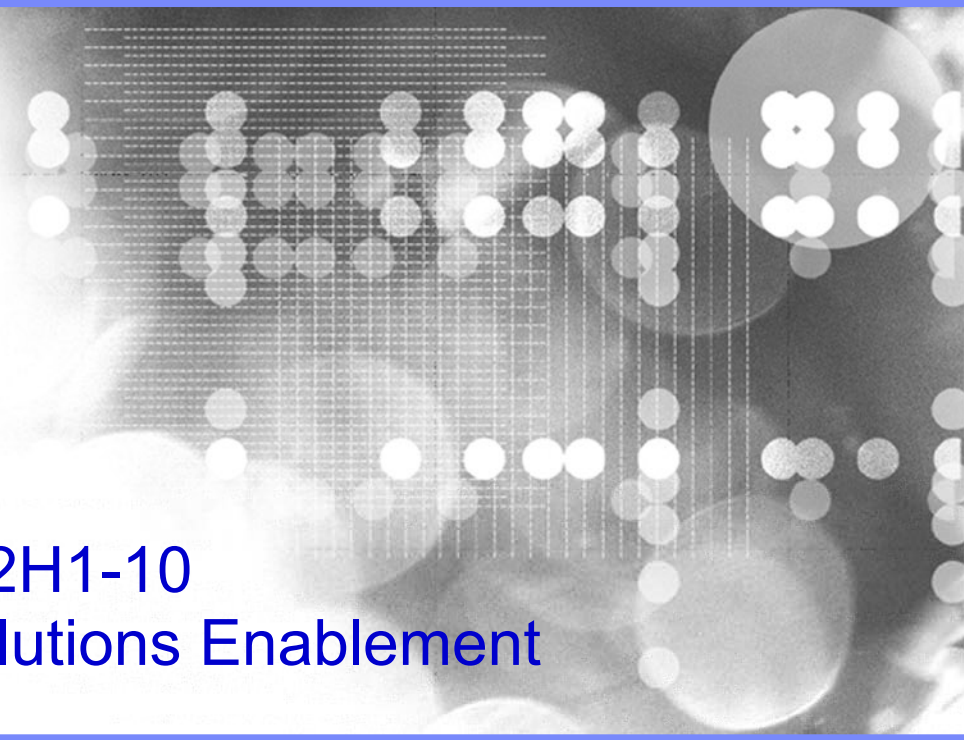
Hello World!



Course Code: L2T2H1-10
Cell Ecosystem Solutions Enablement

Course Objectives

- **You will learn how to write, build and run “Hello World!” on the Cell System Simulator.**
- **There are three different versions of “Hello World!” for the PPE only, SPE only and for the Cell BE, i.e. using PPE and SPE.**

How to get “Hello World!”

- **Pre-requisites**
  - Toolchain
  - Compiler
- **Build Process**
- **Source Code**
  - Makefiles
  - Source PPE
  - Source SPE
- **Simulator**
  - Getting the binary into the simulator
  - Running the binary

SDK Installation Requirements

- **Hardware "official" requirements**
  - At least 2GHz x86 or x86-64 processor
  - At least 1GB RAM
  - At least 5GB available space
- **Software "official" requirements**
  - Fedora Core 4
    - With TCL/TK
  - SDK Installation Files
  - Network connectivity to download 2.6.14 kernel (SDK 1.0) or 2.6.15 (SDK 1.0.1)

SDK Installation Files

- **Barcelona Supercomputing Center website**
  - GNU x86 toolchain — toolchain-2.3-i686.tar.bz2
  - FC4/PowerPC RPMs — ppc-fc4-rpms-1.0.0-1.i386.rpm
  - Cell Linux kernel patches — cell-linux-patches-1.0.tar.bz2
  - SPE runtime lib source — libspe-1.0.tar.bz2
  - Installation script — install.sh
- **IBM alpha works (binary / ILA for early release program)**
  - System simulator — systemsim-cell-1.0-fc4-x86.tar.bz2
  - XLC — xlc-cell-cmp-1.0-1.i386.rpm
xlc-cell-lib-1.0-1.i386.rpm
  - Sample and Library (source / CPL v1.0) — cell-sdk-lib-samples-1.0.tar.bz2
  - SPU instruction timing tool — cell-spu-timing-1.0-fc4-x86.tar.bz2

Your Virtual Machine

- **Contains an installed Fedora Core 4**
  - including the complete cell sdk
- **You can log in using**
  - User: student
  - Password: go4cellNow
- **Settings for Cell**
  - Alias cdsim → changes directory to the simulator start dir
  - Environment variable $TOP → CBE home

Compilers

- **GCC**
  - GNU public compiler
  - x86 toolchain includes PowerPC cross-compiler and SPU-capable cross-compiler
    - /opt/sce/toolchain-2.3/ppu/bin/ppu-gcc
    - /opt/sce/toolchain-2.3/spu/bin/spu-gcc
  - Advantages
    - widely available, open source compiler
    - optimizations for POWER platform are improving
  - Disadvantages
    - auto vectorization capabilities are limited
- **XLC**
  - IBM internal compiler for POWER platform modified to generate SPU object code as well
  - Advantages
    - commercial-level compiler dedicated to generating highly-optimized POWER code
    - auto vectorization capabilities originally designed for VMX instruction set have been implemented for SPU
  - Disadvantages
    - optimizations are slower to be implemented and released
- **Octopiler**
  - A version of XLC that is being developed by IBM Research
  - Intended to perform auto vectorization, auto partitioning, and overlay management to standard sequential code
  - http://www.research.ibm.com/journal/sj/451/eichenberger.html

Build the code

- **TOP set to directory containing make header & footer**
  - make.footer contains all the complicated build rules
- **Place SPU code in a subdirectory of directory containing PPC code**
  - e.g. subdirectory name is ‘spu’
- **Makefile for PPC code:**
  - DIRS = spu
  - PROGRAM_ppu = <PPU_executable_name>
  - IMPORTS = <spu_executable-embed.a> -lspe
  - include $(TOP)/make.footer
- **Makefile for SPU code:**
  - PROGRAM_spu := <SPU_executable_name>
  - LIBRARY_embed = >SPU_executable-embed.a>
  - include $(TOP)/make.footer

Three Different Versions of “Hello World!”

- **PPU only**
- **SPU only**
- **Synergistic**

“Hello World!” – PPU Only

- **PPC program**
  - just like any “Hello World!” program one would write

```
#include <stdio.h>

int main(void)
{
        printf("Hello world!\n");
        return 0;
}
```

- **Makefile**
  - make.footer included to set up compiler and compiler flags
  - PROGRAM_ppu tells make to use PPC cross-compiler

```
PROGRAM_ppu = hello
include $(SDK_TOP)/make.footer
```

“Hello World!” – SPE Only

- **SPE Program**

```
#include <stdio.h>

int main(unsigned long long speid, unsigned long long argp,
unsigned long long envp)
{
        printf("Hello world!\n");
        return 0;
}
```

- **SPE Makefile**

```
PROGRAMS_spu   := hello_spu
IMPORTS        = $(SDKLIB_spu)/libc.a
include $(SDK_TOP)/make.footer
```

“Hello World!” – SPE Only (2)

- **Can only be started directly in the Simulator**
- **Printf()**
  - there is no direct access to linux console by SPE
  - printf() several implementations in different libraries
    - Doing nothing
    - Doing a system call to PPE
  - simulator implements printf() to aid in debugging on simulator console

“Hello World!” – PPU and SPU

- **SPE program**
  - Same as for SPE only
- **SPE Makefile**

```
PROGRAMS_spu    := hello_spu
LIBRARY_embed   := hello_spu.a
IMPORTS         = $(SDKLIB_spu)/libc.a
include $(SDK_TOP)/make.footer
```

“Hello World!” – PPU and SPU (2)

- **PPU program**

```
#include <stdio.h>
#include <libspe.h>
extern spe_program_handle_t hello_spu;
int main(void)
{
        int speid, status;
        speid = spe_create_thread (0, &hello_spu, NULL, NULL, -1, 0);
        spe_wait(speid, &status, 1);
        return 0;
}
```

- **PPU Makefile**

```
PROGRAM_ppu = hello_ppu
IMPORTS = ../spu/hello_spu.a -lspe
include $(SDK_TOP)/make.footer
```

PPE and SPE Synergistic Programming

PPE Code

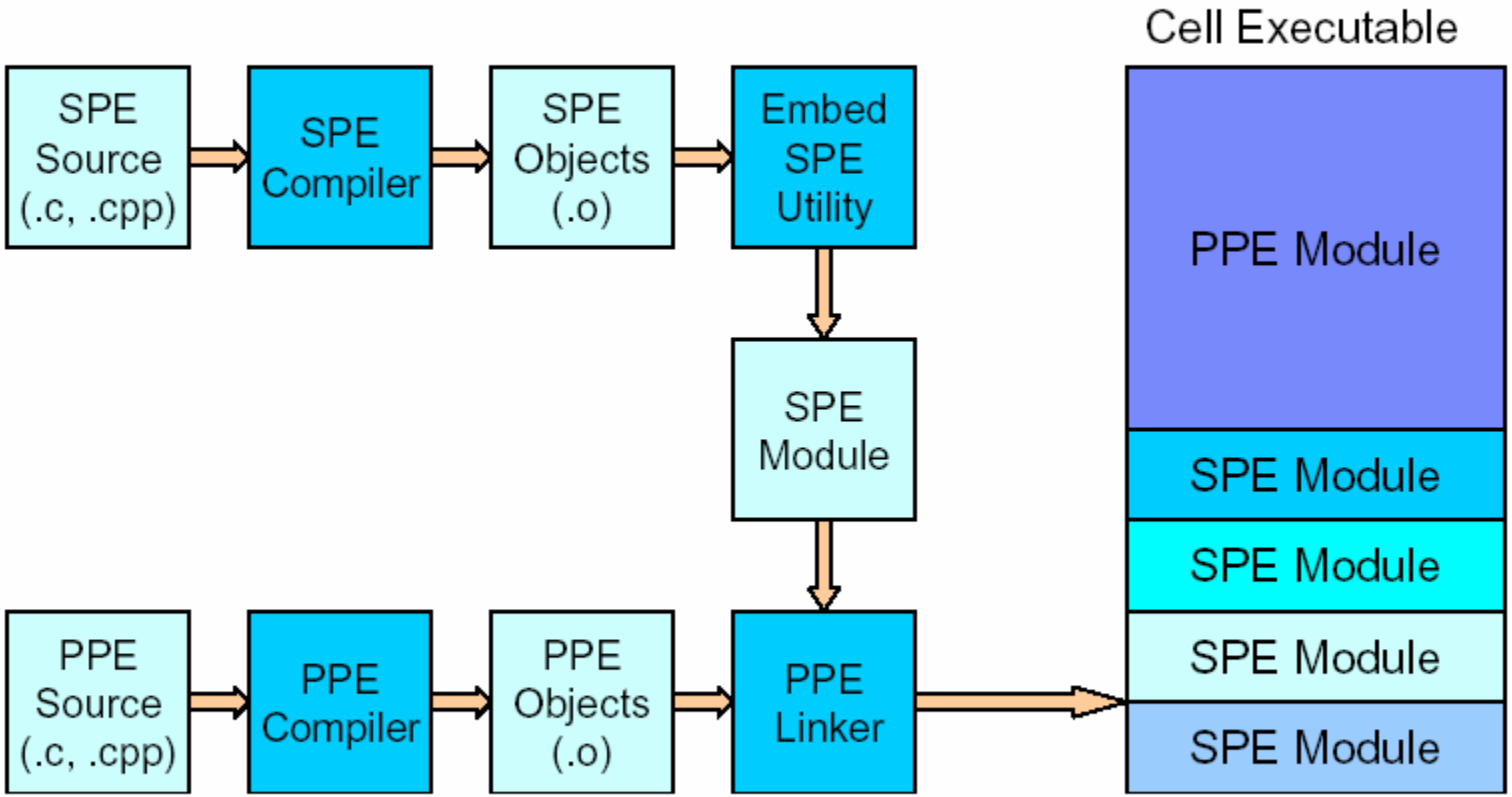
```
#include <stdio.h>
#include <libspe.h>
extern spe_program_handle_t hello_spu;
int main(void)
{
        int speid, status;
        speid = spe_create_thread (0, &hello_spu, NULL, NULL, -1, 0);
        spe_wait(speid, &status, 1);
        return 0;
}
```

SPE Code

```
#include <stdio.h>
#include <cbe_mfc.h>
#include <spu_mfcio.h>

int main(unsigned long long speid, unsigned long long argp,
unsigned long long envp)
{
        printf("Hello world!\n");
        return 0;
}
```

Build Process



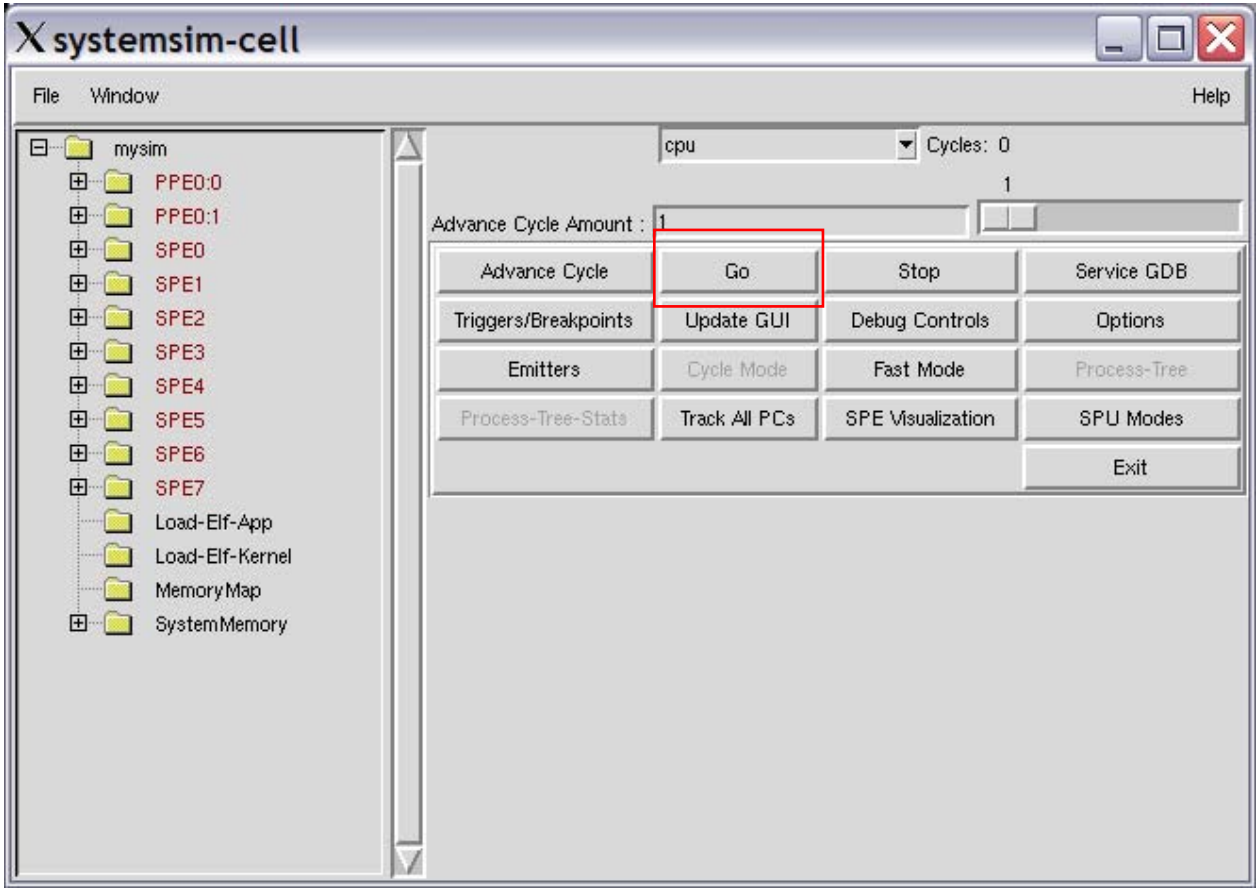
✓Make scripts are available to automate the build process

Two Ways to Exchange Files between Host and Simulator

- **RAMDISK**
  - the Systemsim simulator runs its environment off a ramdisk that is built using “make” in the $SDK_TOP/ramdisk directory
  - files can be inserted into this ramdisk such that when the simulator is started the files will be there already
    - useful for data input files or binaries that are known to work correctly
- **Callthru**
  - “backdoor” communication mechanism for the simulated environment to communicate with the host environment
  - useful for bringing in files to the simulated environment without shutting down and restarting the simulator
  - Example: (binary host → simulator)
    - callthru source /home/systemsim/hello/ppu/hello_ppu > hello_ppu
    - chmod 755 hello_spu
    - ./hello_spu
  - Example (result file simulator → host)
    - callthru sink /home/systemsim/results/result_file < cat result_file
    - exporting result files out of the simulated environment for later inspection

Running the Binary

- Start the simulator
  - **# cd systemsim-cell-release/run/cell/linux**
  - **#../run_gui**
  - **Hit “Go”**



Execute Binary

- Bring executable(s) into the simulator using the callthru utility
  - **callthru source /home/systemsim/hello/ppu/hello_ppu > hello_ppu**
- Execute binary
  - **chmod 755 hello_spu**
  - **./hello_spu**

Tip!
Copy binary to /tmp/´<exe> on host to shorten the filename

Directory Structure

- **hello_ppu**
- **hello_be**
  - spu

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