ESE532: System-on-a-Chip Architecture

Day 12: October 11, 2017
Orchestrating Data in Memories

Today

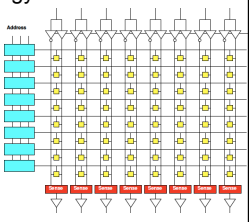
- DRAM
- DRAM Implications
- Tricks for working with DRAM and generally minimizing use of large memories

Message

- How we access data matters
- Reuse data to avoid access to large memories
- Encourage think about how organize computation and data to minimize
  - Use of large memory
  - Data movement
- …and to think about what you expect, so can diagnose unnecessary data bottlenecks

Day 3: 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



Day 3: Lesson

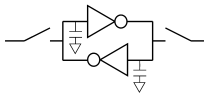
- Cheaper to access wide/contiguous blocks memory
  - In hardware
  - From the architectures typically build
- Can achieve higher bandwidth on large block data transfer
  - Than random access of small data items

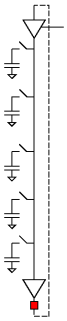
DRAM

Usually the largest memory

Dynamic RAM

- Shares even more logic among bits
- Share refresh/restoration logic as well
- Minimal storage is a capacitor
- "Feature" DRAM process is ability to make capacitors efficiently
- Denser, but slower





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

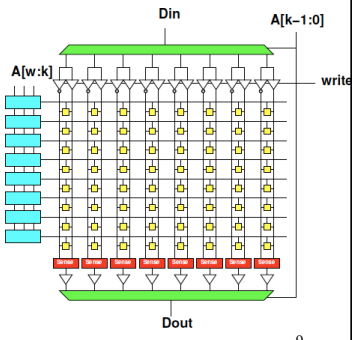
Table 1: Key Timing Parameters

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

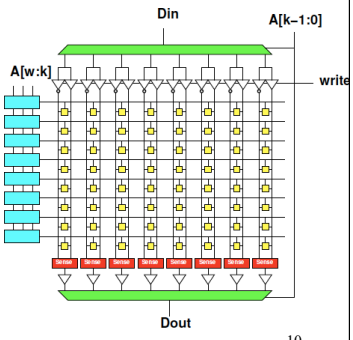
Memory Access Timing

- Access:
  1. Address (RCD)
  2. Row fetch (RP)
  3. Column select (CL)
  4. writeback/ refresh
- Optimization for access within a row



Memory Access Timing

- Access
  1. Address (RCD)
  2. Row fetch (RP)
  3. Column select
     - CL: 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

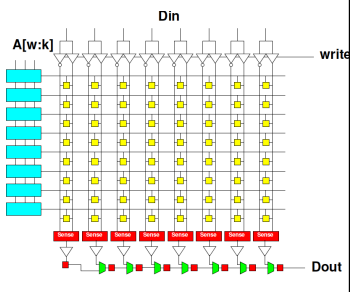
Table 1: Key Timing Parameters

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



DRAM

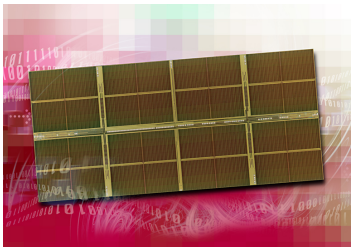
- 1GB DDR3 SDRAM from Micron
  - http://www.micron.com/products/dram/ddr3/
  - 96 pin pakage
  - 16b datapath IO
  - Operate at 500+MHz
  - **37.5ns** random access latency

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

1 Gigabit DDR2 SDRAM



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

Preclass 1

```
uint16_b dram_image[1024*1024]; // assum
for (y=0; y<1024;y++)
  for (x=0; x<1024;x++)
     astream.write(dram_image[1024*y+x])
```

- How many cycles to run loop?

Preclass 2

```
uint16_b dram_image[1024*1024];
for (x=0; x<1024;x++)
  for (y=0; y<1024;y++)
     astream.write(dram_image[1024*y+x]);
```

- How many cycles to run loop?

Preclass 3

```
uint16_b dram_image[1024*1024];
for (y=0; y<1024;y++)
  for (x=0; x<1024;x+=128)
     // assume STREAM_FETCH can setup a streaming read
     //  number of bytes (128*2=256) from the given ad
     //  target stream, with performance as noted in t
     STREAM_FETCH(128*2,&dram_image[1024*y+x],astream)
```

- How many cycles to run loop?

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
  - As we saw for random access on wide-word memory

Maximize Reuse

Minimize need to go to large memory

Data Management

- Large, central memory
  - Bottleneck
  - High energy
- Minimize need to return to it

Window Filter

- Compute based on neighbors
  - F(d[y-1][x-1],d[y-1][x],d[y-1][x+1], d[y][x-1],d[y][x],d[y][x+1], d[y+1][x-1],d[y+1][x],d[y+1][x+1])
- Common idiom
  - Image filtering
  - Numerical simulation
  - Cellular automata
  - Search, matching

Window Filter

- Compute based on neighbors
  - F(d[y-1][x-1],d[y-1][x],d[y-1][x+1], d[y][x-1],d[y][x],d[y][x+1], d[y+1][x-1],d[y+1][x],d[y+1][x+1])
- Generalize for different neighborhood sizes
  - For (yoff=YMIN; yoff<YMAX;yoff++)
    For (xoff=XMIN; xoff<XMAX;xoff++)
    res=fun(res,d[y+yoff][x+xoff]);

Window Filter

- Compute based on neighbors
- For (y=0;y<YMAX;y++)
  For (x=0;x<XMAX;x++)
  o[y][x]=F(d[y-1][x-1],d[y-1][x],d[y-1][x+1], d[y][x-1],d[y][x],d[y][x+1], d[y+1][x-1],d[y+1][x],d[y+1][x+1])
- How work with image in DRAM?

Window Filter

- Compute based on neighbors
- For (y=0;y<YMAX;y++)
  For (x=0;x<XMAX;x++)
  o[y][x]=F(d[y-1][x-1],d[y-1][x],d[y-1][x+1], d[y][x-1],d[y][x],d[y][x+1], d[y+1][x-1],d[y+1][x],d[y+1][x+1])
- How work with image in DRAM?
  - As written references
  - Reads per row

Window Filter

- Compute based on neighbors
- For (y=0;y<YMAX;y++)
  For (x=0;x<XMAX;x++)

```
o[y][x]=F(d[y-1][x-1],d[y-1][x],d[y-1][x+1],
          d[y][x-1],d[y][x],d[y][x+1],
          d[y+1][x-1],d[y+1][x],d[y+1][x+1])
```

- Data reused between inner loop invocations when x increments?

Window Filter

- Compute based on neighbors
- For (y=0;y<YMAX;y++)
  For (x=0;x<XMAX;x++)

```
o[y][x]=F(d[y-1][x-1],d[y-1][x],d[y-1][x+1],
          d[y][x-1],d[y][x],d[y][x+1],
          d[y+1][x-1],d[y+1][x],d[y+1][x+1])
```

- Data reused between y values in outer loop?
  - Reuse distance?

Window Filter

- Compute based on neighbors
- For (y=0;y<YMAX;y++)
  For (x=0;x<XMAX;x++)

```
o[y][x]=F(d[y-1][x-1],d[y-1][x],d[y-1][x+1],
          d[y][x-1],d[y][x],d[y][x+1],
          d[y+1][x-1],d[y+1][x],d[y+1][x+1])
```

- What need to do to store old y offsets so not need to reread from large memory?

Line Buffers

- Only need to save last 2 (window_height-1) rows (d[y][*])
- Store in local buffers
  - Type dy[XMAX], dym[XMAX];
    - dym for dy minus 1
  - Stick value there between uses.

Window Filter

- With line buffers dy, dym
- For (y=0;y<YMAX;y++)
  For (x=0;x<XMAX;x++)

```
dypxm=dypx; dypx=dnew; dnew=d[y+1][x+1];
o[y][x]=F(dym[x-1],dym[x],dym[x+1],
          dy[x-1],dy[x],dy[x+1],
          dypxm,dypx,dnew);
dym[x-1]=dy[x-1];dy[x-1]=dypxm;
```


Window Filter

- With line buffers dy, dym
- For (y=0;y<YMAX;y++)
  For (x=0;x<XMAX;x++)

```
dypxm=dypx; dypx=dnew; dnew=d[y+1][x+1];
o[y][x]=F(dym[x-1],dym[x],dym[x+1],
          dy[x-1],dy[x],dy[x+1],
          dypxm,dypx,dnew);
dym[x-1]=dy[x-1];dy[x-1]=dypxm;
```

- Avoid multiple read dy, dym?

Window Filter

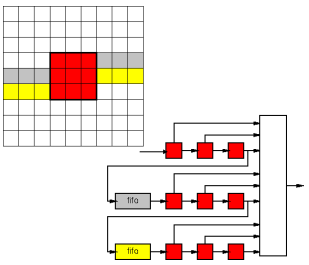
- Single read and write from dym, dy
- For (y=0;y<YMAX;y++)
  For (x=0;x<XMAX;x++)
  dypxm=dypx; dypx=dnew; dnew=d[y+1][x+1];
  dyxm=dyx; dyx=dyxp; dyxp=dy[x+1];
  dymxm=dymx; dymx=dymxp; dymxp=dym[x+1];
  o[y][x]=F(dymxm,dymx,dymxp,
  dyxm,dyx,dyxp,
  dypxm,dypx,dnew);
  dym[x-1]=dyxm;dy[x-1]=dypxm;

Window Filter

- Convert to stream … use Preclass 3 load
- For (y=0;y<YMAX;y++)
  For (x=0;x<XMAX;x++)
  dypxm=dypx; dypx=dnew; astream.read(dnew);
  dyxm=dyx; dyx=dyxp; dyxp=dy[x+1];
  dymxm=dymx; dymx=dymxp; dymxp=dym[x+1];
  o[y][x]=F(dymxm,dymx,dymxp,
  dyxm,dyx,dyxp,
  dypxm,dypx,dnew);
  dym[x-1]=dyxm;dy[x-1]=dypxm;

Pipelined Circuit

- Single read from big memory (DRAM) per result
- Shift registers
- Line buffers



Multiple Filters

- What if want to run multiple filters?
- for (f=0;f<FMAX;f++)
  for (y=0;y<YMAX;y++)
  for (x=0;x<XMAX;x++)
  o[f][y][x]=fun[f](d[y-1][x-1],d[y-1][x],d[y-1][x+1],
  d[y][x-1],d[y][x],d[y][x+1],
  d[y+1][x-1],d[y+1][x],d[y+1][x+1]);
- Forces multiple reads through d
- Avoidable?

Sort

Sort

- Order data
  - 7 3 4 2 1 9 8 0 → 0 1 2 3 4 7 8 9
- Data movement can be challenging
  - Last value may become first (vice-versa)
  - Especially when not fit in small memories
- Many sequential sort solutions access data irregularly
  - Want to avoid with DRAM

Observation: Merge

- Merging two sorted list is a streaming operation
- Int aptr; int bptr;
- For (i=0;i<MCNT;i++)

```
If ((aptr<ACNT) && (bptr<BCNT))
  If (a[aptr]>b[bptr])
   { o[i]=a[aptr]; aptr++; }
  Else
   { o[i]=b[bptr]; bptr++; }
Else // copy over remaining from a or b
```


Merge using Streams

- Merging two sorted list is a streaming operation
- Int ptr; int bptr;
- astream.read(ain); bstream.read(bin)
- For (i=0;i<MCNT;i++)

```
If ((aptr<ACNT) && (bptr<BCNT))
  If (ain>bin)
   { ostream.write(ain); aptr++; astream.read(ain);}
  Else
   { ostream.write(bin) bptr++; bstream.read(bin);}
Else // copy over remaining from astream/bstream
```


Merge and DRAM

- How streaming merge work with DRAM row?
- For (i=0;i<MCNT;i++)

```
If ((aptr<ACNT) && (bptr<BCNT))
  if (ain>bin)
   { ostream.write(ain); aptr++; astream.read(ain);}
  else
   { ostream.write(bin) bptr++; bstream.read(bin);}
```


Building from Merge

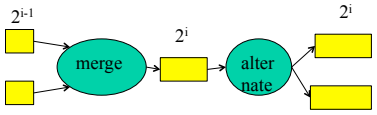
- Know how to get sorted data of length N given two, independently sorted lists of length N/2
- How get sorted lists of length N/2?
- How get sorted lists of length N/4?
- What happens when have lists of length 1?

Merge Sort

- Strategy
  - For i=1 to passes
    - Pass over data creating merged sequences of length 2^i from pairs of length 2^{i-1}
  - What is passes?
- Can, at least, limit to passes reads through data from large memory.

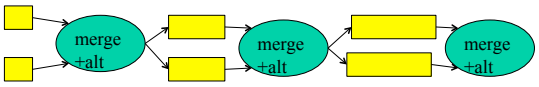
On-chip Merge

- As long as can fit sequence (2^i) on chip, can stream merge stages without going back to big memory (DRAM)



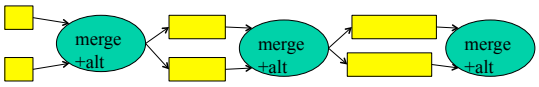
On-chip Merge

- As long as can fit sequence (2^i) on chip, can stream merge stages without going back to big memory (DRAM)
  - E.g. with 36Kb BRAMs ~ first 10-12 stages
    - Up to ~16 using multiple BRAMs



On-chip Merge

- For sorts up to ~100K elements may be able to cover in one read/write stream to/from DRAM
- Larger: ~$\log_2(N)$-15 read/writes



Big Ideas

- Must pay attention to orchestrate data movement
  - How we access and reuse data
- Minimize use of large memories
- Regular access to large memories higher performance
  - Easier to get bandwidth on wide-word reads

Admin

- Reading for Monday on web
- HW5 due Friday
- HW6 out
  - Does into 8 full FPGA builds
  - Plan for that build time