IP/SOC 2005

Session: High Level Modeling

Reusing Verilog IP Cores in SystemC Environment by V2SC

Abstract:

SystemC 2.1 supports all hardware concepts introduced by HDLs such as Verilog and VHDL. V2SC proposes a methodology for automatic conversion of Verilog 2001 constructs into SystemC 2.1 language. This enables SystemC community to import pre-designed (or machine-generated) Verilog IP cores into SystemC development environment. The reuse methodology based on V2SC can dramatically reduce the system development time. As an additional advantage, free SystemC reference simulators can be employed. V2SC offers a large Verilog subset coverage including Verilog test-benches. Innovative solutions have been proposed in this methodology to adopt SystemC 2.1 as a target for conversion of pre-designed Verilog IPs.

1. Introduction

Regarding SystemC as the future choice for design language, existing Verilog IPs need to be translated into SystemC. The methodology discussed in this paper tries to fill in the gap between Verilog and SystemC environments. This way SystemC users are enabled to integrate Verilog cores into their designs and Verilog users are enabled to port their designs into SystemC as another option for IP core customers as well. In summary this conversion methodology makes a very useful and practical extension to any Verilog design environment along with an essential import capability to facilitate IP reuse in any SystemC design environment where simulation is fast and free. Some efforts have been done previous to this work in porting HDLs to C/C++ platform [1-3]. VTOC from Tension [3] is a commercial tool that converts Verilog to C++/SystemC. VTOC converts Verilog RTL to C++/SystemC by interleaving the processes in order to achieve fast simulation speed. V2SC approach is to keep design hierarchy untouched during translation and to support Verilog subset as far as possible, including test-benches. In other words V2SC introduces a transmission from Verilog design environment to SystemC design environment. V2SC is based on our previous work [4] that was our primary efforts in this era.

2. Basic Building Blocks

Verilog most important building blocks are module, always and initial blocks, continuous assignment and gate or module instantiation.

Module is the container that includes other concurrent building blocks. Each module is mapped to its counterpart, *SC_MODULE* in SystemC. *SC_MODULE* is a macro that implements a corresponding SystemC library container class where processes or object can be implemented.

Initial and always blocks and continuous assignments are mapped into SystemC processes. Two kinds of SystemC processes mostly match to these concurrent processes: *SC_METHOD* and *SC_THREAD*. *SC_THREAD* with its own individual thread and stack and local variables is slower than *SC_METHOD* but supports wait statement which is required for delay and dynamic event implementation. This makes use of *SC_THREAD* inevitable for processes including these language constructs. But as long as there is no delay or dynamic event, *SC_METHOD* is the best choice that makes a dramatic enhancement in simulation time.

All the gates that are instantiated in Verilog are mapped into corresponding processes including equivalent signal assignments that assign a logical combination of inputs to output.

Finally, when a module is instantiated in Verilog, in SystemC an object of the corresponding *SC_MODULE* class must be created and then port-mapped accordingly.

3. Signals, Ports and Variables

3.1. Two-Value versus Four-Value Data Types

In Verilog all types of signals, variables and ports are inherently four-value logic containing resolution function. In SystemC *sc_logic* and *sc_lv* types are perfect matches, but most of the time two-value logic sufficiently holds because most parts of circuits are designed with two-value logic. Considering that two-value logic makes simulation faster and more efficient, this methodology uses two-value logic as long as the functionality is not violated. Our methodology contains a two-four value logic recognition algorithm that decides which signals or variables must be declared as four-value and which as two-value ones.
This algorithm recognizes left values assigned by a "Z" either directly or indirectly as four-value logic. Also inout ports are considered as four-value logic signals in this algorithm.
There is a pair of two-value logic data types in SystemC (*bool*, *sc_uint or sc_int*). *bool* is used for single-bit data types where *sc_int* and *sc_uint* are used for signed and unsigned vector data types respectively. Four-value logic signals and variables use *sc_logic* and *sc_lv*.
Table 1 shows coppesponding Verilog and SystemC data types.

Table 1. Scalar and vector types.

Verilog	**SystemC**	
	Two-Value	**Four-Value**
Single bit	Bool	sc_logic
Unsigned vector	sc_uint, sc_biguint	sc_lv
Signed vector	sc_int, sc_bigint	sc_lv

3.2. Verilog Registers in SystemC

Verilog signals(wire) and ports(input, output, inout) have straight forward SystemC equivalent. But an object declared in Verilog with *reg* keyword may be used in the body as a variable or a signal. In order to determine this, the whole Verilog body must be passed once to observe how this object has been treated, i.e., as a variable or as a signal.
V2SC does this task through a built in algorithm. This algorithm sets Verilog *reg* declaration type as signal in following cases:

1. *Reg* variable is used as an event control of an always or an initial block.
2. *Reg* variable is used as a left-value of a non-blocking assignment.
3. *Reg* variable is used in right hand side expression of a continuous assignment.
4. *Reg* variable is used for port binding of a gate/module instantiation.

Table 2 shows Verilog and SystemC signals, ports and variables.

3.3. Other Verilog types in SystemC

There are different types in Verilog such as real, integer, time and so on. The way they are mapped into SystemC is depicted in table 3. Note that type integer in Verilog may be used as a signal or as a variable as well and needs to be cleared out. The way an integer is recognized to be a signal or a variable is the same as the way described for registers.

Table 3. Converted Verilog Types.

Verilog	**SystemC**
event	sc_event
real	double
realtime	double
time	sc_time
integer	int / sc_int<32>

4. Mapping Summary

Verilog is a very open language that makes world easy for users and hard for tool developers. Supporting some of Verilog constructs requires extra knowledge about that specific object that must be collected from other parts of circuit description. V2SC covers a large subset of Verilog constructs.
Wait statement that exists in SystemC 2.1, make dynamic events and delay implementation possible.
Table 4. shows a brief report of covered Verilog language aspects and their match in SystemC.

Table 2. Converted Verilog Variables, Signals and Ports.

Verilog	**SystemC**	
	Two-Value	**Four-Value**
Variables (reg)	Two-value types(Table 1)	Four-value types(Table 1)
Signals (wire/reg)	sc_signal	sc_signal_resolved sc_signal_rv
Ports (input,output,inout)	sc_in,sc_out, sc_inout	sc_in_resolved/sc_in_rv sc_out_resolved/sc_out_rv sc_inout_resolved/sc_inout_rv

Table 4. Mapping Summary.

	Verilog Language Construct	SystemC Target Construct
Building Blocks	Module Declaration	SC_MODULE
	Verilog Process including 1. Continuous assignment 2. Always 3. Initial	SC_METHOD / SC_THREAD
	UDP	SC_MODULE
	Module instantiation	An object of module class is instantiated
	Gate instantiation	SC_METHOD (macro based) / SC_THREAD(macro based)
	Task and Function	C++ class function declaration
	Parameter	C++ template class
	Localparam	C++ static const declaration
Control Flow	Conditional statement	C++ if-then-else
	Case statement	C++ switch statement
	CaseX/CaseZ statement	C++ while statements along with masks to cover intended cases
	For, while, repeat, forever loops	C++ for-while loops
	Disable statement	C++ goto statement
Delay and Event	Static event	Appropriate signal in sensitivity list
	Dynamic event	SystemC wait construct
	Wait statement	SystemC wait construct
	Delay	SystemC wait construct

5. Main Features

Although Verilog is inherently C-like in its nature, there isn't an exact match for each Verilog construct in SystemC and there is no straight forward way from Verilog into SystemC. Several conversion techniques have been employed in order to keep functionality intact and make SystemC output behave the same as Verilog input. Some of the techniques are listed below.

5.1. Signal Handler

SystemC signals and ports lack bit-select and range-select operators. When a single bit or a part of a multi-bit signal/port needs to be written, temporal variable is declared and used instead and then actual signal/port is updated. This case happens when such signals are used in port mapping of a gate or module instantiation too. At the time of port mapping, if expressional signals (e.g., a&b) are used, the same procedure must be done. The updating task is done in a separate process called *signal_handler*. *Signal_handler* is a simple and light-weighted built-in updater engine sensitive to all involved signals. Figure 1 illustrates this more clearly.

Verilog

```
module m1(input a, input b, output
out);
    sub1 X(out,a&b);
endmodule
```

SystemC

```
SC_MODULE(m1)
{
    sc_in<bool> a;
    sc_in<bool> b;
    sc_out<bool> out;

    sub1 X;

    sc_signal<bool>  a_and_b;
    void v2sc_signal_handler() {
        a_and_b = a.read() & b.read();
    }

    SC_CTOR(m1) : X ("X") {
       X (out, a_and_b);

       SC_METHOD(v2sc_signal_handler);
       sensitive << a << b;
    }
}
```

Figure1. Signal Handler Engine.

5.2. Process Merging

Lack of bit/part select of SystemC signals leads to declaring temporary variables for them. These temporary variables do not follow the semantic of a signal and in some cases cause problem in the functionality of the generated SystemC code. For example in the following code the two separated continues assignment with same left hand side should be merged in SystemC generated code to achieve correct SystemC functionality.

Verilog

```
assign  a[0] = exp1;
assign  a[1] = exp2;
```

SystemC

```
void assign_process_complex()
{
    sc_uint<2> a_tmp(a.read());
    a_tmp[0] = exp1;
    a_tmp[1] = exp2;
    a = a_tmp;
}
```

Figure 2. Process Merging Example.

V2SC process merging algorithm recognizes these cases and merges the processes for correct functionality.

5.2. CaseX Handling

V2SC uses built-in masks in order to support Verilog casex construct. V2SC is equipped with an algorithm that generates a mask based on the value of case expression. Case item is ANDed with this mask and the value out of this operation is compared with another mask that is generated by replacing case expression 'x' values by '0'.

Verilog

```
casex(s)
   3'b00x:  x = a;
   3'b1x1:  x = b;
   default: x = c;
endcase
```

SystemC

```
do{
   int value = s.read();
   if( value & 6 == 0 ) {
       x = a.read(); break;}
   if( value & 5 == 5 ) {
       x = b.read(); break;}
   // default
   { x = c.read(); break;}
} while(0);
```

Figure 3. CaseX example.

5.3. Blocking and Non-blocking

V2SC decides whether a Verilog *reg* is used as a signal or as a variable and the corresponding type is declared and used. Therefore when there is no dealy the semantic of blocking and non-blocking assignments are handled automatically. But in presence of a delay, the flow of program differs in blocking and non-blocking assignments.
Using delayed blocking and non-blocking assignment combined in an always block is not common and not supported, but this combination may be used in an initial block for implementing a test-bench that is supported as described in 5.4.

5.4. Test-Benches

Generally a Verilog test-bench is an initial block where there is a combination of delays and a series of assignments to the inputs of the unit under test along with a clock-generator always block that both of them are neatly converted into SystemC. Inside the initial block, a blocking assignment behaves as a normal sequential assignment; it blocks program flow for the amount of delay if there is any, does its task and then transfers the program flow to the next statement. But a non-blocking assignment, as its name suggests, doesn't block the program flow and introduces concurrency into sequential blocks. If there is no delay, the semantic of a blocking assignment is similar to assigning to a variable and the semantic of a non-blocking assignment is equivalent to assigning a signal in SystemC. When there is delayed non-blocking assignment, in order to keep the concurrency it suggest, individual separate processes are generated for each one.

Verilog

```
Initial
begin
   a <= 0;
   b <= 0;
   c <= 0;
   a <= #5 4;
   b <= #8 8;
   c <= #10 1;
end
```

SystemC

```
Void process_a()
{
   a = 0;
   wait(5,SC_NS);
   a = 4;
}

void process_b()
{
   b = 0;
   wait(8,SC_NS);
   b = 8;
}

void process_c()
{
   c = 0;
   wait(10,SC_NS);
   c = 1;
}
```

Figure 5. Non-Blocking assignment in test-benches.

6. Not-Supported Constructs

Not-supported Verilog constructs resides in one of the following categories:

1. Procedural continuous assignment (assign/deassign/force/release)
2. Verilog switch level
3. Specify block
4. Fork/Join constructs in design (fork/join used in test-benches are supported)
5. Four-Value logic operators
6. Verilog 2001 generate statement and elaboration time constructs
7. Some system tasks and functions

Most of above items do not have an equivalent in SystemC and some of them are rarely used in real designs.

7. Experimental Results

In order to examine the V2SC coverage of Verilog constructs, the examples of three well-known books have been selected for test. The following table shows the results.

Table 5. Coverage summary.

Book	**Coverage Percent**
Verilog Digital System Design (by Z. Navabi) [5]	182/243 = 75%
Verilog Quickstart (by J. Lee) [6]	100/131 = 76%
Verilog HDL: A Guide to Digital Design and Synthesis (by S. Palnitkar) [7]	74/88 = 84%

This table shows that V2SC supports about 80% of Verilog constructs.

We selected Synopsys Designware Foundation [8] for testing V2SC conversion skill in the state-of-the-art Verilog IPs. V2SC converts about 90% of this library successfully into SystemC. And finally we tested PicoJava from Sun Microsystems for verifying V2SC in a real fabricated design. V2SC converted all PicoJava [9] codes to SystemC 100% successfully, because whole of this chip is designed at RTL level.

7. Conclusion

The converting methodology just discussed makes re-use of pre-designed Verilog IP cores possible. Along with that they can be simulated in a fast and a free simulation environment that generates executable outputs. In this paper, in pursue of reusing previously designed modules, we have introduced a conversion methodology from Verilog 2001 into SystemC 2.1. The main features of our converter were explained. V2SC provides a homogeneous platform and succeeds in alleviating the load of efforts done in a design flow by converting pre-designed Verilog circuits into SystemC.

Reference:

[1] VHDL to SystemC Compiler, http://www.prosilog.com

[2] VHDLtoSystemC, http://www.tni-valiosys.com

[3] VTOC, http://www.tenison.com

[4] L. Mahmoudi Ayough, A. Haj Abutalebi, O. F. Nadjarbashi and S. Hessabi, "Verilog2SC: A Methodology for Converting Verilog HDL to SystemC," *Proc. of the 11th International HDL Conference (HDL Con 2002)*, pp. 211-217, San Jose, California, USA, March 2002.

[5] Z. Navabi, *Verilog Digital System Design*, McGraw-Hill, 2003.

[6] J. Lee, *Verilog Quickstart*, Kluwer Academic Publishers, 2001

[7] S. Palnitkar, *Verilog HDL: A Guide to Digital Design and Synthesis*, Sunsoft, 1996

[8] Designware Foundation, http://www.synopsis.com/products/designware/

[9] PicoJava Core, http://www.sun.com/microelectronics/picoJava/