Simulating VHDL Models In An Executable C Specifications Environment

Prabhat Jain
AT&T Bell Laboratories
Allentown, PA 18103

Abstract

Most of the commercial VHDL simulators provide some form of interface to C functions and procedures in a VHDL simulation environment. But, there are no methods described to interface VHDL models in an executable C specifications environment. This paper describes an interface that provides transparent simulation of VHDL models like C functions from an executable C specifications environment. The new interface has been successfully used to simulate VHDL models with the executable C specifications of a large design.

1.0 Introduction

There are several situations where simulation of VHDL models with executable C specifications becomes necessary. For instance, there are cases in which the specification models are developed as executable C specifications which may be cycle or phase accurate simulators of a design. In particular, for embedded microprocessor or DSP applications, such simulators are very suitable for code development because they do not require any license for a VHDL simulator. In the framework of executable C specifications, there may be some blocks of a design which are synthesized (e.g., control sections of a design), so the simulation of VHDL models for the synthesizable blocks is necessary to ensure the equivalence of C functions and VHDL models for a block.

This paper describes an interface that provides transparent simulation of VHDL models like C functions from the executable C specifications environment. This new interface scheme uses the interprocess communication primitives, namely, shared memory and semaphores, for data transfer between the executable C specifications and VHDL simulation environment. In addition, this new interface uses the interface functions provided in the VHDL simulation environment to interface C functions. It also uses some techniques in the VHDL simulation environment to allow the transparent simulation of VHDL models like C functions. This interface has been implemented using the Vantage STYX VHDL-C interface and interprocess communication through shared memory and semaphores.

In addition to the advantages offered by VHDL, this interface offers several advantages in the simulation of executable C specifications with VHDL models. This interface can be used with any programming language as long as the interprocess communication primitives are used to communicate the data to the VHDL simulator. Once the interface is set up for simulation with the VHDL models of the desired blocks, changes to VHDL models require only compilation of the designs. Also, new VHDL models can be added for simulation easily. The features of the C simulation environment are not affected by this interface (e.g., user interrupt facility).

2.0 Components of the C-VHDL Interface

The criteria for the design of the C-VHDL interface were that the simulation of VHDL models with the C specifications be very transparent. Also, there should be minimal changes required to the C specifications program to allow simulation with the VHDL models. There are four components of this interface which achieve the control flow, synchronization, data communication, and type conversions functions. These components are listed below.

- The main C specifications program which handles the control flow from the C side and contains the C functions which transfer between the C program and the VHDL simulator.
- The shared memory for data transfer be-
between the C program and the VHDL simulator and semaphores for synchronization.

- The driver block which is part of the VHDL testbench to handle control flow and data transfer between the VHDL model and the C program.
- The VHDL model which represents the C function block to be simulated in the VHDL simulator environment.

Figure 1 shows the organization of the four components for simulation. The data flow is shown among various components. The C function writes the input parameters data to the shared memory and the driver block reads this data from the shared memory and transfers it to the VHDL model. When the simulation of the VHDL model is done, the output parameters data from the VHDL model is transferred to the driver block which then writes this data to the shared memory and the C function reads the output parameters data from the shared memory. The control flow aspects of this interface are discussed later.

2.1 Handling Multiple VHDL Models

The above description focussed on a single driver block and a single VHDL model on the VHDL simulator side. Figure 2 shows the organization of the four components for simulation in the case of multiple VHDL models on the VHDL simulator side. The data flow is similar to the case of a single VHDL model except that each C function writes the input parameters to the shared memory with a flag indicating the function call. The driver block uses the flag information to transfer the input parameters data to the corresponding VHDL model and receive the output parameters data from the same VHDL model. The other VHDL models are not affected because the changes in the input parameters are directed to only one VHDL model. So, one driver block handles all the data transfer from the C program to the VHDL models and vice versa.

3.0 Details of the C-VHDL Interface

This section describes the activities performed by various components in the C-VHDL interface in the simulation process. This section also gives the flavor of the control flow during the simulation process.

3.1 Main C Program

The C specifications program is modified to include the code which creates the semaphores and shared memory for synchronization and data communication between the C program and the VHDL simulator. This interface has been designed such that the C specifications program creates the semaphores and shared memory and then forks a process for the VHDL simulator with the VHDL design library and the VHDL testbench entity and architectures as parameters. The semaphores are initialized for proper communication between the C program and VHDL simulator. The C specifications program and
the VHDL simulator use the same keys to share the semaphore and shared memory.

There are two semaphores and two shared memory areas — one for sending data from C program to VHDL simulator and the other for receiving data from VHDL simulator in the C program. At the end of the simulation, the semaphores and shared memory are freed. One set of semaphore and shared memory is freed in the C specifications program and the other is freed in the VHDL interface code.

3.2 C Functions

On the C specifications side, a function which is to be simulated by a VHDL model is replaced by a new C function which takes the same input parameters as the original C function. This new C function constructs the set of inputs parameters to be sent to the VHDL simulator. It sends the input parameters through the shared memory. The data is transferred after the new C function synchronizes with a function on the VHDL side. Then, it synchronizes for the new data values for the outputs parameters to become available. Then, it receives the output parameters and distributes them to appropriate structures or variables. All the input parameters sent from the new C functions are sent with a flag last_call set to 0.

When the main C program is done with the simulation, it sends the input parameters with the flag last_call set to 1. At this time, the input parameters may not have any useful values. The last transfer of input parameters indicates to the VHDL side synchronizing function that the simulation is over and it can free the semaphore and shared memory and finish the VHDL simulator process. If the C program can exit from several places, then either the exit function call can be replaced by a call to a function which sends the input parameters with last_call set to 1 and frees the shared memory and semaphores and then exits the program or another program can be executed after the C program exits which does the same function. These two programs can be put in a script which executes them in sequence.

3.3 VHDL Testbench

On the VHDL simulator side, the simulation of the VHDL models is done through a testbench which embodies a driver block and one or more VHDL
models. The VHDL models represent the blocks of the C specification program which are to be simulated in the VHDL environment. The VHDL testbench instantiates the driver block and the desired VHDL models. The outputs of the driver block are connected to the inputs of the VHDL models and the outputs of the VHDL models are connected to the inputs of the driver block. If the type conversions are handled at the testbench level, then the necessary VHDL code for type conversion can be added between the ports of the driver block and the ports of the VHDL models. More details of how multiple VHDL models are handled are discussed earlier in the paper.

The VHDL models need no modification to be simulated in the VHDL testbench with the driver block. Some of the input ports of VHDL models and/or output ports of the driver block may need to be initialized to some default values. This can either be done in the VHDL testbench or in the initialization code for the driver block.

The driver block handles all the synchronization and data communication with the C specifications program. It also drives the input ports of the VHDL models and collects the output port values and transfers them to the C program. The architecture of the driver block is written using the C interface functions provided by the VHDL simulator. There are several parts of the driver block which are discussed below.

3.4 The Driver Block
The driver block consists of a collection of functions which are called at different times by the VHDL simulator during the simulation process. The driver block has a data structure which stores the pointers to the input and output ports of the driver block. The functions which comprise the driver block are described below.

The `create_driver` function is called at the elaboration time to elaborate the instance of the driver. This function initializes the instance of the driver block and gets the pointers for the input and output ports of the driver block and stores them in a data structure. Then, it creates a process for the driver. It creates the sources for the output ports of the driver block. The use of these sources would be described later. It then initializes the values of the output ports of the driver block to default values. This also registers the functions to be called at the beginning of simulation and at the end of each cycle.

The function `start_fun` is called at the beginning of simulation. This is the point after the elaboration process, but before the VHDL simulator has started executing any of the processes. This function uses predetermined keys to acquire the semaphore and shared memory identifiers and initializes a semaphore for synchronization.

The main function of the driver block is the `sync_fun` which handles the all the data transfer from the C program to VHDL simulator and vice versa. This function is called by the VHDL simulator after all the processes have been initialized at the start of simulation. Subsequently, this function is called at the end of every simulation cycle. This is at the point where all the processes at the given time have been executed and the simulator is ready.
sync_fun() {
    if (sim_done)
        return;
    if (send_flag) {
        get_next_event();
        if (!end_of_sim)
            return;
        get_driver_input_ports_value();
        send_output_parameters(...);
    }
    set send flag;
    receive_input_parameters(...);
    if (last_call) {
        set_sim_done;
        free_sem_shm;
        return;
    }
    toggle_dummy_sig_value;
    convert_types_input_parameters;
    schedule_input_parameter_events();
}

Figure 4. Function for synchronization and data communication

to advance to the next time at which an event will occur. The activities performed by this function are described below.

The function sync_fun starts with the initial values of the flags sim_done and send_flag as 0. If the flag sim_done is set, then sync_fun returns without changing any values of the flags or the input ports of the VHDL models. At this time, the VHDL simulator is done with the simulation and it can exit the simulation.

The flag send_flag is 0 when sync_fun is called the first time and it is set to 1 in subsequent calls. So, the first time sync_fun is called, it skips the operations which detect the simulation completion and the operations to send the output parameters to the C program. The flag send_flag is set to 1 in all the calls of sync_fun.

If the send_flag is set then the sync_fun gets the next event from the event queue. The flag end_of_sim is set if the next event is an end of simulation event. If end_of_sim is not set then there are still some other events left in the event queue corresponding to the C function call, so sync_fun returns. Essentially, this causes the VHDL simulator to process all the events generated as a result of scheduling the events for the new values of the input parameters received from the C function. If the end_of_sim is set, then it indicates that all the events have been processed and the values of the output ports of the appropriate VHDL model are ready to be transferred to the C program. Then sync_fun gets the new values of the output ports of the appropriate VHDL model, synchronizes with the C function and sends the new values through the shared memory.

The approach of using a function (sync_fun) which is called at the end of every simulation cycle was chosen over the approach of using a function which is called at the end of simulation because in the latter case if the end of simulation event has already occurred (as opposed to next event) the VHDL simulator may not schedule any new events.

Then, the sync_fun synchronizes with the a C function of the C specification program and receives the input parameters from the shared memory. It checks if the last_call is set or not. If last_call is set, then it sets the sim_done flag and frees one set of semaphores and shared memory and returns. If last_call is not set, then it proceeds to process the new values of the input parameters.

In the VHDL testbench, there is a dummy signal sig_dummy which is an output port of the driver block and is not connected to any of the VHDL models. The purpose of this dummy signal is to keep the VHDL simulator active to accept new input param-
The value `sig_dummy` is initialized to 0 in the `create_driver` function at the elaboration time. The value of `sig_dummy` is toggled on each call of the `sync_func` in which new input parameters are received from the C program.

If the type conversion of input parameters is handled in the driver block interface code, then the `sync_func` converts the types of the input parameters to make them compatible with the types of the VHDL models. It then schedules the events for the input parameters of the appropriate VHDL model by inserting the new values of the input parameters in the event queue.

The `driver_process` function, which was declared in the `create_driver` function, is an empty function and it is not sensitized to any input port changes.

4.0 C-VHDL Interface Issues

There are several issues which also play a role in the operation of the C-VHDL interface.

The efficiency of the interface can be improved, if the copying of information is reduced by the use of the shared memory directly by the C program to store its input and output parameter variables for the functions which are to be replaced by VHDL models. Also, some improvements in efficiency can be achieved by scheduling only the changes in the input parameter values. In the case of multiple VHDL models being handled by the same driver block, the data from the VHDL simulation is copied only for the VHDL model which is being called because the signals corresponding to other VHDL models would not have changed.

The local signals of VHDL models can be observed in the C simulation environment by either defining the internal signals of the VHDL model as additional outputs or by using the functions provided in the C interface of the VHDL simulator to probe internal signals.

The VHDL simulator needs to be compiled with the functions to access semaphore and shared memory so that these functions can be used in the interface code of the driver block.

Some input ports of the VHDL models need to be initialized to default values before the start of simulation so that simulation of VHDL models starts properly.

The VHDL simulator is run in the batch mode. The interface has also been used with the interactive version of the VHDL simulator, but the batch mode version of the simulator is very convenient and transparent.

There can be a hierarchy of VHDL models representing a C function. The top level entity can be connected to the driver block.

The debugging features on the VHDL simulator side are limited because of the VHDL simulation environment.

5.0 Summary of Results and Conclusion

The new interface has been successfully used to simulate VHDL models with the executable C specifications of a large design. It has proved very helpful in the debugging of VHDL models for several blocks by allowing a smaller debugging cycle. It has also proved useful in the process of establishing equivalence of C and VHDL models of the same block. This interface can also be used in the gradual transition from executable C specifications to a full VHDL simulation environment because it allows simulation of the VHDL models, as they are developed, with the rest of the C code in a very transparent manner.

There are other ways to interface the executable C specification code to VHDL code. We have tried some other ways of interfacing the VHDL code, but we have found the above method very appropriate for our environment.

The simulation results with the VHDL models of a large design are very encouraging. Even though the simulation speed slows down with the VHDL models compared to the full C simulator, the interface is very useful in many ways.

Acknowledgments

I would like to thank John Fernando for trying out the C-VHDL interface in a large design, Ken Fitch,
Frank Lemmon, and Trevor Little for their help during the development phase of the C-VHDL interface. I would also like to thank Jalil Fadavi for his encouragement to write this paper and suggestions to improve the paper.

References


