An FPGA Host-Multithreaded Functional Model for SPARC v8

Zhangxi Tan
Computer Science Division
UC Berkeley, USA
xtan@cs.berkeley.edu

Krste Asanovic
Computer Science Division
UC Berkeley, USA
krste@cs.berkeley.edu

David Patterson
Computer Science Division
UC Berkeley, USA
pattersn@cs.berkeley.edu

1. Introduction

The RAMP project [1] aims to create a FPGA based hardware emulator for future manycore systems, which will scale to 1000 processor cores on several multi-FPGA boards, such as BEE3 board [2]. At the same time, the emulated processor cores can support running real life workload (e.g OS and unmodified application binaries) in a cycle accurate manner given a user defined timing model. In this work, we introduce a fine-grained multithreading architecture to build 512 independent SPARC v8 integer functional contexts running at over 150 MHz on a single Xilinx Virtex 5 LX110T FPGA with over 1 GIPS peak emulation throughput. The functional model fully complies with the SPARC v8 standard. Therefore, it is easy to run real OS (e.g. Debian Linux) and diagnostics. With a smaller FPGA, we can build a low cost (<$500) practical 64-way or 128-way multicore emulator that has cycle accurate timing model as well as floating point support.

2. Related Work

Besides the timing emulation model, the study of efficient functional emulation model already drew a lot attentions in the community [3, 4, 5]. Most of previous works [3, 4] attempts to directly map a soft-core processor to FPGA fabric with only a few modifications. This approach suffers from poor per FPGA core count and low clock frequency. In addition, it still takes dozens of high-end FPGAs to implement a 1000 core system at a high cost that only large research groups in university or industry can afford. In [5], a software and hardware hybrid simulation technique is discussed. By exploiting time-multiplexed interleaving, a 90 MHz 16-context SPARC v9 functional pipeline is built on a single Virtex-II FPGA. Though the time-multiplexed technique seems promising, we still believe a good understanding of modern FPGA architecture plus careful engineering will help to create a functional model purely in hardware, which will have a much higher density and even better aggregated performance.

3. Design Methodology

One optimization goal in our design is to achieve a high aggregate emulated instruction throughput given the capacity constraint of FPGA, i.e. MIPS/FPGA. It can be written as the following equation

$$MIPS_{per\ FPGA} = \frac{Clock\ Rate}{CPI_{ideal} + Miss\ Rate * Miss\ Penalty} \times \#\ of\ cores$$

We argue that simply duplicating CPU pipelines is suboptimal. First, most of soft-core processors on FPGA run under 100 MHz, which is way slower than state-of-the-art memory interfaces. Second, standard single issue in-order pipeline has a higher actual CPI due to high cache miss penalty (20~30 cycles). On the other hand, we believe fine-grained host-multithreading is a more efficient approach to improve the pipeline utilization on FPGA moreover the computation density. In above equation, the miss penalty can be reduced to $\left\lceil \frac{miss\ latency}{\#\ of\ threads} \right\rceil$. If there are enough threads, the miss penalty is just 1.

We also seriously reconsider some traditional RISC pipeline optimizations on FPGA such as forwarding network and delay slot. Our initial result on LEON processor [6] in Table 1 quantifies the area and clock frequency differences on a Xilinx Virtex II Pro FPGA after removing pipeline forwarding logic. Under different logic synthesis optimization strategies, the pipeline area is reduced by 26%~32% and frequency is boosted by 18%~58%. However, the spirit of RISC "**simpler**, **smaller** and **faster**" still applies.

Over the years, there have been many improvements in FPGA architecture. For example, powerful embedded DSPs are introduced to provide extra computation capacity. Bandwidth and capacity of BRAM and LUTRAM are nearly doubled in each generation. However, few FPGA processor designs make good use of these enhanced structures, which can fundamentally affect design density and clock frequency. Most of designs focus on functional only, and leave design mapping to CAD tools. Our functional model is designed with FPGA fabric in mind. Area, clock frequency, routing delay and reliability were deliberately considered at very early stage of the design phase.

Overall, our design will run at over 150 MHz on Xilinx Virtex 5 LX110T FPGAs. If we assume a perfect memory model (fixed latency), we can have a rough comparison of emulated instruction throughput between 8 multithreaded functional models and 8 standard LEON cores running at 100 MHz on a single FPGA. Besides supporting more emulated contexts, the multithreaded implementation can have an aggregated throughput over 1 GIPS/FPGA, while the standard version can only reach 533 MIPS per FPGA[1].

4. Pipeline Architecture

Figure 1 shows the architecture of the functional model pipeline. Currently, only a single-issue in-order integer pipeline is implemented. Each pipeline supports up to 64 independent SPARC v8 contexts by fine-grained hardware multithreading with zero context switch overhead. Instruction streams from different threads are issued by a static round-robin scheduler. If a long latency instruction (e.g. memory load/store) is executed, the pipeline will tag the issue thread and make no changes to the thread's architecture registers. When the same thread is scheduled in the next round, the instruction will be 'replayed'. If the result is available, the thread will commit the result and continue to execute the next instruction. Ideally, the access latency is hidden by interleaved execution of other threads. On large FPGAs, we can increase the overall throughput by including multiple pipelines. Figure 2 shows eight pipelines grouped into two clusters in a Xilinx Virtex 5 LX110T FPGA on BEE3 board. Four pipelines in each cluster share one DDR2 memory controller. This model is used to emulate a non-cache-coherent distributed memory system (e.g. data centers). With a different cache/memory subsystem, the multithreaded functional model can also be used to emulate shared-memory multicore systems.

Internally, the functional pipeline only implements a subset of SPARC ISA for minimal area. Microcode handles complex instructions that require a sequence of operations (e.g. atomic operations) or special architecture support, or less frequent operations such as traps and interrupts. Although microcode adds a little complexity at decoding stage, it eliminates a lot of more expensive structures are eliminated. For example, 3-read port register file (for some SPARC store

[1] We assume a fixed actual CPI of 1.5 for LEON. Multithreaded model has a higher miss rate due to smaller per thread cache, i.e. 10% miss for I and 30% miss for D. Loads and stores constitute 30% of the instruction mix.

instructions) might potentially double the dual-port BRAM requirements, while BRAM is one of the critical FPGA resources for state storage and cache.

Per-thread State Storage

To minimize the cost of per-thread state storage, we choose to implement three register windows for each hardware SPARC context (thread). The SPARC specification requires at least two register windows. Three is the maximum number that falls in the 64 boundary (low bound of power of two). The 3-window register file is aligned in a 64 32-bit word chunk in BRAM. Eight of the 64 words in one chunk are used as trap base register (TBR) and scratchpad registers (non-architecture register) in microcode mode. In total, each multithreaded register file consumes eight 18kb BRAM blocks. Other special architecture registers, such as PC/nPC and PSR (processor state register), are mapped to distributed LUTRAMs. All registers can be easily indexed by encoding the thread ID as MSBs of register address.

DSP Mapped ALU

To accelerate digital signal processing, most of FPGAs come with dozens of hard DSP blocks, which are greatly enhanced over the generations. DSP blocks are no longer a simple multiplier-accumulator (MAC). In the latest Xilinx Virtex 5 FPGA, each DSP block is extended to perform 48-bit two's complement add/substract and bit-wise logic operations. A 48-bit pattern detector is also included. We find most of SPARC instructions can be mapped to a single DSP block, including all simple arithmetic/logic instructions and address calculation for LD/ST and JMPL/CALL instructions. Besides, the pattern detector in DSP can be used to generate the ALU zero flag, which consumes the most FPGA resource among the four architecture flags, i.e. negate, zero, carry, overflow. Table 2 presents the area save (in 6-input LUT) when mapping 32-bit and 64-bit ALU in DSP block. The baseline SPARC 32-bit v8 pipeline (without forwarding logic) consumes around 1,500 LUTs. With different CAD tools and constraints, DSP based ALU saves around 5%~10% of the LUTs. Moreover, DSP based ALU can work at over 400 MHz compared to <200 MHz implementation in LUT with standard FPGA routing.

Host-level Caching

In order to reduce host memory traffic, we designed small per-thread split I/D host caches. We choose to implement the simplest direct-mapped write-back and write-allocate cache. The block size is 32 bytes, which matches the burst size of DDR2 memory controller on BEE3. Under a 64-thread configuration, each thread can have a 256-byte I-cache as well as a 256-byte D-cache. The cache tag is physical and indexed by either physical address or virtual address, because the cache size is smaller than the page size. The I/D caches (including tags) are mapped to eighteen 18kb BRAM blocks, which accounts for 69% of total per pipeline BRAM usage.

The cache controller is non-blocking to allow accesses from different threads. To be more specific, it allows a total of 64 outstanding requests coming from 64 different threads. In addition, the cache refill and access paths are completely decoupled by using different physical BRAM ports. Figure 3 shows the D-cache controller along with corresponding pipeline stages. Routing and wide bus multiplexers are well planned in order to achieve the highest performance.

Reliability Concerns

As the transistors keep shrinking, soft-errors become more critical than before. Lesea shows BRAM soft-error rates in the recent 65nm Xilinx FPGA nearly doubled compared to previous 90nm generation [7]. Furthermore, eventually we still need dozens of FPGAs in an O(10K) emulation system by stacking up multiple BEE3 boards. The total failure rate for the whole system can be even worse. On Xilinx Virtex 5 FPGA, for the first time hardware ECC primitives are introduced to protect BRAM content. In our design, all the BRAMs are either protected by these embedded ECC blocks or parity bit generated by logic. Error detection/correction status is also outputted to special monitor circuit. Additionally, content in LUTRAMs can also be protected with parity bit by adding extra LUT resource for parity generation and storage.

Fine-tuned Physical Implementation

To achieve an optimal clock frequency, the whole architecture is aggressively pipelined to 11 physical stages. Pipeline stages near dedicated FPGA primitives such as BRAMs and DSPs are heavily engineered. Some of the pipeline stages are dedicated to route global signals and avoid long combinatorial paths.

All the BRAMs and LUTRAMs are either double clocked or work with both clock edges to double bandwidth. DSP blocks are also double clocked to reduce latency. One observation behind this is that hard blocks such as BRAMs can generally run twice as fast as the rest of logic implemented in LUT, which is limited by routing. Double clocking schemes will create a timing-balanced design. Another reason is because BRAM is still the critical resource that limits our design density. Double clocking can avoid the need of duplicating storage in order to provide more read/write ports.

5. Status and Future Work

We have already completed the Systemverilog design of the functional model, which currently has over 6000 lines of code. The Xilinx Virtex5 FPGA is the first target we support because of platform availability, though the code is constructed in a way to support multiple FPGA architectures (e.g. Altera Stratix III). We are now in the process of functional verification using the SPARC v8 verification suite donated from SPARC International. At the same time, we are working closely with major FPGA CAD vendors (Mentor Graphics and Synplicity) in order to deliver a synthesizable RTL release to the community by late this summer under BSD license.

In the near future, we plan to integrate an IEEE 754 FPU and a simple in-order timing model. A 64-way/128-way cycle accurate emulator will be developed on low cost single Xilinx Virtex 5 FPGA board. Adding complete MMU/TLB implementation as well as I/O and interconnect are also on our schedule in order to support manycore OS research. We would also like to answer other emulation performance questions such as trading thread state for more host-level caching.

6. References

[1] RAMP: Research Accelerator for Multiple Processors. http://ramp.eecs.berkeley.edu

[2] BEE3: http://research.microsoft.com/projects/BEE3/

[3] A. Krasnov, A. Schultz, J. Wawrzynek, G. Gibeling, P.-Y. Droz, RAMP Blue: A Message-Passing Manycore System, International Symposium on Field Programmable Logic and Applications, August 2007

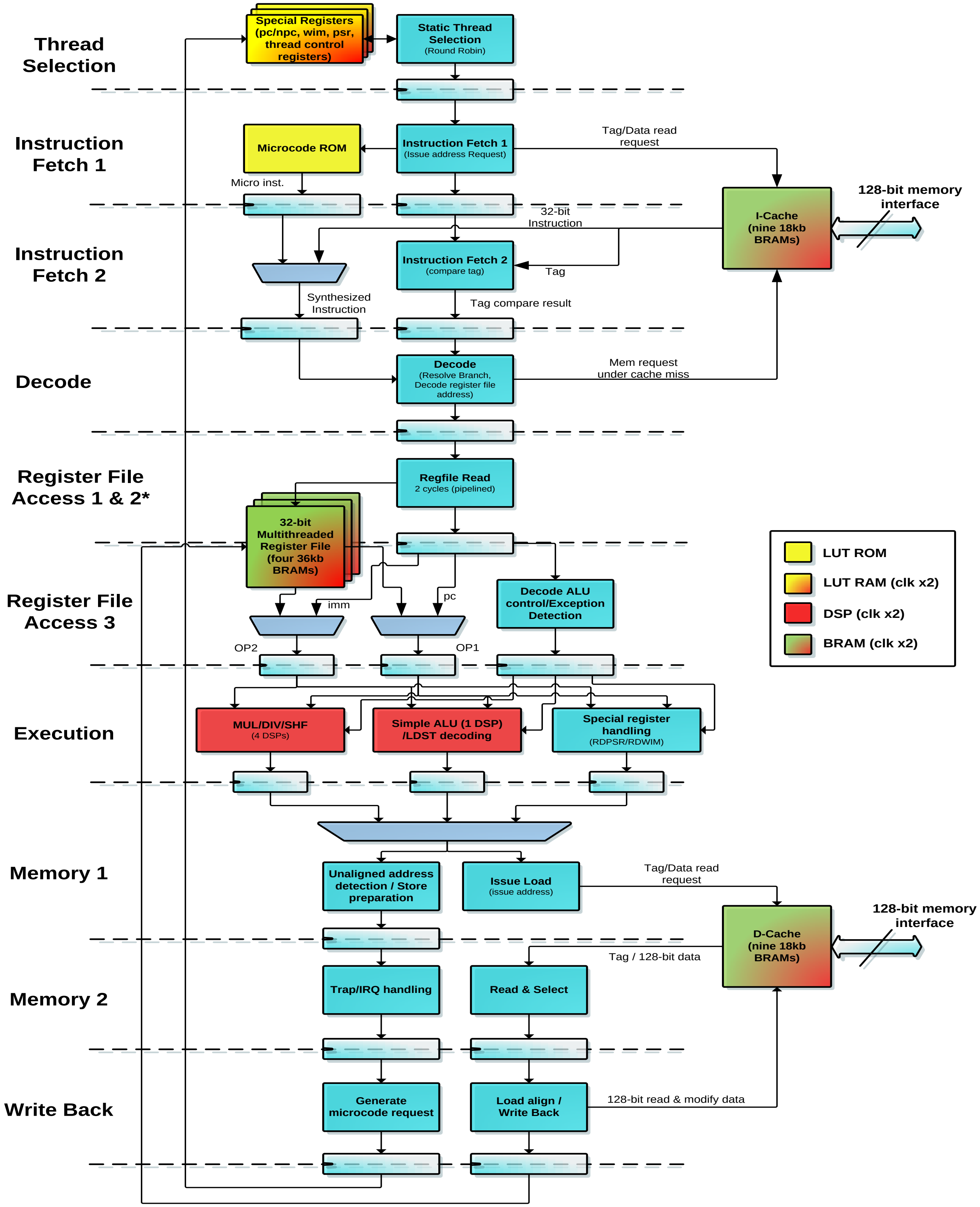
[4] S. L. Lu, P. Yiannacouras, R. Kassa, M. Konow, T. Suh. An FPGA-based Pentium® in a complete desktop system. the International Symposium on Field Programmable Gate Arrays, February 2007.

[5] E. S. Chung, E. Nurvitadhi, J. C. Hoe, B. Falsafi, K. Mai., A Complexity-Effective Architecture for Accelerating Full-System Multiprocessor Simulations Using FPGAs, International Symposium on Field Programmable Gate Arrays, February 2008

[6] Leon3 processor, http://www.gaisler.com.

[7] A. Lesea, Continuing Experiments of Atmospheric Neutron Effects on Deep Submicron Integrated Circuits, Xilinx White Paper 286 for Virtex and Spartan FPGA families, March, 2008

Appendix


Thread Selection
Special Registers (pc/npc, wim, psr, thread control registers)
Static Thread Selection (Round Robin)
Instruction Fetch 1
Microcode ROM
Instruction Fetch 1 (Issue address Request)
Tag/Data read request
Micro inst.
128-bit memory interface
I-Cache (nine 18kb BRAMs)
32-bit Instruction
Instruction Fetch 2
Instruction Fetch 2 (compare tag)
Tag
Synthesized Instruction
Tag compare result
Decode
Decode (Resolve Branch, Decode register file address)
Mem request under cache miss
Register File Access 1 & 2*
Regfile Read 2 cycles (pipelined)
32-bit Multithreaded Register File (four 36kb BRAMs)
Register File Access 3
imm
pc
Decode ALU control/Exception Detection
OP2
OP1
LUT ROM
LUT RAM (clk x2)
DSP (clk x2)
BRAM (clk x2)
Execution
MUL/DIV/SHF (4 DSPs)
Simple ALU (1 DSP) /LDST decoding
Special register handling (RDPSR/RDWIM)
Memory 1
Unaligned address detection / Store preparation
Issue Load (issue address)
Tag/Data read request
128-bit memory interface
D-Cache (nine 18kb BRAMs)
Tag / 128-bit data
Memory 2
Trap/IRQ handling
Read & Select
Write Back
Generate microcode request
Load align / Write Back
128-bit read & modify data

*An extra pipeline stage is dedicated to FPGA routing and demultiplexing operands from double-clocked BRAM.

Figure 1. Multithreaded SPARC v8 functional model (integer pipeline only)

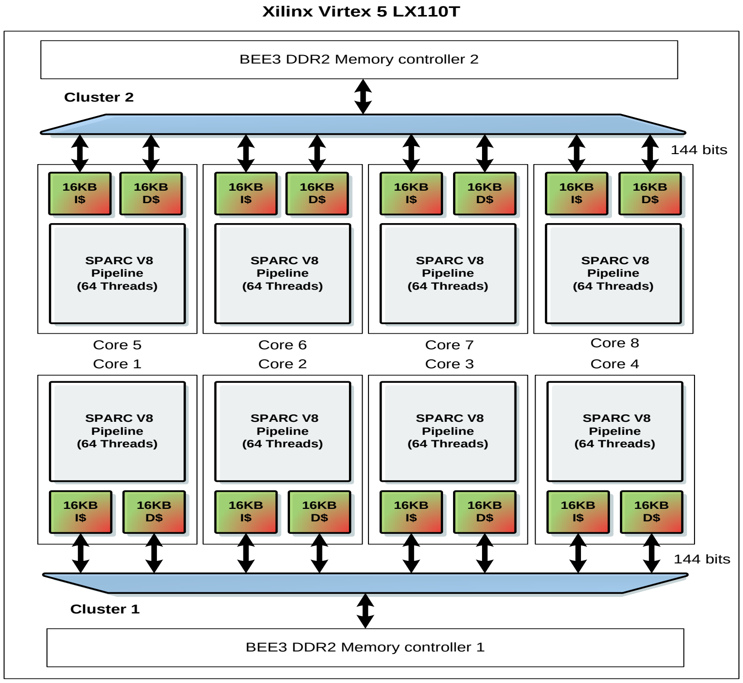


Figure 2. Eight multithreaded pipelines are placed on a single Xilinx Virtex 5 LX110T FPGA for emulating non-cache-coherent distributed memory system. Four pipelines are grouped into one cluster that connects to one DDR2 memory controller. Each pipeline has a 16KB I-Cache and a 16KB D-cache in total, which are equally partitioned among 64 threads.

Table 1. Area and performance comparison of SPARC V8 LEON3 processor after removing standard RISC pipeline optimizations on Xilinx Virtex 2 Pro FPGA. The percentages of improvements are shown in brackets. The results are from Synplify 9.0 with two different optimization strategies (area vs. speed).

		Original (wo. debugging)	No forwarding (wo. debugging)	Original (w. debugging)	No forwarding (w. debugging)
Area Optimized	Frequency	49.3 MHz	77.7 MHz (57.6%)	52.7 MHz	77.7 MHz (47.4%)
	LUT	2637	1943 (26.3%)	3320	2457 (26.0%)
Speed Optimized	Frequency	114.6 MHz	136.1 MHz (18.8%)	123 MHz	144.9 MHz (17.8%)
	LUT	3013	2035 (32.5%)	4198	2938 (30.0%)

Table 2. LUT saving of DSP mapped ALU on Xilinx Virtex 5 FPGA under different tools and constraints

	Synplicity Synplify 9.2		Mentor Graphics Precision 2007a Update 3	
	LUT Savings	Maximum Frequency** implemented in LUT	LUT Savings	Maximum Frequency * implemented in LUT
32-bit ALU*	106	175 MHz	152	210 MHz
64-bit ALU	210	150 MHz	306	170 MHz

* The total base 32-bit pipeline (without forwarding logic) is around 1500 LUTs. The saving is 5%~10%.

** Frequency is after place and route with Xilinx ISE 10.1. There is no significant LUT-count change with Precision when specifying different timing constraints. Relaxing timing constraints in Synplify will reduce the LUT-count a little bit (not shown in the table). However, the numbers are still not as good as those from Mentor Graphics Precision.

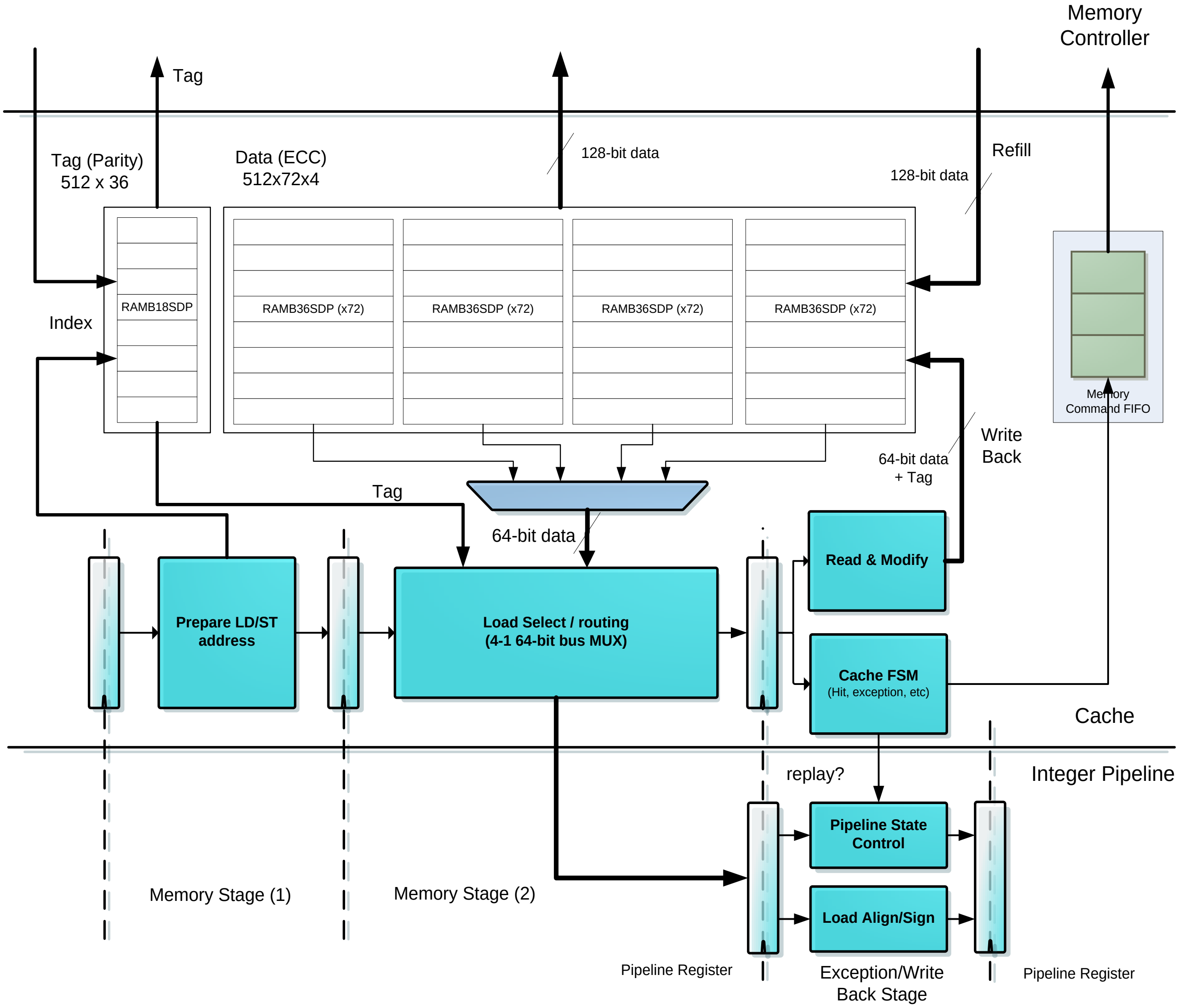


Figure 3. Data cache controller and corresponding pipeline stages.