Design Advantage™

Volume 2, Number 2 The newsletter of Comit Systems, Inc. MAY 1999

Modeling arbitrarily large memories in VHDL

By Vijay A. Nebhrajani

Modeling large memories is a tradeoff. On the one hand, if you model the memory as an array, you statically allocate at least that much memory on the host machine. On the other hand, if you choose to model only part of the memory, leaving the other address lines not connected, you perform a partial simulation on your system. Neither solution provides the flexibility of a *create-on-demand* memory that is described below.

The create-on-demand memory works on the following principle: Whenever there is a write to the memory, it appends to an underlying linked list. The linked list stores both the address and the data that were supplied. Reading is performed by scanning the list till an address match occurs. If no address match occurs, an X is returned. This has the advantage that you allocate only as much computer memory as you really need.

This method is not without its disadvantages, though. Assume that you did use the entire memory for simulation - in that case, the linked list is likely to consume more computer memory than a simple array. Further, as the list length increases, the real time required for searching the list increases. The real time problem can be reduced by sorting and storing, using a binary search, or by using hashing functions.

Even with this, the memory consumption problem does not go away. A partial solution is possible if you know beforehand that the address and data will not exceed 32 bits. If so, you can choose to store the address and data as integers; this is cheaper than storing std_logic_vectors. Assuming MVL9, each bit in a std_logic_vector would require 4 bits in real memory to store. If the simulator is efficient, 32 address lines would require 16 bytes of real memory. If you stored it as an integer however, 32 bits would require only 4 bytes. The address and a pointer to the next element in the list would require another 4 + 4 = 8 bytes. Thus, a total of 12 bytes per structural element are required, in lieu of 16 per array element. This is certainly cheaper, but of course, there would be the computational overhead of conversion between std_logic_vector and integer for every read and write.

You can choose to make your own tradeoffs when you make models using linked lists. The good news is that these tradeoffs can result in a faster, more resource-efficient model.

Either way you choose, you will need to build a linked list in VHDL to be able to successfully model the memory without the penalties described above. A simple linked list based memory package: *'mempkg.vhd'*, written in VHDL 93 is available at the Comit Systems website for free download (www.comit.com). A VHDL memory model, 1 Meg locations x 32 bits wide, that uses *mempkg* is also provided. The test bench and associated vectors file is also available at the same location as a ready to use bundle. The memory package defines the linked list and write and read functions for the memory. You can choose to build whatever memory you want around this; the package does not impose restrictions about whether you want to model an SRAM or DRAM or FIFO, or any other kind of memory element.

FREE!
Download

Details on compiling, running and using *mempkg* are available on the website in a README file. ■

◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆

Scripting Java

By Vivek Popli & Dinesh Monga

TCL is a very powerful, flexible, un-typed, high level scripting language. By providing an interpreter and a UI builder, TCL simplifies integration of components with a uniform interface and provides a rapid application development environment. TCL is used to 'glue' together components to make larger applications.

But TCL is not a solution in itself, as it does not provide facilities to efficiently generate and process large data structures and doesn't scale well to complex algorithms.

Traditionally TCL has been used widely with C/C++ for application development in EDA industry. This interface is not 100% platform independent, requiring porting, builds and release on multiple platforms.

Comit realizes that to simplify software development management and usage, a truly platform independent solution is essential

Java with its platform-independent features makes an ideal partner with TCL. This next generation programming language has large collections of programming APIs and technologies. It has built in safety checking, complex data structures and is ideal for building components from scratch.

Java Command Language (Jacl) is a scripting language that combines the power of both TCL and Java. It is designed to glue together Java components using TCL like syntax. Jacl is a pure 100% Java implementation of the TCL command

Comit now designs AMBA compliant interfaces for on-chip digital peripherals!

Design Advantage™

The newsletter of:

COMIT ®
SYSTEMS

Comit Systems, Inc.
3375 Scott Blvd. Suite 330
Santa Clara, CA 95054
Phone : (408)-988-7966

Editor : Jaishankar Iyer
We welcome your comments and suggestions. Please email us at jiyer@comit.com

Download White Paper:
Competitive Positioning for the New Millennium.
http://www.comit.com

language that allows users to interact with the Java Virtual Machine. By adding scripting capabilities to Java applications, Java developers can add a great deal of power, flexibility and end user customization capability to their Java applications. . Users can load Java classes, create new instances of Java classes, and access public members of Java objects. Jacl allows users to write portable Java extensions.

Applications for Jacl are limitless e.g GUI development, CGI scripting, small application development. An example to illustrate the use of Jacl for GUI development is shown below. This demonstrates the advantage of using JACL over plain Java.

```
 # Create a top level frame to hold menu bar
set frame [java::new java.awt.Frame "Jacl Demo"]

# Create the menu bar
set menuBar [java::new java.awt.MenuBar]
$frame setMenuBar $menuBar

# Create the 'File' menu and add items to it
set fileMenu [java::new java.awt.Menu "File"]
set editMenu [java::new java.awt.Menu "Edit"]
set viewMenu [java::new java.awt.Menu "View"]
set stupMenu [java::new java.awt.Menu "Setup"]
set helpMenu [java::new java.awt.Menu "Help"]
set fileMenuOpen [java::new java.awt.MenuItem "Open..."]
$fileMenu add $fileMenuOpen

set fileMenuNew [java::new java.awt.MenuItem "New..."]
$fileMenu add $fileMenuNew

set fileMenuSave [java::new java.awt.MenuItem "Save"]
$fileMenu add $fileMenuSave

set fileMenuSaveAs [java::new java.awt.MenuItem "Save
As..."]
$fileMenu add $fileMenuSaveAs

$menuBar add $fileMenu
$menuBar add $editMenu
$menuBar add $viewMenu
$menuBar add $stupMenu
$menuBar add $helpMenu

$frame pack
$frame toFront
$frame setLocation 100 100
$frame setSize 400 100
```

At Comit, we understand the time constraint to develop and market the product. Our knowledge in Jacl and Java helps customers to develop platform independent applications, quickly. ■