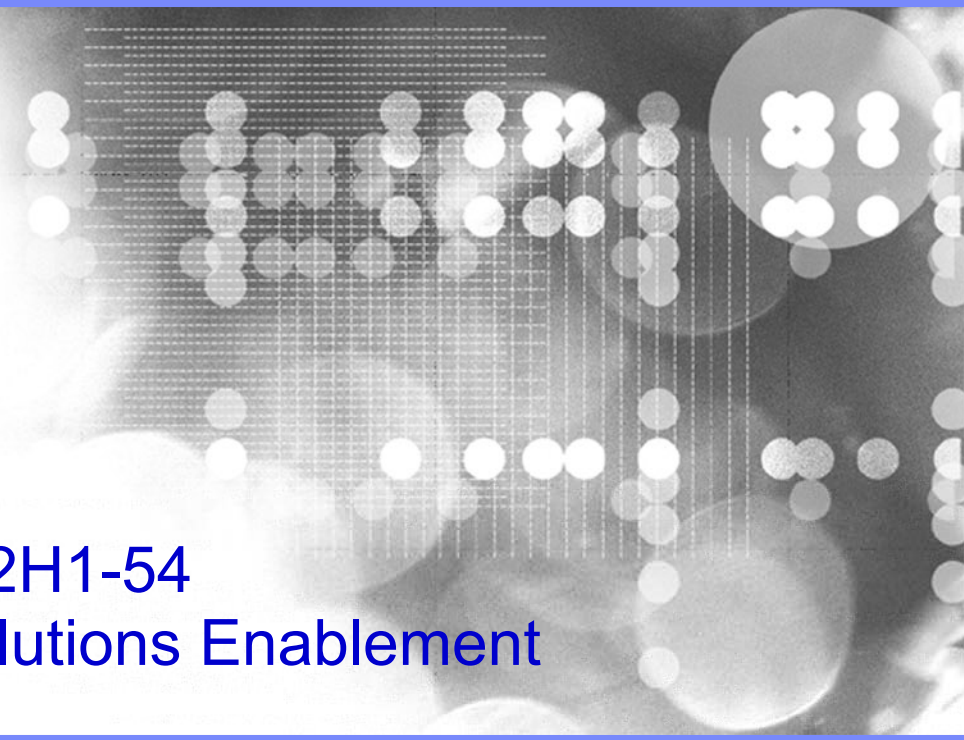


Using the SDK

Course Code: L3T2H1-54
Cell Ecosystem Solutions Enablement

Course Objectives

- **Get to know the structure of SystemSim**
- **Learn how to use gdb to debug cell threads**
- **Get to know the structure of the SDK**
- **Learn some tips and techniques to better use the SDK**

Course Agenda

- **System Sim**
- **GDB**
- **SDK contents**
  - Samples
  - Tests
  - Tools
  - Workloads
  - Application-oriented code samples
- **SDK tips and techniques**

SystemSim

System Sim - Overview

- **Two simulation modes for simulation**
  - Functional simulation
  - Cycle accurate simulation (performance & timings)
- **Including PPE, SPEs, MFCs, PPE caches, bus, memory controller**
- **It can simulate and capture operational details on (among others)**
  - instruction execution
  - Cache
  - Memory subsystems
  - interrupt subsystems
  - Communications
- **Two different modes for OS support**
  - Linux mode: operating system is booted and simulated
  - Standalone model: no operating system support
- **Internal name “Mambo”**

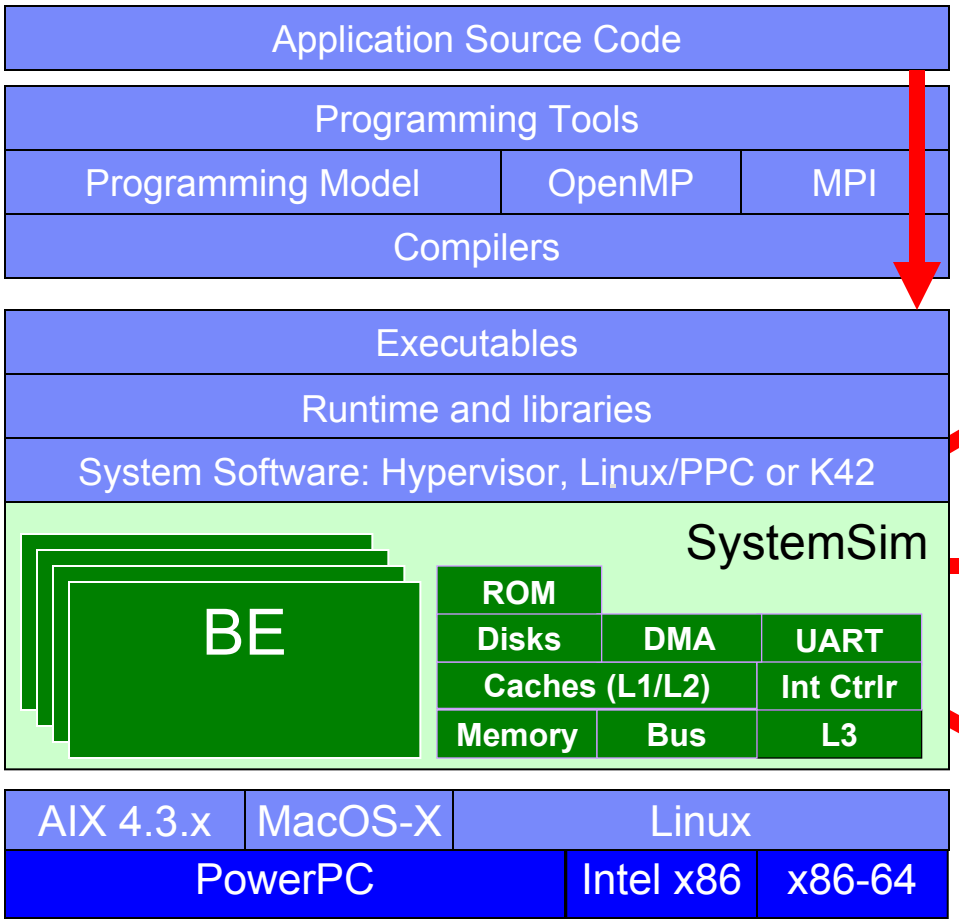
System Sim – Simulator Stack

Development Environment:

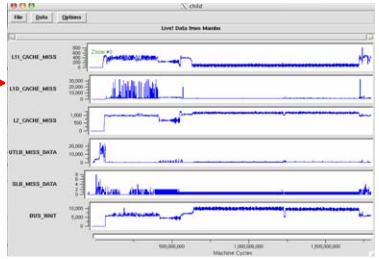
Software Stack: Running on SystemSim

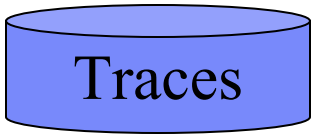
SystemSim: *Simulation of hardware*

Real Systems:

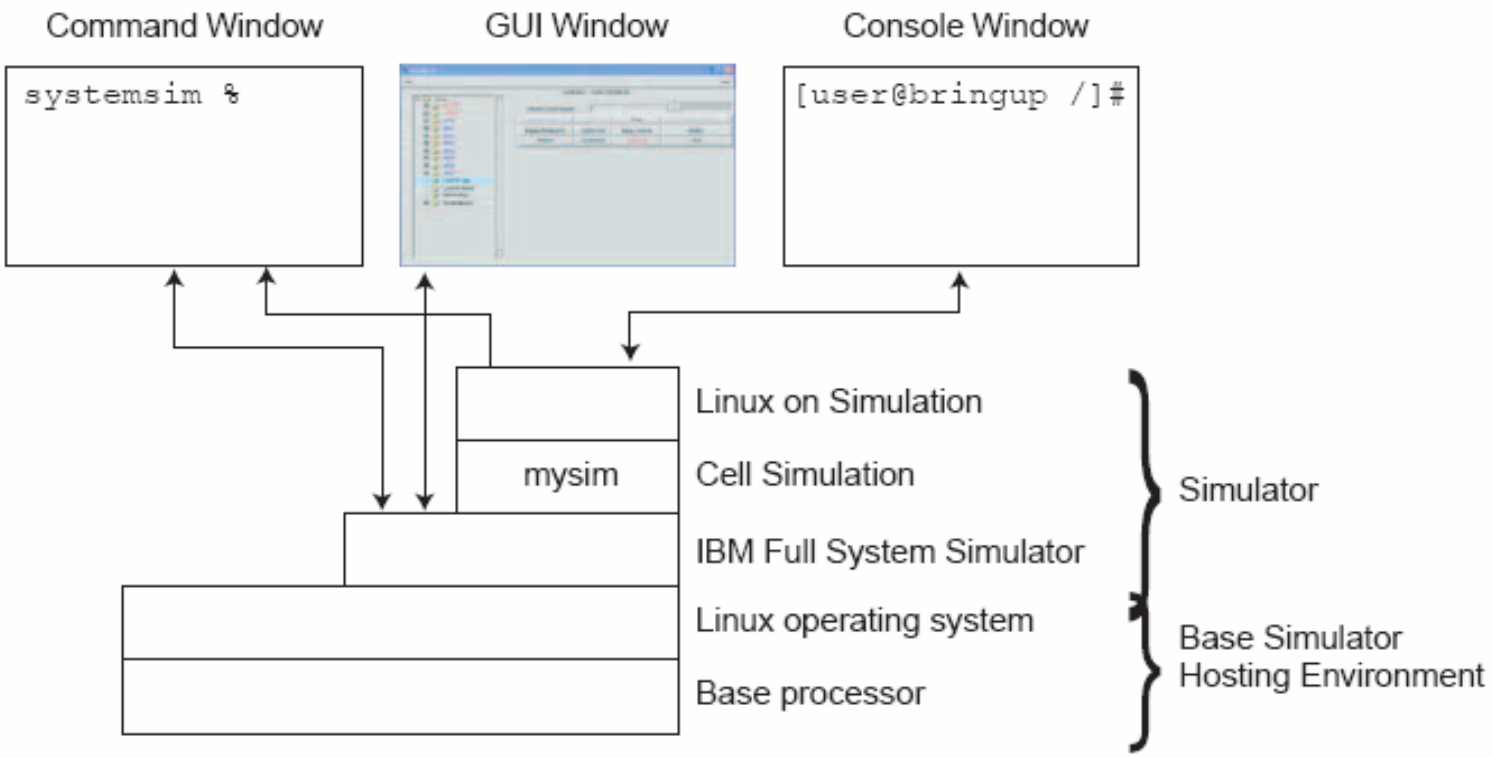


Standard Terminal





System Sim - Interfaces



Interacting with the Simulator

- **Issuing commands to the *simulated system***
  - in the *console window which is* a Linux shell of the simulated Linux operating system.
  - The simulated system is the Linux environment on top of the simulated cell, where you run and debug programs.
- **Issuing commands to the *simulator***
  - in the *simulator command window*, or using the equivalent actions in the graphical user interface (GUI).
  - To control the simulator itself, configuring it to do such tasks as collect and display performance statistics on particular SPEs, or set breakpoints in code.

Simulator console commands

Simulator Command	Meaning
Quit	Closes the simulation and exits the simulator.
help	Displays a list of the available simulator commands.
mysim go	Starts or continues the simulation. The first time it is issued, the simulator boots the Linux operating system on the simulation.
mysim spu *n* set model *mode*	Sets SPEn into model mode, where n is a value from 0 to 7 and mode is either pipeline or instruction.
mysim spu *n* display statistics	Displays to the simulator command window, the performance analysis statistics collected on SPEn, where n is a value from 0 to 7. Statistics are only collected when the SPE is executing in pipeline mode.

GDB

Debugging SPU or PPU Threads

- **PPU thread**
  - gdb <ppu_exe>
- **SPU thread**
  - spu-gdb <spu_exe>

Debuging BE Threads

- SPU_INFO=1
  - Implemented within libspe runtime library
  - When loading SPE ELF executable, prints message
    ```
    Loading SPE program : NNN
    SPU LS Entry Addr   : NNN
    ```
  - Before starting up new SPE thread, prints message
    ```
    Starting SPE thread 0x..., to attach debugger use:
    spu-gdb -p NNN
    ```
- *SPU_DEBUG_START=1*
  - *Includes everything done by SPU_INFO=1*
  - *Waits until debugger is attached (or signal received)*

Example: Bus Error due to DMA

```
sid@localhost:~
lcs2114:/tmp/be_example/ppu # ./main
Bus error
lcs2114:/tmp/be_example/ppu # SPU_DEBUG_START=1 ./main &
[4] 608
lcs2114:/tmp/be_example/ppu # Starting SPE thread 0x1817050, to attach debugger
use: spu-gdb -p 609

lcs2114:/tmp/be_example/ppu # spu-gdb spu_dmado -p 609
GNU gdb 6.1.1
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=powerpc64-unknown-linux-gnu --target=spu"...
Attaching to program: /tmp/be_example/ppu/spu_dmado, process 609
0x0003fe00 in ?? ()
(gdb) c
Continuing.

Program received signal SIGBUS, Bus error.
0x00000240 in get_work (buffer=0xa80 "\001\201` ", ef_addr=0xffd3c0c0)
    at spu_dmado.c:20
20        mfc_read_tag_status_all ();   /* Wait for DMA to complete */
(gdb) list get_work
12       * description:
13       *      - get buffer from main memory to work on.
14       */
15      void
16      get_work (unsigned char *buffer, unsigned int ef_addr)
17      {
18        mfc_get (buffer, ef_addr, DMA_SIZE, 31, 0, 0);
19        mfc_write_tag_mask ((1 << 31));
20        mfc_read_tag_status_all ();   /* Wait for DMA to complete */
21      }
(gdb) p ef_addr
$1 = 0xffd3c0c0
(gdb) p buffer
$2 = (unsigned char *) 0xa80 "\001\201` "
(gdb)
```

```
sid@localhost:~
      if (!buffer[i])
        {
          perror ("memory low");
          return -1;
        }
      memset (buffer[i], 0, DMA_SIZE);


      speid[i] =
        //spe_create_thread (spegid[i], &spu_dmado, buffer, NULL, 0, 0);
        spe_create_thread (spegid[i], &spu_dmado, buffer[i], NULL, 0, 0);
      if (speid[i] < (spe_gid_t) 0)
        {
          printf ("Failed spe_create_thread(rc=0x%x, errno=%d)\n",
                  (int) speid[i], errno);
          perror ("failed creating thread\n");
          exit (1);
        }
    }

/*
 * Tell the spes to start working on their buffers.
 * ------------------------------------------------------------------------
 */
"main.c" 201L, 4522C                                  68,1-8        31%
```

error

The Software Development Kit

SDK Contents

- **The SDK source code is organized into the following categories:**
- **samples ($TOP/src/samples)**
  - simple and concise code examples to demonstrate specific functions, use of tools, libraries, and/or HW features
- **tests ($TOP/src/tests)**
  - self-verifying tests use to assure standards compliance, validate libraries and tools
- **tools ($TOP/src/tools)**
  - utilities used to generate content or ease programming burden
- **workloads ($TOP/src/workloads)**
  - code samples used to characterize the performance of the architecture
- **lib ($TOP/src/lib)**
  - libraries and reusable header files

Samples

- **cesof (CBE™ Embedded SPU Object Format)**
  - sample code to demonstrate the object format used to embed SPU objects into PowerPC binaries
- **DMA**
  - sample code to demonstrate non-trivial DMA calls
- **resample**
  - audio resampling code for SP/DP monotonic/stereo audio samples
- **simpleDMA**
- **spu_clean**
  - sample SPU program that clears the register file and local store (including itself)
- **spu_entry**
  - sample crt0 – initializes the stack and stack pointer; calls main; returns main's return value to a controlling PU program in an ABI compliant fashion (exit function).
- **spu_interrupt**
  - sample first level interrupt handler and second level interrupt handler registration function. Demonstration second level decrementer interrupt handler.
- **spulet**
  - C-library functions made to run on SPU (printf(), read(), etc.)
- **sync**
  - conditional wait, mutex, and atomic operation sample code
- **tutorial**
  - contains some of the source code used within the tutorial document

Tests

- **abi**
  - set of tests used to validate conformance to the SPU and BE ABI standards
- **asm**
  - set of tests used to verify assembler support of all instructions, parameter forms, and parameter ranges
- **events**
  - set of tests used to validate and demonstrate the handling of user-defined SPU events
- **intrinsics**
  - set of tests used to validate all VMX and SPU intrinsics
- **lib**
  - suite of self-validating tests used to verify correct operation of the libraries

Tools

- **callthru**
  - callthru source code
- **idl**
  - IDL compiler tool reads a high-level specification describing an interface to a SPU function
  - produces special stub functions to implement the interface in C
  - stubs allow the PU and SPU to communicate through what appear to be ordinary, local procedure calls or method invocations
- **oprofile (in progress)**
  - system-wide profiler for Linux
  - kernel dirver and daemon for collecting data
  - several post-profiling tools

Workloads

- **FFT16M**
  - hand-tuned program performing 4-way SIMD SP complex FFT of 16M elements
- **matrix_mul**
  - workload calculates C = A * B where A, B, and C are N x N squared matrices comprised of SP floats.
  - uses block-partitioning algorithm to reduce bandwidth (block size fixed to 64)
- **oscillator**
  - workload used to synthesize two stereo sound files
- **vse_subdiv**
  - workload demonstrating subdivision using contours of variable sharpness
  - displays result in OpenGL output window

Application-oriented Code Samples

- **C**
  - SPE-only library containing functions typically found in standard C99 library
  - includes functions executed by the SPE natively, functions initiated by the SPE but executed by the PPC, and SPE local store functions
  - provides or enhanced common high-level programming functionality
- **audio resample**
  - provides sample audio resampling functions that include
    - monophonic and stereophonic audio
    - unsigned short or FP samples
    - SP and DP computation
- **curves and surfaces**
  - support routines for evaluating quadratic and cubic Bezier curves as well as biquadric/bicubic Bezier surfaces and curved point-normal triangles
- **FFT**
  - highly tuned 1-D FFT as well as base kernel functions that can be used to implement 2-D FFTs

Application-oriented Code Samples (cont.)

- **game math**
  - set of routines implemented with the notion that precision and mathematical accuracy can at times be sacrificed for performance
- **image**
  - includes routines for various size convolutions as well as generation of histograms of byte data
- **large matrix**
  - various utility functions that operate on large vectors/matrices of SP FP numbers
  - size of input vectors and matrices limited by SPE local storage size (no matrix partitioning)
- **math**
  - general purpose math routines tuned to exploit SIMD features
  - most only support SP
  - intended to mimic standard math library functions

Application-oriented Code Samples (cont.)

- **matrix**
  - utility library to operate on matrices and quaternions including inversion, identity, perspective projection, and multiplication
- **misc**
  - routines that do not logically fit into other categories (min, max, rand, clamp, etc.)
- **multi-precision math**
  - performs mathematical functions on unsigned integers of a large number of bits
- **noise**
  - 1-D, 2-D, 3-D, and 4-D noise
  - lattice and non-lattice noise
  - turbulance

Application-oriented Code Samples (cont.)

- **oscillator**
  - two oscillator libraries to create a synthetic environment of configurable directional microphones, a large number of oscillators moving along defined paths, all relative to static microphones
  - computes time delays, volume changes, doppler effects
- **sim**
  - services useful to the full system simulator such as callthru
- **sync**
  - libraries making use of the load-with-reservation and store-conditional functions within CBEA
  - atomic operations, mutexes, conditional variables, and completion variables
  - sample code included in samples dir
- **vector**
  - 15 general purpose routines to operate on vectors

SDK Tips and Techniques

- **$TOP/make.env contains environment variables to change compiler and compiler settings**
  - SPU_COMPILER
    - tells make to use gcc or xlc for SPU code
  - SPU_TIMING
    - if timing tools RPM is installed will generate a static timing analysis of SPU code to determine pipe stalls and register dependencies
  - SCE_VERSION
    - used to change the toolchain version employed
- **Several internal Systemsim parameters can be accessed and changed via the TCL/TK command line prompt**
  - use the HELP subsystem to access general information about these commands
- **Debugging tools are limited to GDB although there are both PPC and SPU versions**
  - PPC and SPU code must be debugged separately

IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY 12533-6351

The IBM home page is http://www.ibm.com
The IBM Microelectronics Division home page is
http://www.chips.ibm.com