Ping Pong: 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 Ping Pong Application

After the *Simple* application has introduced the basic operation of the *Cmpware CMP-DK*, the *Ping Pong* application introduces some of the basics of multiprocessor simulation and software development. This applications consists of two compiled *ELF* executable files called *Ping.elf* and *Pong.elf*. These each execute on neighboring *MIPS32* processors and communicate with each other. Specifially, they send a single value back and forth, incrementing it once on each round trip.

The Ping Pong code is all in the compressed ZIP archive under the *PingPong* 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* compiler 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 Ping Pong 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.

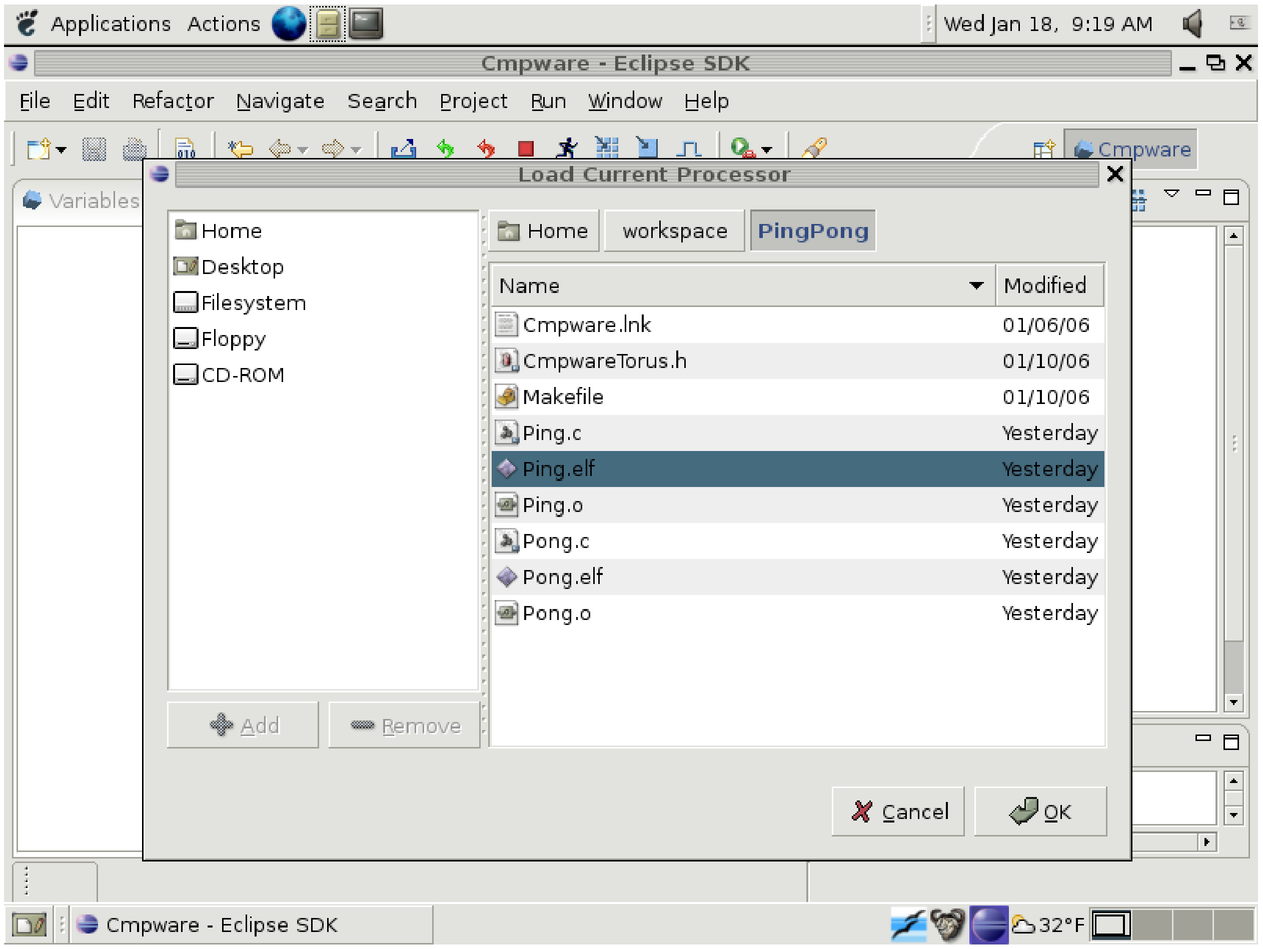
This *Cmpware CMP-DK* used in this example is of the demonstration version of the software and begins with the default 3 x 3 array of processors. The first row contains three *MIPS32* processors, the second row three *Sparc-8* processors, and the third row contains another three *MIPS32* processors.

Like the previous *Simple* example, 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.elf* file from the list of files for the *Ping Pong* 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 row 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 MIPS32 processor at (0,1) is selected. Again using the **Load** button (), bring up a file selection dialog. Using this file selection dialog, select the *Pong.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 MISP32 processor at location (0,1).

At this point, the executable file *Ping.elf* is loaded into the processor in the upper left corner, and the executable file Pong.elf is loaded into the adjacent processor. This location is important because the software expects to communicate with the neighboring node in a specific manner.

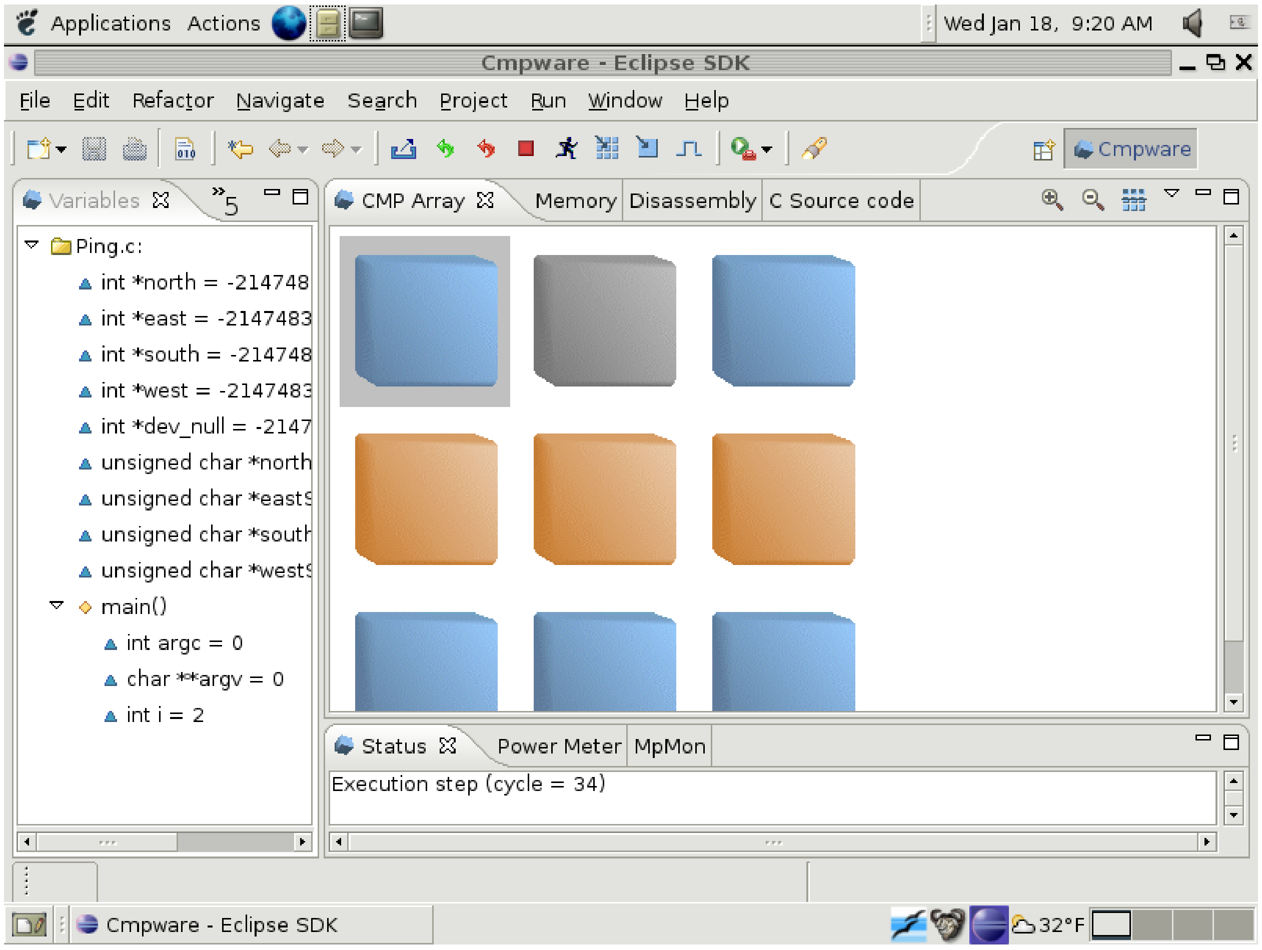


Name	Modified
Cmpware.lnk	01/06/06
CmpwareTorus.h	01/10/06
Makefile	01/10/06
Ping.c	Yesterday
Ping.elf	Yesterday
Ping.o	Yesterday
Pong.c	Yesterday
Pong.elf	Yesterday
Pong.o	Yesterday

Figure 1: Loading the Ping Pong ELF executable files.

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 34 cycles, as indicated by the **Status** window, and the value of the incremented token passed back and forth between the two processors is now 2. Also note that the second processor is displayed in a greyed color. As the processors communicate and wait for communication, they will alternate active and waiting (grey and colored).

Rather than single stepping with the **Step** button (⎍), the **Run** button (🏃) can be used to animate the display. Here the "ping pong" back and forth of the application is more obvious. in addition, the value of the variable *i* in the **Variables** window can be watched as it increments.



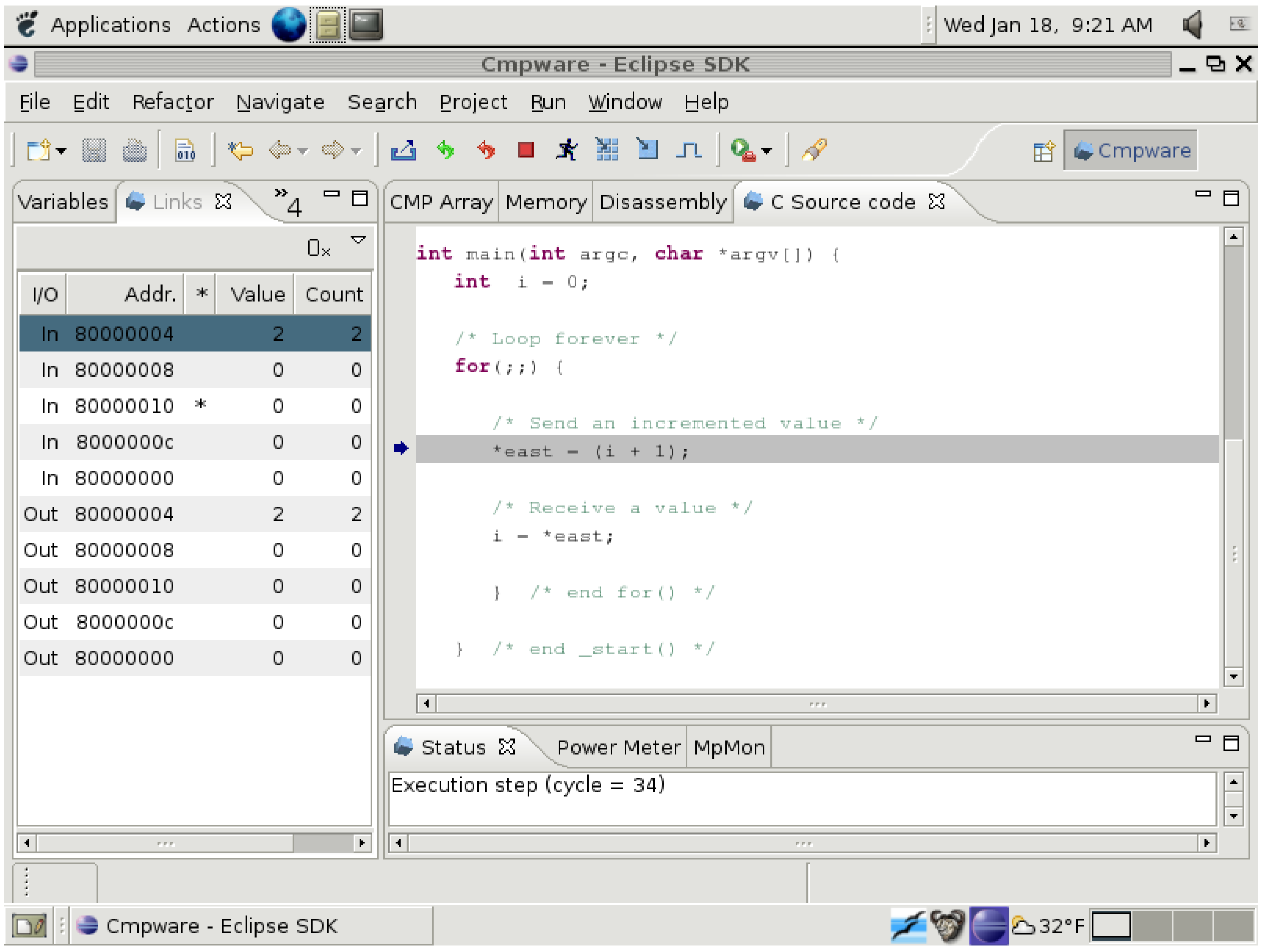
```
Ping.c:
    int *north = -214748
    int *east = -2147483
    int *south = -214748
    int *west = -2147483
    int *dev_null = -2147
    unsigned char *north
    unsigned char *east
    unsigned char *south
    unsigned char *west
  main()
      int argc = 0
      char **argv = 0
      int i = 2
```

```
Execution step (cycle = 34)
```

Figure 2: Executing the *Ping.elf and Pong.elf* files.

Note that even though the underlying simulation engine runs at approximately two million instructions per second, the updating of the displays is typically the limiting factor in performance when such an animation is run. But this view can be very valuable in watching communication patterns and analyzing the utilization of the processing resources. The **Preference Page** for the *Cmpware CMP-DK* contains a parameter to set the **Step Size**, which changes the number of simulation cycles per update of the IDE. The default is one, but it can be changed using the **Windows --> Preferences** menu item and selecting the **Cmpware** preference page. The **Step Size** field is one of the settable parameters on this page.

While the previous example showed animation on the main display, all displays in the IDE are 'live' and update with each step of the simulation.



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

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

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

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

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

      }  /* end for() */

   }  /* end _start() */
```

```
Execution step (cycle = 34)
```

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

Figure 3 shows the **C Source Code** window and the **Links** window. Note that these windows can be selected even while the simulation is running and updating will continue. In the case of the **Source Code** window, the source code line being executed wil lbe displayed, highlighted, in the center of the **Source Code** window.

The **Link** window shows information on the interprocessor communication links. These are essentially synchronized registers that may be written from one processor and read from another. In the default configuration, they are set up in a nearest neighbor pattern producing a mesh or torus topology.

Another way to view these links is a one word FIFO. These links are memory mapped to various addresses, in this case starting at 0x8000000. The **Link** display in this case shows four input links and four output links corresponding to the four directions in the mesh network. An additional input and output link are provided as a conveninece to software. These are "null" links that may be read and written at any time. These may be considered analogous to the *"/dev/null"* file in Unix systems. It is a handy place to send data that is no longer needed.

The simulation model and the software must agree on the memory map of system, including the memory mapped IO of the link registers. The memory map of the model is beyond the scope of this document, but the memory map of the software is specified in the *CmpwareTorus.h* include file as in Appendix C. The links are defined as *north*, *south*, *east* and *west* and are defined at addresses starting at 0x8000000. These values are used in the Ping and Pong source code to define the communication. Since these are memory mapped, they can be accessed as a simple pointer such as **north*.

It should also be noted that communication across processors is as simple as assigning a value to a variable. But some care must be taken to be sure that for each variable written, one is read. This is similar to uniprocessor code, where variable may also be overwritten or ignored. But when the event spans two processors, some extra care must be taken.

This is where the *Cmpware CMP-DK* has its most valuable use. If mis-communication occurs across processors, it is easy to select the involved processors and inspect these variables and code and see where the problem is. It is the ability to rapidly inspect all processors in the system and view high and low level data quickly that permits rapid development and debug of such multiprocessor applications.

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 east port, then reads a value from the
**  west port.  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 */
      *east = (i + 1);

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

      }  /* end for() */

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

Appendix B: Pong.c Source Code

```
/*
**  This implements the 'Ping' half of the 'Ping Pong' demo.  It
**  sends a value to the east port, then reads a value from the
**  west port.  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 */
      *east = (i + 1);

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

      }  /* 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_ */
```