Memory system enhancements

First Annual gem5 User Workshop, MICRO

Vancouver, December, 2012

Andreas Hansson

Goal

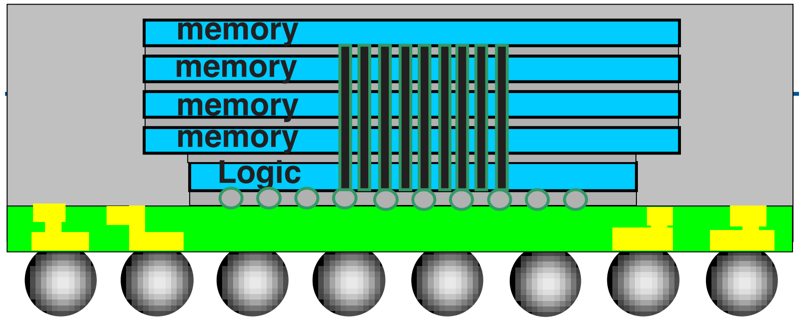
- Highlight changes and additions to the memory system
- Make it clear that memory systems are not all about CPUs
- Showcase the new DRAM controller model and point out why we don't just integrate an existing model

Outline

- Overview
- Ports and transport interfaces
- Traffic toolkit
- Memory controllers

Overview

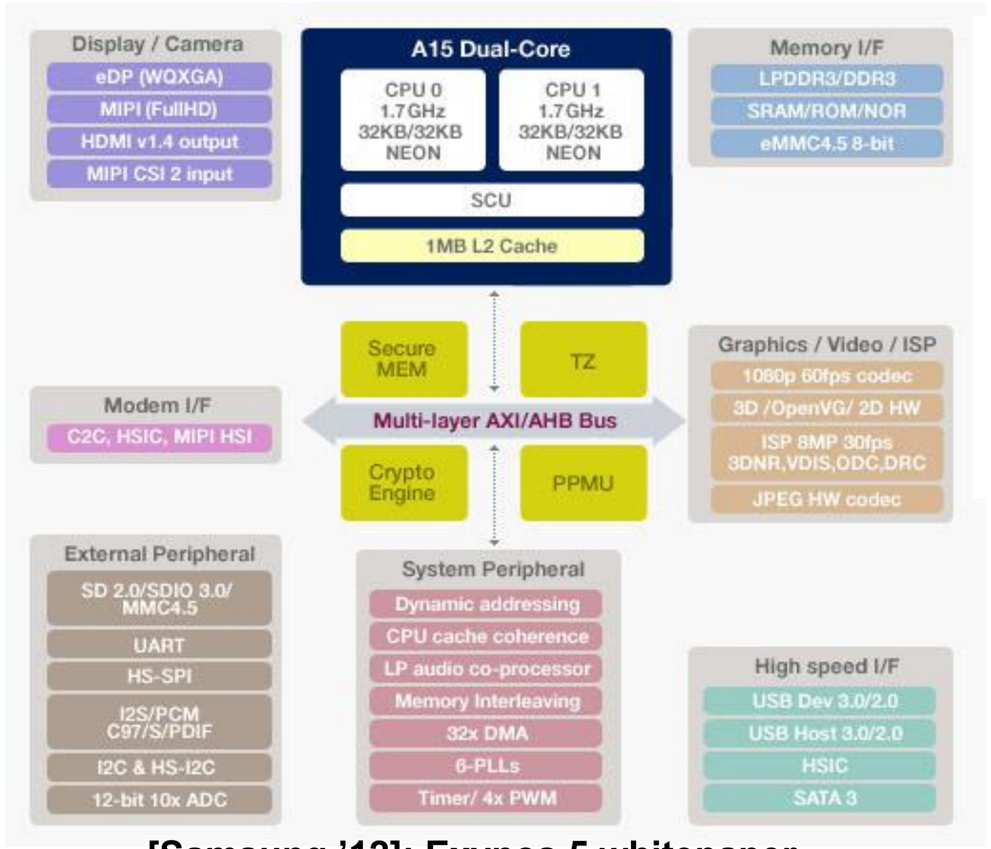
- Memory system has big impact on system performance
  - CPUs (and GPUs) spend a lot of their time waiting for memory accesses to complete
- DRAM is getting increasingly heterogeneous
  - DDRx, LPDDRx, WideIO, Hybrid Memory Cube (HMC)
  - Many high-level architecture trade-offs
- Memory performance closely linked with access patterns
  - Latency and bandwidth varying greatly depending on both memory, controller and accesses
  - The number of IO streams is growing with the number of cores



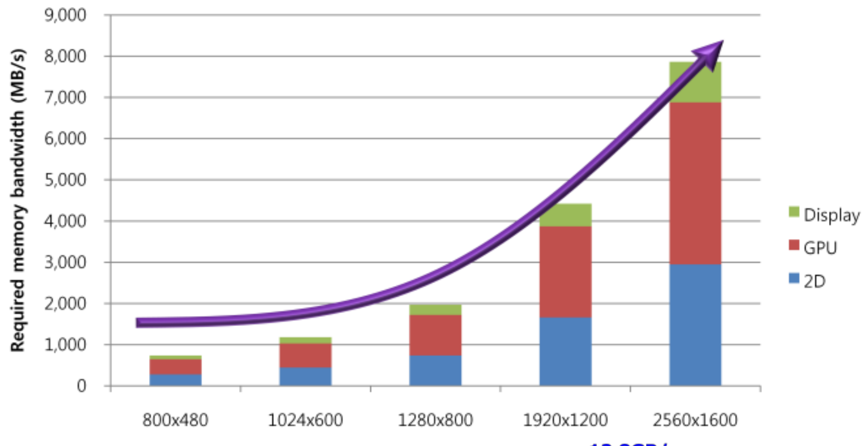
[Sematech '12]: M. Suh, Technology challenges for WideIO

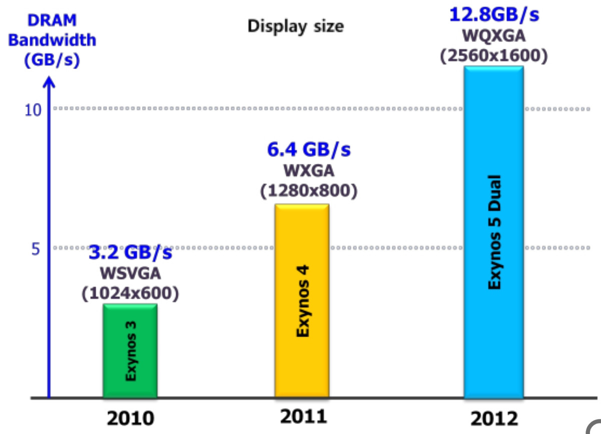
It’s a changing world

- GPU/video/camera and connectivity is the key driver for memory-system performance in portable devices
  - High-speed interfaces and increasing resolutions drive increasing bandwidth (and latency) demand



[Samsung ’12]: Exynos 5 whitepaper



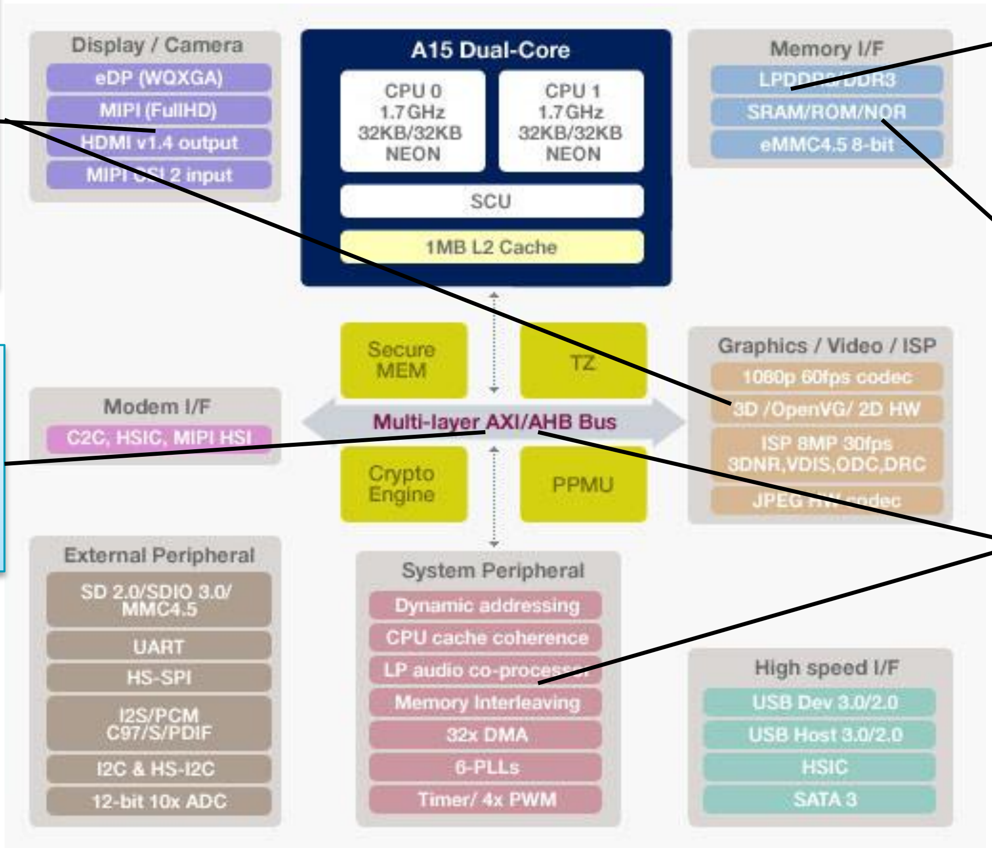


All in one place

- Our goal is to make gem5 the environment of choice for evaluating future heterogeneous memory architectures

TrafficGen
Highly-configurable traffic generation and trace playback

CommMonitor
Traffic trace capture, statistics and profiling



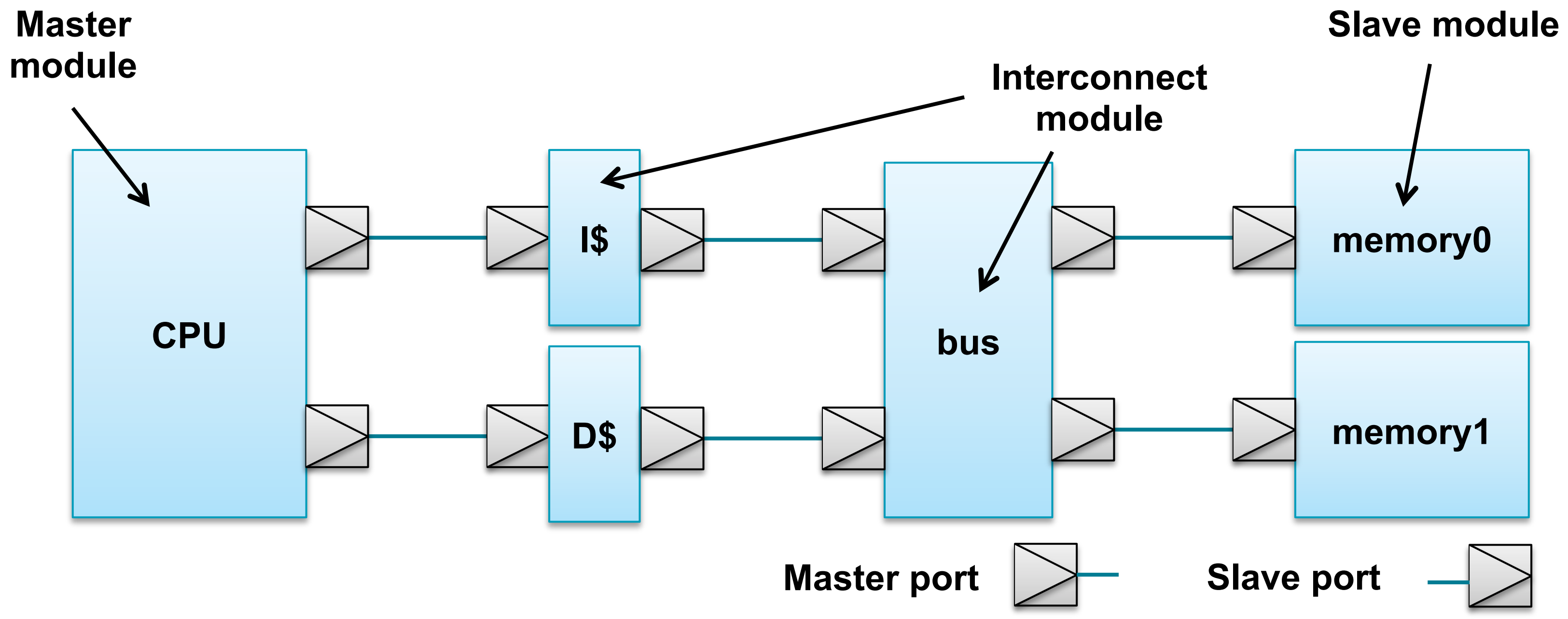
SimpleDRAM
High-level DRAM controller

SimpleMemory
Fixed latency, fixed bandwidth

CoherentBus
On-chip fabric with address interleaving support

Ports, Masters and Slaves

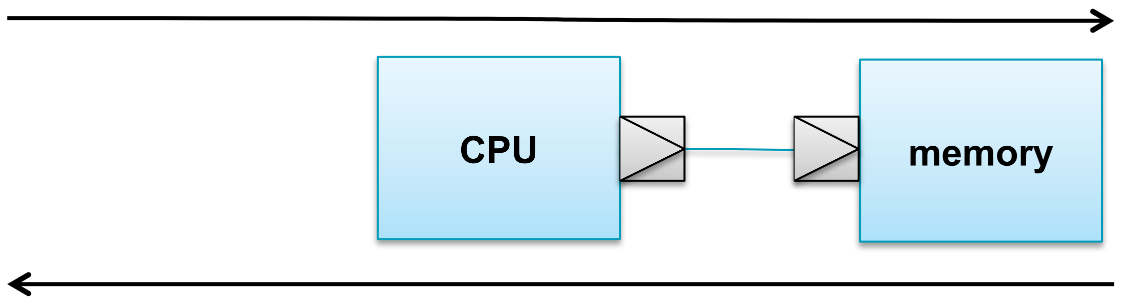
- MemObjects are connected through master and slave ports
  - A master module has at least one master port, a slave module at least one slave port, and an interconnect module at least one of each
    - Similar to TLM-2 notation (4-phase semantics is work-in-progress)
  - A master port always connects to a slave port
    - Binding checked already in Python and reflected in C++ classes



Requests & Packets

- Protocol stack based on Requests and Packets
  - A master module, e.g. a CPU, changes the state of a slave module, e.g. a memory through a Request transported between master ports and slave ports using Packets
  - Addition of a MasterID uniquely identifying the module behind the request
    - Used for statistics in caches, busses, memory controllers etc

```
Request req(addr, size, flags, masterId);
Packet* req_pkt = new Packet(req, MemCmd::ReadReq);
...
```



```
if (req_pkt->needsResponse()) {
   req_pkt->makeResponse();
} else {
   delete req_pkt;
}
...
```

```
...
delete resp_pkt;
```

Transport interfaces

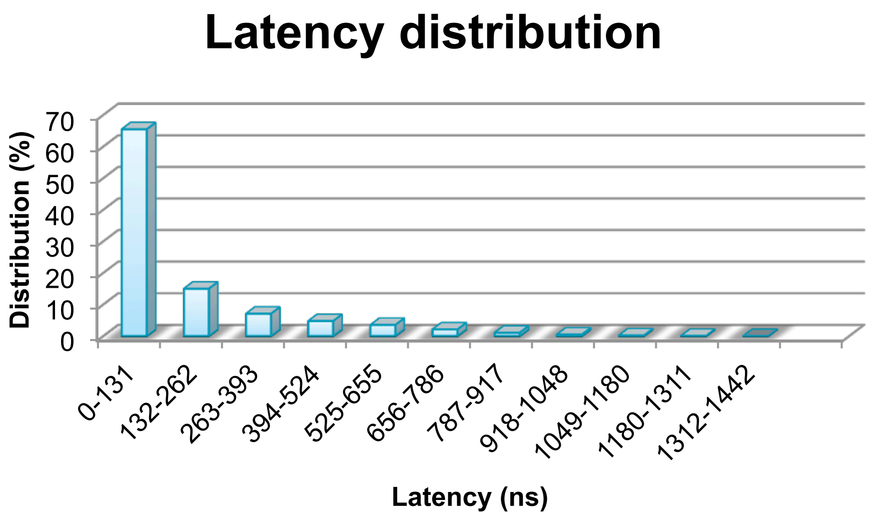
- Split into a master and slave interface, and distinguishing direction (request/response), and snoop/no snoop
  - E.g. only masters can be snooping or not; only slaves have address ranges
- Atomic/functional
  - Master uses `sendAtomic` to send requests and responses are returned in the call
  - Slave uses `sendAtomicSnoop` to send snoop requests and snoop responses are returned in the call
- Timing
  - Master uses `sendTimingReq` to send requests and receives responses through `recvTimingReq`
  - Slave uses `sendTimingSnoopReq` to send snoop requests and receives snoop responses through `recvTimingSnoopResp`

Communication monitor

- Insert as a structural component where stats are desired

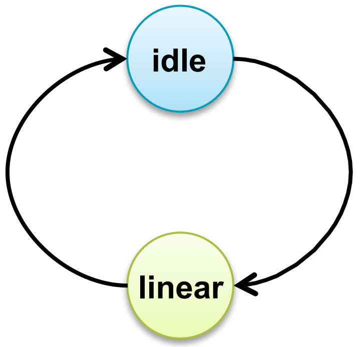
```
memmonitor = CommMonitor()
membus.master = memmonitor.slave
memmonitor.master = memctrl.slave
```

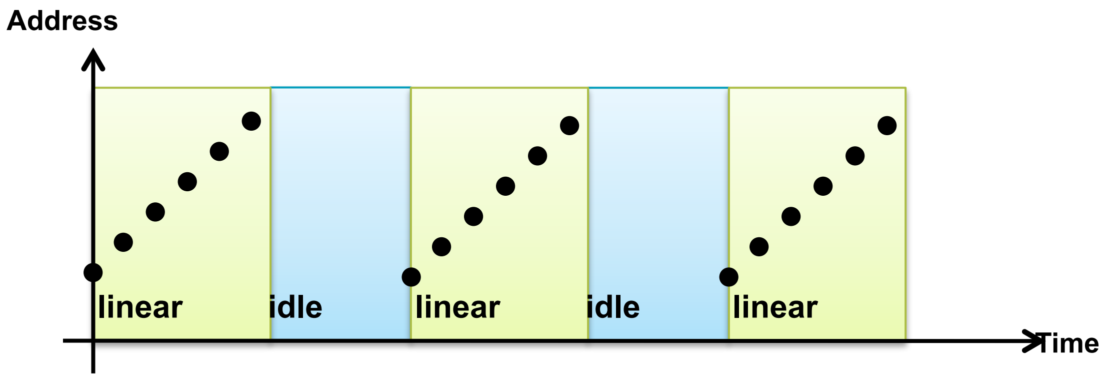
- A wide range of communication stats
  - bandwidth, latency, inter-transaction (read/write) time, outstanding transactions, address heatmap, etc
- Also captures traffic traces to file
  - using protobuf (about to be posted)



Traffic generator

- Test scenarios for memory system regression and performance validation
  - High-level of control for scenario creation
- Black-box models for components that are not yet modeled
  - Video/baseband/accelerator for memory-system loading
- Inject requests based on (probabilistic) state-transition diagrams
  - Idle, random, linear and trace replay states



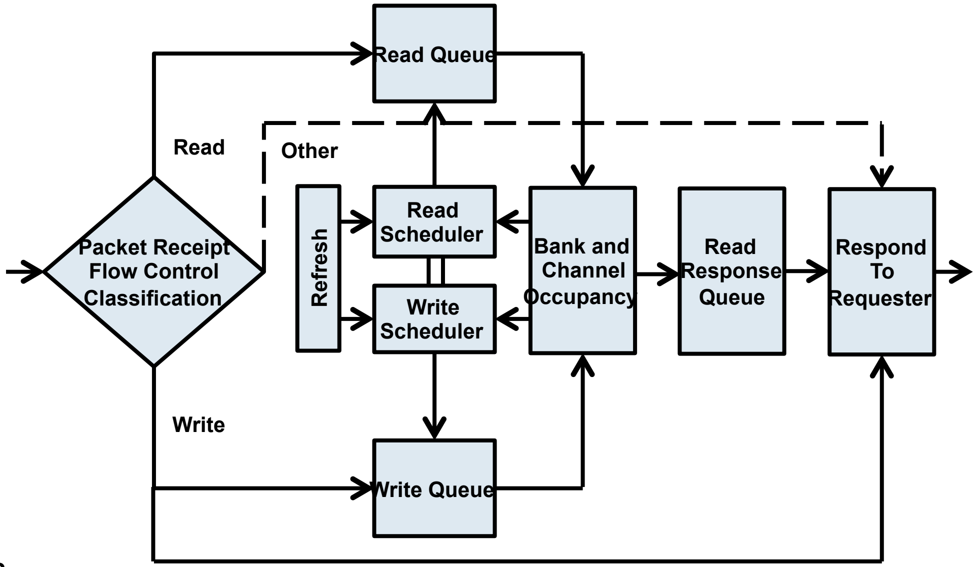


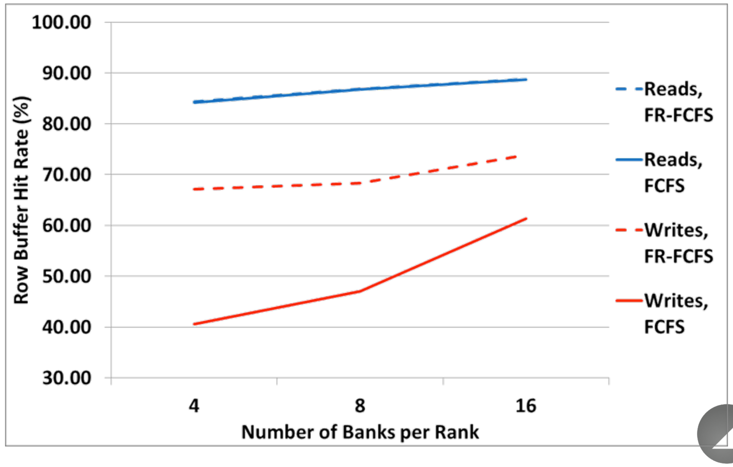
Memory controllers

- All memories in the system inherit from AbstractMemory
  - Basic single-channel memory controller
    - Instantiate multiple times if required
    - Interleaving support added in the bus/crossbar (to be posted)
- SimpleMemory
  - Fixed latency (possibly with a variance)
  - Fixed throughput (request throttling without buffering)
- SimpleDRAM
  - High-level configurable DRAM controller model to mimic DDRx, LPDDRx, WideIO, HMC etc
    - Memory organisation: ranks, banks, row-buffer size
    - Controller architecture: Read/write buffers, open/close page, address mapping, scheduling policy
    - Key timing constraints: tRCD, tCL, tRP, tBURST, tRFC, tREFI, tTAW/tFAW

SimpleDRAM

- Focus is not on modeling the DRAM, but the impact on the system performance
  - NOT cycle-callable for faster simulation
    - Determine decision points from timing parameters and schedule appropriate events
  - Enable rapid exploration and evaluation of new memory system architectures
- Statistics provide insight into memory usage





Questions?