RAMP Gold
Hardware and Software Architecture

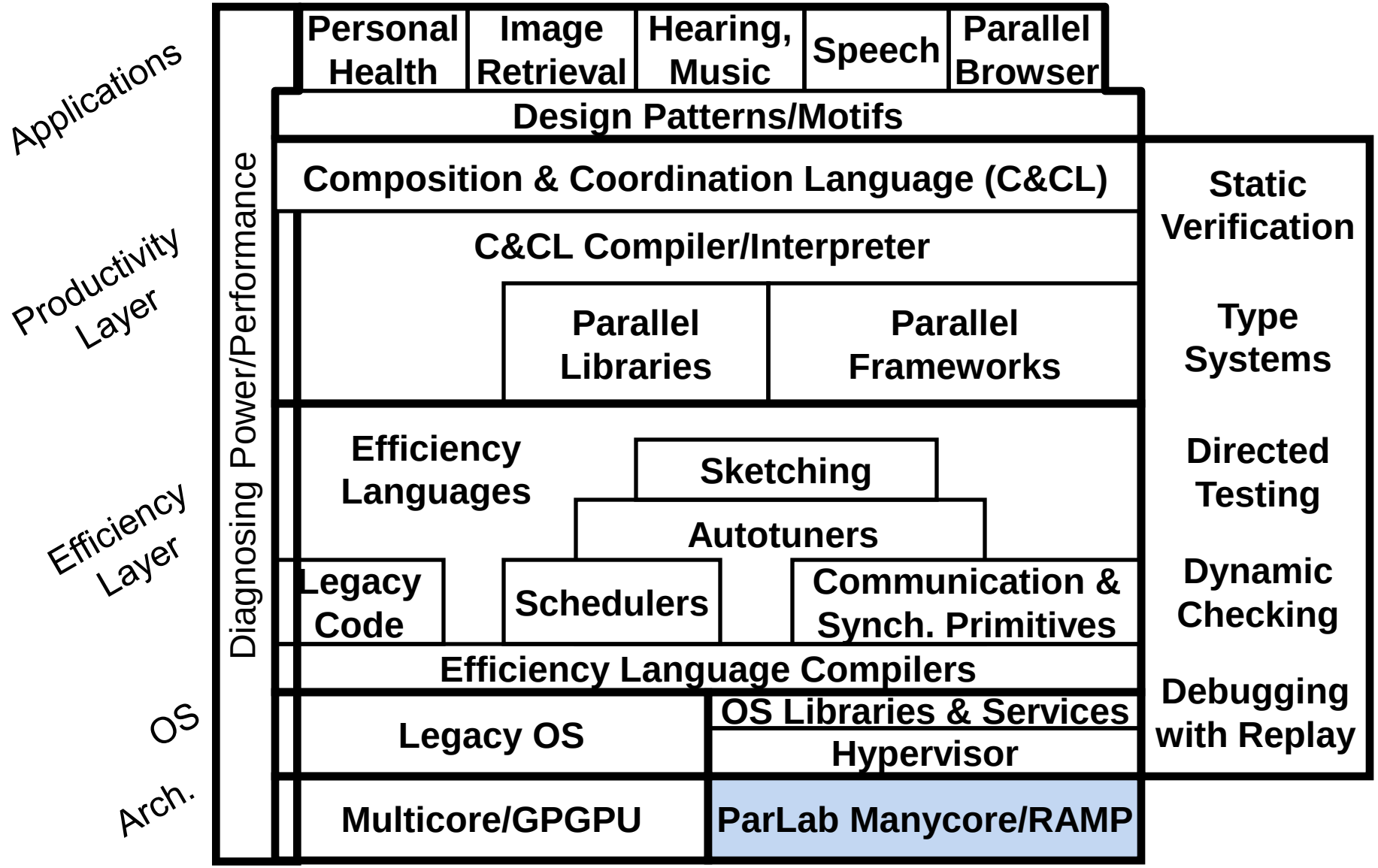
Zhangxi Tan, Yunsup Lee, Andrew Waterman, Rimas Avizienis,
David Patterson, Krste Asanovic

UC Berkeley

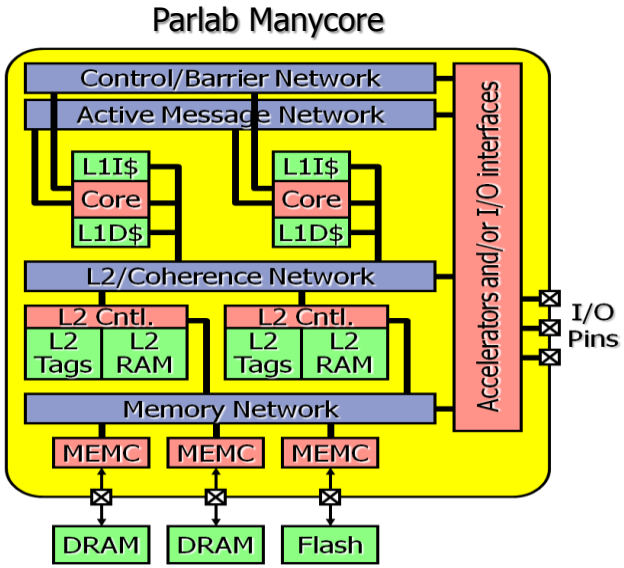
Jan 2009

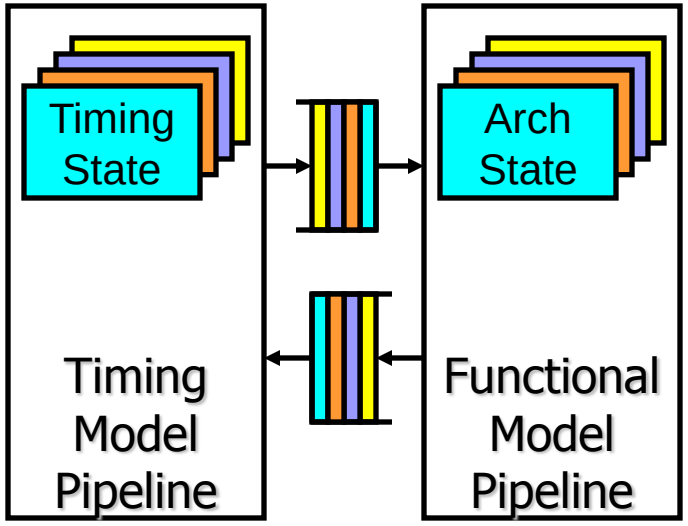
Par Lab Research Overview

Easy to write correct programs that run efficiently on manycore



RAMP Gold : A Parlab manycore emulator



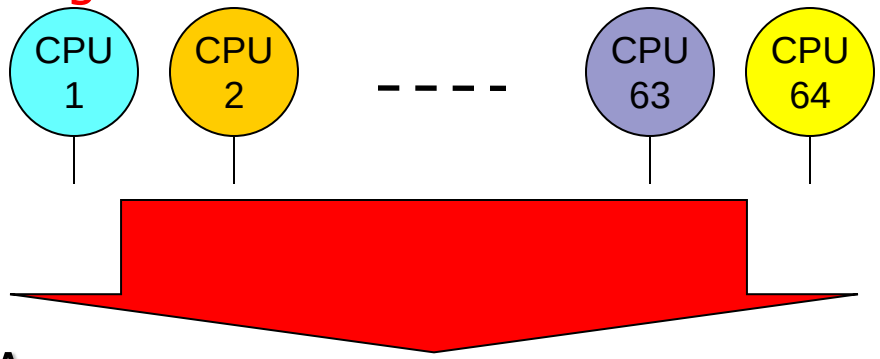


- Leverage RAMP FPGA emulation infrastructure to build prototypes of proposed architectural features
  - Fast enough to run real apps
  - "tapeout" everyday
- RAMP Gold
  - Single-socket tiled manycore target
    - SPARC v8 -> ISA neutural
    - Shared memory, distributed coherent cache
    - Multiple on-chip networks and memory channels
  - Split functional/timing model, both in hardware
  - Host multithreading of both functional and timing models

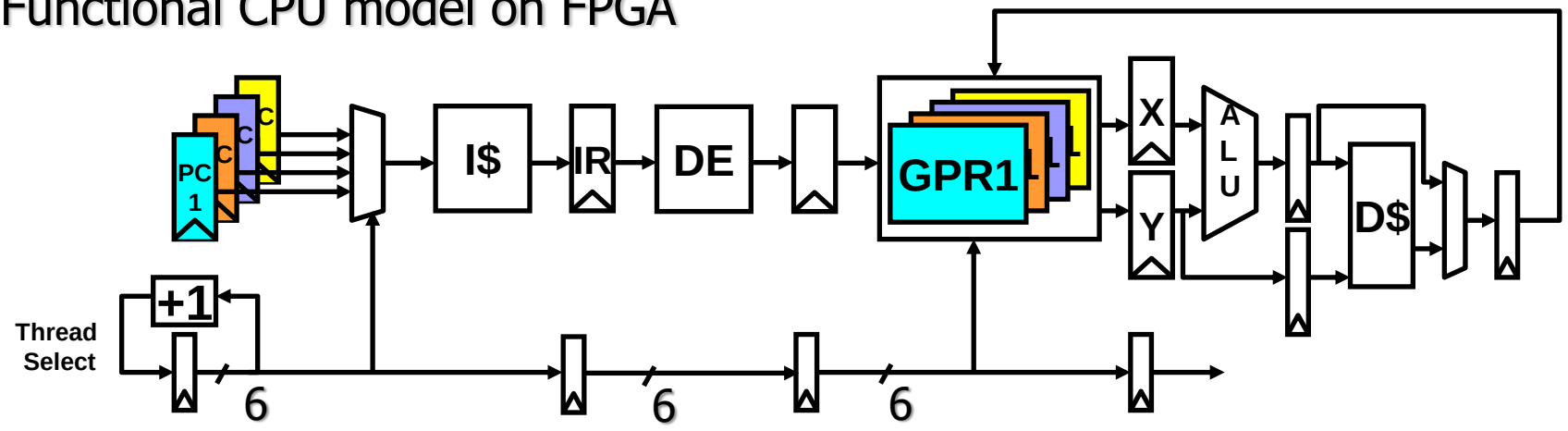
Host multithreading

- Single hardware pipeline with multiple copies of CPU state
  - Virtualized the pipeline with fine-grained multithreading to emulate more target CPUs with high efficiency (i.e. MIPS/FPGA)
  - Hide emulation latencies
  - Not multithreading target

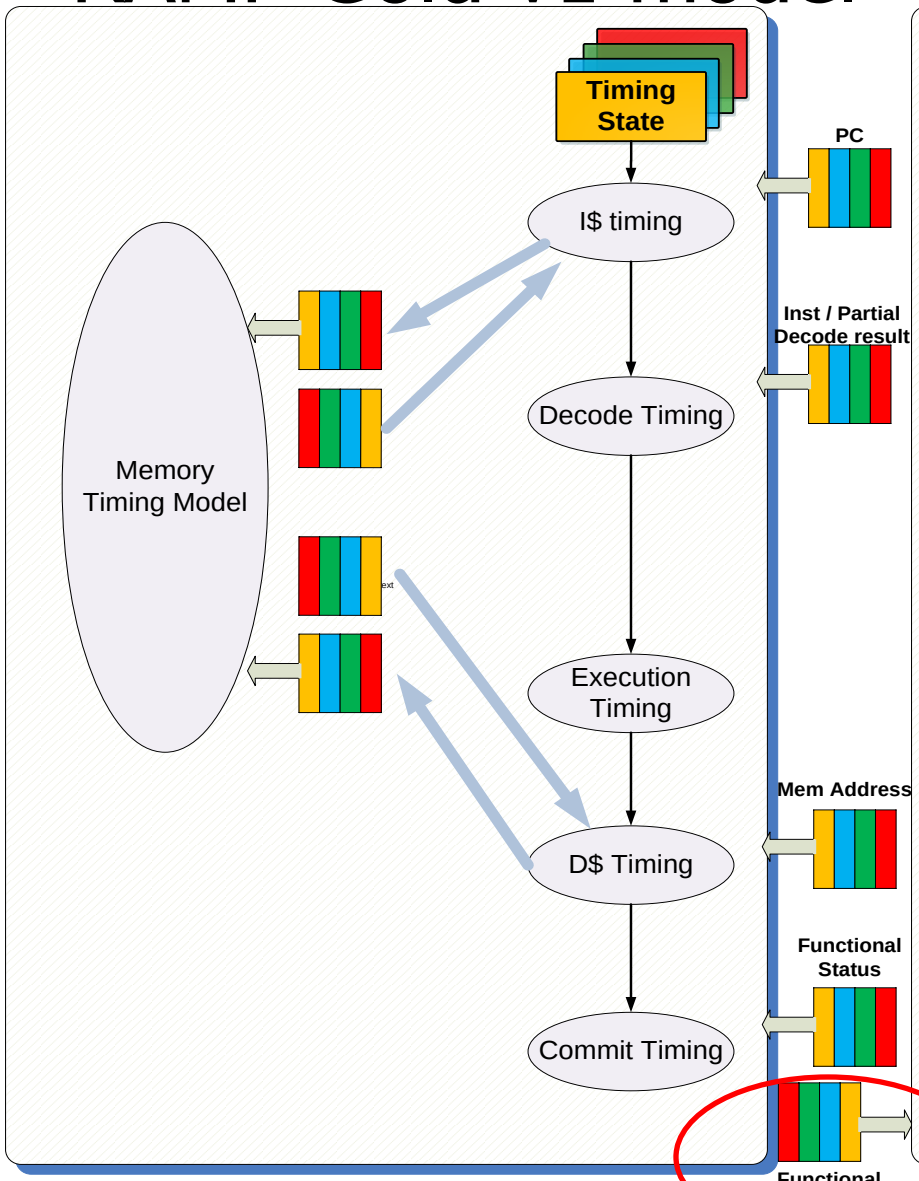
Target Model

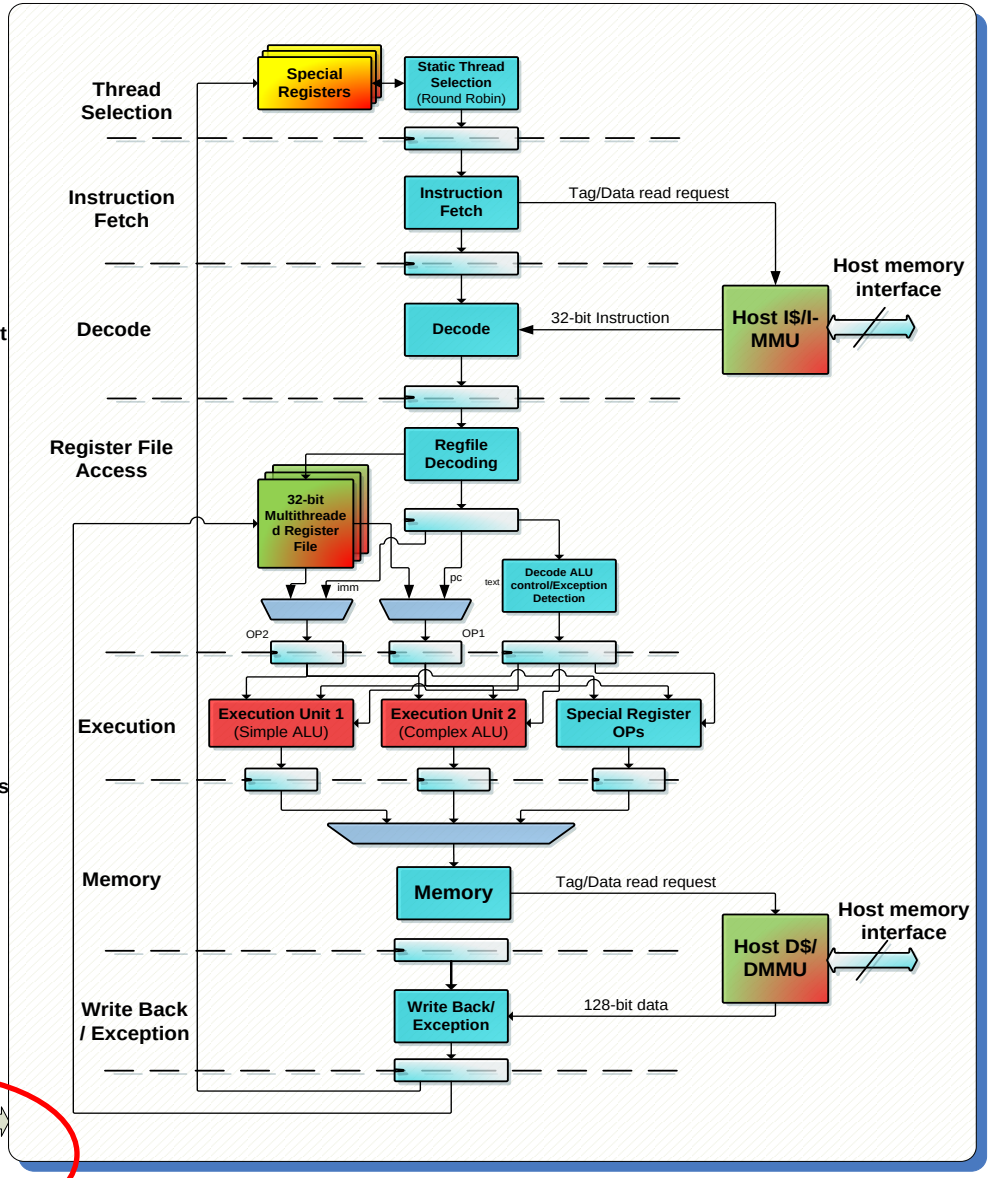


Functional CPU model on FPGA



RAMP Gold v1 model

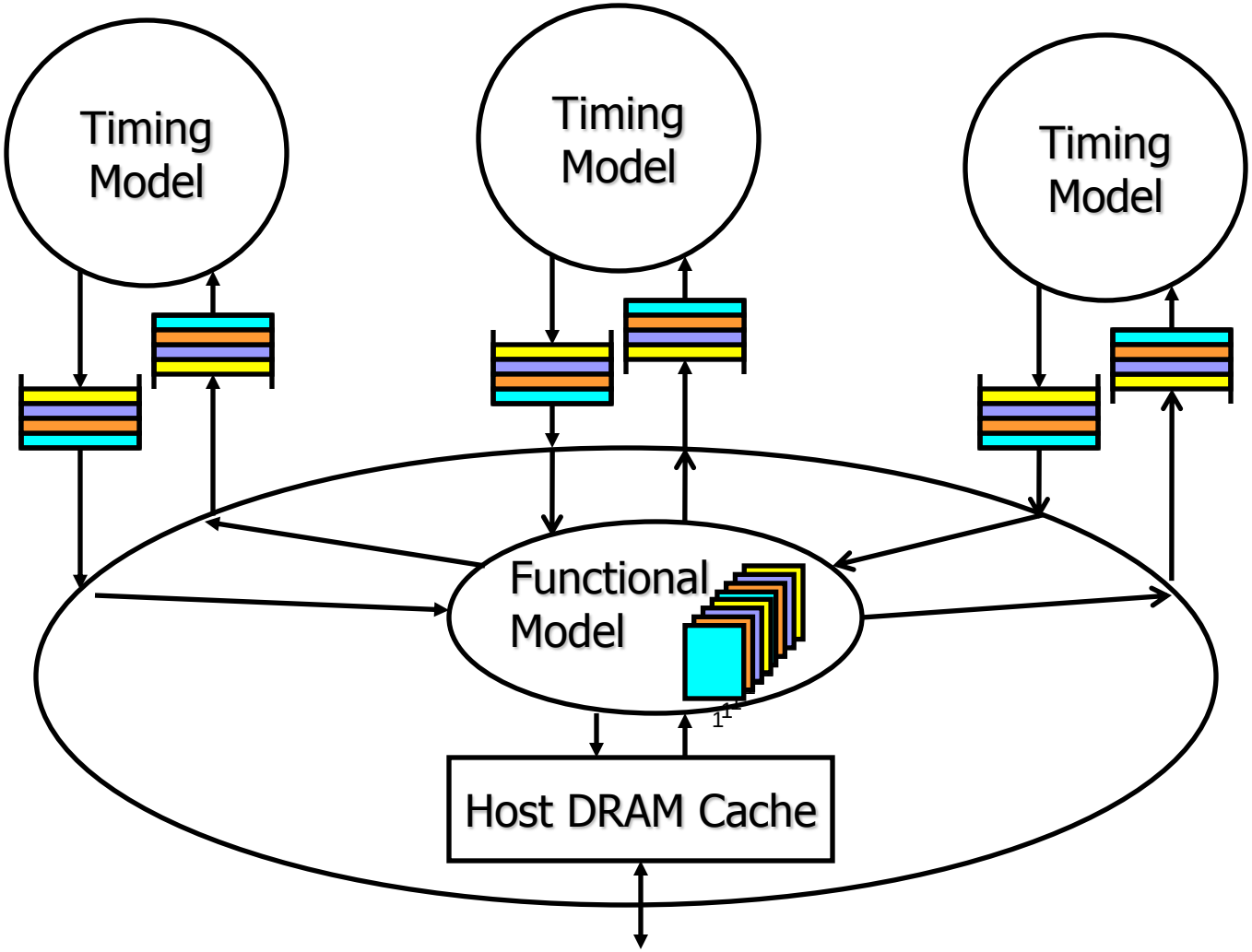




Timing Model

Functional Model

Balancing Func. And Timing Models



- Single functional model supports multiple timing models on FPGAs

RAMP Gold Implementation

- Single FPGA Implementation
  - Low cost Xilinx ML505 board
  - 64~128 cores, 2GB DDR2, FP, timing model, 100~130 MIPS
  - A 64-core functional model demo

- Multi-FPGA Implementation
  - BEE3 : 4 Xilinx Virtex 5 LX155T
  - 1K~2K cores, 64GB DDR2, FP, timing model
    - Higher emulation capacity and memory bandwidth

RAMP Gold Prototyping

- Version 1
  - Single FPGA implementation on Xilinx ML505
  - 64~128 cores, integer only, 2G byte memory, running at 100 MHz
  - Simple timing model
    - Single in-order issue CPU with an ideal shared memory (2-cycle access latency)
  - RAMP performance counter support
  - Full verification environment
    - Software simulator (C-gold)
    - RTL/netlist verification
    - HW on-chip verification
  - BSD license : Everything is built from scratch!

CPU Functional Model (1)

- 64 HW threads, full 32-bit SPARC v8 CPU
  - The same binary runs on both SUN boxes and RAMP
  - Optimized for emulation throughput (MIPS/FPGA)
  - 1 cycle access latency for most of the instructions on host
  - Microcode operation for complex and new instructions
    - E.g. trap, active messages
- Design for FPGA fabric for optimal performance
  - "Deep" pipeline : 11 physical stages, no bypassing network
  - DSP based ALU
  - ECC/parity protected RAM/cache lines and etc.
  - Double clocked BRAM/LUTRAM
  - Fine-tuned FPGA resource mapping

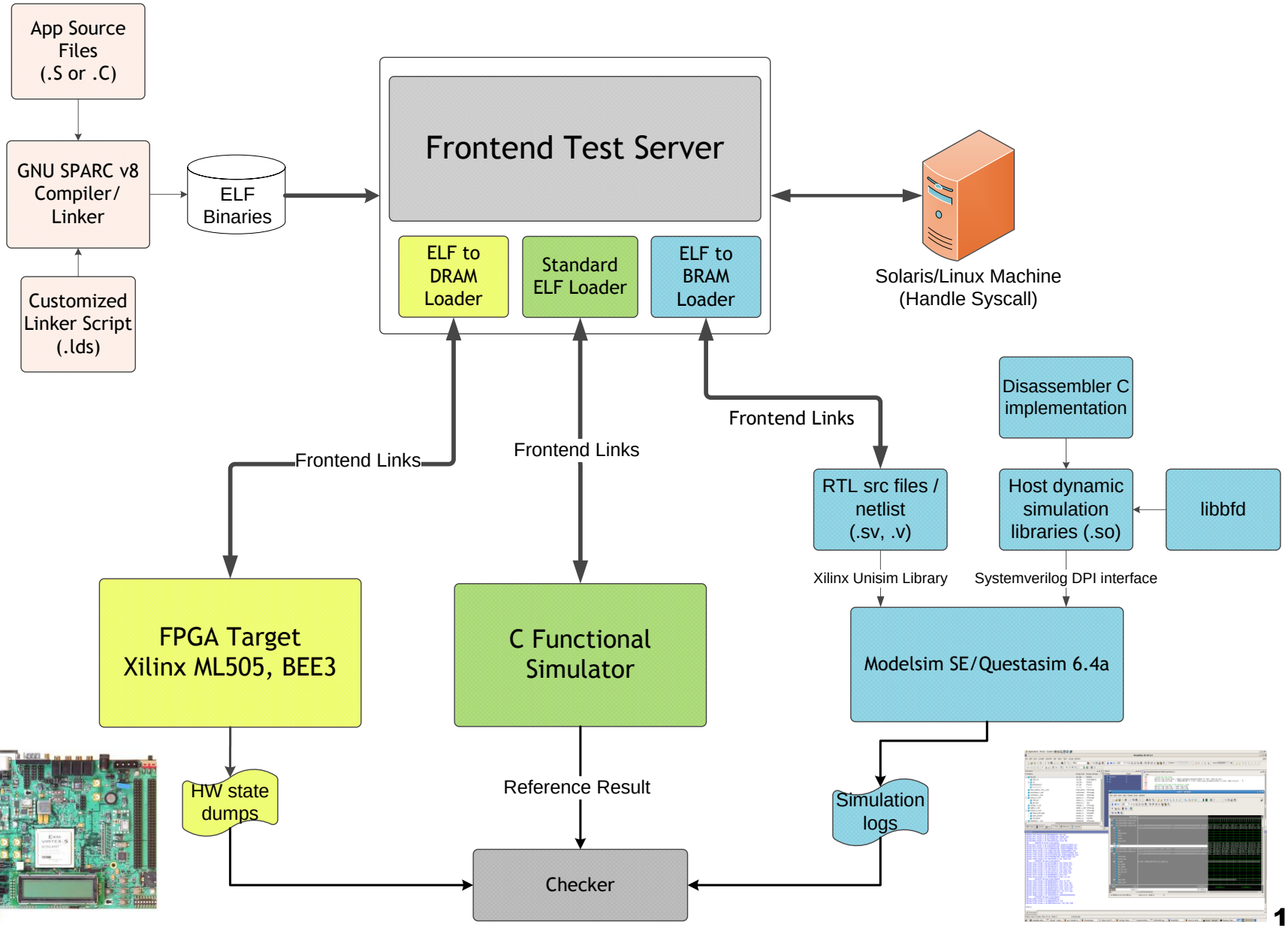
CPU Functional Model (2)

- Status
  - Coded in Systemverilog
  - Passed the verification suite donated by SPARC International
  - Verified against our C functional simulator
  - Mapped and tested on HW @ 100 MHz
    - Maximum frequency > 130 MHz

- FPGA resource consumption (XCV5LX50T)
  - 1 CPU + SRAM controller + memory network

LUT	Register	BRAM	DSP
2,719	4,852	13	2
9%	16%	22%	4%

Verification/Testing Flow

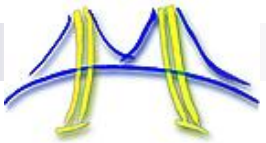


GCC Toolchain

- SPARC cross compiler with newlib
  - Built with (binutils-2.18, gcc-4.3.2/gmp-4.3.2/mpfr-2.3.2, newlib-1.16.0)
  - sparc-elf-{gcc, g++, ld, nm, objdump, ranlib, strip, …}
- Link newlib statically
  - newlib is a C library intended for use on embedded systems
  - C functions in newlib are narrowed down to 19 system calls
    - _exit, close, environ, execve, fork, fstat, getpid, isatty, kill, link, lseek, open, read, sbrk, stat, times, unlink, wait, write
- Now we can compile our C, C++ source code (w/ standard C functions) to a SPARC executable
  - sparc-elf-gcc –o hello hello.c –lc –lsys –mcpu=v8

Frontend Machine

- Multiple backends
  - C-gold functional simulator (link: function calls)
  - Modelsim simulator (link: DPI)
  - Actual H/W (Xilinx ML505, BEE3) (link: gigabit ethernet)
- Narrow interface to support a new backend
  - Host/Target interface
    - CPU reset
  - Memory interface
    - {read,write}_{signed,unsigned}_{8,16,32,64}
- Execute system calls received from the backend
  - Signaled by the backend proxy kernel
  - Map a Solaris system call to a Linux system call and execute



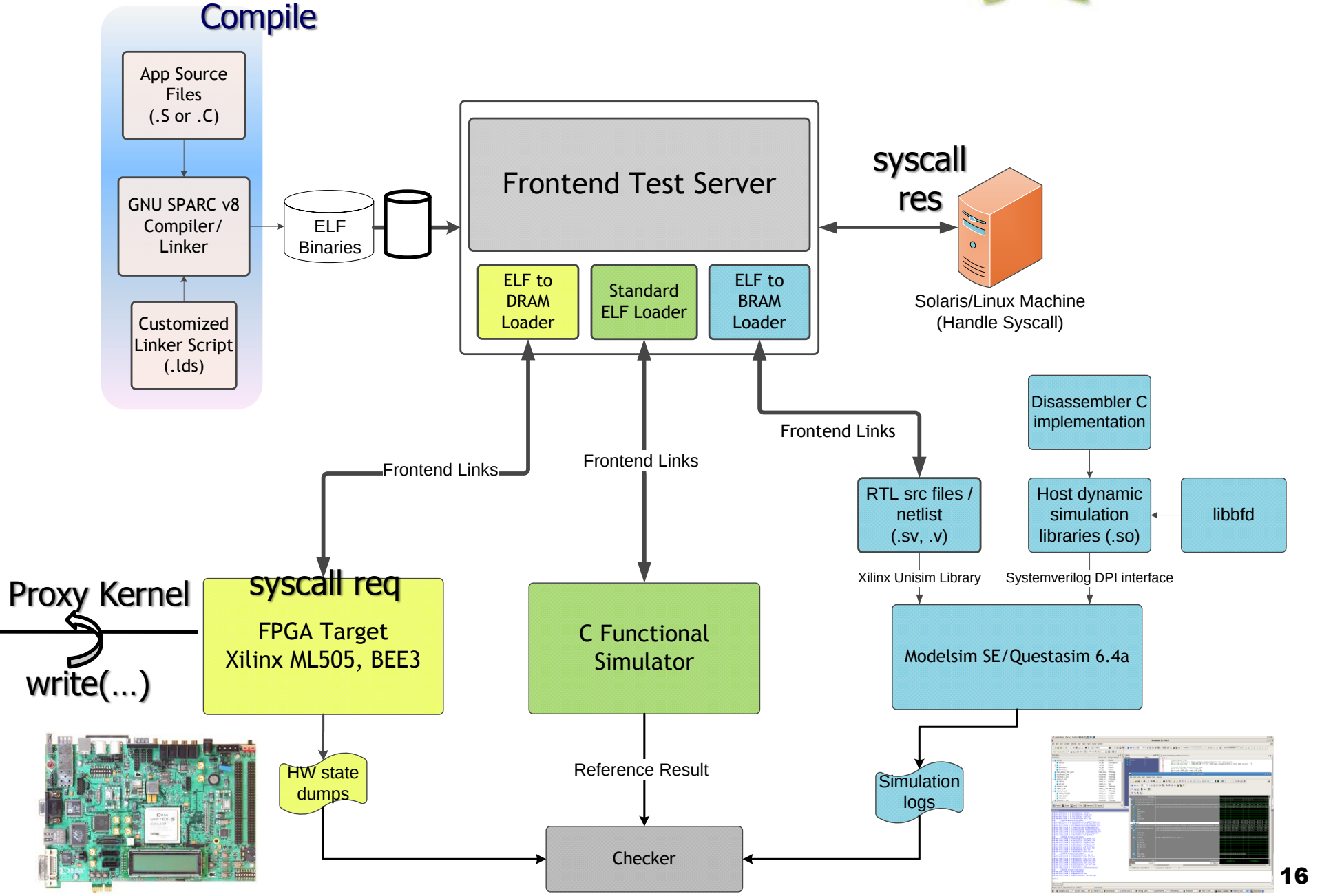
Proxy Kernel

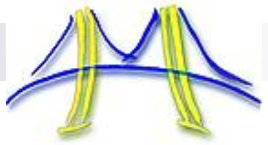
- How could we support I/Os?
  - The target doesn't have any peripherals (e.g. disks)
  - It would be a pain to program a system which can't read or write to anything…
  - It would be more pain to make the peripherals work with the actual H/W
- A minimal kernel which acts as a proxy for system calls invoked by newlib
  - Proxy kernel sends the arguments and the system call number to the frontend machine
  - The frontend machine does the actual system call and returns the results back to the proxy kernel
  - Finally the PC is moved back to the application and everybody is happy

C-gold Functional Simulator

- Baseline functional model to verify our functional model written in system verilog
  - Full 32-bit SPARC v8
  - Includes an IEEE 754 compatible FPU
  - New instruction introduced to support active messages
    - SENDAM
- Written from scratch, no junk in it
  - Very fast, ~25 MIPS
  - Easy to understand
  - Easy to add/modify modules for experiments
  - Flexible parameters (Number of target threads, host threads, …)

Code & Cross-Compile

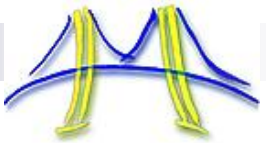




On the fly DEMO

Plans for the Next Version

- Try out several research projects
  - Performance Counter
  - Virtual Local Stores
- Enhance the functional model and timing model
  - Add FPUs
  - Memory/cache timing models

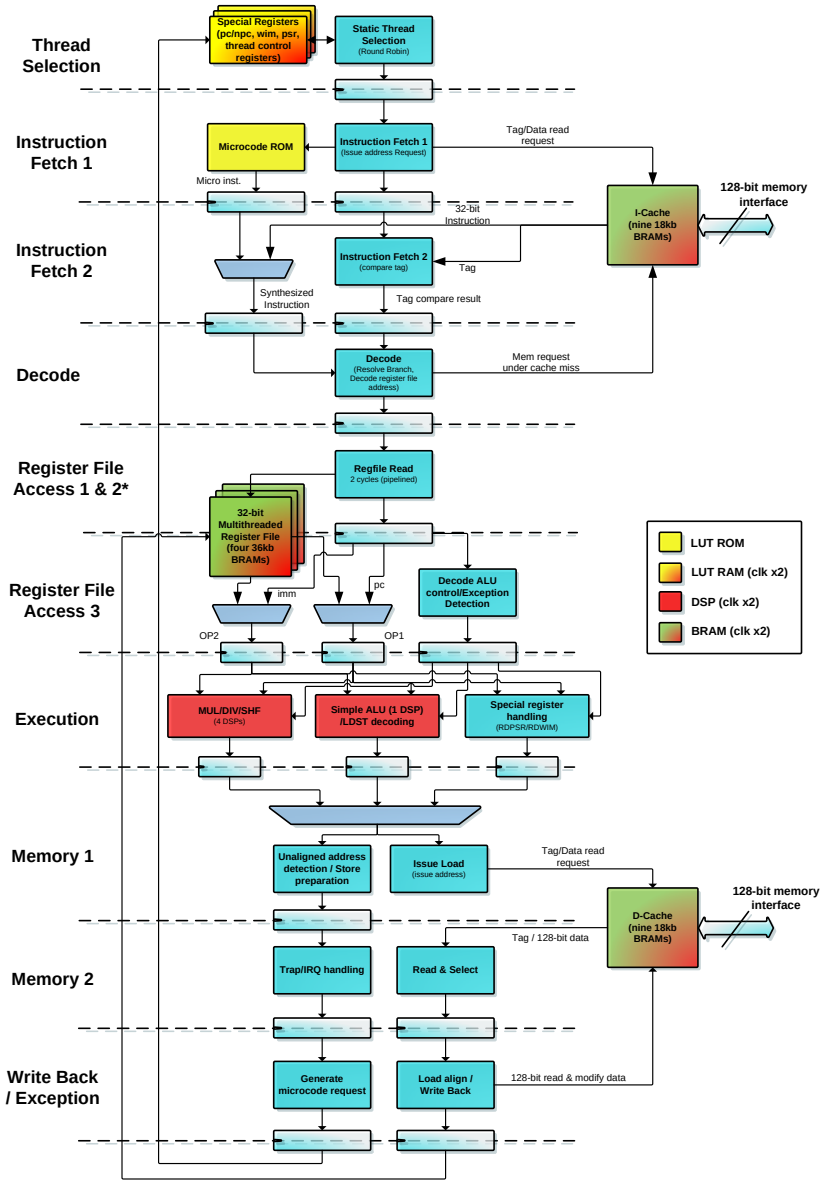


Acknowledgement

- Special thanks to Prof. Kurt Keutzer for help from Synopsys!

Backup Slides

Pipeline Architecture



- Single issue in order pipeline (integer only)
  - 11 pipeline stages (no forwarding) -> 7 logical stages
  - Static thread scheduling, zero overhead context switch
  - Avoid complex operations with "microcode"
    - E.g. traps, ST
  - 32-bit I/O bus (threaded) with interrupt support
- Physical implementation
  - All BRAM/LUTRAM/DSP blocks in double clocked or DDR mode
    - Manually-controlled BRAM mapping
    - LUTRAM mapping by memory compiler
  - Extra pipeline stages for routing
  - ECC/Parity protected BRAMs
    - Deep submicron effect on FPGAs

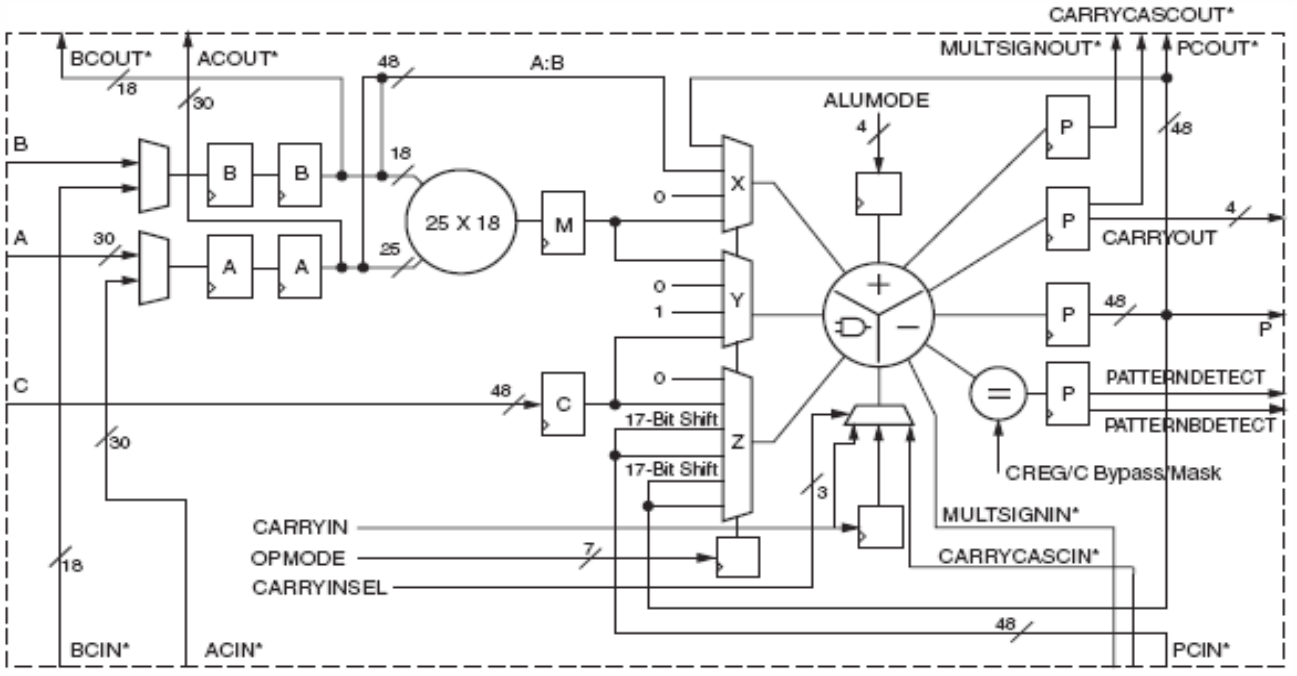
Implementation Challenges

- CPU state storage
  - Where?
  - How large? Does it fit on FPGA?
- Minimize FPGA resource consumption
  - E.g. Mapping ALU to DSPs
- Host cache & TLB
  - Need cache?
  - Architecture and capacity
  - Bandwidth requirement and R/W access ports
    - host multithreading amplifies the requirement

State storage

- Complete 32-bit SPARC v8 ISA w. traps/exceptions
- All CPU states (integer only) are stored in SRAMs on FPGA
  - Per context register file -- BRAM
    - 3 register windows stored in BRAM chunks of 64
      - 8 (global) + 3*16 (reg window) = 54
  - 6 special registers
    - pc/npc -- LUTRAM
    - PSR (Processor state register) -- LUTRAM
    - WIM (Register Window Mask) -- LUTRAM
    - Y (High 32-bit result for MUL/DIV) -- LUTRAM
    - TBR (Trap based registers) -- BRAM (packed with regfile)
- Buffers for host multithreading (LUTRAM)
- Maximum 64 threads per pipeline on Xilinx Virtex5
  - Bounded by LUTRAM depth (6-input LUTs)

Mapping SPARC ALU to DSP



- Xilinx DSP48E advantage
  - 48-bit add/sub/logic/mux + pattern detector
    - Easy to generate ALU flags: < 10 LUTs for C, O
  - Pipelined access over 500 MHz

DSP advantage

- Instruction coverage (two double clocked DSPs / pipeline)
  - 1 cycle ALU (1 DSP)
    - LD/ST (address calculation)
    - Bit-wise logic (and, or, …)
    - SETHI (value by pass)
    - JMPL, RETT, CALL (address calculation)
    - SAVE/RESTORE (add/sub)
    - WRPSR, RDPSR, RDWIM (XOR op)
  - Long latency ALU instructions (1 DSP)
    - Shift/MUL (2 cycles)

- 5%~10% logic save for 32-bit data path

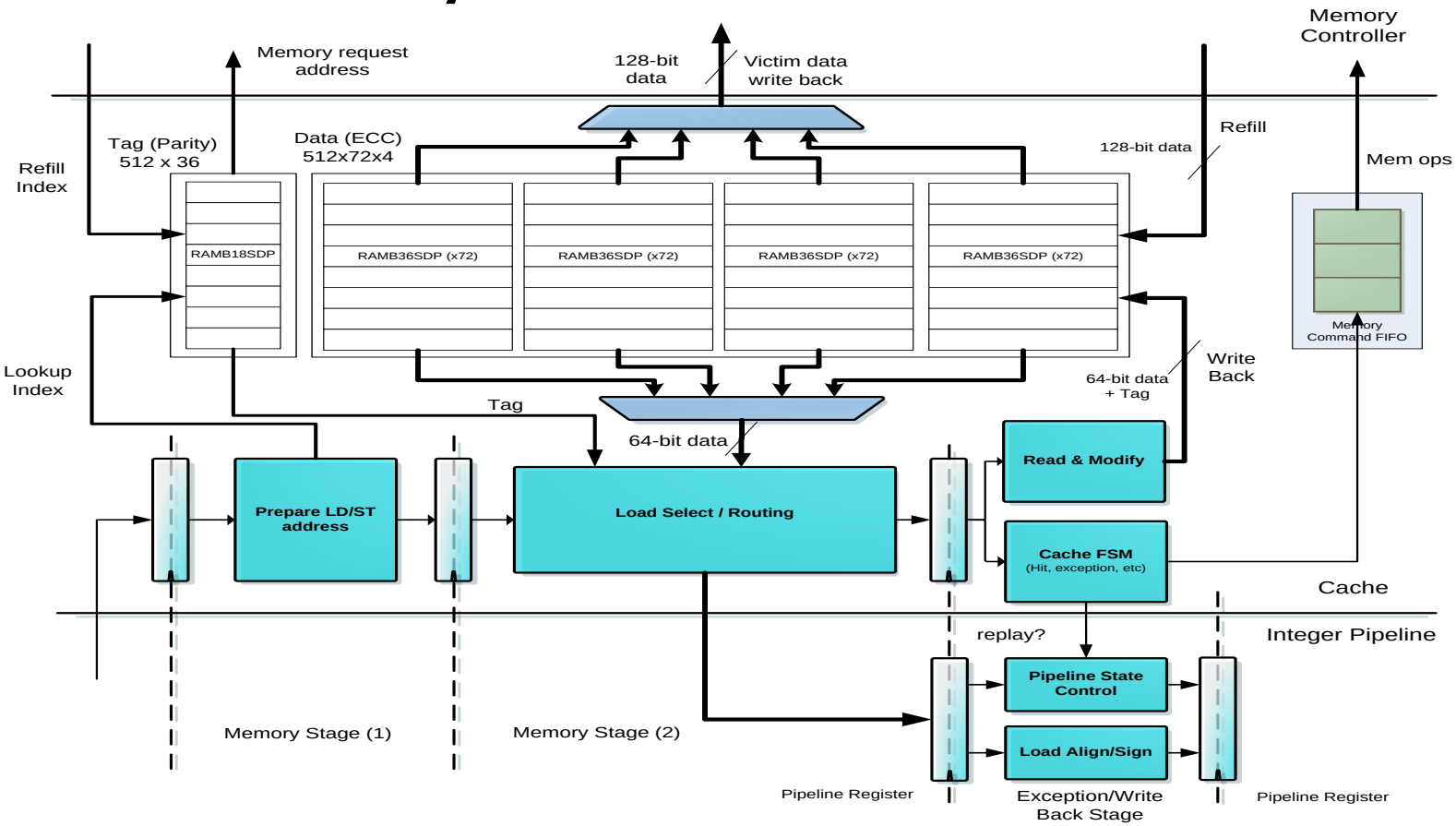
Host Cache/TLB

- Accelerating emulation performance!
  - Need separate model for target cache

- Per thread cache (Partitioned)
  - Split I/D direct-map write-allocate write-back cache
    - Block size: 32 bytes (BEE3 DDR2 controller heart beat)
    - 64-thread configuration: 256B I$, 256B D$
      - Size doubled in 32-thread configuration
    - Non-blocking cache, 64 outstanding requests (max)
    - Physical tags, indexed by virtual or physical address
      - $ size < page size
  - 67% BRAM usage

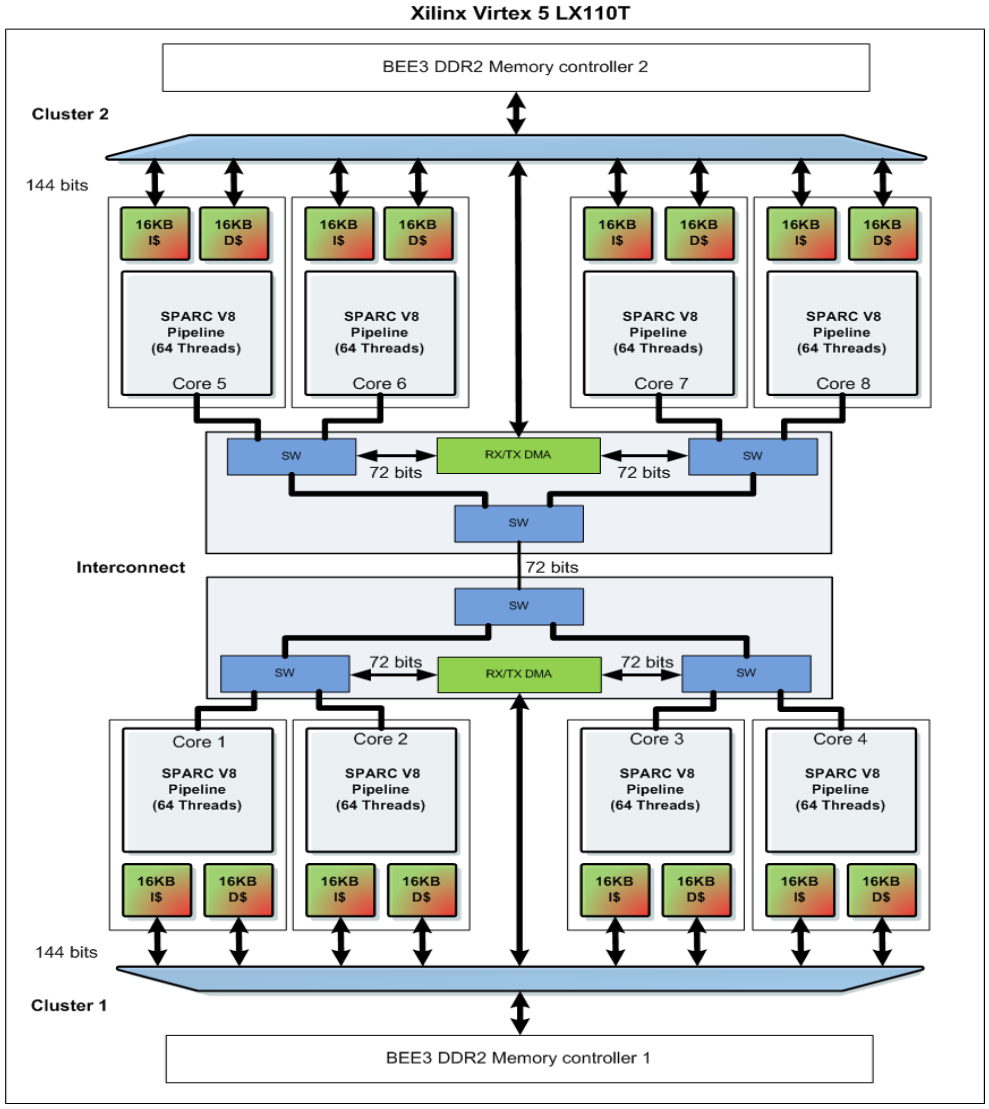
- Per thread TLB
  - Split I/D direct-map TLB: 8 entries ITLB, 8 entries DTLB
  - Dummy currently
    - Static translation for Solaris virtual address layout

Cache-Memory Architecture



- Cache controller
  - Non-blocking pipelined access (3-stages) matches CPU pipeline
  - Decoupled access/refill: allow pipelined, OOO mem accesses
  - Tell the pipeline to "replay" inst. on miss
  - 128-bit refill/write back data path
    - fill one block at 2x clk rate

Example: A distributed memory non-cache coherent system



- Eight multithreaded SPARC v8 pipelines in two clusters
  - Each thread emulates one independent node in target system
  - 512 nodes/FPGA
  - Predicted emulation performance:
    - ~1 GIPS/FPGA (10% I$ miss, 30% D$ miss, 30% LD/ST)
    - x2 compared to naïve manycore implementation
- Memory subsystem
  - Total memory capacity 16 GB, 32MB/node (512 nodes)
  - One DDR2 memory controller per cluster
  - Per FPGA bandwidth: 7.2 GB/s
  - Memory space is partitioned to emulate distributed memory system
  - 144-bit wide credit-based memory network
- Inter-node communication (under development)
  - Two-level tree network with DMA to provide all-to-all communication

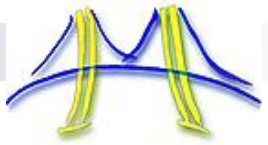
Project Status

- Done with RTL implementation
  - ~12,000 lines synthesizable Systemverilog code
  - FPGA resource utilization per pipeline on Xilinx V5 LX110T
    - ~4% logic (LUT), ~10% BRAM
    - Max 10 pipelines, but back off to 8 or less

- Built RTL verification infrastructure
  - SPARC v8 certification test suite (donated by SPARC international) + Systemverilog
  - Can be used to run more programs but very slow (~0.3 KIPS)

Project Status

- Passed all SPARC v8 integer diagnostics in pre-synthesized RTL simulation
- Run single threaded Solaris apps (5 syscall supported so far)
- Working on HW verification after synthesis and P&R
  - Synthesized with an alpha version of Synplify
  - Will support MentorGraphics Precision 2008a Update 2 in late Nov
- Planned Release in Jan 09
  - 64/128 emulated CPUs on Xilinx ML505 board @ $500 + 2GB DDR2 DRAM cost
  - Source code will be available under BSD license



Thank you