Hetero: Demonstration Application for the Cmpware CMP-DK (Demo Version 2.0 for Eclipse 3.0)

Cmpware, Inc.

Introduction

The *Cmpware Configurable Multiprocessor Development Kit (CMP-DK)* is a multiprocessor simulation and software development environment. It provides fast and efficient modeling of multiprocessor architectures as well as support for software development on such systems. The goal of supporting software development is achieved by providing an interactive, display-rich environment that permits large amounts of information to be displayed in a fast, simple and uncluttered format. Such capabilities are essential in analyzing the behavior of multiprocessor systems.

This demonstration version of the *Cmpware CMP-DK* (version 2.0) for Eclipse 3.0 and higher contains all features of the standard toolkit, but restricts the simulation model to a 3 x 3 heterogeneous array of MIPS32 and SPARC-8 processors. All simulation capabilities and displays are included. This includes:

- Source Code Tracing
- Source Code Variables
- Disassembly
- Memory Display
- Power Estimator
- General Purpose Registers
- Special Purpose Registers
- Command Line Interface
- Link Utilization

Demonstration Applications

Avaliable for use with the *Cmpware CMP-DK* version 2.0 is a series of demonstration applications which are presented to introduce some of the features in the *CMP-DK*. These applications start with small, simple programs gradually building up to more complex applications exploiting relatively low-level parallelism. These demonstrations stand alone and can be studied in any order, but it is best to start with the early examples, which are smaller and simpler and build up to the larger ones. This provides a tutorial-like introduction to the features in the *Cmpware CMP-DK*.

While these demonstrations cover the application development aspects of this tool, much of the power in the *Cmpware CMP-DK* is in the ability to quickly model relatively complex multiprocessor systems. This modeling activity is reserved for licensed copies of the software. For more information on getting licensed copies of the *Cmpware CMP-DK,* contact Cmpware at *info@cmpware.com*.

The groups of files in this tutorial package are as follows:

- **Introduction** - An introduction to all of the applications
- **Simple** - A simple, single processor test application
- **Ping Pong** - a simple two processor application
- **Hetero** - the Ping Pong application on two different types of processors
- **FIR Filter** - A multiprocessor Finite Impulse Response (FIR) Filter
- **AES Encryption** - A multiprocessor AES encryption implementation
- **FFT Filter** - a multiprocessor FFT filter using shared memory
- **FFT Filter 2** - a multiprocessor FFT filter using communication channels

These example applications assume that the *Cmpware CMP-DK* has already been successfully installed on your system. For more information on acquiring and installing either the free demonstration version or the fully licensed verrsion, see the Cmpware web site.

The source and compiled code for these demonstration applications can be downloaded from the Cmpware Web site as a compressed ZIP archive at:

http://www.cmpware.com/Apps/CmpwareApps_2_0.zip

The Hetero Application

After the *Ping Pong* application has introduced multiprocessor operation of the *Cmpware CMP-DK*, the *Hetero* application introduces so-called *Heterogeneous* multiprocessor operation. This simply means a multiprocessor with more than one type of processor. Conversely, a multiprocessor with only a single type of processing element is often called a *Homogeneous* multiprocessor. In this case, the previous *Ping Pong* application will be run on two different types of processors, a *MIPS32* and a *SPARC-8*. This demonstrates the ability of the *Cmpware CMP-DK* to support multiple processor types transparently in the IDE. It also shows that source code can be transparently, and often with no modification, be moved to a new processor architecture with only a simple recompilation.

The *Hetero* code is all in the compressed ZIP archive under the *Hetero* directory, and contains source code, Makefile, linker directives file, compiled relocatable object files and finally, fully linked executable ELF files. In fact, all of the demonstration applications will contain these types of files. All have been built using the *Gnu GCC* compiler with a version higher than 3.0. If you have access to a *MIPS32* and *Sparc-8* compilers which produces standard *ELF* executable with *DWARF2* debug information, you may modify these files and re-compile them and test the results and use them in the *Cmpware CMP-DK*.

Running the Hetero Application

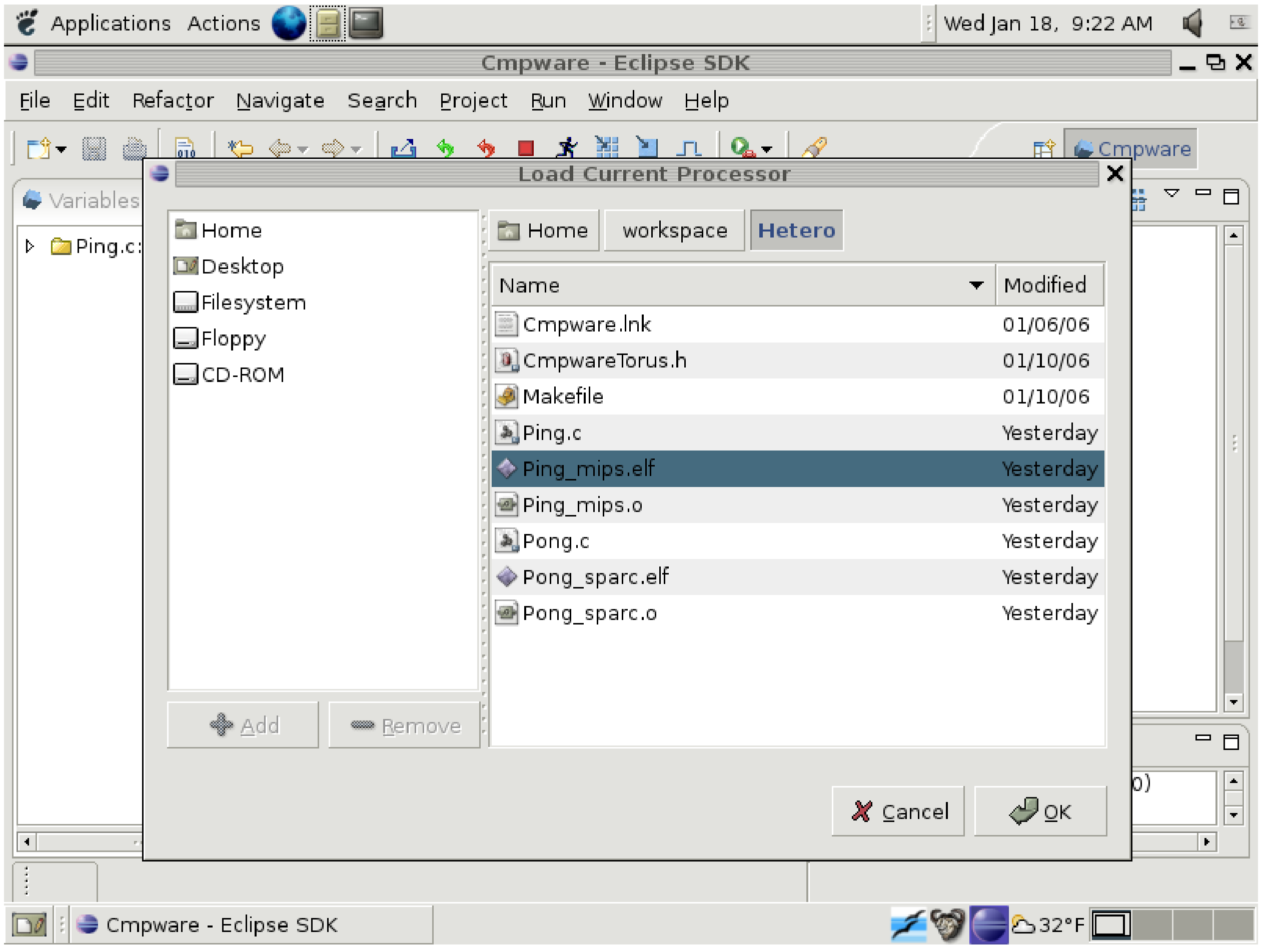
First, the *Cmpware* perspective in Eclipse should be opened. This is typically done from the Eclipse main menu using the **Window --> Open Perspective --> Cmpware** menu command. If you have problems getting this view to come up, or have not installed the *Cmpware CMP-DK*, see the installation guide available on the *Cmpware* web site. It will guide you in installing the software.

The *Cmpware CMP-DK* used in this example is the demonstration version of the software and begins with the default 3 x 3 array of processors. The first row contains three *MIPS32* processors, the second row three *Sparc-8* processors, and the third row contains another three *MIPS32* processors.

Like the previous examples, executable code is loaded into the first processor in the upper left corner of the array. To load this processor with executable code, select the processor with the mouse. It should be highlighted with a grey background and the **Status** window at the bottom should indicate that the processor **MIPS32(0,0)** is selected.

Use the **Load** button () to bring up a file selection dialog. Using this file selection dialog, select the *Ping_mips.elf* file from the list of files for the *Hetero* demonstration as shown in Figure 1. A message in the **Status** window at the bottom of the IDE should indicate that the file was successfully loaded into the MISP32 processor at location (0,0).

Next, select the second processor on the first column with the mouse. It should be highlighted with a grey box much like the (0,0) processor was initially. The **Status** window should indicate that the Sparc-8 processor at (1,0) is selected. Again using the **Load** button (), bring up a file selection dialog. Using this file selection dialog, select the *Pong_sparc.elf* file from the list of files for the *Ping Pong* demonstration. A message in the **Status** window should indicate that the file was successfully loaded into the Sparc-8 processor at location (1,0). Note that these two executable ELF files are compiled for completely different architectures. Loading them to the wrong type of processor will typically result in illegal opcode errors upon execution.

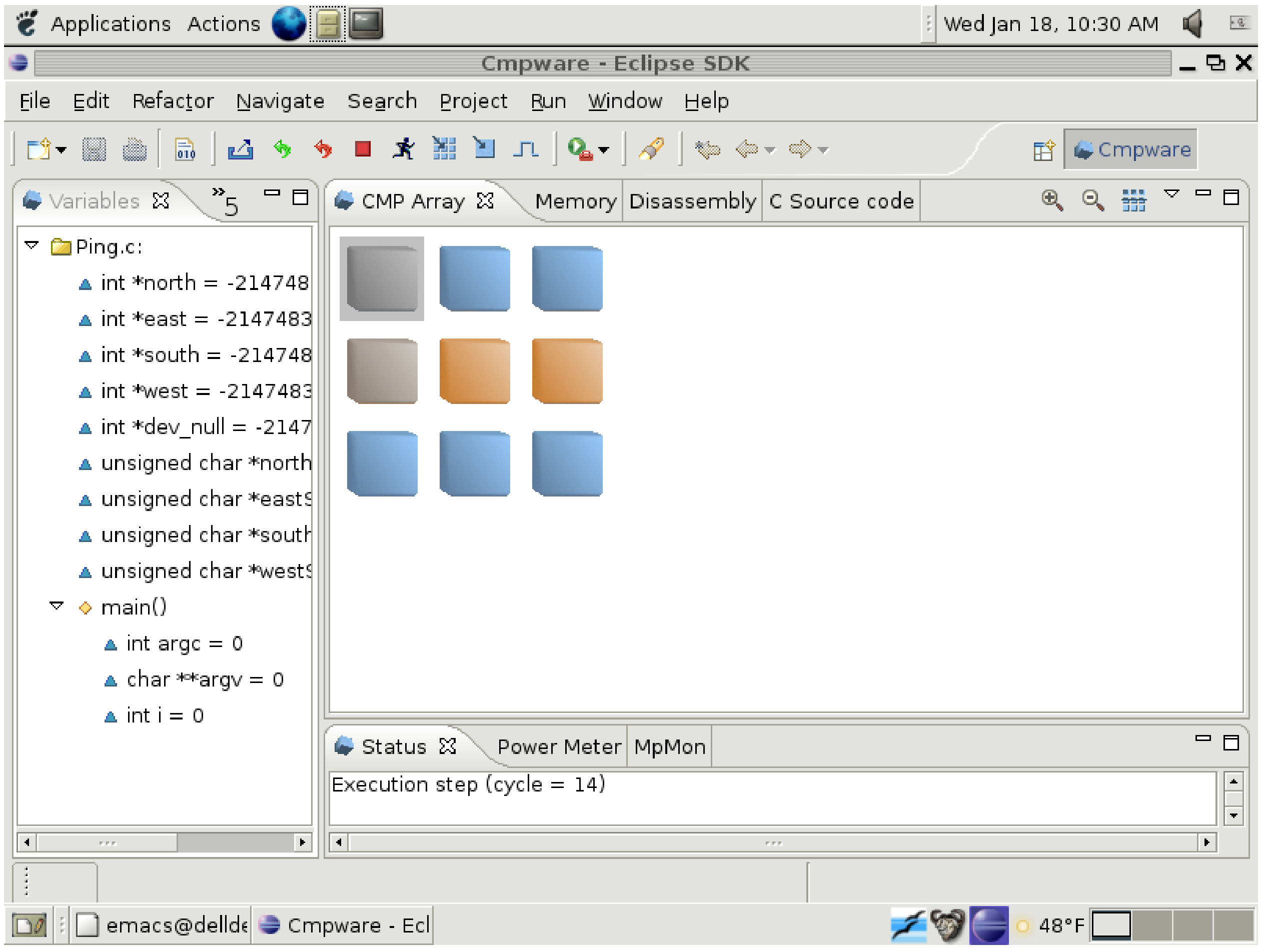


Name	Modified
Cmpware.lnk	01/06/06
CmpwareTorus.h	01/10/06
Makefile	01/10/06
Ping.c	Yesterday
Ping_mips.elf	Yesterday
Ping_mips.o	Yesterday
Pong.c	Yesterday
Pong_sparc.elf	Yesterday
Pong_sparc.o	Yesterday

Figure 1: Loading the Hetero ELF executable files.

At this point, the executable file *Ping_mips.elf* is loaded into the processor in the upper left corner, and the executable file Pong_sparc.elf is loaded into the processor just below the MIPS32 at (0,0). This location is important because the software expects to communicate with the neighboring node in a specific manner.

Clicking on the **Step** button (⎍) advances the global clock. Each step updates the displays in the *Cmpware CMP-DK*. In the view in Figure 2, the multiprocessor has been stepped through 14 cycles, as indicated by the **Status** window. Unfortunately, both processors which were loaded with executable code appear to be idle and displayed in a greyed color. Some investigation will be necessary to find out why this multiprocessor code is not executing.



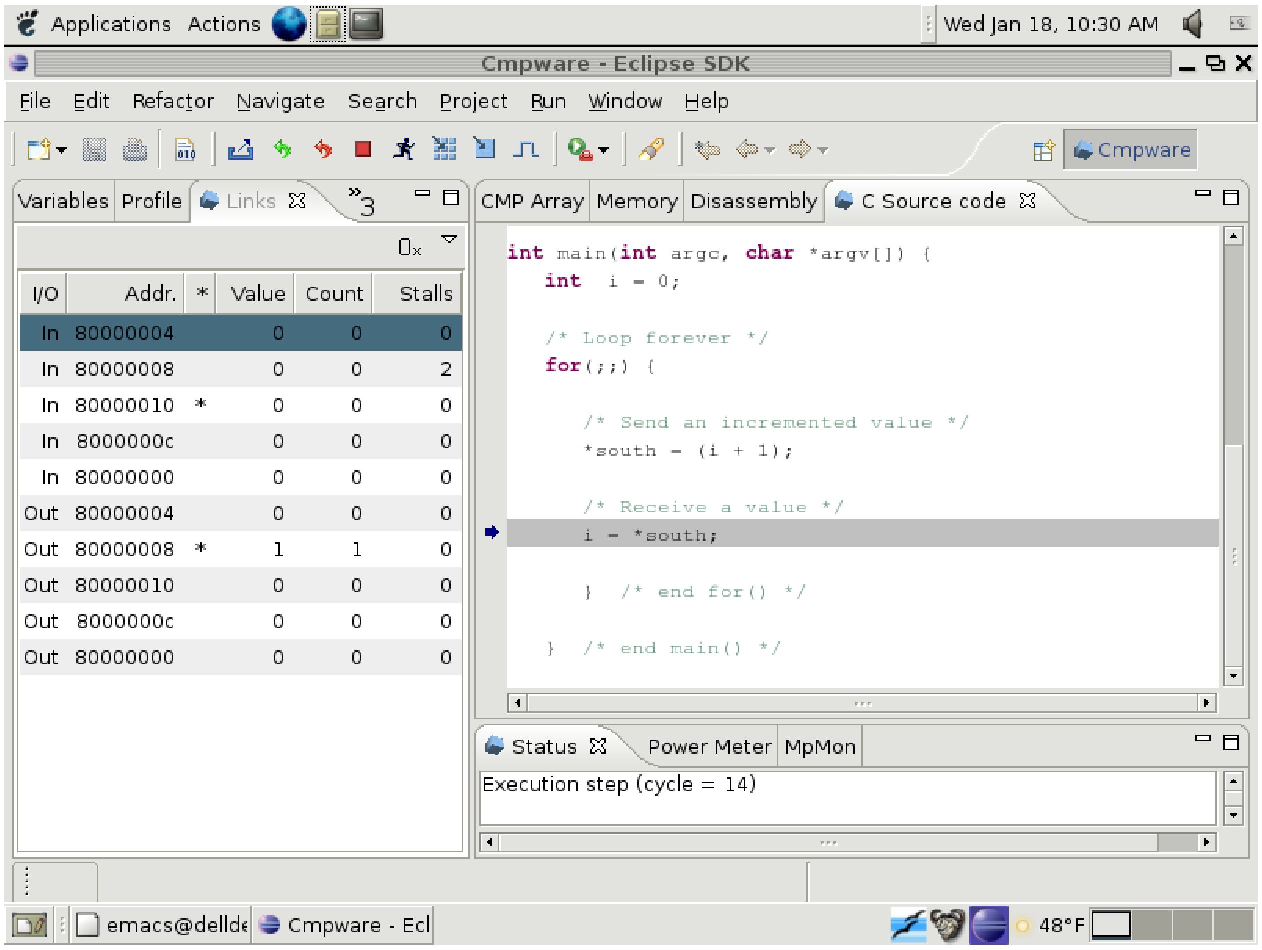
```
Ping.c:
  int *north = -214748
  int *east = -2147483
  int *south = -214748
  int *west = -2147483
  int *dev_null = -2147
  unsigned char *north
  unsigned char *east$
  unsigned char *south
  unsigned char *west$
  main()
    int argc = 0
    char **argv = 0
    int i = 0
```

```
Execution step (cycle = 14)
```

Figure 2: Executing the *Hetero* files.

The first thing to do is to find out exactly where in the code the processors have stalled. Figure 3 shows the MIPS32 processor **Source Code** and **Links** views. The code appears to be stalled after sending a value to the *south* link, and is waiting for data to be returned on the *south* link.

The **Link** display shows that data has indeed been written to the Output port at address 0x80000008. The *CmpwareTorus.h* definition in *Appendix C* file indicates that this is the south port, as expected. The asterisk character ("*") indicates that the port contains data ready to be read. Interestingly, there seems to be no input data ready to be read. The asterisk in the Input port at 0x80000010 corresponds to the "*/dev/null*" port, which can always be read and is part of the network model simply as a software convenience. None of the four communication ports appears to have available data.



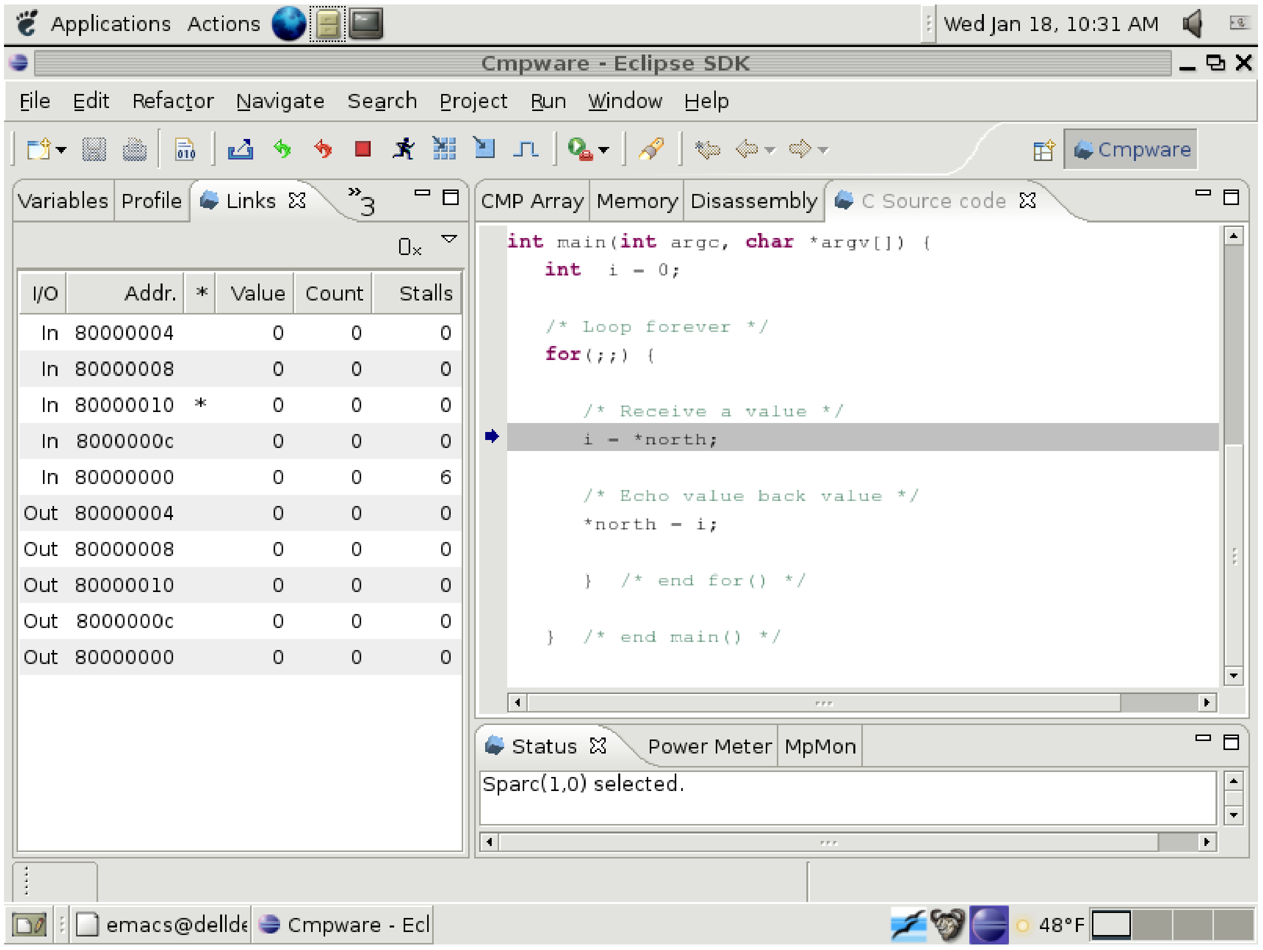
I/O	Addr.	*	Value	Count	Stalls
In	80000004		0	0	0
In	80000008		0	0	2
In	80000010	*	0	0	0
In	8000000c		0	0	0
In	80000000		0	0	0
Out	80000004		0	0	0
Out	80000008	*	1	1	0
Out	80000010		0	0	0
Out	8000000c		0	0	0
Out	80000000		0	0	0

```
int main(int argc, char *argv[]) {
   int  i = 0;

   /* Loop forever */
   for(;;) {

      /* Send an incremented value */
      *south = (i + 1);

      /* Receive a value */
      i = *south;

      }  /* end for() */

   }  /* end main() */
```

```
Execution step (cycle = 14)
```

Figure 3: The Cmpware CMP-DK Links and Source Code views.

Figure 4 shows the same **Source Code** and **Link** views, but for the *Sparc* processor at index (1,0) in the processor array. This view is changed by returning to the main **CMP Array** view, selecting the Sparc processor, the returning to the **Source Code** view. This ability to quickly change processor views is one of the strengths of the *Cmpware* IDE. Because all of the views are continuously updated, a unified view of the multiprocessor is always presented. It is never possible to become confused by looking at windows and not understanding which processor they are associated with. This can become a significant problem in other similar IDEs, particularly when the number of processors becomes large.



I/O	Addr.	*	Value	Count	Stalls
In	80000004		0	0	0
In	80000008		0	0	0
In	80000010	*	0	0	0
In	8000000c		0	0	0
In	80000000		0	0	6
Out	80000004		0	0	0
Out	80000008		0	0	0
Out	80000010		0	0	0
Out	8000000c		0	0	0
Out	80000000		0	0	0

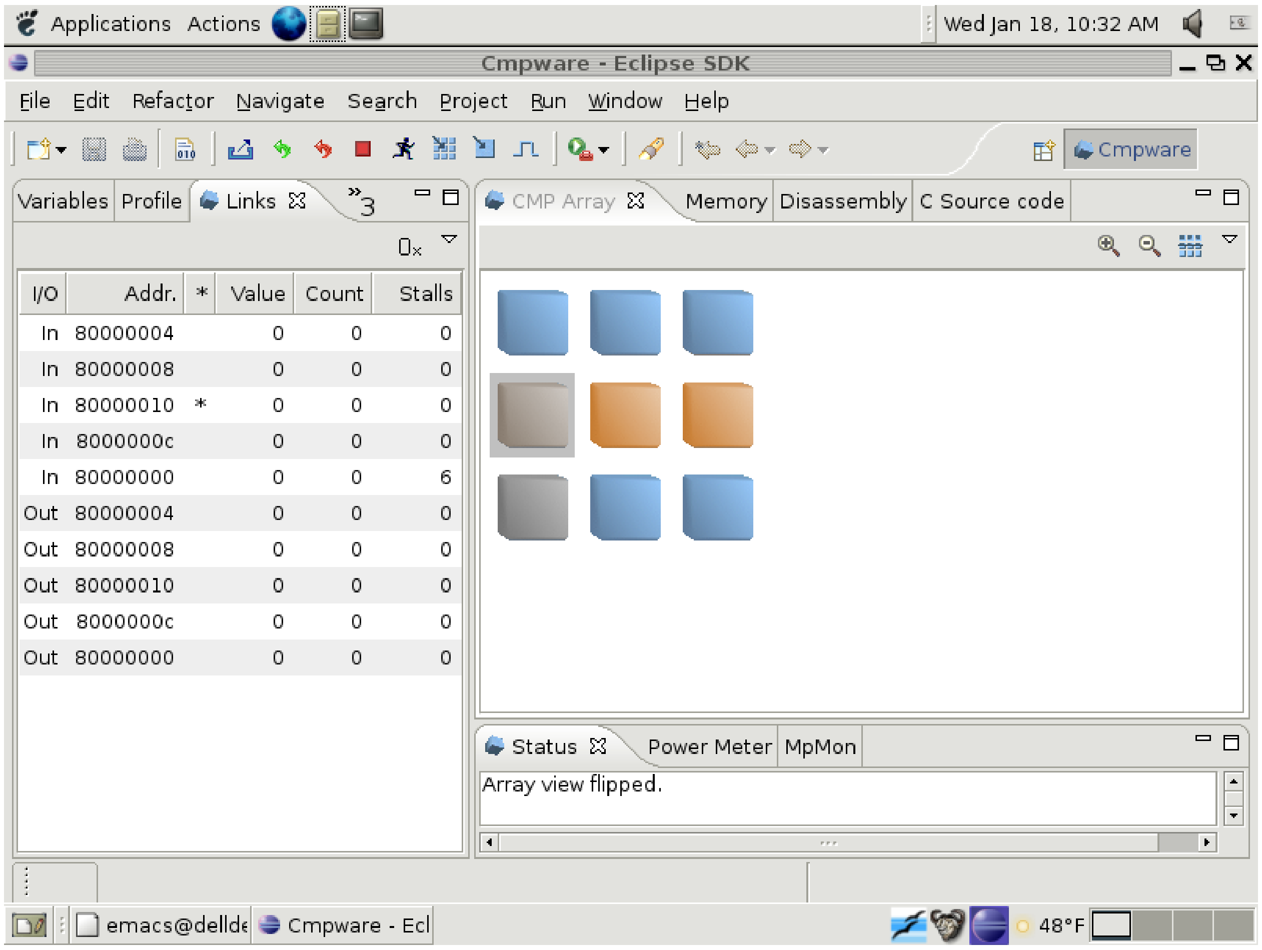
```
int main(int argc, char *argv[]) {
   int   i = 0;

   /* Loop forever */
   for(;;) {

      /* Receive a value */
      i = *north;

      /* Echo value back value */
      *north = i;

      }  /* end for() */

   }  /* end main() */
```

```
Sparc(1,0) selected.
```

Figure 4: The Cmpware CMP-DK Links and Source Code views.

The Sparc processor in Figure 4 is even more perpelxing. It is stalled on the first attempt to read from the *north* Link. And no data seems to be available on any of the other communication links. This unusual situation has a simple solution, and one that is a common source of error in such multiprocessor systems.

Inspecting some of the other nearby processors in the system indicates that data is available on some of their communication links. So the data has gone somewhere, just not to the place it was expected. The reason is mostly in the definition of the ordinal points. The hardware model appears to believe that a Cartesian coordinate system with (0,0) in the lower left corner is the orientation, with "north" being "up". Unfortunately, the default software presentation assumes processor (0,0) in the upper left corner, again with "north" being "up".


Applications Actions
Wed Jan 18, 10:32 AM
Cmpware - Eclipse SDK
File Edit Refactor Navigate Search Project Run Window Help
Cmpware
Variables Profile Links »3
0x
CMP Array Memory Disassembly C Source code
Status Power Meter MpMon
emacs@dellde Cmpware - Ecl
48°F

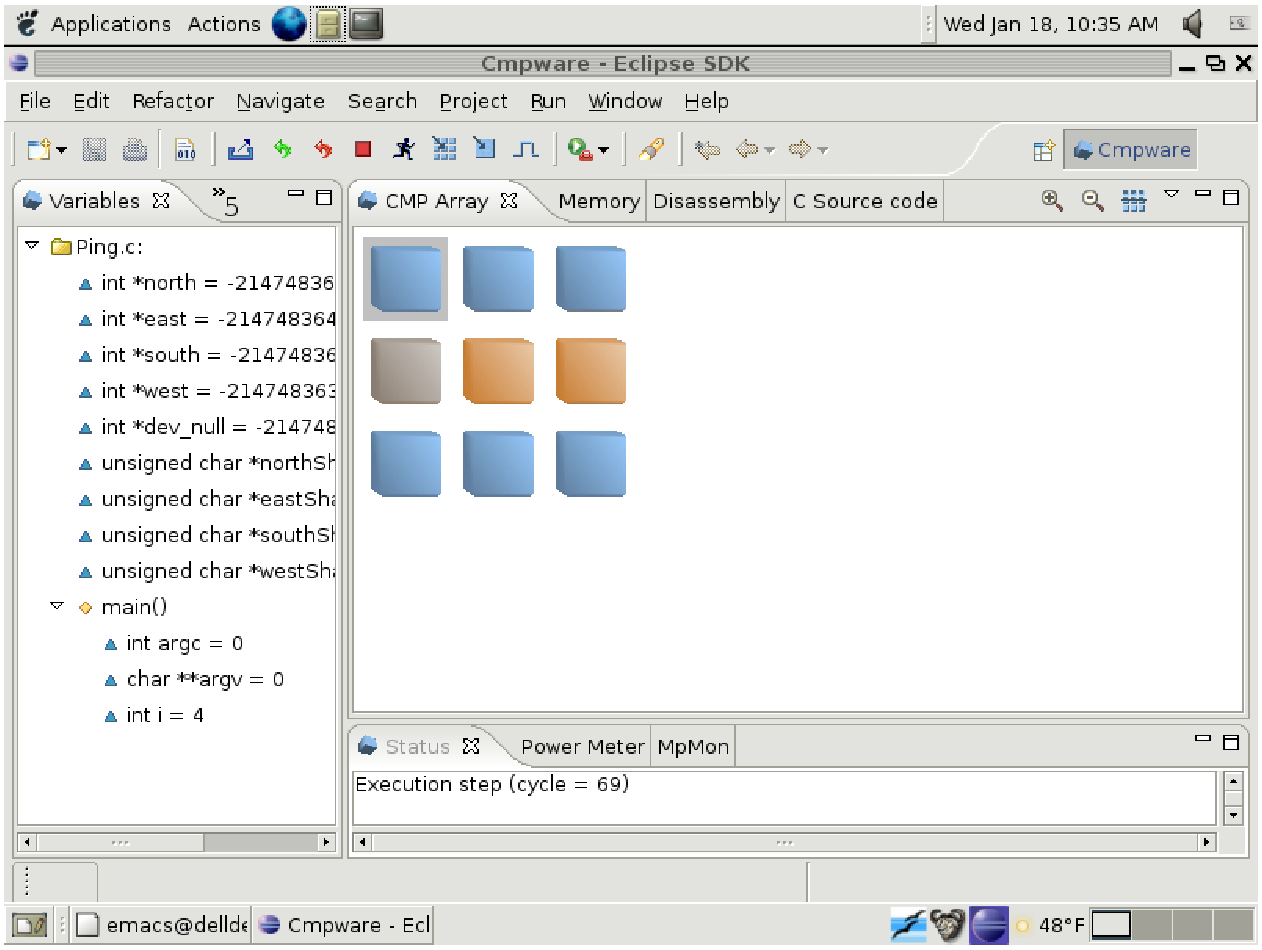
I/O	Addr.	*	Value	Count	Stalls
In	80000004		0	0	0
In	80000008		0	0	0
In	80000010	*	0	0	0
In	8000000c		0	0	0
In	80000000		0	0	6
Out	80000004		0	0	0
Out	80000008		0	0	0
Out	80000010		0	0	0
Out	8000000c		0	0	0
Out	80000000		0	0	0

```
Array view flipped.
```

Figure 5: The flipped array view.

The solution is just to flip the array with the **Flip** button (▦) in the **CMP Array** display window. This puts the (0,0) processor index at the lower left. as in Figure 5. Note that this is merely a change in the way that the processor array is displayed. No changes to the model have been made. It is just a simple way to re-orient the display in two common arrangements. Unfortunately, the code is now loaded in the wrong processors.

First, the array should be reset using the **Reset** button ( ) to prevent the erroneously loaded code from continuing to execute. Note that in the in the demonstration version the default *MIPS32* and *Sparc* models load a simple loop of assembly language code into address zero upon reset. This means that all of the processors in the array will be executing some code on each step, but most will be executing this default assembly language loop. This prevents various errors should the processors attempt to execute from uninitialized memory and is just a developer convenience.



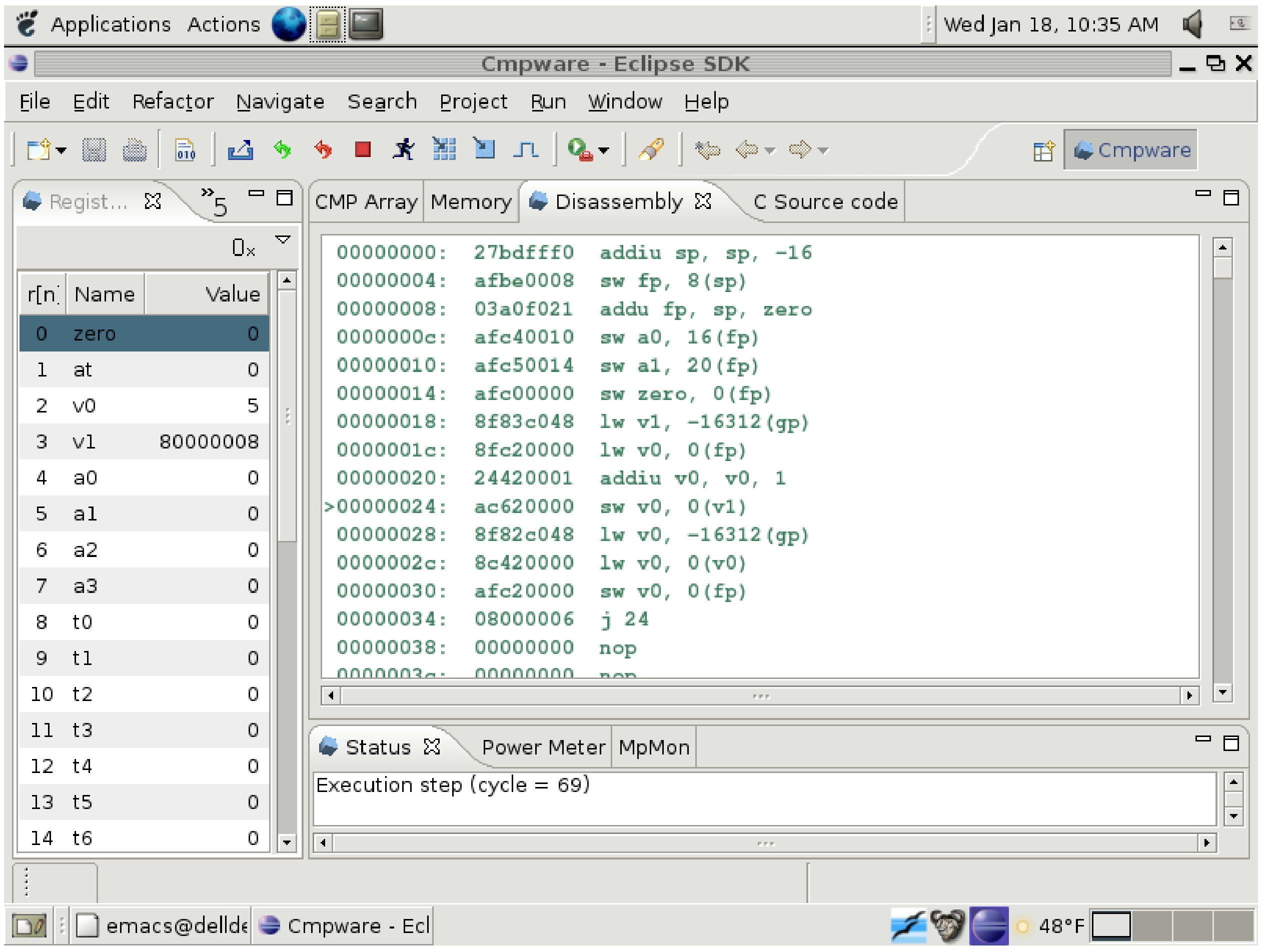
```
Ping.c:
    int *north = -21474836
    int *east = -214748364
    int *south = -21474836
    int *west = -214748363
    int *dev_null = -214748
    unsigned char *northSh
    unsigned char *eastSha
    unsigned char *southSh
    unsigned char *westSha
  main()
    int argc = 0
    char **argv = 0
    int i = 4
```

```
Execution step (cycle = 69)
```

Figure 6: The working Hetero application after flipping and reloading.

Figure 6 shows the flipped array with the *Ping_mips.elf* code loaded into the processor in the upper left (which is now processor (0, 2)) and the *Ping_sparc.elf* code loaded into the processor below. Figure 6 shows the heterogeneous *Ping Pong* application executing successfully, with the processors taking turns execuuting and going idle, and with the variable *i* incrementing with each round-trip.

As in the previous *Ping Pong* application, the **Run** button (🏃) may be pressed and an animated view of the executing can be observed. The **Stop** button (🟥) is used to halt execution of a running multiprocessor program.


Applications Actions
Wed Jan 18, 10:35 AM
Cmpware - Eclipse SDK
File Edit Refactor Navigate Search Project Run Window Help
Cmpware
Regist... »5
0x
CMP Array Memory Disassembly C Source code
Status Power Meter MpMon
emacs@delld Cmpware - Ecl
48°F

r[n]	Name	Value
0	zero	0
1	at	0
2	v0	5
3	v1	80000008
4	a0	0
5	a1	0
6	a2	0
7	a3	0
8	t0	0
9	t1	0
10	t2	0
11	t3	0
12	t4	0
13	t5	0
14	t6	0

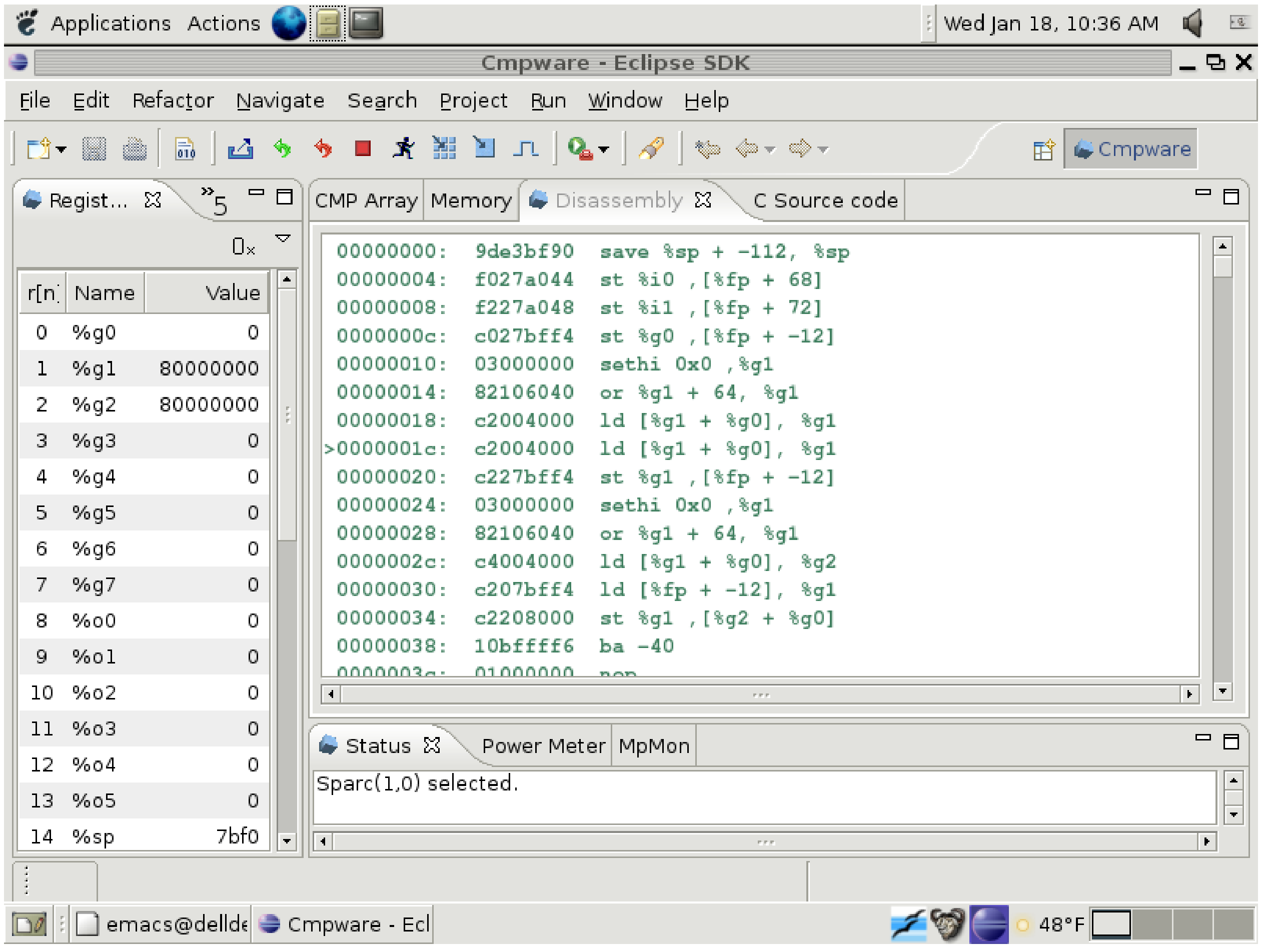
```
 00000000:  27bdfff0  addiu sp, sp, -16
 00000004:  afbe0008  sw fp, 8(sp)
 00000008:  03a0f021  addu fp, sp, zero
 0000000c:  afc40010  sw a0, 16(fp)
 00000010:  afc50014  sw a1, 20(fp)
 00000014:  afc00000  sw zero, 0(fp)
 00000018:  8f83c048  lw v1, -16312(gp)
 0000001c:  8fc20000  lw v0, 0(fp)
 00000020:  24420001  addiu v0, v0, 1
>00000024:  ac620000  sw v0, 0(v1)
 00000028:  8f82c048  lw v0, -16312(gp)
 0000002c:  8c420000  lw v0, 0(v0)
 00000030:  afc20000  sw v0, 0(fp)
 00000034:  08000006  j 24
 00000038:  00000000  nop
 0000003c:  00000000  nop
```

```
Execution step (cycle = 69)
```

Figure 7: The registers and disassembly for a MIPS32 processor.

Lastly, the *Cmpware* IDE is notable for its transparent support for heterogeneous multiprocessors. If the *MIPS32* processor is selected in the **CMP Array** view and the main window is switched to the **Disassembly** view, the expected *MIPS32* assembly language can be seen. Similarly, the **Register** and **Special Register** tabs on the left bring up *MIPS32* registers and special registers with the most recent values as in Figure 7.

If the *Sparc-8* processor is similarly selected in the **CMP Array** view and the views returned to the **Disassembly** and **Registers** as in Figure 8, the expected *Sparc* assembly language and registers, all with the most recent values, can be viewed.



r[n]	Name	Value
0	%g0	0
1	%g1	80000000
2	%g2	80000000
3	%g3	0
4	%g4	0
5	%g5	0
6	%g6	0
7	%g7	0
8	%o0	0
9	%o1	0
10	%o2	0
11	%o3	0
12	%o4	0
13	%o5	0
14	%sp	7bf0

```
 00000000:  9de3bf90  save %sp + -112, %sp
 00000004:  f027a044  st %i0 ,[%fp + 68]
 00000008:  f227a048  st %i1 ,[%fp + 72]
 0000000c:  c027bff4  st %g0 ,[%fp + -12]
 00000010:  03000000  sethi 0x0 ,%g1
 00000014:  82106040  or %g1 + 64, %g1
 00000018:  c2004000  ld [%g1 + %g0], %g1
>0000001c:  c2004000  ld [%g1 + %g0], %g1
 00000020:  c227bff4  st %g1 ,[%fp + -12]
 00000024:  03000000  sethi 0x0 ,%g1
 00000028:  82106040  or %g1 + 64, %g1
 0000002c:  c4004000  ld [%g1 + %g0], %g2
 00000030:  c207bff4  ld [%fp + -12], %g1
 00000034:  c2208000  st %g1 ,[%g2 + %g0]
 00000038:  10bffff6  ba -40
 0000003c:  01000000  nop
```

```
Sparc(1,0) selected.
```

Figure 8: The registers and disassembly for a Sparc-8 processor.

This tightly integrated model and display approach permits a wide variety of different processing elements to work together while providing fast, uniform access to the processor data. Such fast and uniform access to data greatly improves the ability to produce and program such heterogeneous multiprocessor systems. In addition, the use of a single family of windows all displaying data from the same processor greatly helps to avoid confusion common in multi-window approaches. The vast amount of data produced by a multiprocessor quickly overwhelms users who attempt to view the operation of more than a small number of processors simultaneously. Fast switching between processors and a self-consistent interface help avoid many of the problems associated with multiprocessor software development and debug.

Conclusions

The *Cmpware CMP-DK* is a rich display environment combining fast simulation and flexible multiprocessor modeling. This makes it an ideal environment for architecture modeling and software development for these systems.

While the execution and display features of the *Cmpware CMP-DK* are notable, much of the power of the system lies in its ability to quickly and flexibly construct processor, network, link and multiprocessor models. This modeling capability is a large part of the commercial version of the *Cmpware CMP-DK*.

For more information on the commercial version of the *Cmpware CMP-DK* see our web site at:

http://www.cmpware.com/

or send an email to:

info@cmpware.com

Appendix A: Ping.c Source Code

```
/*
**  This implements the 'Ping' half of the 'Ping Pong' demo.  It
**  sends a value to the south port, then reads a value from the
**  north port.  In the heterogeneous processor demo using the
**  default Cmpware configuration, the executable ELF file from
**  this code should be compiled for a MIPS32 processor and loaded
**  into the (0,0) processor.
**
**  Copyright (c) 2005, 2006 Cmpware, Inc.  All rights reserved.
**
*/

#include "CmpwareTorus.h"

int main(int argc, char *argv[]) {
   int  i = 0;

   /* Loop forever */
   for(;;) {

      /* Send an incremented value */
      *south = (i + 1);

      /* Receive a value */
      i = *south;

      }  /* end for() */

   }  /* end main() */
```

Appendix B: Pong.c Source Code

```
/*
**  This implements the 'Pong' half of the 'Ping Pong' demo.  It
**  sends a value to the north port, then reads a value from the
**  south port.  In the heterogeneous processor demo using the
**  default Cmpware configuration, the executable ELF file from
**  this code should be compiled for an SPARC-8 and loaded into
**  the (1,0) processor.
**
**  Copyright (c) 2005, 2006 Cmpware, Inc.  All rights reserved.
**
*/

#include "CmpwareTorus.h"

int main(int argc, char *argv[]) {
   int  i = 0;

   /* Loop forever */
   for(;;) {

      /* Receive a value */
      i = *north;

      /* Echo value back value */
      *north = i;

      }  /* end for() */

   }  /* end main() */
```

Appendix C: CmpwareTorus.h Source Code

```
/*
**
**  This defines the shared memory and links in the 'Torus' topology.
**  This must agree with the values in the memory map for the
**  hardware and the simulation model.
**
**  Copyright (c) 2005, 2006 Cmpware, Inc.  All rights reserved.
**
*/

#ifndef _CMPWARETORUS_H_
#define _CMPWARETORUS_H_


/* The Memory Mapped IO Ports */
typedef volatile int *Port;

/* A shared memory address */
typedef unsigned char *Address;


/* The size of the local memory */
#define LOCAL_MEMORY_SIZE   (32 * 1024)

/* The size of the shared memory */
#define SHARED_MEMORY_SIZE  (8 * 1024)


/* Memory Mapped IO ports */
#ifdef SELFHOSTED
   /* For self-hosted testing, just make a pointer to an int */
   Port  north[1];
   Port  east[1];
   Port  south[1];
   Port  west[1];
   Port  dev_null[1];
#else
   /* Point to links in hardware memeory map */
   Port  north = (Port) 0x80000000;
   Port  east  = (Port) 0x80000004;
   Port  south = (Port) 0x80000008;
   Port  west  = (Port) 0x8000000c;
   Port  dev_null  = (Port) 0x80000010;
#endif  /* SELFHOSTED */


/* Shared Memory */
#ifdef SELFHOSTED
   /* For self-hosted testing, just allocate arrays */
   #include <stdio.h>
   Address northSharedMemory[SHARED_MEMORY_SIZE];
   Address eastSharedMemory[SHARED_MEMORY_SIZE];
   Address southSharedMemory[SHARED_MEMORY_SIZE];
   Address westSharedMemory[SHARED_MEMORY_SIZE];
#else
   /* else define shared memory addresses corresponding to the hardware */
   Address northSharedMemory = (Address) LOCAL_MEMORY_SIZE;
   Address eastSharedMemory = (Address) (LOCAL_MEMORY_SIZE +
SHARED_MEMORY_SIZE);
   Address southSharedMemory = (Address) (LOCAL_MEMORY_SIZE +
(2*SHARED_MEMORY_SIZE));
   Address westSharedMemory =  (Address) (LOCAL_MEMORY_SIZE +
(3*SHARED_MEMORY_SIZE));
#endif  /* SELFHOSTED */


#endif /* _CMPWARETORUS_H_ */
```