HDL / C Interface Exploration

by

A report submitted in partial fulfillment of the requirements of

ICS212

2002

Introduction to Embedded Systems

Department of ICS
University Of California, Irvine
Contents

0. Acknowledgements 2
1. Goal 2
2. Background 2
3. Interface issues 2
4. Available interfaces 3
5. VHDL – CLI 3
6. Verilog – PLI 5
7. System Model 6
8. HW-SW partition 6
9. Datapath 7
10. FSM 8
11. Implementation in VSS 8
12. Implementation in VCS 9
13. Observations and results 10
14. Future work and points 11
15. Bibliography 12
0.0 Acknowledgements

The author wishes to express a deep sense of gratitude to Sumit Gupta, for helping install and run vcs on the runner machines. And Samar Abdi in helping me debugging code.

1.0 Goal

HDL / C interface exploration with VHDL - CLI and Verilog - PLI. Focussing on efficiency issues between the two interfaces using a FSM datapath system model as an example.

2.0 Background

What is a HDL / C interface?

Characteristically hardware are designed, simulated and tested using a hardware description language (HDL) like Verilog or VHDL and software designed, executed, and tested using high level languages like C / C++ and Java.

As systems get more and more complicated and faster a need for the following is felt:

1. Software portions need to be moved to hardware for better performance
2. Test software on not yet designed hardware platforms
3. Test partially translated software, to hardware, against its original software specification. (e.g. mpeg decoder)
4. Complete system level simulation.
5. Adding more functionality to the given simulator (e.g. checking activity on specific nets etc)

Such cases warrant the use of co – simulation techniques that model a portion of the design in C/C++ constructs and the rest in an HDL. Thus creating the HDL / C interface.

3.0 Interface issues

Language embodies methodology [1]. Thus HDLs are better suited towards discrete event capture, unbuffered “wire” communication and other structural and procedural styles, modeling hardware concurrency. While software languages are more sequential and provide concurrency through scheduler, semaphores, and multi-threading.
The basic differences observed during the modeling and simulation here, as detailed later in the report were two fold.

1. Different set of data types on both HDL and C side.
2. Incorporating correct triggering (execution) of the C code with the HDL during the co-simulation.
3. The absence of higher abstract datatypes on the HDL side like objects and classes

4.0 Available interfaces

The system hardware was modelled in both VHDL and Verilog. While the software was implemented in C.

The VHDL simulator used was VHDL Synopsys Simulator (VSS) and the verilog simulator was Verilog Compiler and Simulator (VCS).

VSS supports C interfacing through its “C language interface (CLI)” and VCS supports the same through the standard verilog “programming language interface (PLI)”. Both the interfaces are detailed in subsequent sections later in the report.

Both the VSS and VCS versions used were from the Synopsys 2000.06 – SIM1 package.

5.0 Synopsys VHDL – C language interface (CLI)

“The C-Language Interface (CLI) enables … to describe component models in C and interface them in VHDL source files for simulation on VSS” as stated in [2].

As per the VHDL design methodology each hardware component modelled has an entity which defines its ports that connect the component with the outside world and it also allows for passing any generic values that might be set at run time. This entity is linked to an architecture with the help of a configuration file. By modifying the configuration the entity could be linked to any other architecture. Where the architecture defines the internal functionality of the component.

Through CLI the functionality of the component can be specified in C and needs to include a header file <cli.h> kept in the “include” directory parallel to the “bin” directory. The architecture in this case is dummy and needs to contain five attributes and is constrained to be named CLI. Of which three define the C routines that specify the architecture while the fifth is used to specify the sensitivity on the input pins of the component.

The five attributes are:

attribute CLI_ELABORATE of CLI : architecture is “elaborate_routine_name”;
attribute CLI_EVALUATE of CLI : architecture is “evaluate_routine_name”;
attribute CLI_ERROR of CLI : architecture is “error_routine_name”;
attribute CLI_CLOSE of CLI : architecture is "close_routine_name";
attribute CLI_PIN of signal is sensitivity;

Where the routine names are the names of the corresponding C routines.

Elaborate routine:
The elaborate routine is used to set up the basic data structure for exchange of signals across the HDL/C interface. CLI implements structs in C that combine VHDL data types with some enumeration data types on the C side. This is done to solve the data type differences on either side of the boundary and to incorporate the missing physical data type of time. There are limitations to the interface data exchange as defined in [2]. The basic elaborate step is to define a structure in C which is similar to the VHDL entity.

Evaluate routine:
The evaluation routine carries the real functionality of the component and requires an instantiation of the struct defined above.

Error and Close routines:
The error routine provides exception handling and procedures for debugging. While the close routine provides procedures for database cleanup and memory restoration, post execution.

CLI data types and CLI functions:
Predefined CLI composite data types provide the missing link between data types on either side of the interface.

e.g.

<table>
<thead>
<tr>
<th>CLI type</th>
<th>C type</th>
<th>VHDL type</th>
<th>Usable C values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>BIT</td>
<td>BIT</td>
<td>BIT_0, BIT_1</td>
</tr>
<tr>
<td>Std_logic</td>
<td>STD_LOGIC</td>
<td>STD_U/LOGIC</td>
<td>STD_LOGIC_X STD_LOGIC_Z</td>
</tr>
<tr>
<td>Time</td>
<td>TIME</td>
<td>TIME</td>
<td></td>
</tr>
</tbody>
</table>

The CLI functions use CLI data types to provide the required functionality across the interface. The four most important functions are described below:

1. cliVALUE: provides value transfer across the interface with fields for cli data type, length, direction, value, units etc.
2. cliAFTER: equivalent to VHDL code : F<= v after 5 ns. The struct has fields for time value and the timebase. It is used in conjunction with the two cliScheduleOutput() and the cliScheduleWakeup() calls.
3. cliScheduleOutput(): used to schedule the output value across the interface to the HDL side. Uses both the above calls for value and time while scheduling outputs.
4. cliScheduleWakeup(): similar to the above but for non-event driven wakeups.
For a complete list see [2].

CLI limitations: There are a few limitations to the interface modelling in CLI but the most important ones impacting my system design in question was the lack of support for two or more dimensional arrays.

6.0 Verilog - C language interface

The Verilog Programming Language Interface (PLI) is a system of C routines that allows access to the Verilog simulation [4]. The routines interact with the internal data representation of the Verilog modules of the given design and extract information about the simulation environment. A major difference between the PLIs and the CLIs, discussed in the previous section, is that the PLI’s interface with the simulator engine and thus provide system tasks and functions which could be called from inside the design code. This allows more flexibility and a way towards simulator modification. Something not possible with the CLI.

PLI is an IEEE 1364 standard and thus supported by almost all Verilog simulators. It has two versions ver1.0 and 2.0. Where version 1.0 is more widespread and supported by majority of simulators, 2.0 is supported only by VerilogXL and NC-Verilog of Cadence Inc [5].

PLI version 1.0 consists of a set of access and utility routines that can be called from custom C routines. The basic extra feature in 2.0 is the support for Verilog Procedural Interface (VPI), which basically support the same features as the access and utility routines.

The access routines (starting with acc_) are used to access the information contained in the Verilog design while the utility routines (starting tf_) are used for more mundane house keeping functions.

The routines are specified in the include files <veriuser.h> (VerilogXL) and in <vcsuser.h> (VCS) [7]. In VCS a table file specifying the mapping between the defined function/task and its corresponding C code is specified as argument at the command line with the \(-P\) option. This creates the new Verilog simulator binary default named simv.

The most relevant access routines are:

- acc_fetch_value (object_handle, format_string): Returns a pointer to a character string indicating the logic or strength value of a net register or variable

- acc_set_value (object_handle, value_p, delay_p): Sets and propagates a value on a register or a sequential UDP (user defined primitive).

For a more detailed list routines check [8].
7.0 System model

The system defined for the exploration in question is a small incremental average computing system [9]. Here the average is computed online as new values keep coming in, instead of waiting for all and then providing the average at the end.

A system like this is usually employed in DSP and GPS systems where a few parameters need to be averaged on an online basis. The flowchart below captures the system assuming the stream of incoming numbers is provided by a memory.

![System flowchart](image-url)

Figure1: System flowchart

8.0 System modelled as a FSM - D

The above system can be implemented as a finite state machine controlling a datapath. To speedup a given software implementation of the given system, a choice is made to
implement the datapath in hardware. Also the number of data elements to be averaged over, though online was limited to N (=2,5,9).
Thus the hardware (HDL) implements:

- Datapath
- Memory

The decision to move memory to hardware was due to the CLI interface issues as described later.

The C portion contains:
- Finite State Machine controller

9.0 Datapath

The datapath for the system is shown in the diagram below. In the implementation the lowest structural models are the regs.x, compa.x, pe.x (x = vhd & v). These are instantiated in a bigger module, datapath.x. The datapath had problems during the initialisation portion on the start of the simulation. Remedies are discussed in the observations and constraints section.

A, I: Registers
PE: Processing Element
LA, LI: Load regs
S(I) : Memory data
Add: Memory address
Avg: Final average
Done: N values done
C: Clear
Memory not shown!

Figure 2 : Datapath
10.0 Finite state machine controller

The finite state machine of the controller for the datapath contains five states. In the first state the datapath is initialised, second reads the first data element from the memory by sending out a read (rd) signal to the memory and a value is computed in the processing element. In state two this computed value is stored in the two registers (regs) and a new value read from memory in the state three. If the number if inputs has reached the pre-specified amount then the in signal (done) goes high pushing the fsm to the final state (s4). The fsm is shown below. For the signals not specified in a specific state, its assumed to be tied to zero. And the triggering is done on the positive edge of clock (clk) with a period of 10ns.

![Finite state machine controller diagram](image)

11.0 Implementation in VSS

The vhdl files are divided as per the blocks shown and described in the datapath, along with an fsm and a memory function instantiated in a top level testbench.vhd. Before implementing the fsm in C it was first tried in VHDL.

Initialisation issues:
If a signal value or an input port is not initialised it blocks the processing element’s multiplication and gives an overflow/underflow error.

Memory issues:
The memory was originally moved to hardware as it was required to be a two dimensional array of bits. One dimension for the bit_vector and the other for addressing. As the CLI had a limitation of passing only one dimensional arrays across the interface, it constrained to keep the memory in hardware. But later to speed up the design process of the processing element (pe) the memory had to be moved to handle real data instead of bit vectors.

The average computation formula utilised the memory address and thus required to be of the same data type as the data and is thus modelled as real. This constrained to use only an if-then-else construct for the memory function.

CLI routines:
In the fsm block the error and close attributes and their corresponding routines where not implemented. This was left to the default settings. Only the clk and done signal of type bit move to the C side and all the datapath control signals move out of the C side along with the memory read signal (rd) of type bit.

Datapath:
To enable the average to be made available in the fifth state (s4) a new pin en was added. It goes high in s4 to enable the dataout on the average port.

The part of the compilation sequence is given below:

Assuming that all the other modules have been compiled before. For the cli first

cli –nc –add –cv fsmc fsm
cli –nc build
vhdlan –nc fsm.vhd
vhdlan –nc testbench.vhd
vhdlldb ctop

12.0 Implementation in VCS

First the implementation in VSS was explored and thus some of the decisions taken during that phase were carried forward in this phase as well. Thus the memory was kept in hardware and the data and address in real. It was later modified to the verilog memory representation.

VCS execution:
The VCS PLI requires the C compiler to be in its path and thus the below mentioned flags and environment variables had to be set.

setenv VCS_CC /opt/local/bin/gcc
vcs –CFLAGS “-I /opt/synopsys/vcs5.2/include”
+incdir+/opt/synopsys/vcs5.2/include/ -P mypli.tab
The functions defined also required the use of access routines not provided in the Open Verilog Initiative (OVI) but added in the vcs simulator. This made the implementation vcs simulator dependent. The access routines used were acc_fetch_tfarg, acc_clearmem_int, acc_handle_tfarg, acc_fetch_tfarg_int, acc_get_mem_int.

The functions were:
1. set_mem
2. get_mem
3. clear_mem

They were defined in the pli table. And the vcs compilation used the –Mupdate option.

vcs +acc+4 –Mupdate –P mypli.tab at_croutes.c testbench.v

The results are discussed in the following section.

13.0 Observations and results

Observations:

Both VCS and VSS have their pros and cons with the data exchange across the interface. VHDL requires a robust structure to be put in place first while verlog requires a handle to be instantiated on the data structure accessed.

The CPU time was logged in the PLI implementation and is as shown in the log below while cpu time in the vhdlssim could not be extracted.

The simulation time cannot be used to make the comparison as both the simulations could be made to run for the same time duration. Moreover the implementations in vhdl and veilog differ in the memory definition.

Although accessing and setting data is a lot more easier in the PLI than in CLI as the functions are better defined and do not require any intermediate data structures to be implemented.

Secondly, PLI functions and tasks are implemented and linked with the simulator they can be used across a larger variety of designs and can also be used for simulator modification which is not feasible with CLI.

Also the PLI being an IEEE standard allows porting PLI routines across simulators.

In both tweaking and debugging the C side is challenging.

Log portions:
Vhdlsim:

# run
75 NS
Assertion NOTE at 65 NS in design unit TOP(BEHAV) from process /TOP/....
“done = 1 and en = 1”
(vhdlsim): Simulation complete, time is 75 NS.
# quit

VCS(simv):

    V C S   S i m u l a t i o n   R e p o r t
Time: 1000
CPU Time: 0.040 seconds; Data structure size: 0.0Mb
Fri Mar 22 18:45:44 2002

14.0 Future work and points

- Better benchmarks to help compare the two simulators.
- Synthesis tool from Synopsys Design Analyzer does not support the HDL/C interfaced designs in its synthesis flow.
- I have not come across any other synthesis tool to support the same.
- C to VHDL translation can be achieved with Spark [10]
- A more seamless hw/sw interface is envisaged in SystemC towards complete system modelling.

---
15.0 Bibliography

[3] Chapter 7, Using the PLI, VCS user’s guide
[4] A Brief Introduction to PLI (Programming Language Interface), Celia Clause
   www.europa.com/~celiac/pli.htm
[10] Using Global Code Motions to Improve the Quality of Results for Higher-Level Synthesis, Sumit Gupta et al.
   http://www.angelfire.com/in/rajesh52/verilogvhdl.html
[14] DATE99: C-to-HDL translation tools emerge, EETimes