DRAMsim: DRAM Memory System Simulation Framework Code Review

• Why are we doing a code review?
• What is wrong with the old code?
• What is new?
• What is the new hardware architecture?
• How does the new software architecture reflect/realize the new hardware architecture?
• Goal: Seek comments/feedback to ensure that all old features can be transparently graft onto new code
Why Are We Doing A Code Review?

- DRAM memory system simulation code written 3 years ago, some parts rather ugly. Needs to be “fixed”.
- Fix or Replace? Replace core with new simulation core, retain added features. {FBD support, power computations}
What are the problems?

Current simulation code attempts to model DRAM commands phase by phase. Each phase acquires/releases resources.

Column-read becomes \{command transport : 1 cycle, data fetch : t_{cas} cycles, data burst : t_{burst} cycles\}

Command bus ready? acquire/release data bus. etc.

Problem: Coding requires understanding of not just DRAM protocol, but intricate DRAM device operation!

Needs a better way to simulate DRAM commands.
Replace with Protocol Table

<table>
<thead>
<tr>
<th>Pre</th>
<th>Ext</th>
<th>Rank</th>
<th>Rank</th>
<th>Minimum Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>s</td>
<td>s</td>
<td>t_{RC}</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>s</td>
<td>d</td>
<td>t_{RRD}</td>
</tr>
<tr>
<td>P</td>
<td>A</td>
<td>s</td>
<td>d</td>
<td>t_{RP}</td>
</tr>
<tr>
<td>F</td>
<td>A</td>
<td>s</td>
<td>s</td>
<td>t_{RFC}</td>
</tr>
<tr>
<td>A</td>
<td>R</td>
<td>s</td>
<td>s</td>
<td>t_{RCD} - t_{AL}</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>s</td>
<td>a</td>
<td>t_{BURST}</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>d</td>
<td>a</td>
<td>t_{BURST} + t_{RTP}</td>
</tr>
<tr>
<td>W</td>
<td>R</td>
<td>s</td>
<td>a</td>
<td>t_{CWD} + t_{BURST} + t_{WTR}</td>
</tr>
<tr>
<td>W</td>
<td>R</td>
<td>d</td>
<td>a</td>
<td>t_{CWD} + t_{BURST} + t_{RTP} - t_{CAS}</td>
</tr>
<tr>
<td>A</td>
<td>W</td>
<td>s</td>
<td>s</td>
<td>t_{RCD} - t_{AL}</td>
</tr>
<tr>
<td>R</td>
<td>W</td>
<td>a</td>
<td>a</td>
<td>t_{CAS} + t_{BURST} + t_{RTP} - t_{CAS}</td>
</tr>
<tr>
<td>W</td>
<td>W</td>
<td>a</td>
<td>a</td>
<td>t_{BURST}</td>
</tr>
<tr>
<td>A</td>
<td>P</td>
<td>s</td>
<td>s</td>
<td>t_{RAS}</td>
</tr>
<tr>
<td>R</td>
<td>P</td>
<td>s</td>
<td>s</td>
<td>t_{AL} + t_{BURST} + t_{RTP} - t_{INT-BURST}</td>
</tr>
<tr>
<td>W</td>
<td>P</td>
<td>s</td>
<td>s</td>
<td>t_{AL} + t_{CWD} + t_{BURST} + t_{WR}</td>
</tr>
<tr>
<td>A</td>
<td>RP</td>
<td>s</td>
<td>a</td>
<td>MAX(t_{RCD} + t_{RAS} - t_{RTP} + t_{INT-BURST}) - t_{AL}</td>
</tr>
<tr>
<td>A</td>
<td>WP</td>
<td>s</td>
<td>a</td>
<td>MAX(t_{RCD} - t_{RAS} - t_{CWD} - t_{BURST} - t_{WR} - t_{AL})</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>s</td>
<td>a</td>
<td>t_{RFC}</td>
</tr>
<tr>
<td>P</td>
<td>F</td>
<td>s</td>
<td>a</td>
<td>t_{RP}</td>
</tr>
</tbody>
</table>

**Legend**
- **F** = refresh
- **A** = row access
- **R** = column read
- **P** = column write
- **RP** = Read-and-Prec
- **WP** = Write-and-Prec
- **P** = precharge
- **s** = same
- **d** = different
- **a** = any

Command to command timing... Compute “gaps”

Much easier to deal with.
New Controller Architecture

If bank queue is empty, insert RAS, CAS, PREC sequence for transaction request. Open page systems can delay PREC indefinitely, or delay for number of cycles.

Bank queues are FIFO, except open page can insert CAS ahead of PREC to same bank.

Algorithm selects between commands at head of queues in different banks.

Select command: Compute “gap” based on DRAM system states. Advance clock to now+gap. “execute” DRAM command by updating DRAM system state.
Files

- LICENSE (GPL)
- address.c (previous address mapping code)
- algorithm.c (which bank do we select?)
- commandline.c (process command line)
- dramsim.c (main code)
- dramsim.h (header file very similar to mem-system.h)
- files.c (read trace and spd files)
- inits.c (init data structures)
- protocol.c (literal implementation of protocol table)
- queues.c (everything now use abstract FIFO queue)
- random.c (random number generator)
- stats.c (need to be done)
algorithm.c

```c
if(config->command_ordering_algorithm == STRICT_ORDER){ /* look for oldest command
   oldest_command_time = -1;
   oldest_rank_id = -1;
   oldest_bank_id = -1;
   for(rank_id = 0; rank_id< config->rank_count; rank_id++){
     for(bank_id = 0; bank_id< config->rank_count; bank_id++){
       candidate_bank_q = ((channel->rank[rank_id]).bank[bank_id]).per_bank_q;
       temp_c = (command_t *) q_read(candidate_bank_q,0);
       if(temp_c != NULL) {
         if((oldest_command_time < 0) || (oldest_command_time>temp_c->enqueue_time)){
           oldest_command_time = temp_c->enqueue_time;
           oldest_rank_id = rank_id;
           oldest_bank_id = bank_id;
         }
       }
     }
   }
   candidate_bank_q=(((channel->rank[oldest_rank_id]).bank[oldest_bank_id]).per_bank_q;
   return (command_t *) dequeue(candidate_bank_q);
} else if(config->command_ordering_algorithm == RANK_ROUND_ROBIN){ /*

```
The protocol.c uses the protocol table to compute "gaps" that we need before the selected command can be issued.

Note: t_rfc_gap missing, refresh not yet added
Now, commit “state” by setting this bank’s last_ras_time, and

Also, add entry to that rank’s last_ras_times queue. Happens to be 4 entries deep.
Thank You

- Comments?
- Suggestions?
- To do: bring up web site.
- To do: clean up manual
- To do: Migrate FBD support
- To do: Migrate power computations for SDRAM, DDR, and DDR2 . . .