A Hardware Accelerator Store for Low Power Processors

Mike Lyons

Kevin Brownell

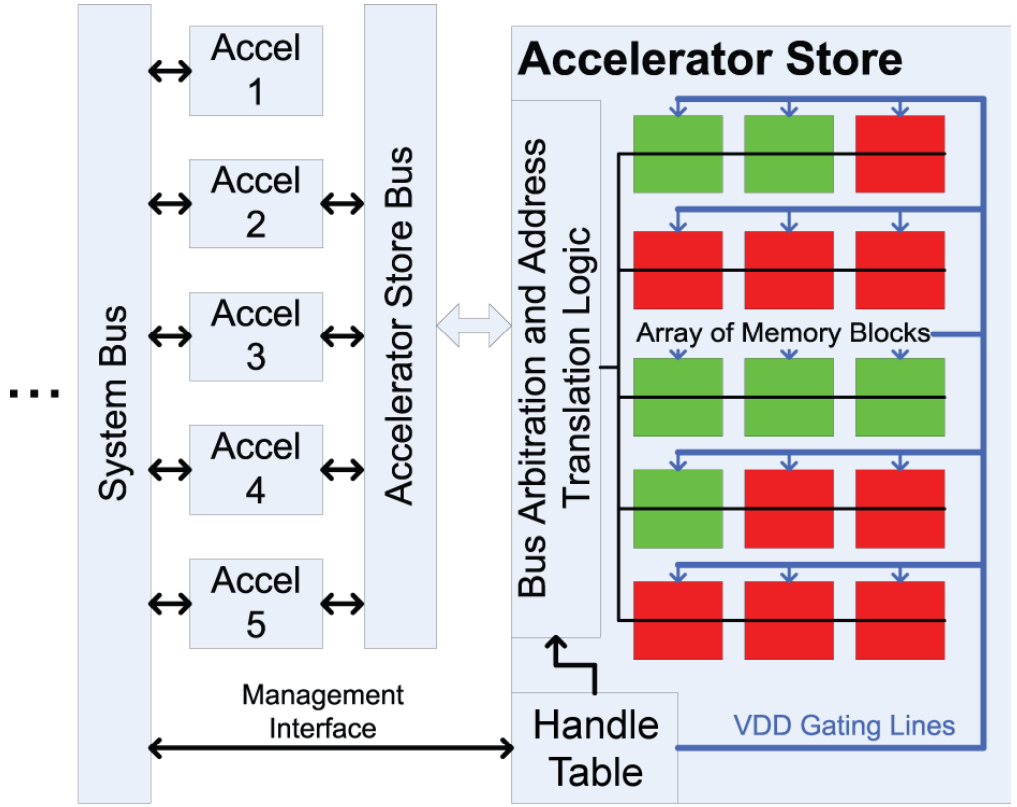
Durlov Khan

Motivation

- ASICs often use custom hardware / accelerators
- Include required memory internally
  - Results in inefficient use of area and power due to redundancy if multiple accelerators don't fully utilize their private memory
  - Excessive transport of data between accelerators if they operate on common data
- Lower power embedded systems, such as sensor networks, have very limited energy budgets

Proposed Shared Memory Framework

- Accelerator Store for low power, low frequency systems
- Shared memory with a common interface to multiple accelerators



Shared Memory Framework

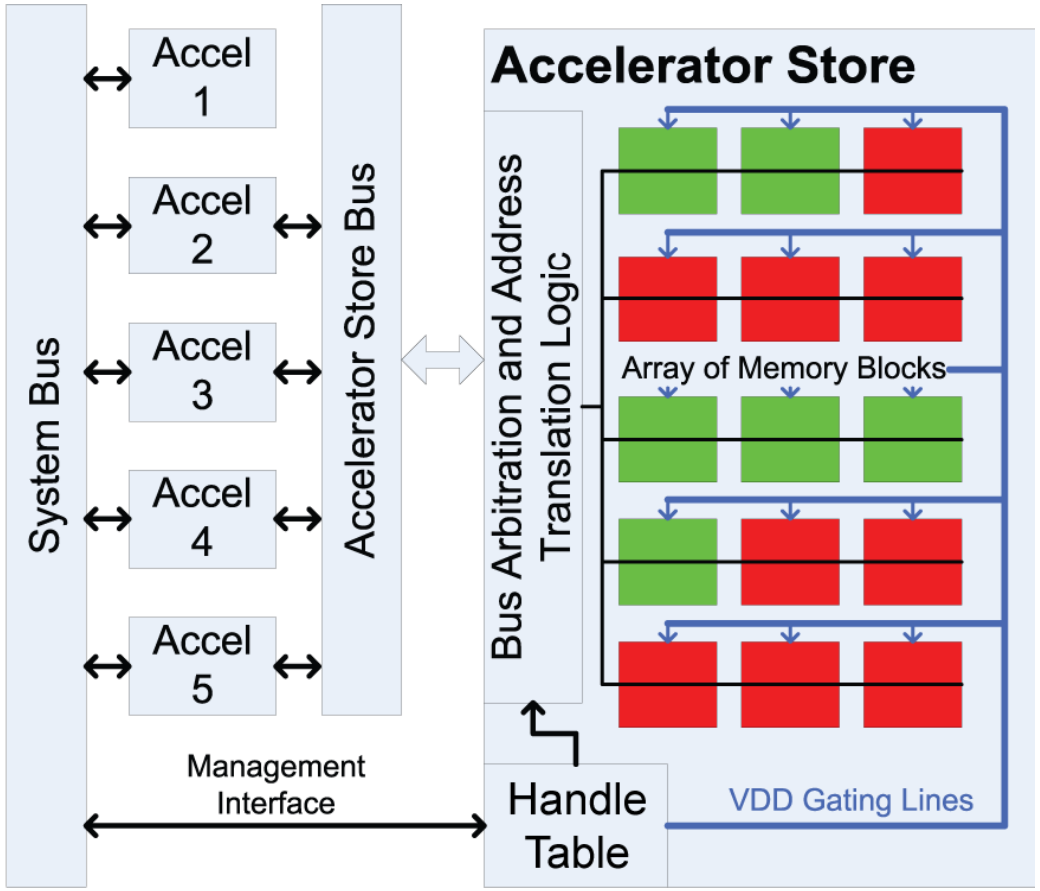
- A shared memory framework would allow for:
  - Separation of accelerator logic and state allows more opportunities to VDD gate logic
  - Automatic VDD gating of unused memory blocks
  - Support for complex data structures in memory, such as queues and stacks
    - Allows for communication abstractions
      - Producer/Consumer model

Simple application behavior

- Sensor reading every 1ms
- 1024 point FFT roughly every second
- Compress FFT frequency response
- Queue compressed frequency response
- Transmit queued frequency response after 8KB of data is stored

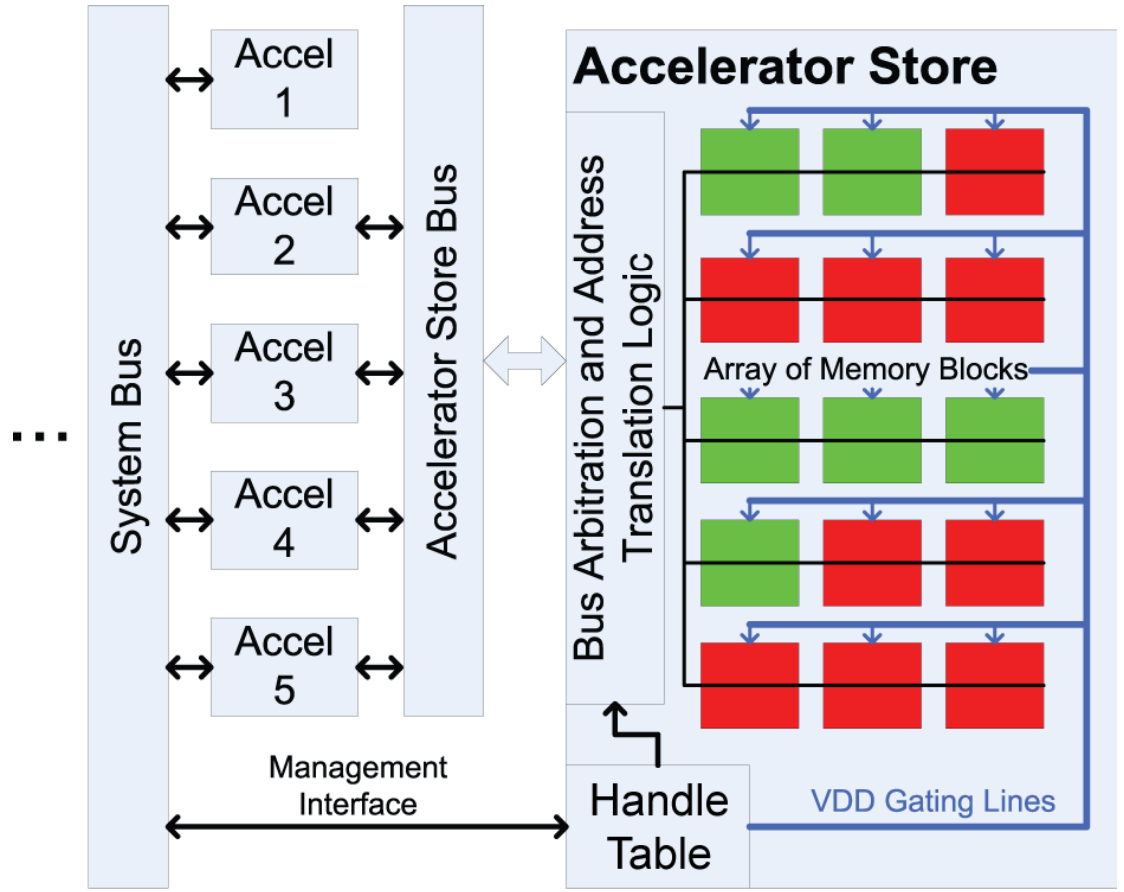
High Level System Description

- General Purpose CPU talks over system bus with accelerators and accelerator store



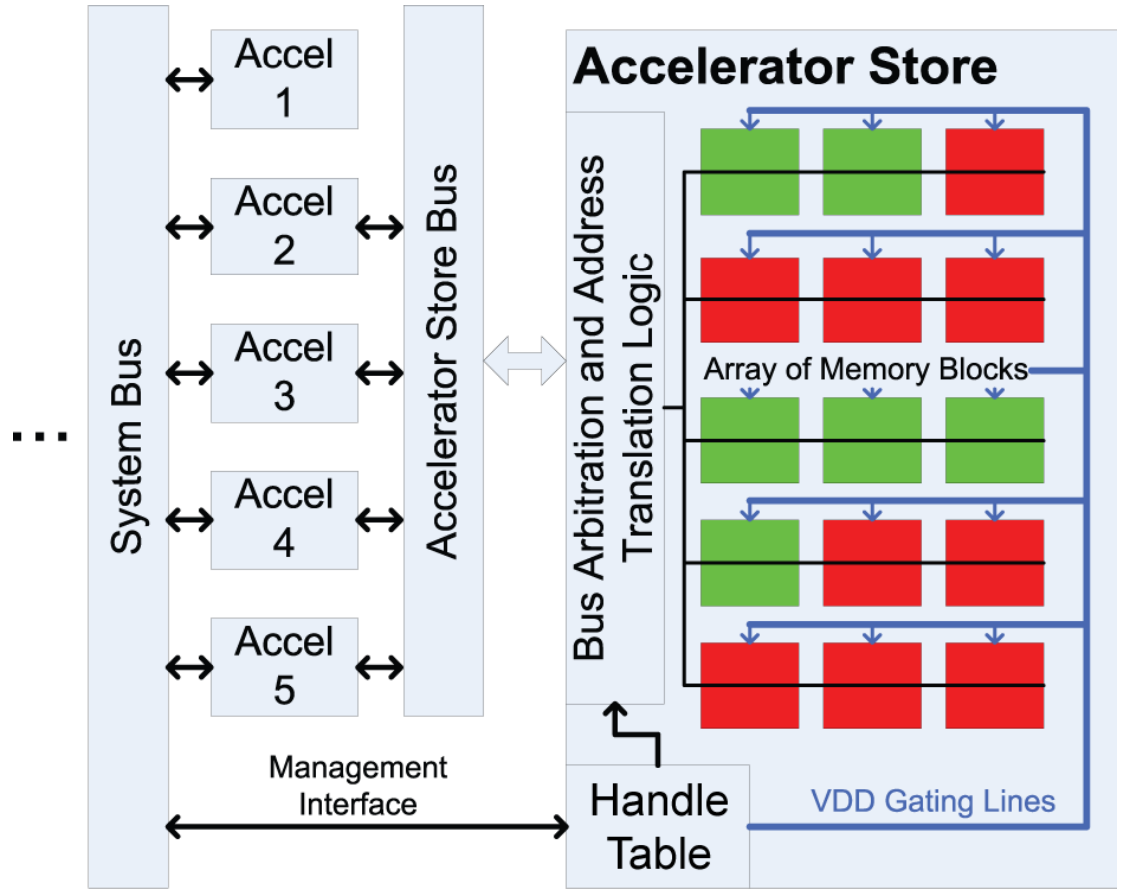
High Level System Description

- CPU manages “handles” in the Handle Table
- Handles represent portions of memory in the accelerator store
- Handle Table maintains list of active handles and meta data



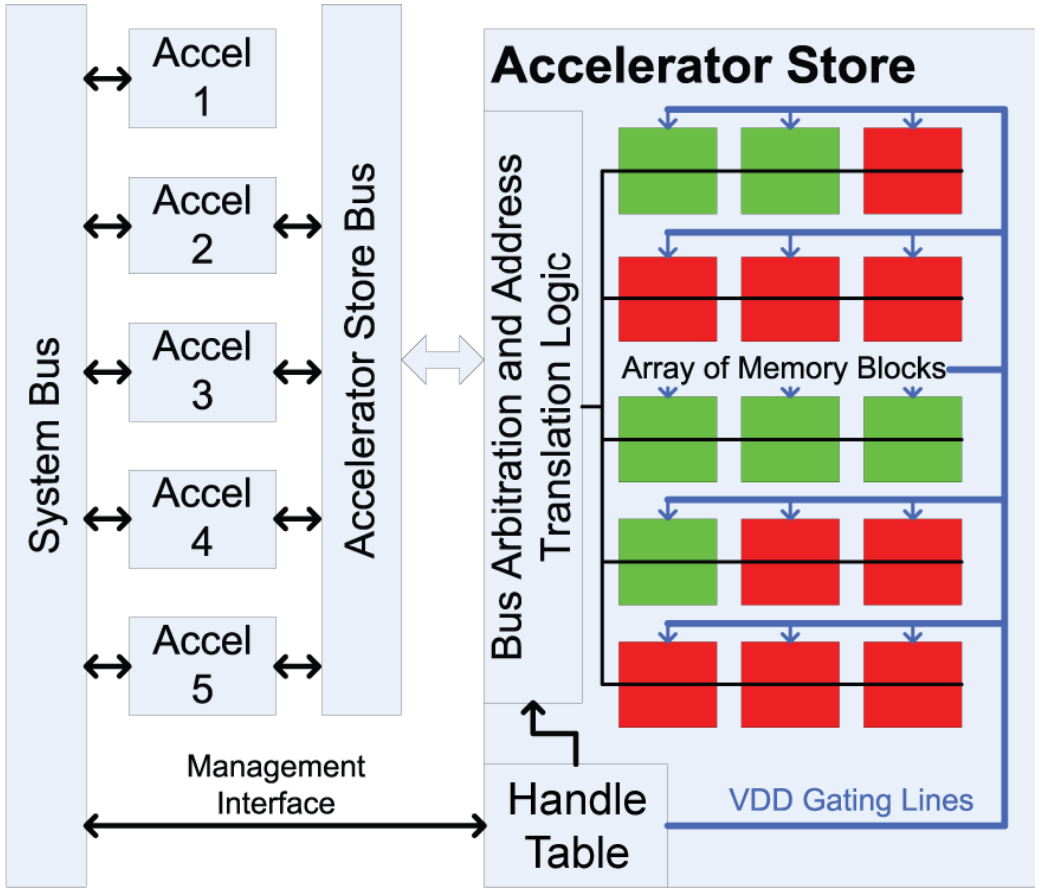
High Level System Description

- Accelerator Store can accept multiple requests per cycle
- Each accelerator can make multiple requests per cycle



High Level System Description

- Accelerator store uses a faster clock internally than rest of system
- System runs off of slow frequency (100 … KHz)
- Faster memory clock doesn't present difficult timing targets



System Software Architecture

- System software responsible for:
  - Configuration of accelerator store
    - Handle table management
    - Accelerator port priority management
  - Coordination between accelerators
  - Interrupt handling from accelerator store
- Accelerators issue read/write operations to accelerator store
  - Not guaranteed a response, depending on memory congestion

Handle Table

- Handle ID is a unique identifier for each active handle
- Valid bit to indicate active entries

Handle Id	*Valid*	*Type*	*Starting Address*	*Size (bytes)*	*Head Offset*	*Element Count*	*Trigger Level*
0	Y	Unconstrained	0	1024	0	512	0
1	Y	FIFO	1024	1024	2	1	5
2	Y	Stack	2048	2048	0	1	5
3	Y	CFIFO	4096	4096	0	0	5
...	N	0	0	0	0	0	0

Handle Table

- Handle Type
  - Unconstrained
  - FIFO
  - Circular FIFO
  - Stack

Handle Id	*Valid*	*Type*	*Starting Address*	*Size (bytes)*	*Head Offset*	*Element Count*	*Trigger Level*
0	Y	Unconstrained	0	1024	0	512	0
1	Y	FIFO	1024	1024	2	1	5
2	Y	Stack	2048	2048	0	1	5
3	Y	CFIFO	4096	4096	0	0	5
...	N	0	0	0	0	0	0

Handle Table

- Starting address
  - Corresponds to physical address of start of allocated memory
- Size
- Head offset
- Element count

Handle Id	*Valid*	*Type*	*Starting Address*	*Size (bytes)*	*Head Offset*	*Element Count*	*Trigger Level*
0	Y	Unconstrained	0	1024	0	512	0
1	Y	FIFO	1024	1024	2	1	5
2	Y	Stack	2048	2048	0	1	5
3	Y	CFIFO	4096	4096	0	0	5
...	N	0	0	0	0	0	0

Handle Table

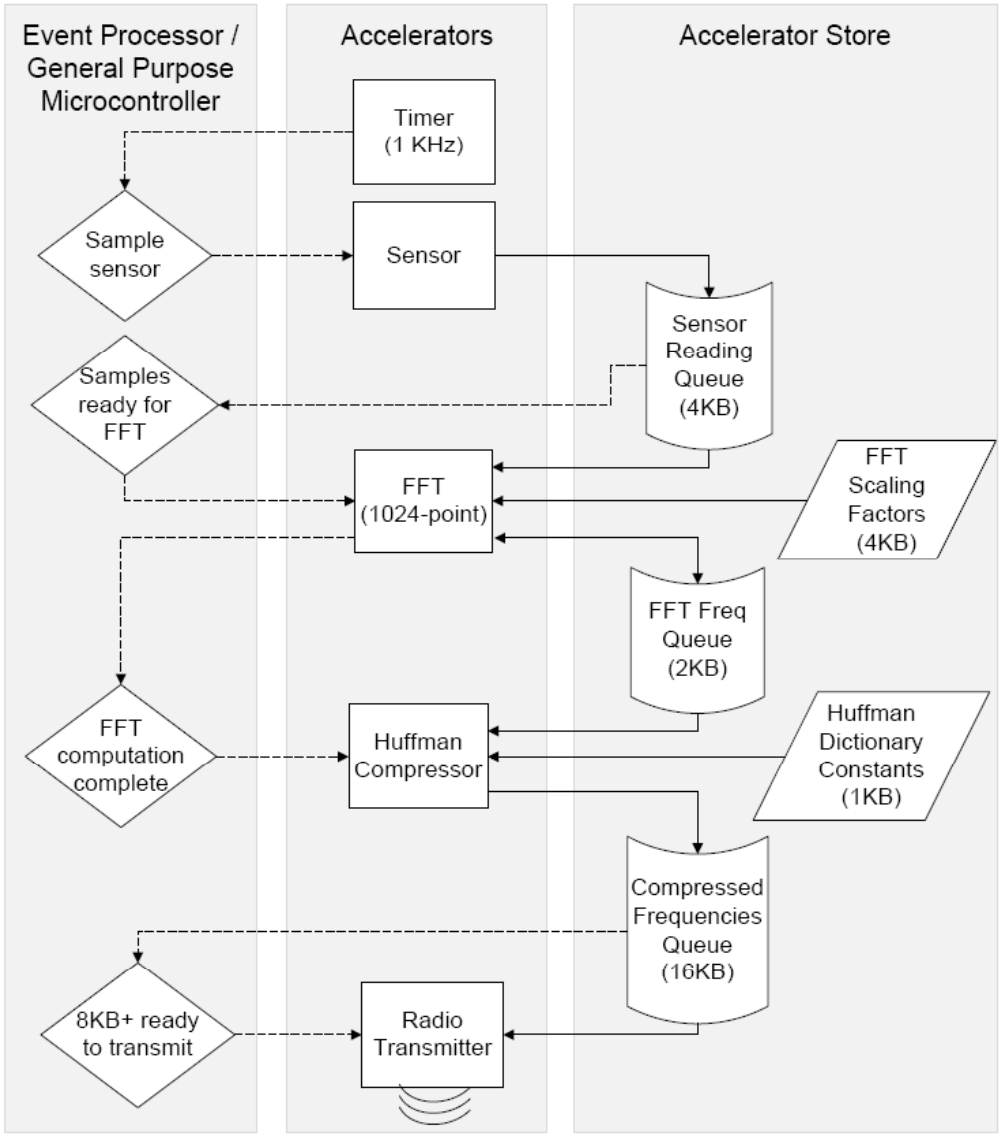
- Trigger level

```
if (element count == trigger level) {
    FireInterruptToCPU();
    }
```

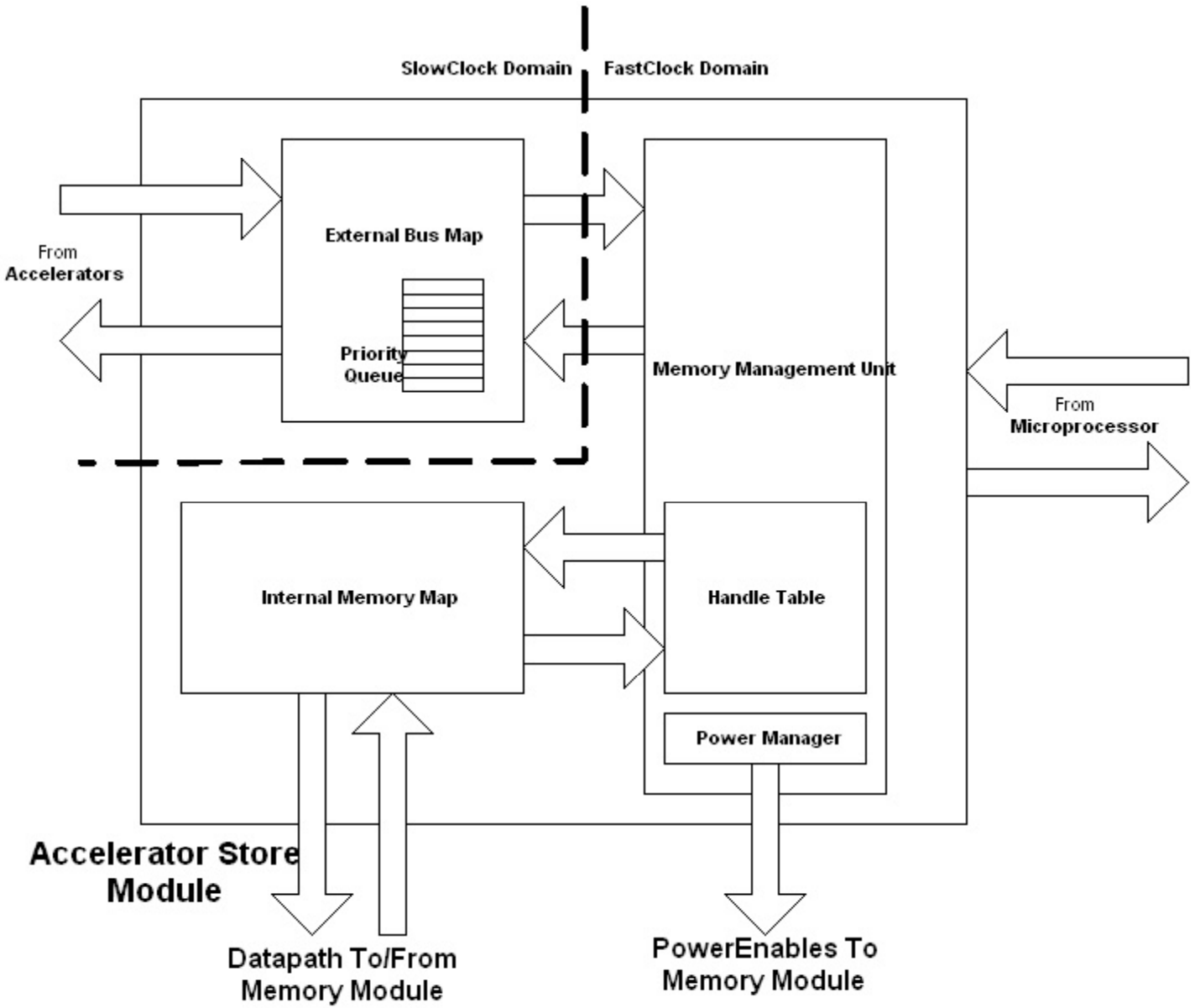
- Notification for nearly full or empty FIFOs/stacks

Handle Id	*Valid*	*Type*	*Starting Address*	*Size (bytes)*	*Head Offset*	*Element Count*	*Trigger Level*
0	Y	Unconstrained	0	1024	0	512	0
1	Y	FIFO	1024	1024	2	1	5
2	Y	Stack	2048	2048	0	1	5
3	Y	CFIFO	4096	4096	0	0	5
...	N	0	0	0	0	0	0

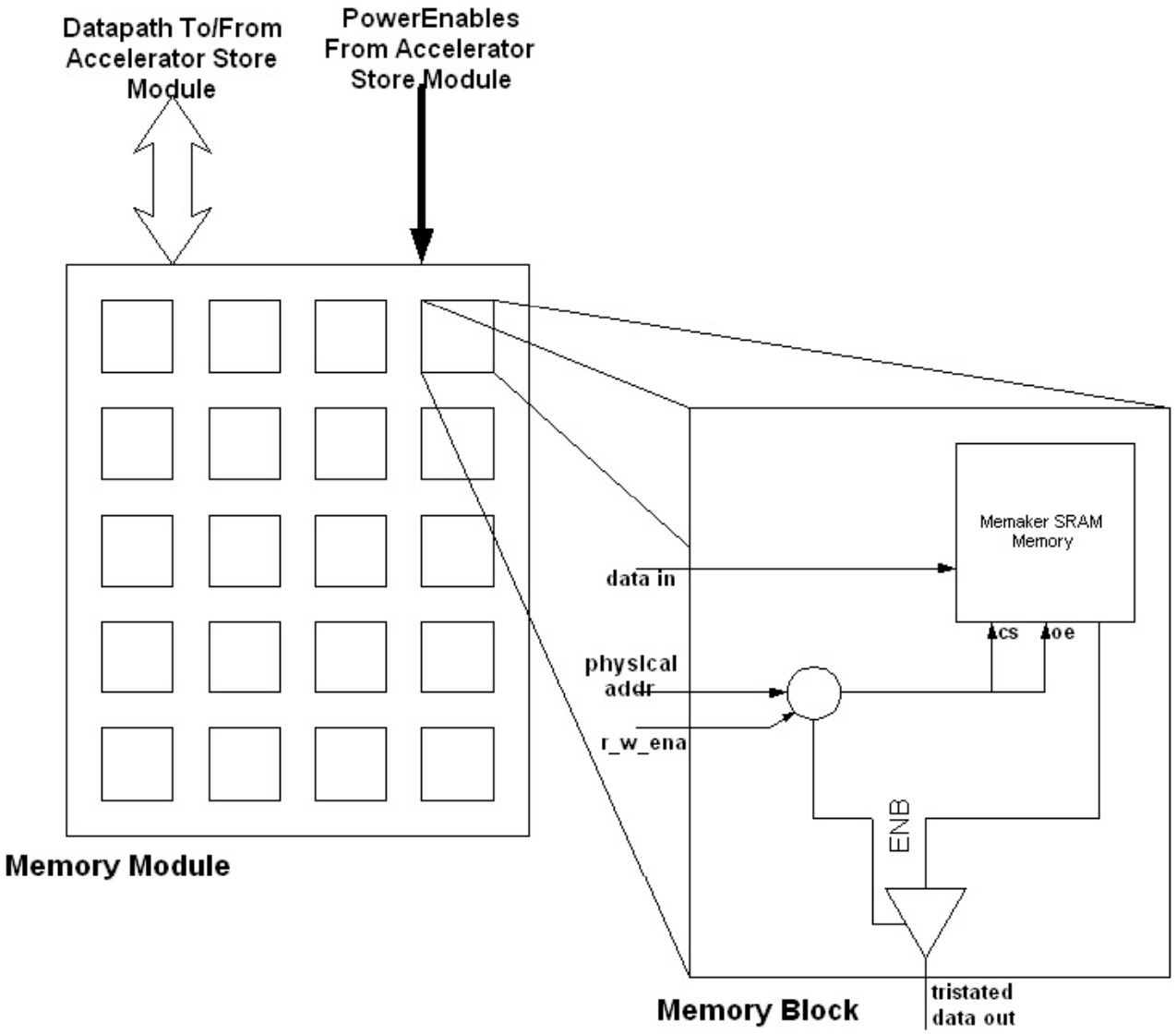
Simple application layout



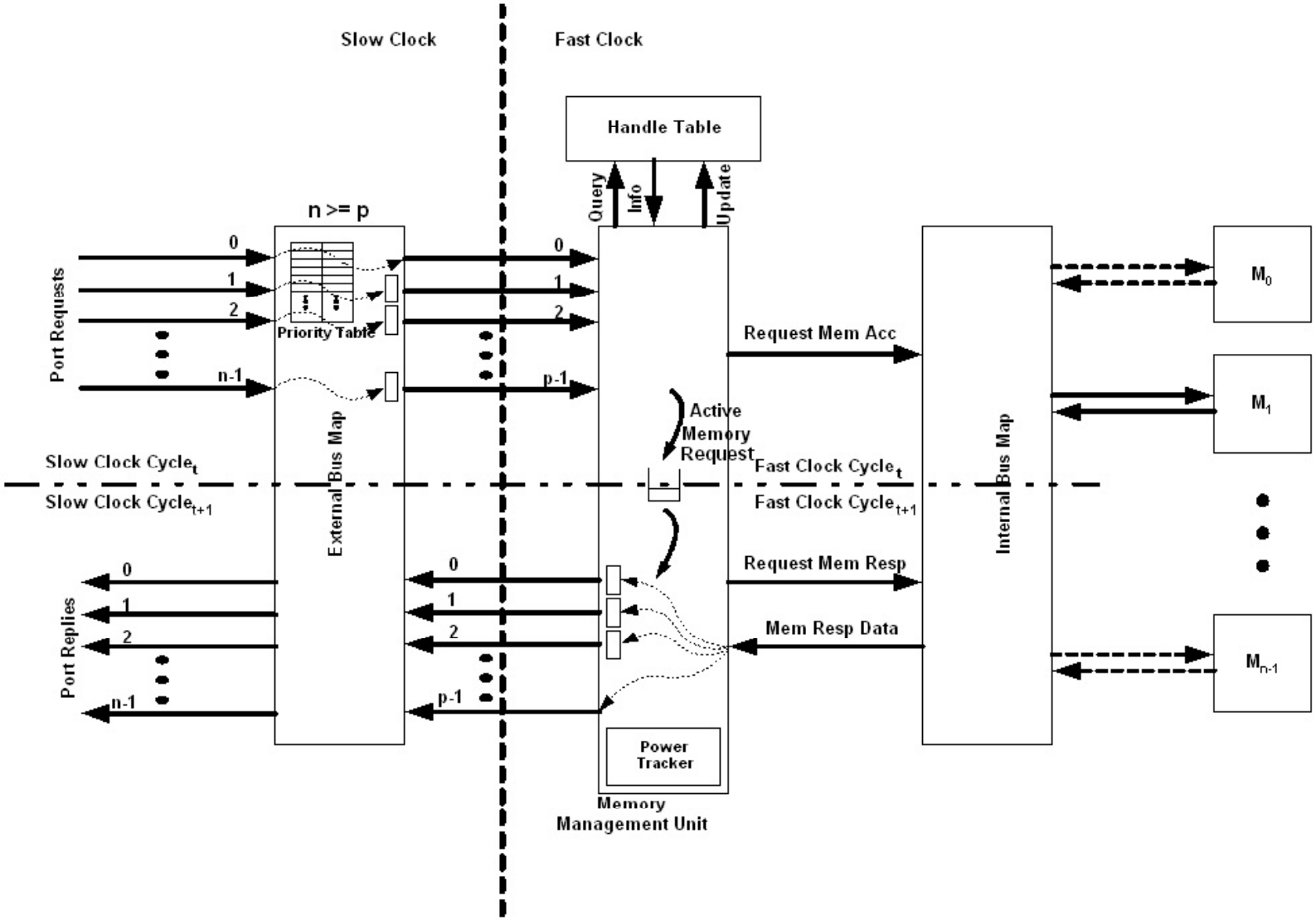
Block Level Architecture



Memory Module

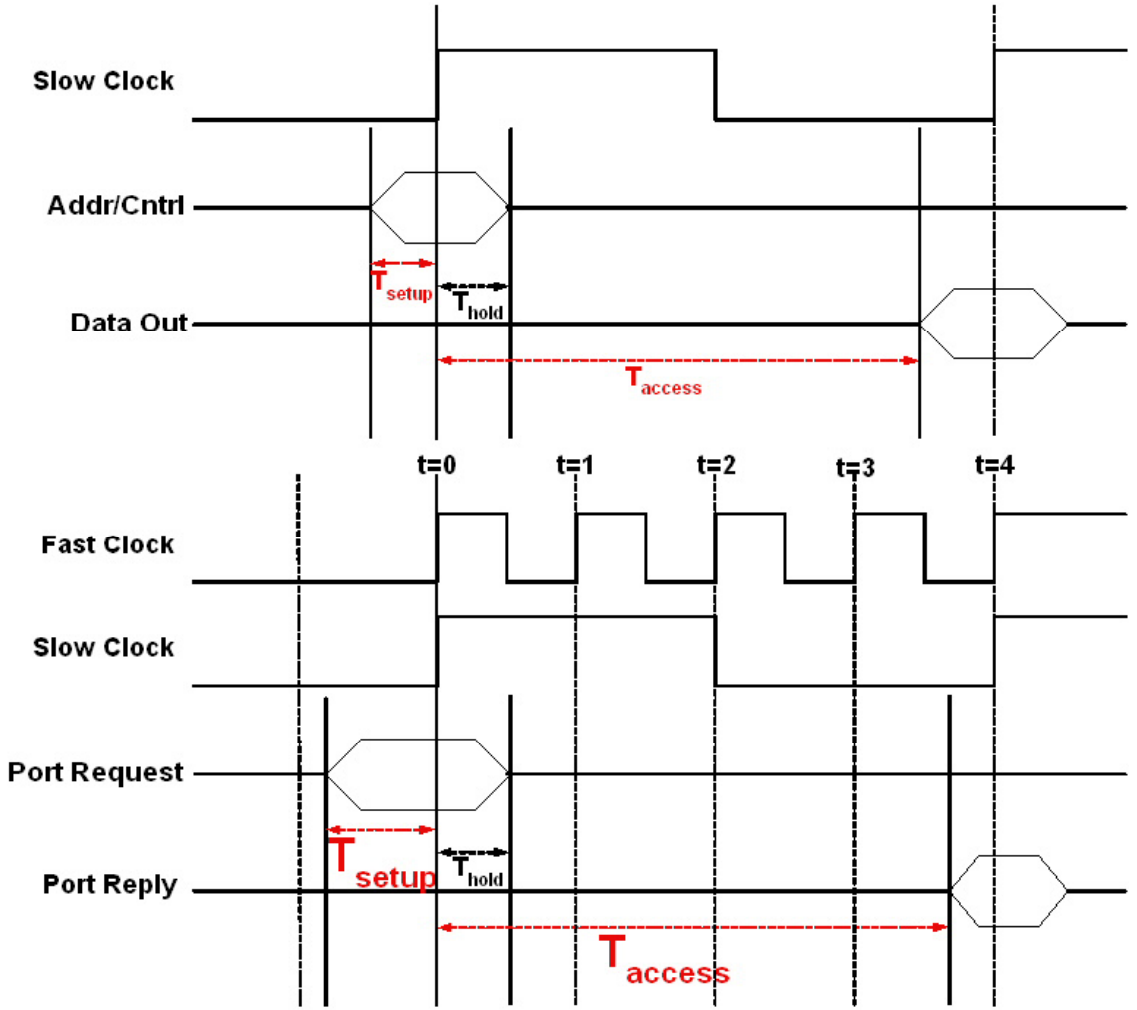


Detailed Datapath



Timing

- Accelerator store has the timing of a normal SRAM
- Only difference: Request may be rejected



Implementation

- Implementation includes:
  - Design in Bluespec
  - Memory from SRAM models generated from a commercial memory compiler
- Parameterized to allow simple varying of:
  - Number of accelerator ports into accelerator store
  - Number of memory accesses per clock cycle
  - Number of handles in the handle table
  - Memory power up and power down time
  - Size and quantity of SRAM blocks

Validation

- Fully unit tested
- Traces from cycle accurate simulator used to verify correctness of accelerator store design
  - 150k cycles

Results

- For validated configuration, used eight 4KB memory blocks
- In 180nm:
  - Dynamic power = 105 uW
  - Leakage power = 400 nW
  - Area = 21,700 um^2
  - Max Frequency = 15.7 MHz
- Estimate for 130nm:
  - System power = 55 uW

Results

- Accelerator store logic uses just 33% of the power of one 4KB memory block
- If through automatic VDD-gating, on average, one memory block is turned off, the accelerator store has justified its power cost