OpenSPARC Slide-Cast

In Twelve Chapters
Presented by OpenSPARC designers, developers, and programmers
• to guide users as they develop their own OpenSPARC designs and
• to assist professors as they teach the next generation

Sun microsystems

OpenSPARC

Chapter Seven

OPENSPARC FPGA IMPLEMENTATIONS

Thomas Thatcher
Staff Engineer
OpenSPARC Evangelist
Sun Microsystems

Agenda

- OpenSPARC T1 hardware package
  - > Download Contents
- Simulation
- Synthesis
- Implementation of an OpenSPARC T1 system on FPGA

OpenSPARC T1 Hardware Package

- Documentation
  - doc/
- Full RTL
  - design/sys/iop
- Simulation scripts & full verification suite
  - verif/env — Simulation Environment files
  - verif/diag — Test lists and assembly code for tests
- Synthesis scripts (Design Compiler, Synplicity, and XST)
  - design/sys/synopsys — Synopsys synthesis scripts
  - design/sys/synplicity — Synplicity FPGA synthesis scripts
  - design/sys/xst — Xilinx XST synthesis scripts
- Xilinx EDK Project for full system on FPGA
  - design/sys/edk

RTL Hierarchy

- **Top level block for FPGA implementation**
  - design/sys/iop/iop_fpga.v
- **RTL Path**
  - design/sys/iop/
    - l2b/rtl — Level-2 Cache
    - ccx/rtl — Cache Crossbar
    - ccx2mb/rtl — Cache Crossbar to MicroBlaze adapter
    - fpu/rtl — Floating-Point Unit
    - dram/rtl — DDR2 DRAM controller
- **SPARC Core**
  - design/sys/iop/sparc/
    - rtl/ — Top-level code
    - ifu/rtl — Instruction Fetch Unit – Includes ITLB and I-Cache
    - exu/rtl — Execution Unit
    - lsu/rtl — Load-Store Unit – Includes DTLB and D-Cache
    - ffu/rtl — Floating-Point Front-end – Includes FP register file
    - tlu/rtl — Trap Logic Unit

Verification Environments

- Core1: Simulate a single SPARC core
  - One SPARC core
  - Level 2 cache
  - Memory Controller
  - Memory model
- Thread1: Simulate one single-thread SPARC core
  - Same as Core 1, except that SPARC core is single thread
- Chip8: Simulate the entire OpenSPARC T1
  - Eight SPARC cores
  - Level 2 Cache
  - I/O Subsystem
  - Memory controller
  - Memory model
- For the netlist (gate level) simulation, use vector playback methodology (provided in DV guide)

Synthesis Scripts

- Scripts to run a synthesis tool
  - tools/bin/rsyn — Run Synopsys Design Compiler
  - tools/bin/rsynp — Run Synplicity FPGA Synthesis
  - tools/bin/rxil — Run Xilinx XST FPGA synthesis
- Input scripts for the synthesis tools
  - design/sys/synopsys — Input scripts for Design Compiler
  - design/sys/synplicity — Input scripts for Synplicity
  - design/sys/xst — Input scripts for XST
- Example Synthesis command:
  - rsynp sparc — Synthesize the SPARC core with Synplicity

Agenda

- OpenSPARC T1 hardware package
- Download Contents
- > Simulation
- Synthesis
- Implementation of an OpenSPARC T1 system on FPGA

OpenSPARC Verification Environment

- Every test is written as an assembly language program
  - Must include “hboot.s” for reset code
- The SPARC assembler is run to generate an executable
- The ELF executable is converted to a memory image
  - Virtual memory tables are added at this point
- The memory image is loaded into the memory model
- The simulation starts with a reset
- An architectural simulator (SAS) runs in lock-step with the RTL, checking the state at the end of each instruction

Typical Test Flow

- The Reset pin is asserted and the chip is initialized.
- The I/O block sends a Power-On Reset (POR) interrupt to a core (usually core 0, thread 0)
- The core wakes up, and begins fetching from address 0xfff0000020 (POR trap handler in the Boot ROM)
- Reset code will turn on caches, TLBs
- Control will be passed to the user code for the test

Test Completion

- At the end of the test, code will perform a software trap
- The trap is to one of two locations
  - GOOD_TRAP address indicates success
  - BAD_TRAP address indicates a problem
  - NOTE: There are two or three addresses for GOOD_TRAP and two or three for BAD_TRAP
    - User trap table, Supervisor trap table, Hypervisor trap table
- Example:

```
good_end:
   ta      T_GOOD_TRAP
   nop
```

How to Run Diagnostic Tests

- Simulations run with sims
  - Full Regresssion:
    - % sims -sim_type=vcs -group=core1_mini
  - Common regressions
    - thread1_mini thread1_full
    - core1_mini core1_full
    - chip8_mini chip8_full
  - Reporting Results:
    - % regreport $PWD/2006_01_25_0 > report.log
  - Single Test:
    - % sims -sim_type=vcs -sys=core1 -sas verif/diag/assembly/arch/exu/exu_add.s

Simulation Output

- A directory is created for each test
  - Important Files:
    - diag.s — Copy of the original assembly language file
    - diag.exe — ELF executable of the test created by assembler
    - mem.image — Memory image of the test, including virtual memory tables
    - symbol.tbl — Symbol table for the elf executable
    - sim.log — Simulation log file
    - sims.log — Log from sims program: including simulation log
    - sas.log — Log file created by the architectural simulator
- Simulation Log file output
    - Time 47800 Test reached GOOD_TRAP

Agenda

- OpenSPARC T1 hardware package
- Download Contents
- Simulation
- > Synthesis
- Implementation of an OpenSPARC T1 system on FPGA

SPARC Core Options

- The SPARC core contains the following options
  - > Set by compiler defines
- Options:
  - > FPGA_SYN — Optimize code for FPGA
    - – Required for all other options
  - > FPGA_SYN_1THREAD — Create a single-thread core
  - > FPGA_SYN_NO_SPU — Do not include the SPU
  - > FPGA_SYN_8TLB — Reduce # of TLB entries to 8 (from 64)
  - > FPGA_SYN_16TLB — Reduce # of TLB entries to 16
  - > CONNECT_SHADOW_SCAN — Connect shadow scan in RTL

Running Synplicity FPGA Synthesis

- Setting Compile Options:
  - Edit file:
    - design/sys/synplicity/env.prj
  - Add Line:
    - set_option -hdl_define -set *"FPGA_SYN FPGA_SYN_1THREAD"*
- Synthesis Command:
  - %rsynp -all — Synthesize all blocks
  - %rsynp -device=XC5VLX110 sparc — Synthesize sparc core, specify device
- Synthesis Output
  - design/sys/iop/sparc/synplicity — Directory where output files found
  - XC5VLX110/ — Directory for target device
  - sparc.edf — EDIF output netlist
  - sparc.srr — Synthesis log file

Running XST Synthesis

- Setting Compile Options
  - > Edit File:
    - design/sys/iop/include/xst_defines.h
  - > Add Lines:
    - `define FPGA_SYN
    - `define FPGA_SYN_1THREAD
- Synthesis Command:
  - %rxil -all — Synthesize all blocks
  - %rxil -device=XC5VLX110 sparc — Synthesize sparc core, specify device
- Synthesis Output
  - design/sys/iop/sparc/xst — Directory where output files found
  - XC5VLX110/ — Directory for target device
  - sparc.ngc sparc.v — Xilinx/Verilog output netlists
  - sparc.srp — Synthesis Log file

Agenda

- OpenSPARC T1 hardware package
- Download Contents
- Simulation
- Synthesis
- > Implementation of an OpenSPARC T1 system on FPGA

Why FPGAs

- Community Requested it
  - Primarily academic community
  - As a platform for course-work design and research
  - Gentler introduction to commercial server class design
  - Basic building block for more interesting variants
- Need a hardware prototype board, that is
  - Re-configurable – FPGA based
  - Scalable – Available in large quantity throughout the world
  - Standardized – Supports standardized peripherals
  - Economical

FPGA Implementation: Goals

- Proliferation of OpenSPARC Technology
- Proliferation of Xilinx FPGA Technology
  - Make OpenSPARC FPGA-Friendly
  - Create reference design with complete system functionality
  - Boot Solaris/Linux on the reference design
  - Open it up ..
  - Seed ideas in the community

Enable multi-core research

FPGA Implementation: Benefits

- FPGAs provide a flexible design environment
  - > Fast turnaround for changes
  - > Enables experimentation in hardware
  - > Speeds up verification time
- Cost Savings
  - > Don't have to pay fabrication costs for each new chip

Creating an FPGA-friendly Design

The following changes were made to the OpenSPARC T1 code

- Re-code sections for more efficient FPGA synthesis
  - Use Block RAMs effectively
  - Efficiently synthesize logic
- Put in options to reduce size
  - Four threads --> one thread
  - Reduce TLB entries from 64 to 8
  - Remove modular arithmetic unit from design

FPGA Implementation

- Initial version released May 2006

  (on OpenSparc.net website)

  > full 8-core, 32-thread

  > First-cut implementation;
  not yet optimized for Area/Timing

  > Synplicity scripts for Xilinx/Altera FPGAs

- Reduced version released Mar 2007 – Release 1.4

  > Single-core, single-thread

  > Reduced size TLB

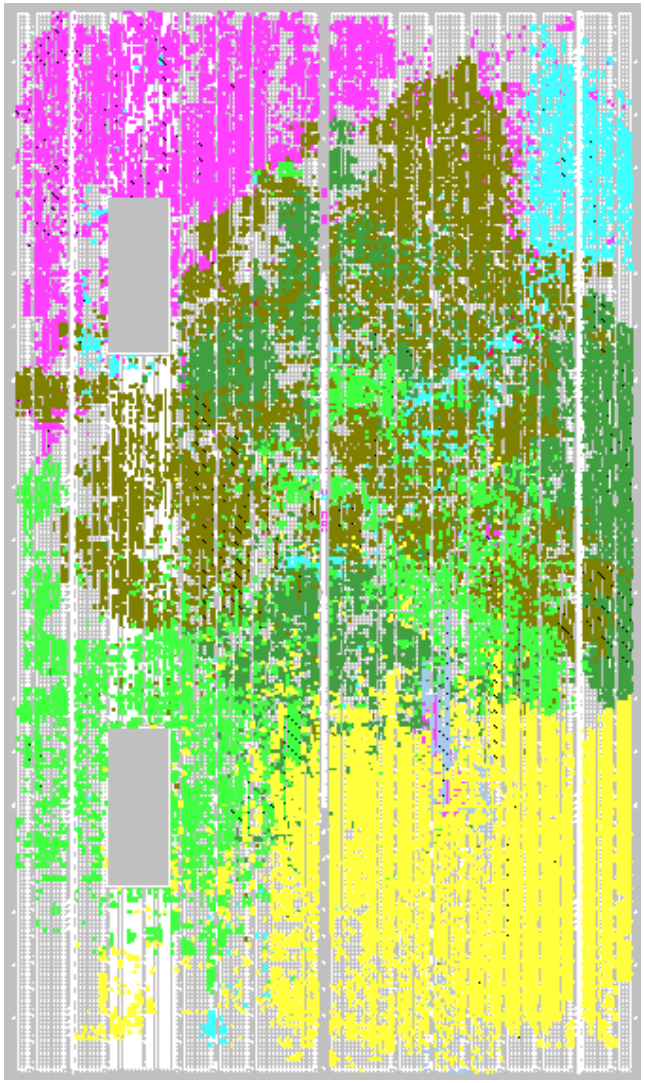
  > Optimizations for Area

New Features of Release 1.6

- Support for Virtex-5 ML505 board
  - Upgraded to XC5VLX110T
- Implementation of 4-thread core on FPGA
- Complete OpenSolaris Image
- Quick-start files, enable you to boot OpenSolaris on day one.

OpenSPARC T1 on FPGAs

- Single thread version
  - ~40K Virtex-2/4 LUTs, 30K Virtex-5 LUTs
  - Optimized for area
    - No modular arithmetic (MA), reduced TLBs
    - Easily meets 20ns cycle time (50MHz)
    - Fits into a Xilinx XC4VFX60
  - Full TLB and MA included: 50K Virtex-4 LUTs



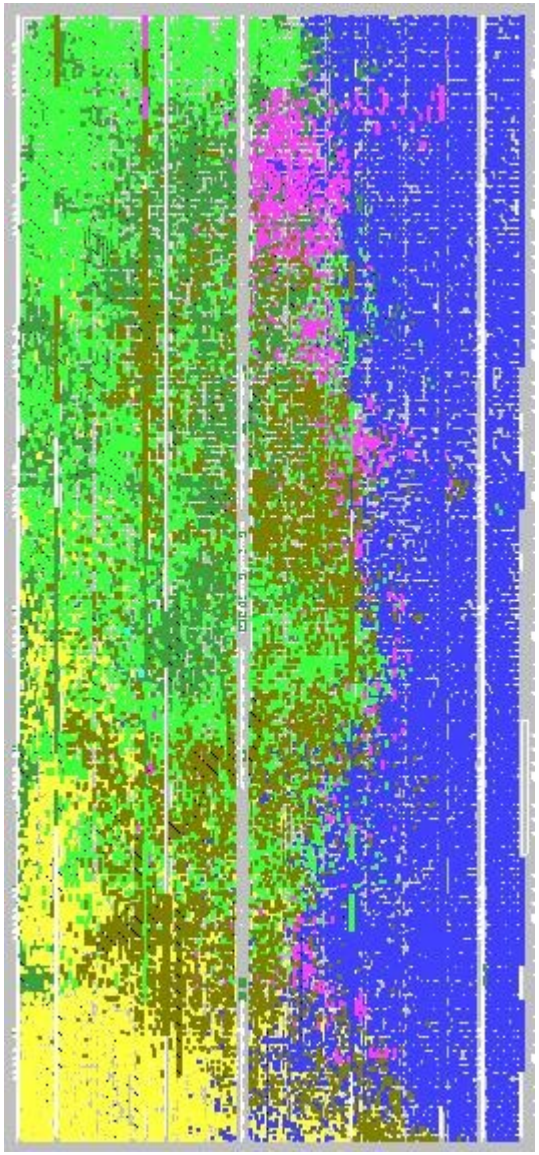
Plot of 1-thread design on XC4VFX60

Single Thread T1 on FPGAs

- Functionally stable
  - Passing mini and full regressions
- Completely routed
  - No timing violations
  - Easily meets 20ns (50MHz) cycle time
- Expandable to more threads
  - Reasonable overhead for most blocks (~30% for 4 threads)
  - Some bottlenecks exist (Multi-port register files)

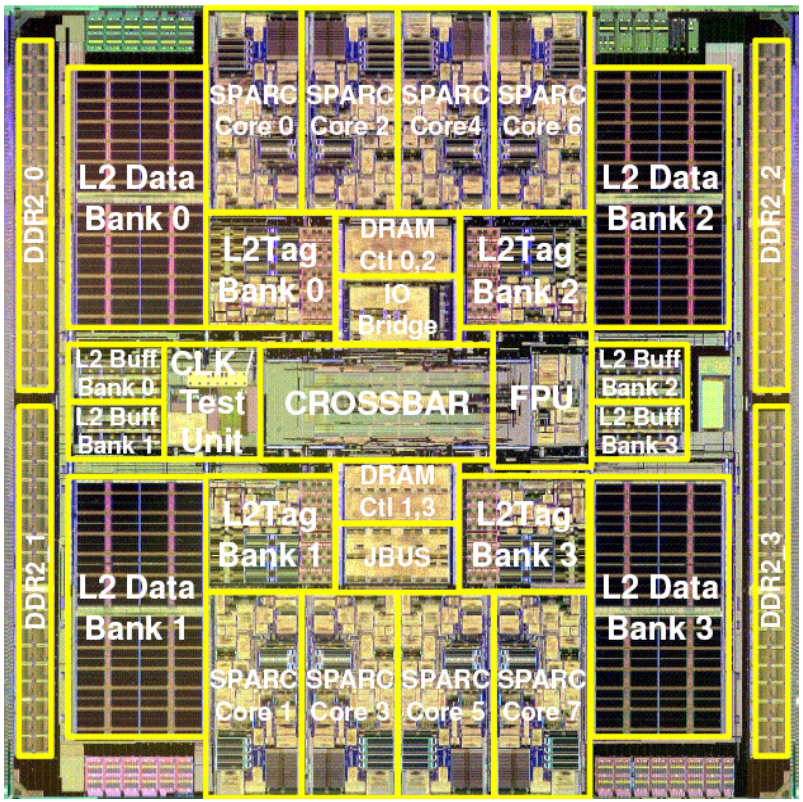
OpenSPARC T1 on FPGAs

- Four thread version
  - Functionality identical to Niagara1 core – on FPGAs
  - No Modular Arithmetic unit
  - 16-entry TLB
  - 69K Virtex-2/4 LUTs, 51K Virtex-5 LUTs
  - 40%+ reduction in area compared to original design
  - Runs at 10 MHz
  - Block RAMs used: v4: 127, v5: 115

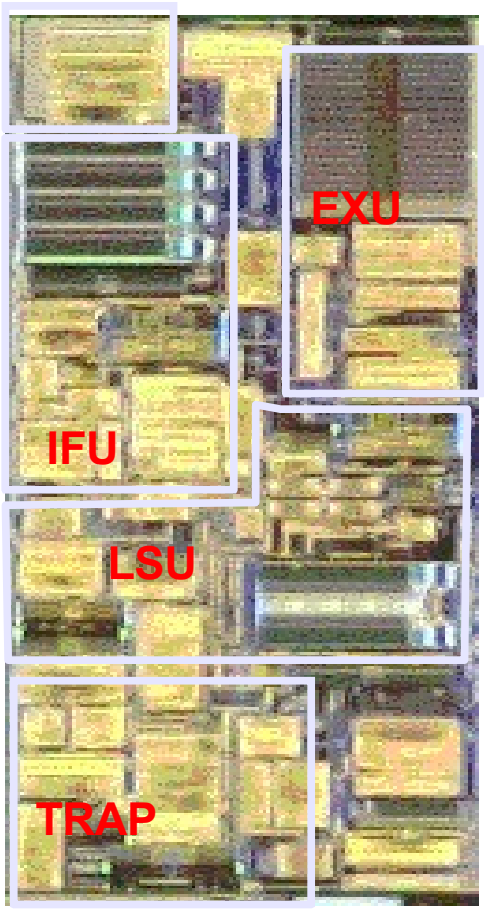


Plot of 4-thread design on XC5VLX110T

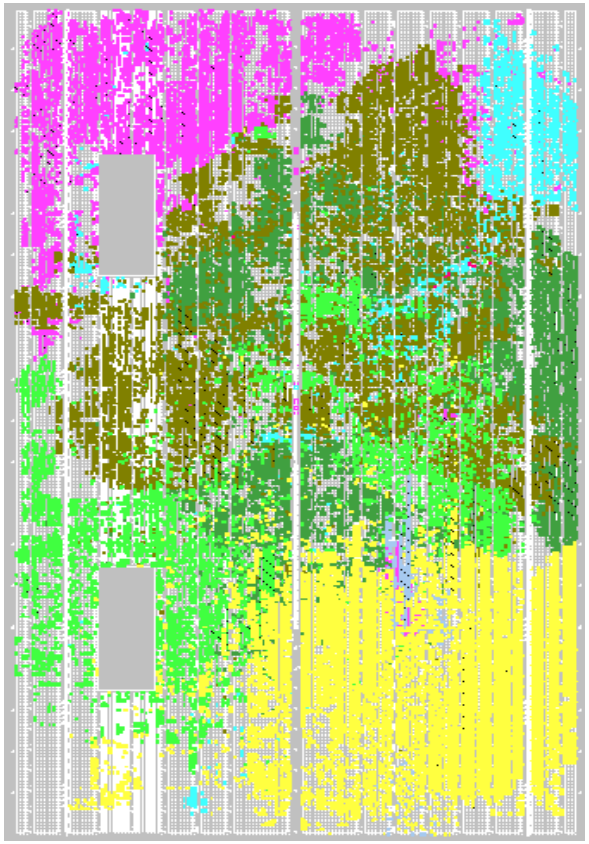
OpenSPARC T1 FPGA variants



T1 – 8 cores, 32 threads



T1 – 1 core, 4 threads

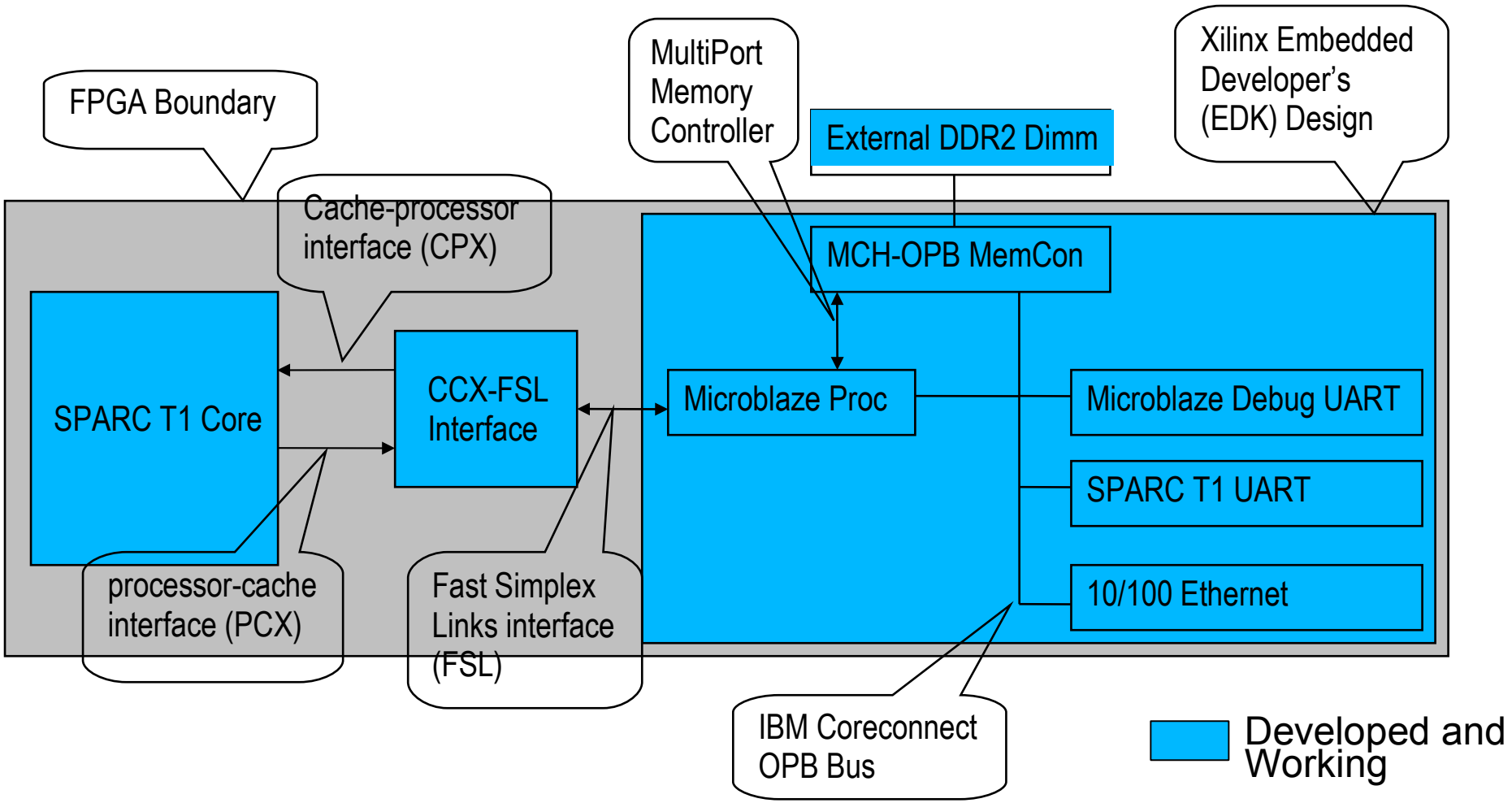


1 core, 1 thread

System-on-FPGA

- Goal: Create a working system on an FPGA Board
  - Requires: core, memory interface, peripherals
  - Core requires L2 cache for coherence, and connectivity to memory controller
    - This won't fit on the FPGA
- Needed a small replacement for L2
  - And we had an aggressive schedule
- Solution:
  - Use a Xilinx MicroBlaze Core to process memory transactions

System Block Diagram

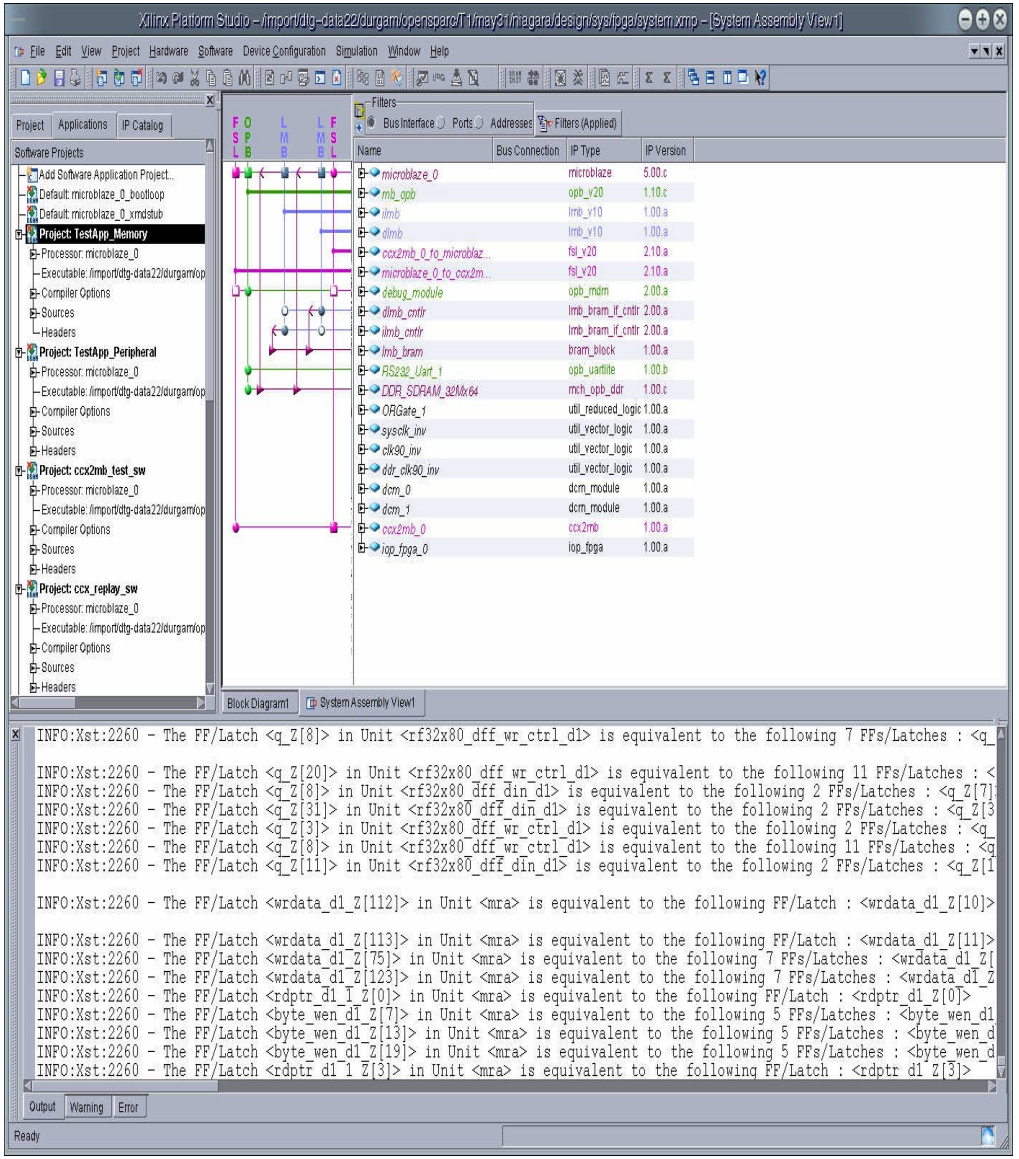


System Operation

- OpenSPARC T1 core communicates exclusively via cache-crossbar interface (CCX)
  - PCX (processor-to-cache), CPX (cache-to-processor)
  - Glue logic block forwards packets between OpenSPARC core and Microblaze
- Microblaze firmware polls T1 core and system peripherals
  - Services memory and I/O requests
  - Performs address mapping
  - Returns results to the core
  - Maintains L1 cache coherence

T1 EDK Project

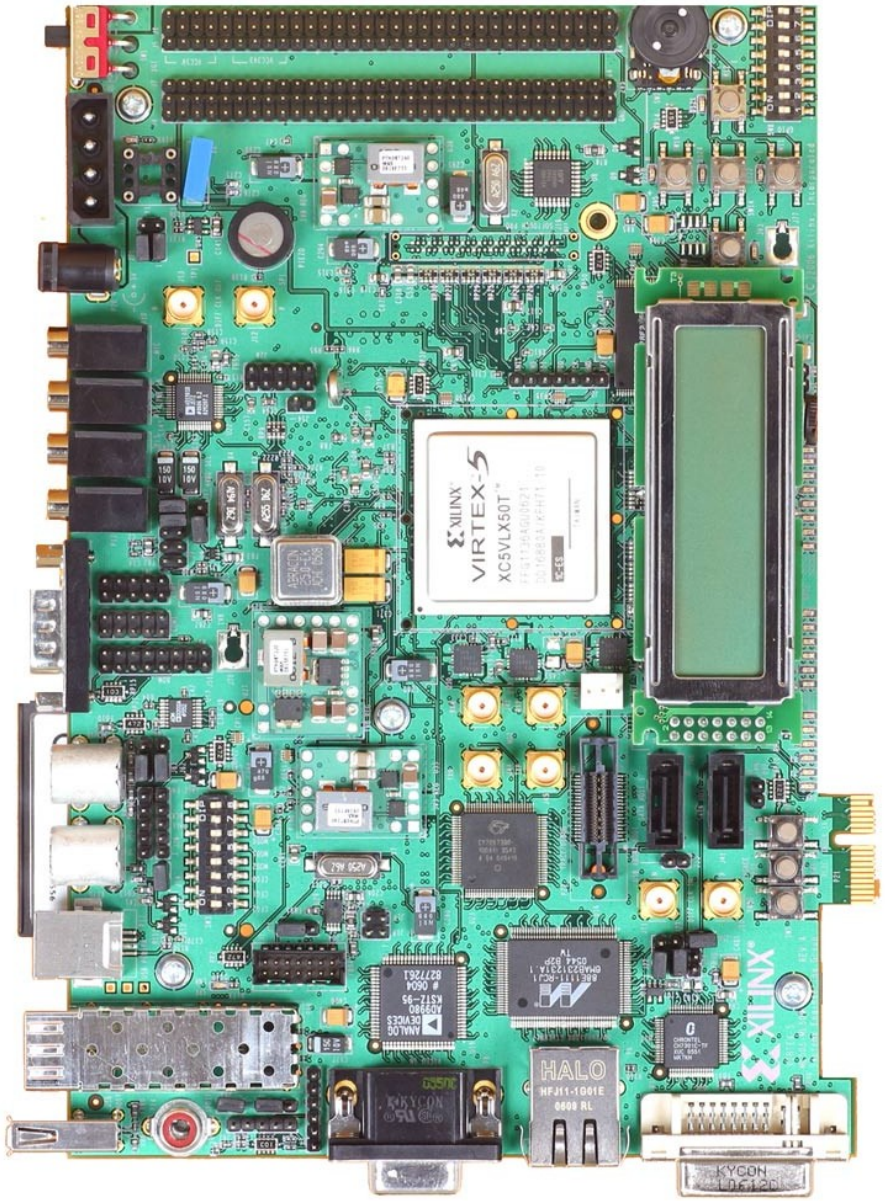
- System captured in Xilinx EDK project
  - > T1 core and Microblaze glue logic defined as Xilinx peripheral cores (“pcores”)
  - > T1 netlist generated via Synplicity or Xilinx XST
  - > Implemented on a Xilinx XC5VLX110T



T1 EDK Project (Cont'd)

- Entire system placed & routed
- Downloaded to FPGA on ML505 board
- Use Debugger to load software into memory
- Run!

- View program output via serial cable connected to a PC



ML505 Board (not upgraded)

Included in EDK Project

- SystemAce file for quick start-up
- EDK system setup files
- Synplicity-generated netlist:
  - 4 threads, 16 TLB entries, no SPU.
- Firmware to process cache crossbar packets
  - Setup to run stand-alone tests on the board
  - Setup to boot Hypervisor
- Full-system simulation setup using Modelsim

FPGA Board

Latest OpenSPARC Development Kit

- • A kit for OpenSPARC development now available
- • Manufactured by Digilent Inc.
- > Commercial interests can directly order from http://www.digilentinc.com
- > Board based on the ML505, but with an XC5VLX110T FPGA
- • Kit Includes:
- > Board, with power supply and 256 MB DRAM
- > Platform USB download cable
- > Host to host SATA crossover cable
- > Compact flash card with OpenSPARC T1 1.6 ace files

OpenSPARC Development Kit Contents

Xilinx ML505-V5LX110T

Platform USB programming cable

5V switching power supply

Quick Start-Up

- Files:
  - design/sys/edk/ace/
    - OpenSPARCT1_1_6_os_boot.ace
      - OpenSolaris Boot on a 4-thread core (on ML505-110T)
    - OpenSPARCT1_1_6_Hello_World.ace
      - Run a standalone program under hypervisor
- Procedure:
  - Format a compact flash card with Xilinx filesystem
  - Copy a file to the compact flash card
  - Insert CF card into board socket (set DIP switches)
  - Connect Serial port on board to a computer
    - Using a null-modem serial cable
    - Use Hyperterminal or some other terminal to connect

Quick Start-Up (Cont'd)

- Boot Process
  - Turn on the board
  - At OBP “OK” prompt, type “boot”
    - boot -m milestone=none (Fast single-user boot)
    - boot -mverbose (Enable networking)
  - At login prompt (30-60 minutes later) login as root
  - Interesting commands
    - psrinfo Will show 4 processors
    - uname

Running Stand-alone Tests

- We use the ELF executable and the memory image created by the simulation
- Memory Map table created
  - > Maps different program segments into 256 MB DRAM
  - > Compiled into firmware executable.
- Download and run the firmware
  - > Firmware will send wake-up to core
  - > Will process packets
  - > Will report success or failure *(GOOD_TRAP/BAD_TRAP)*

%

How to Run Stand-alone tests

- Run the simulation of the test using sims
- Generate the memory table for the test
  - genmemimage.pl -single -f *memory-image-file* -name *test_name*
- Copy the memory table to the EDK project
  - % cp mbfw_diag_memimage.c ccx-firmware-diag/src
- Re-build the firmware
- Download
- Run

Running Hardware Regressions

- Run the sims regression
- Generate the memory tables for each test
  - genmemimage.pl -d *regression-dir*
  - Creates a directory named `diags`
- Edit the diag list
  - design/sys/edk/scripts/
    - thread1_mini.list, thread1_full.list, core1_mini.list, or core1_full.list
- Run the regression script
  - % xtclsh *edk-project-dir*/scripts/rundiags.tcl -edk *edk-project-dir* -list *edk-project-dir*/scripts/diag_mini.list -d *diag_dir* -model core1 -suite {thread1_mini thread1_full core1_mini core1_full}

Memory Allocation (256 MB DDR2)

- 256 MB DDR2 DRAM is at MicroBlaze Address 0x50000000
- DRAM Utilization

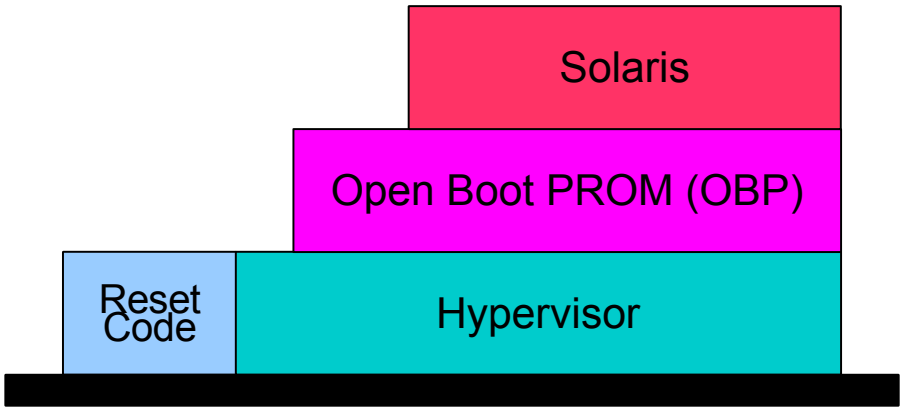
MicroBlaze Address	Function
0x5000_0000	MicroBlaze Firmware
0x5010_0000	OpenSPARC Memory Space: 174 MB 0x00_0000_0000 – 0x00_0fdf_ffff
0x5aef_ffff	Ram Disk Image (80 MB)
0x5ff0_0000	OpenSPARC Boot Prom: 0xff_f000_000

Booting Solaris on an FPGA Board

- MicroBlaze firmware is compiled and loaded into DRAM
  - A fixed memory translation table is used to map OpenSPARC addresses to MicroBlaze addresses
- Boot PROM image and RAM disk images loaded as data into DRAM
- The firmware program is started

Software Stack

- Use Standard software installation
- Use a virtual disk in RAM to hold the Solaris binaries
- Some memory copy sections performed by MicroBlaze
- MicroBlaze firmware now performs floating-point operations, so emulation is not needed



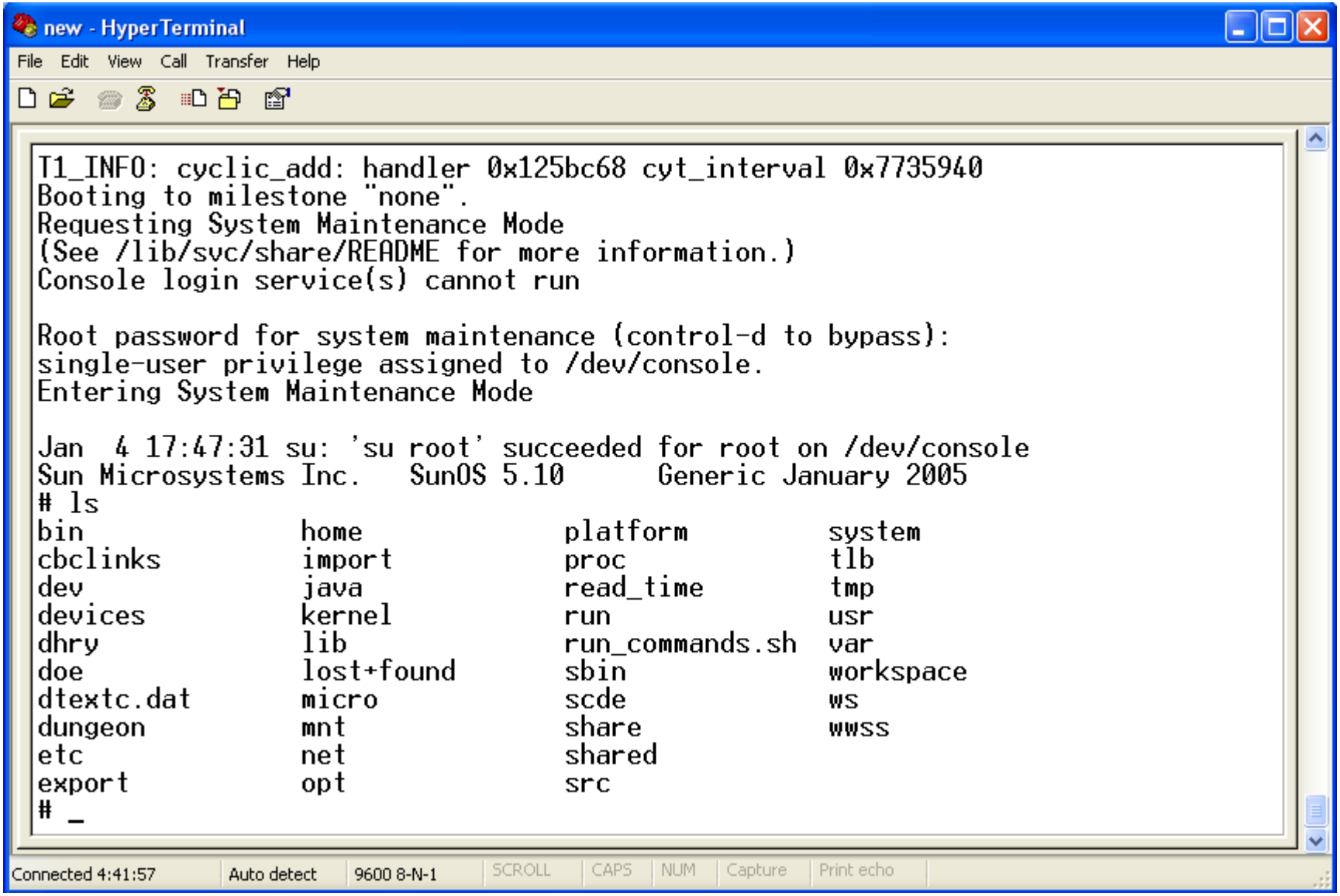
Boot Sequence

- The processor starts at the Power-On Reset (POR) trap handler
- Reset code is executed: Caches & TLBs enabled
- Control passed to Hypervisor
  - > Hypervisor copies itself from PROM to RAM area
  - > Passes control to Open Boot PROM (OBP)
- OBP then loads the operating system

Steps to Boot the Operating System

- Download the bit file to the FPGA
- Start the debugger
  - Download the MicroBlaze firmware
    - % dow mb-firmware-hv/executable.elf
  - Download the PROM image
    - % dow -data prom.bin 0x5ff00000
  - Download the RAM disk image
    - % dow -data ramdisk_image.bin 0x5af00000
  - Start the firmware
    - % run
  - At OBP prompt, type the boot command
    - Ok boot -m milestone=none

Solaris Boot



Thanks for watching the OpenSPARC Slide-Cast!

Let us hear from you! The OpenSPARC Team would appreciate your feedback on this in the http://www.OpenSPARC.net forum.