A Comparison of VHDL and Verilog Resource Usage by Behavioral Memory Models

Richard Munden
Free Model Foundry
www.FreeModelFoundry.com

Abstract

The amount of memory included in embedded systems today is often in the gigabytes and is continuing to grow creating a strain on board-level simulation resources. This paper discusses four methods of modeling memory components at the behavioral level and the relative performance of each in terms of computer memory consumed and simulation speed. Two methods use VHDL and two use Verilog. Likewise, two use static memory allocation and two use dynamic memory allocation. Two popular mixed language simulators are used in making the performance comparisons.

1.0 Introduction

An engineer and user of FMF (www.FreeModelFoundry.com) component models was recently having difficulties simulating a design that included a large amount of memory. His simulations would die during elaboration because his PC did not have enough memory. He was advised by an applications engineer from his simulator vendor that Verilog models would consume less computer memory than VHDL models.

This seemed contrary to my personal experience. I wanted to say the applications engineer was wrong but, I did not have the numbers to back me up. I told the engineer I would get the numbers and report back to him. Using models for two 1Gb flash memory parts, one 8-bits wide and the other identical expect 16-bits wide, I ran some experiments and recorded the results. The numbers I got were surprising. The experiments and the results are described below.

2.0 A Comparison of Modeling Methods

The models used were recently written for two new 1Gb flash memories from Spansion. I cannot give the part numbers at this time because they have not yet been announced. Although the parts are both 1Gb, one is organized in 8-bit words and the other in 16-bit words. They are identical in all other respects.

2.1 VHDL

Each part has a single VHDL model with two architectures. The first architecture used dynamic memory allocation.

2.1.1 VHDL Dynamic Allocation

Here the memory is implemented as a linked list as shown in the following declarations:

```
-- -------------------------------------------------------------------------
    -- Data types required to implement link list structure
    -- -------------------------------------------------------------------------
    TYPE mem_data_t;
    TYPE mem_data_pointer_t IS ACCESS mem_data_t;
    TYPE mem_data_t IS RECORD
        key_address  :  INTEGER;
        val_data     :  INTEGER;
        successor    :  mem_data_pointer_t;
    END RECORD;

    -- -------------------------------------------------------------------------
    -- Array of linked lists.
    -- Support memory region partitioning for faster access.
    -- -------------------------------------------------------------------------
    TYPE mem_data_pointer_array_t IS
        ARRAY(NATURAL RANGE <>) OF mem_data_pointer_t;
```

Memory locations are created only when they are written to. For simulations of large memory devices in which less than about 15% of the locations are accessed, this method conserves substantial computer memory resources as will be shown later.

2.1.2 VHDL Static Allocation

The second VHDL architecture implemented memory as a static array of integers as shown in the following declarations:

```
-- Page Array
    TYPE PageArr IS ARRAY (0 TO PageSize) OF Integer RANGE -1 TO MaxData;
    -- Flash Memory Array
    TYPE MemArr IS ARRAY (0 TO PageNum) OF PageArr;
    TYPE IDArr IS ARRAY (0 TO 3) OF integer RANGE -1 TO MaxData;
```

In this architecture, each word is stored as an integer with possible values ranging from -1 to a value that depends on the size of the word. As such, the computing memory consumed by a 16-bit word (or even a 30-bit word) is no more than that consumed by an 8-bit word. As you can see, this method is more efficient for larger word sizes.

In case you are wondering, the -1 value is used for corrupted locations. In other types of memories, there is also a -2 that is used for unwritten locations so the user can tell the difference. However, in this technology, unwritten locations contain "1"s.

A full discussion of this technique as compared to the traditional array of std_logic_vector, can be found at: ww.freemodel-foundry.com/pdf/mem_model.pdf.

2.2 Verilog

For each part there are also two Verilog models.

2.2.1 SystemVerilog Dynamic Allocation

As in VHDL, memory is implemented as a linked list. Verilog does not have the facilities for this so SystemVerilog must be used as shown below:

```
// abstract memory region model
    class linked_list_c;
        // memory element model
        reg[31:0] key_address;
        integer val_data;
        // organize memory storage elements into a linked list
        linked_list_c successor;

        function new(
            integer address_a,
            integer data_a);
        begin
            key_address = address_a;
            val_data = data_a;
            successor = null;
        end
        endfunction
    endclass
```

Full code can be seen by downloading any of the SystemVerilog memory models from the FMF website.

2.2.2 Verilog Static Allocation

For static allocation, traditional (v2k) Verilog is used. However, rather than describing memory as the usual register arrays, the FMF models following our VHDL convention of using an array of integers:

```
// Mem(Page)(Address)
    integer Mem[0:(PageSize+1)*(PageNum+1)-1];
```

The improvements in computer memory conservation are similar to those achieved in VHDL.

3.0 How the Models were Tested

For each model, tests were run to find the memory footprint of the bare model, the maximum footprint of the model and testbench during simulation, and the CPU time consumed by running each simulation. The test machine was a Sun Ultra60 with dual 450MHz processors and 2GB of memory. The operating system was Solaris10.

Two simulators, from different vendors were utilized. The purpose of the experiment is to understand the implications of different modeling methods, not to compare the performance of the simulators. Therefore, the simulators shall be referred to only as Simulator A, and Simulator B.

3.1 Footprint Size

The footprint size was measured using only Simulator A. Each model of each component was compiled and a simulation was run without stimulus for 1 nanosecond. The simulation was determined necessary because Verilog did not allocate static memory during elaboration.

No compiler optimizations were employed beyond what came standard in the out-of-the-box system initialization file.

Memory footprint was measured by running the Unix “top” program and monitoring the size of the working set of the simulation. Only Simulator A was used in making footprint size measurements.

When full simulations were run, total footprints were measured.

3.2 Simulation Time

Simulation time was measured for each model on each simulator. The testbenches were all VHDL and used the same storage mechanism as the model being tested, that is either dynamic or static. In every case, compilation was done ahead of time and is not included in the results.

The simulations were run in batch mode and the graphical user interfaces were not invoked. The Unix “time” command was used to get the user and system CPU run times for each run while the “top” program was used to monitor memory usage.

All simulations were run with full SDF backannotated timing.

4.0 The Results

Here are the results of the tests.

4.1 Footprint Size of Bare Models

Table 1 shows a comparison of the computer memory consumed by each model. As can be seen from the data, for static

Table 1: Footprint Size

1Gb x8 model		
static allocation	VHDL	1089MB
	Verilog	1145MB
dynamic allocation	VHDL	25MB
	SystemVerilog	25MB
1Gb x16 model		
static allocation	VHDL	561MB
	Verilog	585MB
dynamic allocation	VHDL	25MB
	SystemVerilog	25MB

memory allocations VHDL had an advantage in memory footprint but, only by about 5%. This is because both the VHDL and Verilog models used the same memory representation method. The 16-bit wide model consumed about half the computer memory as the 8-bit wide model. This is because an integer was allocated for each memory location and the 16-bit model has half as many locations. Either model would show a significant advantage over a model in either language that used the traditional array of vectors or array of registers representation.

The dynamic allocation method produced the same footprint in either language and in either model. The footprint of the dynamic models will grow as data is written into them but, as long as only a small portion of the total capacity is used, there will be a size advantage over the static method. This advantage can allow a simulation to run on a given computer where the static model will not.

4.2 Performance and Footprints of Models and Testbenches

Table 2 shows a comparison of the computer memory consumed by each model and testbench along with the total CPU run time (user time plus system time) of each simulation. I was unable to get the SystemVerilog models to run on Simulator B.

Table 2: Full Simulation Footprints and Run Times

1Gb x8 model					
		Simulator A		Simulator B	
		footprint size	CPU run time	footprint size	CPU run time
static allocation	VHDL	1621MB	68 sec	1255MB	36 sec
	Verilog	2225MB	277 sec	N/A	N/A
dynamic allocation	VHDL	25MB	39 sec	203MB	100 sec
	SystemVerilog	25MB	210 sec	N/A	N/A
1Gb x16 model					
static allocation	VHDL	829MB	21 sec	727MB	19 sec
	Verilog	1137MB	94 sec	1208MB	710 sec
dynamic allocation	VHDL	25MB	16 sec	195MB	34 sec
	SystemVerilog	25MB	54 sec	N/A	N/A

This may be due to operator error on my part as I have not tried using it before for SystemVerilog. Also, Simulator B is a 32-bit simulator and as such, was unable to run the 1Gb x8 static Verilog model do to that model exceeding the simulator's capacity.

It can be seen from the numbers above that for static memory allocation, VHDL provided a mild advantage in simulation footprint size. It is not known why the size difference, 30% vs. 5%, was greater for full simulation than for the bare models. In all cases the testbenches were the same VHDL code. For Simulator B, the footprint size was enough to make the difference between being able to run the simulation or not.

For both static and dynamic allocations, VHDL provided a significant advantage in simulation performance.

5.0 Summary

When I began this experiment, I expected to see a large difference between VHDL and Verilog in footprint size of bare models but did not. I had forgotten that FMF was using similar modeling methods for both languages. However, when full simulations with backannotated timing were run, a larger difference of about 30% in memory footprint emerged. The cause of this difference remains to be discovered.

A much more significant difference was found between the VHDL and Verilog (or SystemVerilog) run times for full simulations. The VHDL simulation was 3 to 5 times faster than the equivalent Verilog simulation on Simulator A. The 37x difference in one case on Simulator B should be mostly attributed to the effects of swapping.

It would appear, even engineers using Verilog for the RTL design of FPGAs and ASICs might benefit from using a mixed language approach to board-level verification and verifying the interfaces to their chips.

These tests were conducted using behavioral models. Nothing in this experiment should be assumed to apply to RTL simulation.