ESE532:
System-on-a-Chip Architecture

Day 11: October 7, 2019

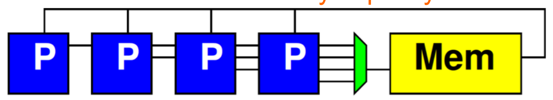
Data Movement

(Interconnect, DMA)

Penn

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?

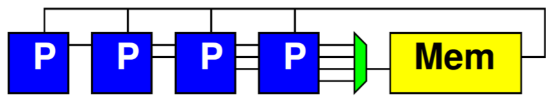


Schedule Memory Port

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 2nd iteration											
				P2 compute f on 2nd iteration									

Bottleneck

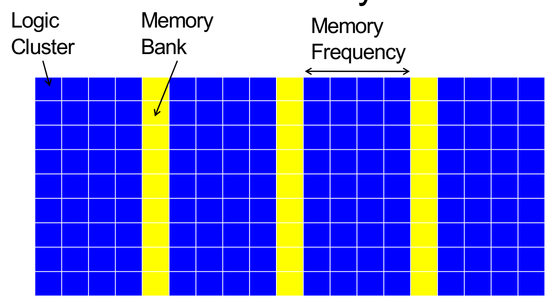
- Sequential access to a common memory can become the bottleneck



Previously

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

Embedded Memory in FPGA



ZU3EG (Ultra96) has 216 36Kb BRAMs

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

Today

- Interconnect Infrastructure
- Data Movement Threads
- Peripherals
- DMA -- Direct Memory Access

Message

- Need to move data
- 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)

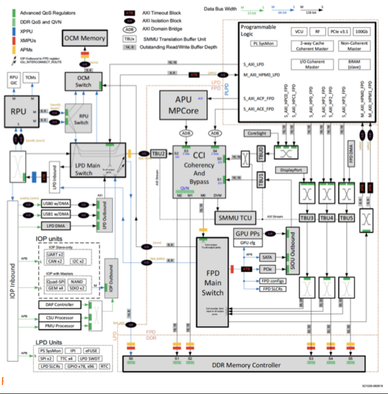
Memory and I/O Organization

- 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

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, …

Programmable SoC



UG1085 Xilinx UltraScale Zynq TRM (p27)

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

How move data?

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

Dedicated Wires?

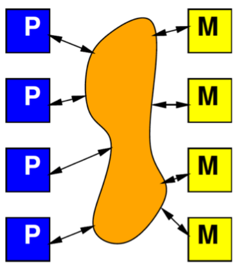
- What might prevent us from having dedicated wires between all communicating units?

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

Model

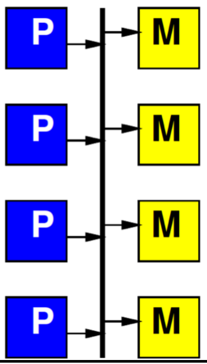
- Programmable, possibly shared interconnect



Simple Realization

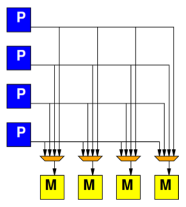
Shared Bus

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



Alternate: Crossbar

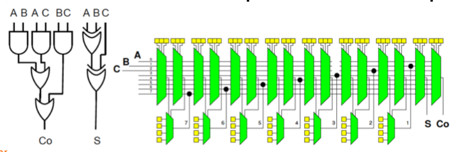
- Provide programmable connection between all sources and destinations
- Any destination can be connected to any single source



Simplistic FPGA (illustrate possibility)

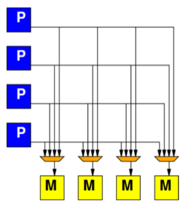
Day 8

- 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

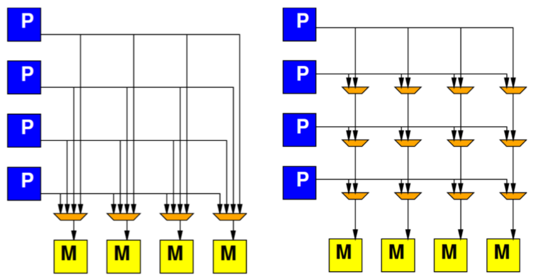


Alternate: Crossbar

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

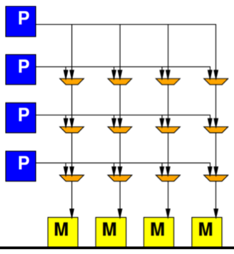


Crossbar



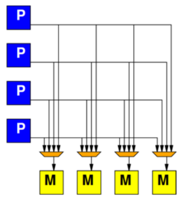
Preclass 2

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



Crossbar

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



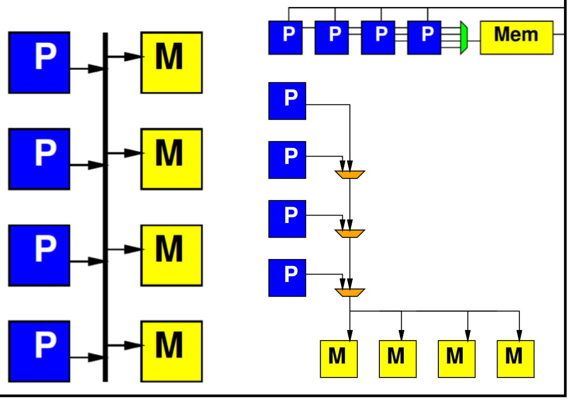
General Interconnect

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

Interconnect

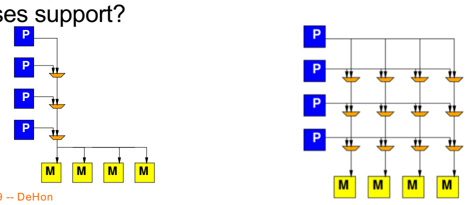
- How might get design points between bus and crossbar?
- How could reduce number
  - Inputs to crossbar?
  - Outputs from crossbar?

Mux-based Bus



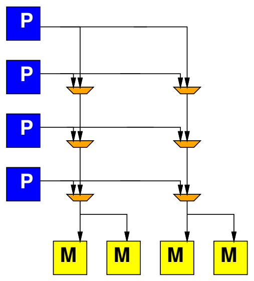
Multiple Busses

- Think of crossbar as one bus per output
- Simple bus is one bus total
- In between,
  - How many simultaneous busses support?



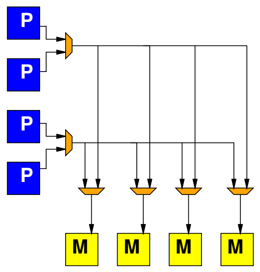
Share Crossbar Outputs

- Group set of outputs together on a bus



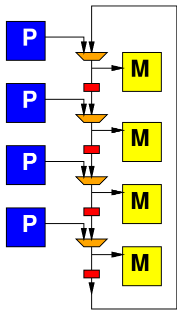
Share Crossbar Inputs

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



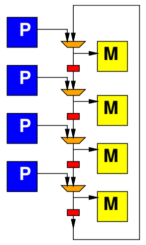
Delay

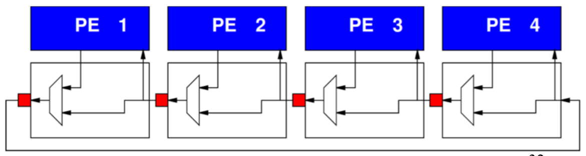
- Delay proportional to distance
- Pipeline bus to keep cycle time down
  - Take many cycles to travel long distance
  - …but fewer cycles when distance small



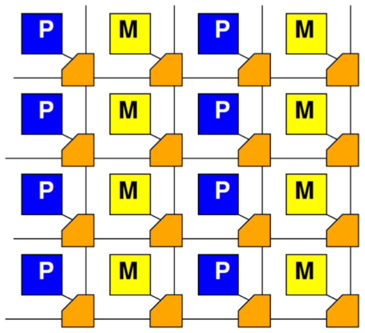
Local Interconnect

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

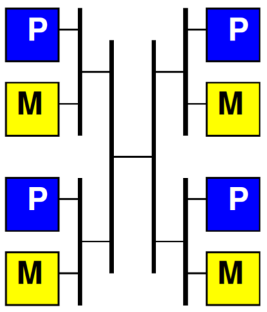




Mesh

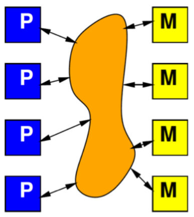


Hierarchical Busses



Interconnect

- Will need an infrastructure for programmable connections
- Rich design space to tune area-bandwidth-locality
  - Will explore more later in course



Long Latency Memory Operations

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

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;
 }
```

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]);
```

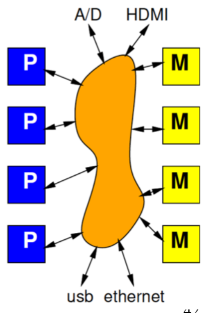
Fetch (Write) Threads

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

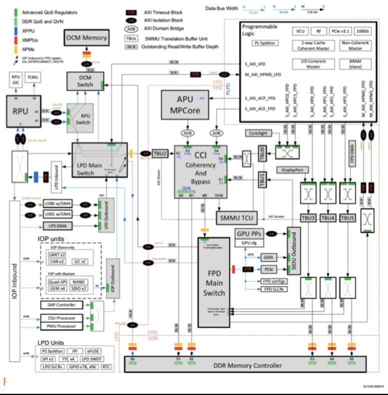
Peripherals

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

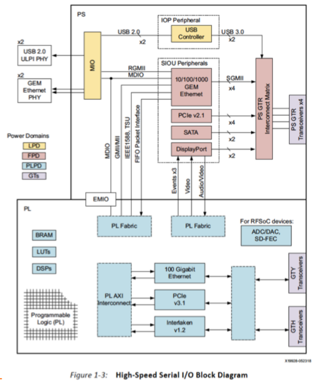


Figure 1-3: High-Speed Serial I/O Block Diagram

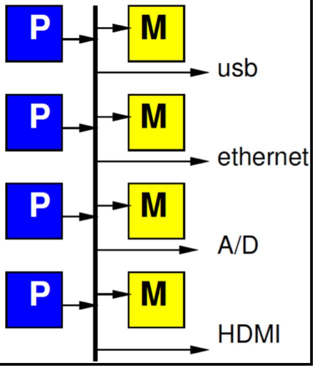
UG1085
Xilinx
UltraScale
Zynq
TRM
(p27)

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

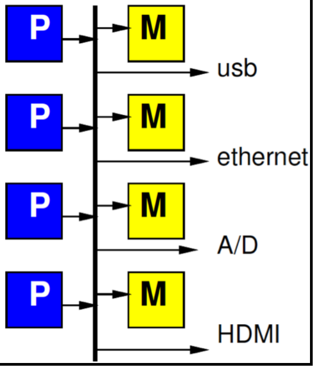
Simple Peripheral Model

- 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

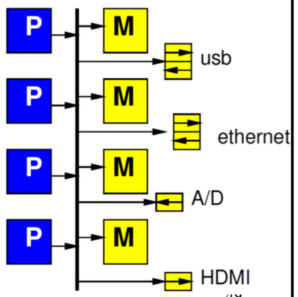


Timing Demands

- 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

Refine Model

- Give each peripheral local FIFO
- Processor must still move data
- How does this change requirements and impact?

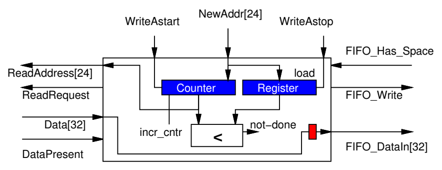


DMA

Direct Memory Access

Preclass 4a

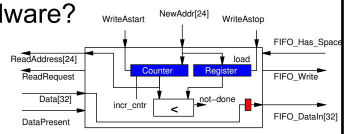
```
P1:  for(i=0;i<MAX;i++) Astream.write(a[i]);
```



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

Preclass 4

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



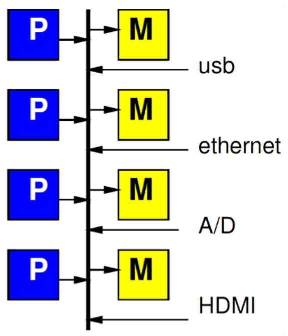
- Compare to MicroBlaze
  - small RISC Processor optimized for Xilinx
  - minimum config 630 6-LUTs

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

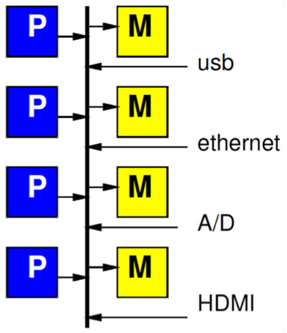
DMA

- 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



DMA

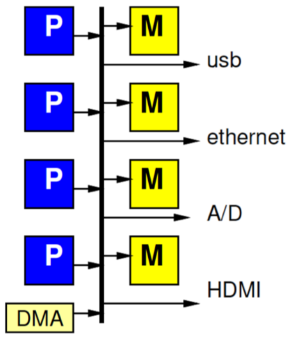
- Direct Memory Access (DMA)
- Peripheral as Master
  - Can write **directly** into (read from) memory
  - Saves processor from copying
  - Reduces demand to schedule processor to service



DMA Engine

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

DMA Engine



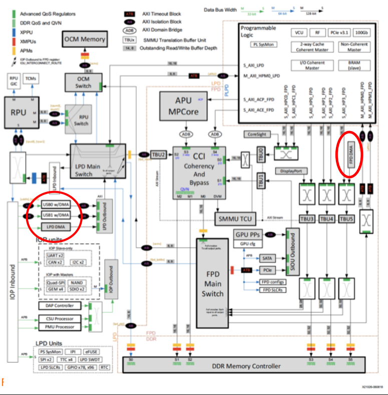
Programmable DMA Engine

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

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, ….

Programmable SoC



UG1085 Xilinx UltraScale Zynq TRM (p27)

Hardwired and Programmable

- Zynq has hardwired DMA engine
  - 8 channels
- Can also add data movement engines (Data Movers) in FPGA fabric

UG1085
Xilinx
UltraScale
Zynq
TRM
Ch. 10
(p519)

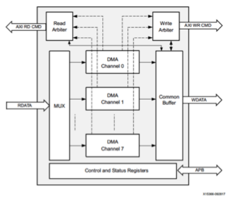
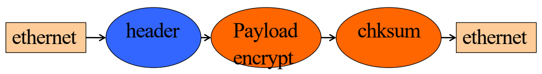


Figure 19-1: **DMA Block Diagram**

Example

- Networking Application



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

Automation

- SDSoC will automatically take care of DMA of memory to and from accelerators in FPGA fabric
  - Inserting logic, programming DMA
- Mostly need to be aware is happening
  - Understand impacts on performance
- Have some options to control
  - With pragmas
  - With choice of data sizes
  - Explore HW6

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

Admin

- Midterm Wednesday, here, classtime
  - Review Tuesday evening office hours
- Day 13 (Monday) reading
  - Chapter nine of *Parallel Programming for FPGAs* (available on web)
  - DRAM reading if not read on Day 3
- *HW6 out soon??*
  - We learned quite a bit putting it together
  - Hopefully, assignment passes on to you