ESE532:
System-on-a-Chip Architecture

Day 7: February 6, 2017
Memory

Penn

Today

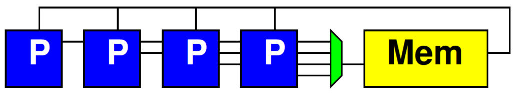
Memory
- Memory Bottleneck
- Memory Scaling
- DRAM
- Memory Organization
- Data Reuse

Message

- Memory bandwidth and latency can be bottlenecks
- Minimize data movement
- Exploit small, local memories
- Exploit data reuse

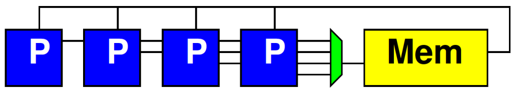
Preclass 1

- 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



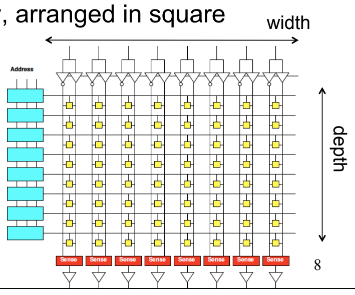
Memory Scaling

On-Chip Delay

- Delay is proportional to distance travelled
- Make a wire twice the length
  - Takes twice the latency to traverse
  - (can pipeline)
- Modern chips
  - Run at 100s of MHz to GHz
  - Take 10s of ns to cross the chip
- What does this say about placement of computations and memories?

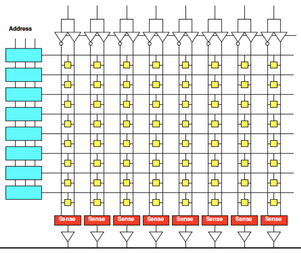
Memory Block

- Linear wire delay with distance
- Assume constant area/memory bit
- N-bit memory, arranged in square
- Width?
- Depth?
- Delay scale N?



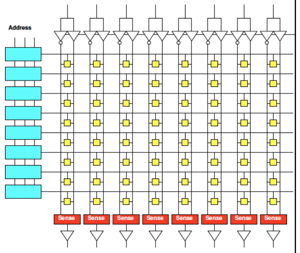
Memory Block Bandwidth

- N-bit memory, arranged in square
- Bits/cycle scale with N?



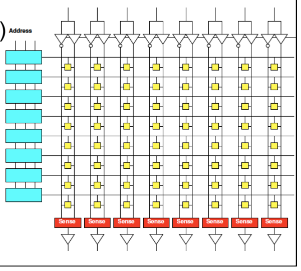
Memory Block Energy

- Assume read full width,
  - Energy scales with N
    - Activate sqrt(N) wires
    - Each wire sqrt(N) long
  - Or, energy/bit scales as sqrt(N)
- Larger memories cost more energy



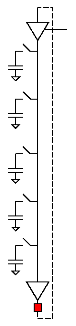
Memory Block Density

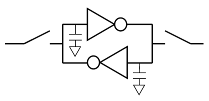
- Address, sense amplifies are large
- Scale slower than bit area
  - Bits scale N
  - Address sqrt(N)*log(N)
  - Sense amps sqrt(N)
- Large memories pay less per bit
  - denser

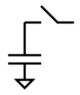


Dynamic RAM

- Goes a step further
- Share refresh/restoration logic as well
- Minimal storage is a capacitor
- “Feature” DRAM process is ability to make capacitors efficiently
- Denser, but slower

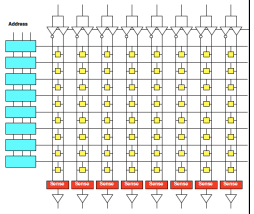






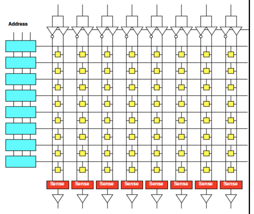
Memory Scaling

- Small memories are fast
  - Large memories are slow
- Small memories low energy
  - Large memories high energy
- Large memories dense
  - Small memories cost more area per bit
- Combining:
  - Dense memories are slow



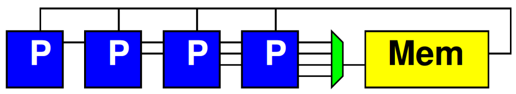
Wide Memory

- Relatively easy to have a wide memory
- As long as we share the address
- SIMD-like
  - One address to select wide word or bits
- Efficient if all read together



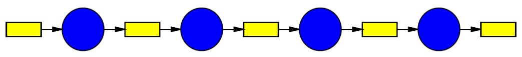
Preclass 2a

- 256b wide memory read/write in cycle
- N processors
- Random 32b read; 10 cycle; 32b write
- How many processors can support?



Preclass 2b

- 256b wide memory read/write in cycle
- N processors
- Local memory (>=1024b)
- FIFO 32b read; 10 cycle; 32b write
- How exploit wide memory transfer?
- How many processors can support?



Lesson

- Cheaper to access wide/contiguous blocks memory
  - In hardware
  - From the architectures built
- Can achieve higher bandwidth on large block data transfer
  - Than random access of small data items

DRAM

- 1GB DDR3 SDRAM from Micron
  - http://www.micron.com/products/dram/ddr3/
  - 96 pin pakage
  - 16b datapath IO
  - Operate at 500+MHz
  - 37.5ns random access latency

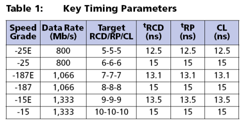
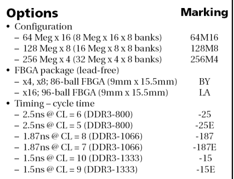


Table 1: Key Timing Parameters

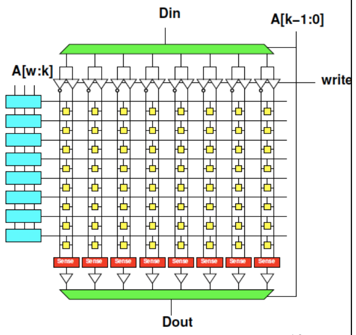
Speed Grade	Data Rate (Mb/s)	Target RCD/RP/CL	^tRCD (ns)	^tRP (ns)	CL (ns)
-25E	800	5-5-5	12.5	12.5	12.5
-25	800	6-6-6	15	15	15
-187E	1,066	7-7-7	13.1	13.1	13.1
-187	1,066	8-8-8	15	15	15
-15E	1,333	9-9-9	13.5	13.5	13.5
-15	1,333	10-10-10	15	15	15



Options	Marking
• Configuration	
– 64 Meg x 16 (8 Meg x 16 x 8 banks)	64M16
– 128 Meg x 8 (16 Meg x 8 x 8 banks)	128M8
– 256 Meg x 4 (32 Meg x 4 x 8 banks)	256M4
• FBGA package (lead-free)	
– x4, x8; 86-ball FBGA (9mm x 15.5mm)	BY
– x16; 96-ball FBGA (9mm x 15.5mm)	LA
• Timing – cycle time	
– 2.5ns @ CL = 6 (DDR3-800)	-25
– 2.5ns @ CL = 5 (DDR3-800)	-25E
– 1.87ns @ CL = 8 (DDR3-1066)	-187
– 1.87ns @ CL = 7 (DDR3-1066)	-187E
– 1.5ns @ CL = 10 (DDR3-1333)	-15
– 1.5ns @ CL = 9 (DDR3-1333)	-15E

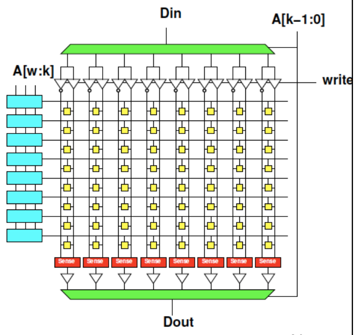
Memory Access Timing

- RAS/CAS access
  1. Address
  2. Row fetch
  3. Column select
  4. writeback/ refresh
- Optimization for access within a row



Memory Access Timing

- RAS/CAS access
  1. Address
  2. Row fetch
  3. Column select
     - Can repeat
  4. writeback/ refresh
- Row 1024b—8192b
- Faster to access within row



DRAM

- 1GB DDR3 SDRAM from Micron
  - http://www.micron.com/products/dram/ddr3/
  - 96 pin pakage
  - 16b datapath IO
  - Operate at 500+MHz
  - **37.5ns** random access latency

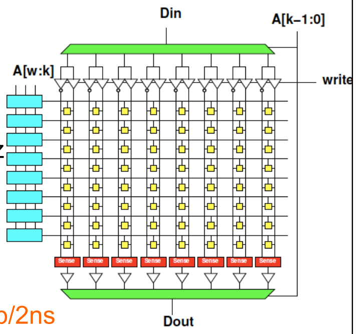
Table 1: Key Timing Parameters

Speed Grade	Data Rate (Mb/s)	Target RCD/RP/CL	^{t}RCD (ns)	^{t}RP (ns)	CL (ns)
-25E	800	5-5-5	12.5	12.5	12.5
-25	800	6-6-6	15	15	15
-187E	1,066	7-7-7	13.1	13.1	13.1
-187	1,066	8-8-8	15	15	15
-15E	1,333	9-9-9	13.5	13.5	13.5
-15	1,333	10-10-10	15	15	15

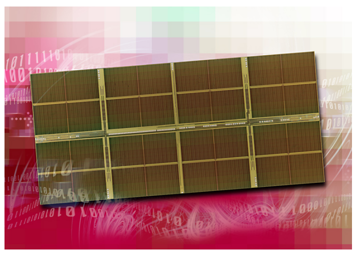
Options	Marking
• Configuration	
– 64 Meg x 16 (8 Meg x 16 x 8 banks)	64M16
– 128 Meg x 8 (16 Meg x 8 x 8 banks)	128M8
– 256 Meg x 4 (32 Meg x 4 x 8 banks)	256M4
• FBGA package (lead-free)	
– x4, x8; 86-ball FBGA (9mm x 15.5mm)	BY
– x16; 96-ball FBGA (9mm x 15.5mm)	LA
• Timing – cycle time	
– 2.5ns @ CL = 6 (DDR3-800)	-25
– 2.5ns @ CL = 5 (DDR3-800)	-25E
– 1.87ns @ CL = 8 (DDR3-1066)	-187
– 1.87ns @ CL = 7 (DDR3-1066)	-187E
– 1.5ns @ CL = 10 (DDR3-1333)	-15
– 1.5ns @ CL = 9 (DDR3-1333)	-15E

DRAM Streaming

- Reading row is 15ns
- 16b @ 500MHz
- 1024b row
- 1024/16
  - 64 words per row
- How supply 16b/2ns



1 Gigabit DDR2 SDRAM



[Source: http://www.elpida.com/en/news/2004/11-18.html]

DRAM

- Latency is large (10s of ns)
- Throughput can be high (GB/s)
  - If accessed sequentially
  - If exploit wide word block transfers
- Throughput low on random accesses

Preclass 4

- 10 cycle latency to memory
- Throughput in each case?

```
Case 1:

   for(i=0;i<MAX;i++) {

      in=a[i]; // memory read
      out=f(in); // 10 cycle compute
      b[i]=out;
    }
```

```
Case 2:
next_in=a[0];
for(i=1;i<MAX;i++) {
   in=next_in;
   next_in=a[i];
   out=f(in);
   b[i-1]=out;
 }
b[MAX-1]=f(next_in);
```

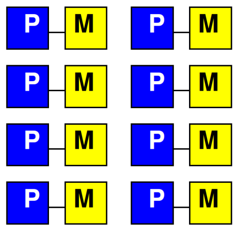
Lesson

- Long memory latency can impact throughput
  - When must wait on it
  - When part of a cyclic dependency
- Pipeline memory access when possible
  - exploit higher bandwidth of memory
  - (compared to 1/latency)

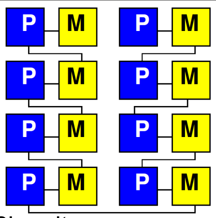
Memory Organization

Local Memories

- Cheapest way to get low latency and high bandwidth
  - Small memories local to compute



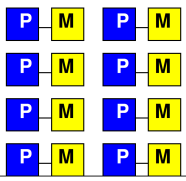
Like Preclass 2b



- N processors
- Local memory
- FIFO 32b read; 10 cycle; 32b write
- FIFO memories local to each processor pair
- How many processors can these memories support?

Local Memories

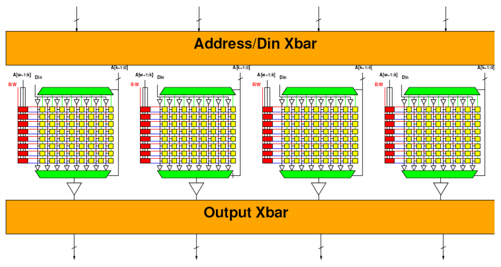
- Given a choice, to the extent we can get away with it
  - Want small memories local to compute
    - Bandwidth scales with compute
    - Latency remains small



Bandwidth to Shared

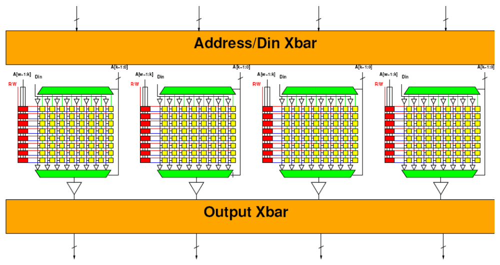
- High bandwidth is easier to engineer than low latency
  - Wide-word demonstrated
  - Banking
    - Decompose memory into independent banks
    - Route requests to appropriate bank

Independent Bank Access



- Impact on: Bandwidth, Latency, Area?

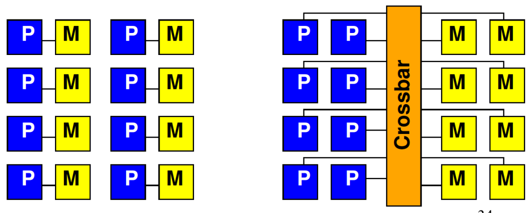
Independent Bank Access



- Compare same capacity, single memory with 4x wide (same bw) output: area, latency

Compare Local and Shared

- Compare banked, shared and local
  - Latency, area ?



Data Reuse

Preclass 5ab

- How many reads to x[] and w[]?
- How many distinct x[], w[] read?

```
for (i=0;i<MAX;i++) {
  t=0;
  for (j=0;j<WSIZE;j++)
    t+=x[i+j]*w[j];
  y[i]=t;
  }
```

Preclass 5c

- How use local memory to reduce to 5b?
- How much local memory need?

```
for (i=0;i<MAX;i++) {
  t=0;
  for (j=0;j<WSIZE;j++)
    t+=x[i+j]*w[j];
  y[i]=t;
  }
```

Preclass 5

•

```
for (i=0;i<MAX;i++) {
  t=0;
  for (j=0;j<WSIZE;j++)
    t+=x[i+j]*w[j];
  y[i]=t;
  }
```

```
for (j=0;j<WSIZE;j++)
  local_w[j]=w[j];
for (j=0;j<WSIZE-1;j++)
  local_x[j+1]=w[j];
for (i=0;i<MAX;i++) {
  t=0;
  for (j=0;j<WSIZE-1;j++)
    local_x[j]=local_x[j+1];
  local_x[WSIZE-1]=x[i+WSIZE-1];
  for (j=0;j<WSIZE;j++)
    t+=local_x[j]*local_w[j];
  y[i]=t;
  }
```

Lesson

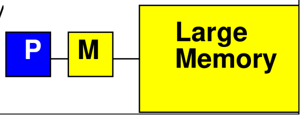
- Data can often be reused
  - Keep data needed for computation in
    - Closer, smaller (faster, less energy) memories
  - Reduces bandwidth required from large (shared) memories
- Reuse hint: value used multiple times
  - Or value produced/consumed

Enhancing Reuse

- Computations can often be (re-)organized around data reuse
- Parallel decomposition can often be driven by data locality
  - Extreme case: processors own data
    - Send data to processor for comptuation

Processor Data Caches

- Traditional Processor Data Caches are a heuristic instance of this
  - Add a small memory local to the processor
    - It is fast, low latency
  - Store anything fetched from large/remote/ shared memory in local memory
    - Hoping for reuse in near future
  - On every fetch, check local memory before go to large memory



Processor Data Caches

- Demands more than a small memory
  - Need to sparsely store address/data mappings from large memory
  - Makes more area/delay/energy expensive than just a simple memory of capacity
- Don't need explicit data movement
- Cannot control when data moved/saved
  - Bad for determinism

Memory Organization

- Architecture contains
  - Large memories
    - For density, necessary sharing
  - Small memories local to compute
    - For high bandwidth, low latency, low energy
- Need to move data
  - Among memories
    - Large to small and back
    - Among small

Big Ideas

- Memory bandwidth and latency can be bottlenecks
- Exploit small, local memories
  - Easy bandwidth, low latency, energy
- Exploit data reuse
  - Keep in small memories
- Minimize data movement
  - Small, local memories keep distance short
  - Minimally move into small memories

Admin

- Reading for Day 8 from Zynq book
- HW4 due Friday