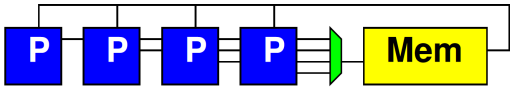
ESE532: System-on-a-Chip Architecture

Day 11: October 9, 2017
Data Movement
(Interconnect, DMA)

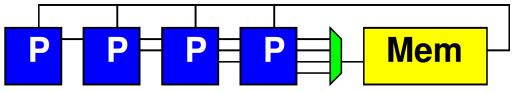
Preclass 1

- N processors
- Each: 1 read, 10 cycle, 1 write
- Memory: 1 read or write per cycle
- How many processors can support?



Bottleneck

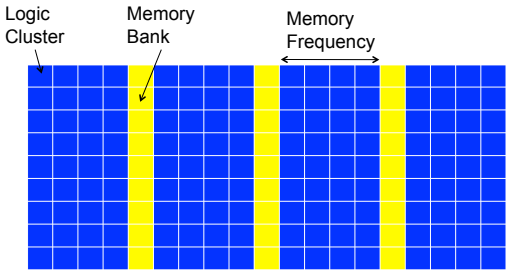
- Sequential access to a common memory can become the bottleneck



Previously

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

Embedded Memory in FPGA



XC7Z020 (Zed Board) has 140 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

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: 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

How move data?

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

Dedicated Wires?

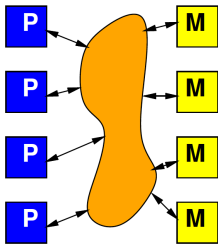
- Why might we not be able to have dedicated wires?

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

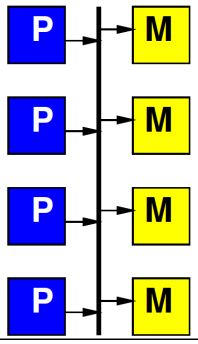
- Programmable, possibly shared interconnect



Simple Realization

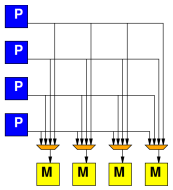
Shared Bus

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

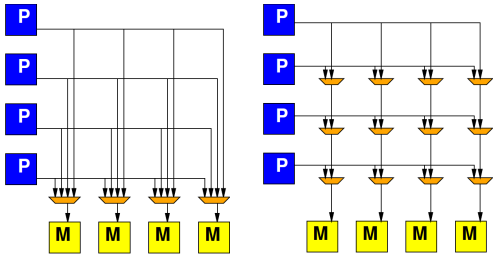


Alternate: Crossbar

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

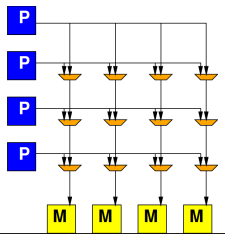


Crossbar



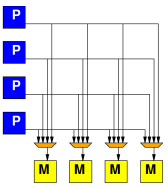
Preclass 2

- 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

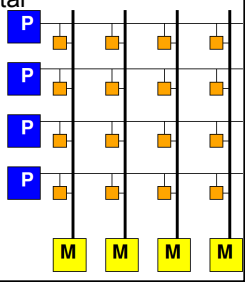
- Generally, want to be able to parameterize designs
- Here: tune area-bandwidth
  - Control how much bandwidth provide

Interconnect

- How might get design points between bus and crossbar?

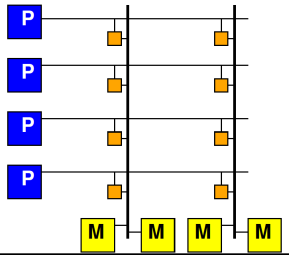
Multiple Busses

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



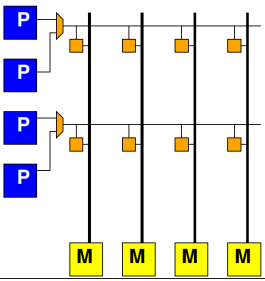
Share Crossbar Outputs

- Group set of outputs together on a bus



Share Crossbar Inputs

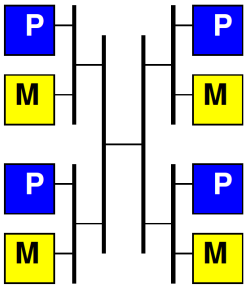
- Group number of inputs together on an input port to crossbar



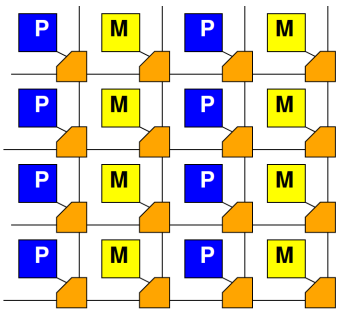
Locality in Interconnect

- How allow physically local items to be closer?

Hierarchical Busses

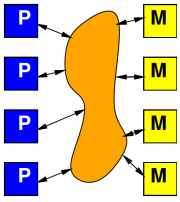


Mesh



Interconnect

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



Masters and Slaves

- Regardless of form, potentially have 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 requset

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?

```
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]);
```

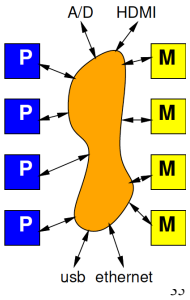

Fetch (Write) Threads

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

Peripherals

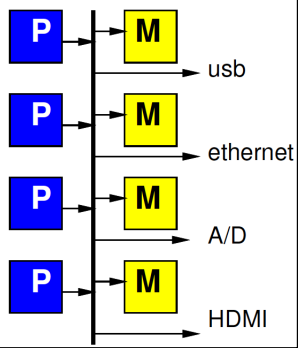
Input and Output

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



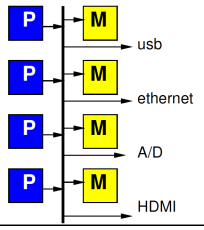
Simple Peripheral Model

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



Simple Model Implications

- What implication to processor grabbing/ moving each input (output) value?

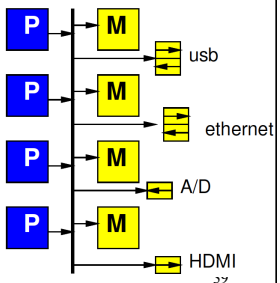


Timing Demands

- 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 inputs for 1Gb/s network and 32b, 1GHz processor?

Refine Model

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



DMA

Preclass 4

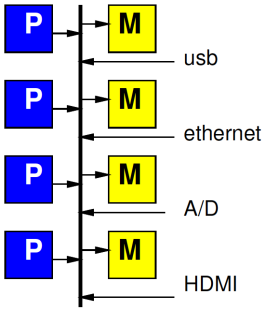
- How much hardware to support fetch thread:
  - Counter bits?
  - Registers?
  - Comparators?
  - Other gates?
- Compare to MicroBlaze
  - (minimum config 630 6-LUTs)

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

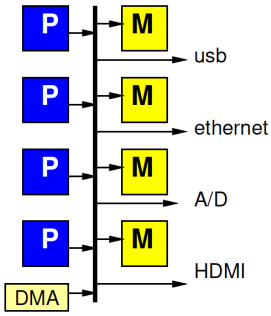
- Direct Memory Access (DMA)
- Peripheral as Master
  - Can write directly into (read from) memory
  - Saves processor from copying
  - Reduces demand to schedule processor to service



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

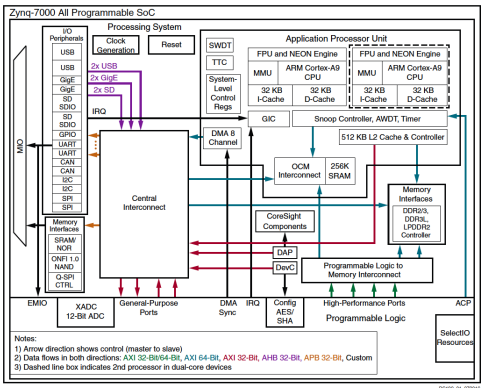


Programmable DMA Engine

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

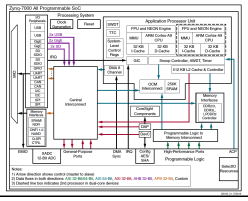
Multithreaded DMA Engine

- 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
  - E.g., round-robin



Hardwired and Programmable

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



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

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

- Day 12
  - DRAM reading if not read on Day 3
- HW5 due Friday