon 45

45

# Day 3

- Large memories are slow
  - Latency increases with memory size
- Distant memories are high latency
  - Multiple clock-cycles to cross chip
  - Off-chip memories even higher latency

Penn ESE5320 Fall 2023 -- DeHon 46

46

# Day 3, Preclass 2

- 10 cycle latency to memory
- If must wait for data return, latency can degrade throughput
- 10 cycle latency + 10 op + (assorted)
  - More than 20 cycles / result

```
for(i=0;i<MAX;i++) {
   in=a[i]; // memory read
   out=f(in); // 10 cycle compute
   b[i]=out;
}
```

Penn ESE5320 Fall 2023 -- DeHon 7

47

# Preclass 3

- Throughput using 3 threads on 3 processors: P1, P2, P3?

```
P1:  for(i=0;i<MAX;i++) Astream.write(a[i]);
P2:  while(1) {Astream.read(aval); Bstream.write(f(aval));}
P3:  for(i=0;i<MAX;i++) Bstream.read(b[i]);
```

Penn ESE5320 Fall 2023 -- DeHon 48

48

# Fetch (Write) Threads

- Potentially useful to move data in separate thread
- Especially when
  - Long (potentially variable) latency to data source (memory)
- Useful to split request/response



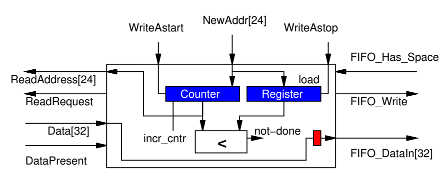
# DMA
# Part 4

Direct Memory Access



# Preclass 4a

```
P1:  for(i=0;i<MAX;i++) Astream.write(a[i]);
```



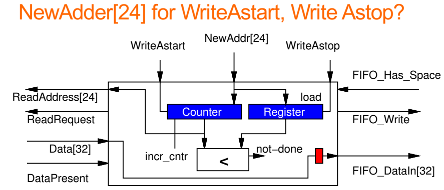

```
int *p;
P1:  for(p=&(a[0]);p<&(a[MAX]);p++) Astream.write(*p);
```



# Preclass 4a

```
P1:  for(i=0;i<MAX;i++) Astream.write(a[i]);
```

NewAdder[24] for WriteAstart, Write Astop?



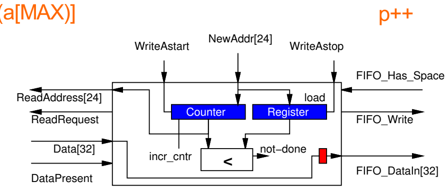

```
int *p;
P1:  for(p=&(a[0]);p<&(a[MAX]);p++) Astream.write(*p);
```



# Preclass 4a

```
P1:  for(i=0;i<MAX;i++) Astream.write(a[i]);
```

p<&(a[MAX)]    p++



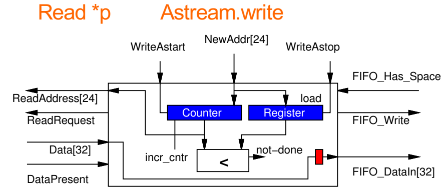

```
int *p;
P1:  for(p=&(a[0]);p<&(a[MAX]);p++) Astream.write(*p);
```



# Preclass 4a

```
P1:  for(i=0;i<MAX;i++) Astream.write(a[i]);
```

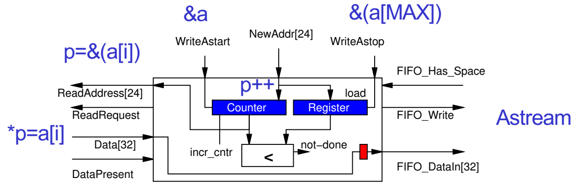
Read *p    Astream.write




```
int *p;
P1:  for(p=&(a[0]);p<&(a[MAX]);p++) Astream.write(*p);
```

## Preclass 4a

```
P1:  for(i=0;i<MAX;i++) Astream.write(a[i]);
```

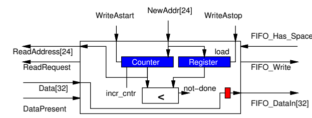



```
int *p;
P1:  for(p=&(a[0]);p<&(a[MAX]);p++) Astream.write(*p);
```



55

## Control Logic

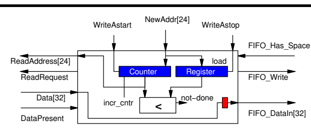



- FIFO_Write =
- ReadRequest =
- Incr_cnt =



56

## Control Logic



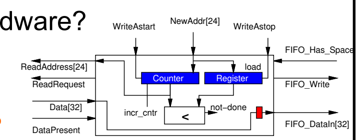

- FIFO_Write = DataPresent & FIFO_Has_Space
- ReadRequest = not-done
- Incr_cnt = not-done
- (AckDataPresent = FIFO_Write)



57

## Preclass 4

- How much hardware?
  - Counter bits?
  - Registers?
  - Comparators?
  - Control Logic gates? (4cd)




- Compare to MicroBlaze
  - small RISC Processor optimized for Xilinx
  - minimum config 630 6-LUTs



58

## Observe

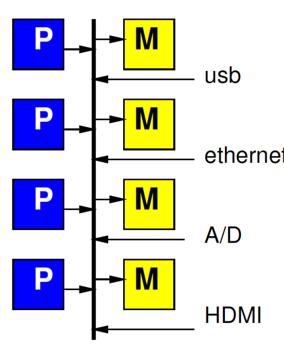
- Modest hardware can serve as data movement thread
  - Much less hardware than a processor
  - Offload work from processors
- Small hardware allow peripherals to be **master** devices on interconnect



59

## DMA

- Direct Memory Access (DMA)
- “Direct” – inputs (and outputs) don’t have to be indirectly handled by the processor between memory and I/O
- I/O unit directly reads/write memory

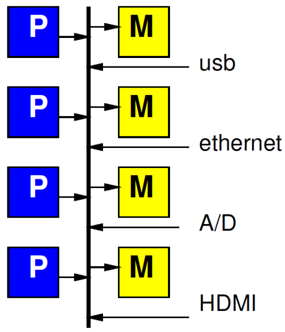





60

## DMA

- Direct Memory Access (DMA)
- Peripheral as Master
  - Can write **directly** into (read from) memory
  - Saves processor from copying
  - Reduces demand to schedule processor to service






61

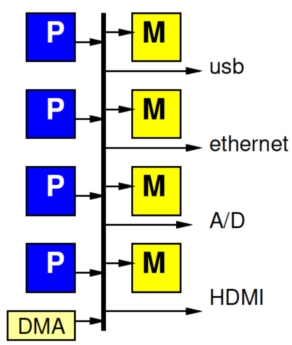
## DMA Engine

- Data Movement Thread
  - Specialized Processor that moves data
  - Another heterogeneous processing engine
- Act independently (hence thread)
- Implement data movement
- Can build to move data between memories (Slave devices)
- E.g., Implement P1, P3 in Preclass 3



62

## DMA Engine






63

## Programmable DMA Engine

- What copy from?
- How much?
- Where copy to?
- Stride?
- What size data?
- Loop?
- Transfer Rate?



64

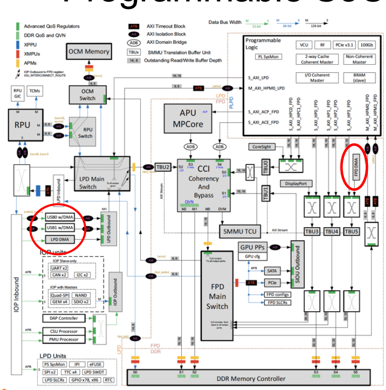
## Multithreaded DMA Engine

- One copy task not necessarily saturate bandwidth of DMA Engine
- Share engine performing many transfers (channels)
- Separate transfer state for each
  - Hence thread (or channel)
- Swap among threads
  - Simplest: round-robin:
    - 1, 2, 3, .. K, 1, 2, 3, .. K, 1, ….



65

## Programmable SoC




UG1085 Xilinx UltraScale Zynq TRM (p27)



66

## Hardwired and Programmable

- Zynq has hardwired DMA engine
  - 8 channels
- Also build data movement engines (Data Movers) in FPGA fabric

UG1085 Xilinx UltraScale Zynq TRM Ch. 10 (p519)

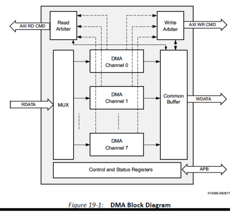


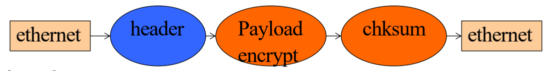

Figure 19-1: DMA Block Diagram



67

## Example

- Networking Application: inline packet encrypt




- Header on processor
- Payload (encrypt, checksum) on FPGA
- DMA from ethernet→main memory
- DMA main memory→BRAM
- Stream between payload components
- DMA from encrypted packet and chksum to ethernet out



68

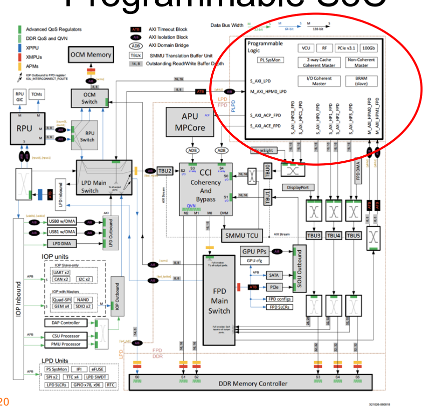
## AXI

- Advanced eXtensible Interface
  - Originally developed by ARM
  - On-chip communication bus standard
  - Particular communication protocol
- Full AXI
  - Read/write operations with bursts
    - Burst = single address + length
      - Large, contiguous block of memory
  - Separate send/receive data channels
- AXI-S – for streaming connections
- AXI-lite – simpler, not burst



69

## Programmable SoC

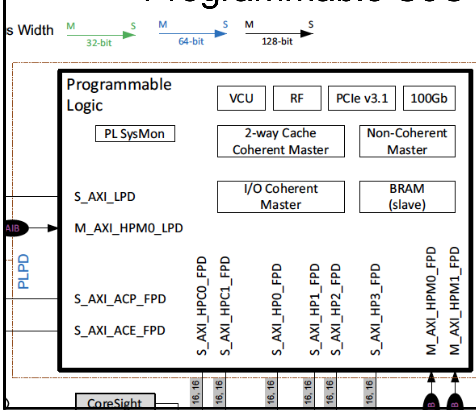



UG1085 Xilinx UltraScale Zynq TRM (p27)



70

## Programmable SoC




AXI Channels
128b wide
@ 333 MHz
=5.3GB/s

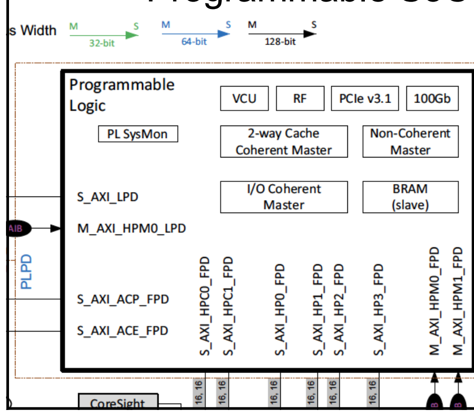
Separate Data in Data out Paths

5.3GB/s Each way



71

## Programmable SoC




AXI Channels
128b wide
@ 333 MHz
=5.3GB/s
Per direction
Per channel

S_ act as master for PL (connects to slave ports in PS)

M_ as slave for PL



72

## DMA in Vitis

- Vitis/OpenCL demands that we write code to perform DMA of data to and from accelerators in FPGA fabric
- We will see specifics on Monday
- Have some options to control
  - With pragmas
  - With choice of data and burst sizes
  - Explore HW6



73

## Big Ideas

- Need to move data
- Shared Interconnect to make physical connections – can tune area/bw/locality
- Useful to
  - move data as separate thread of control
  - Have dedicated data-movement hardware: DMA



74

## Admin

- Feedback
- Midterm on Wednesday
  - Here, classtime
- HW6 coming soon
- Fall Break on Friday
  - No homework do this week



75