Modeling a Highly Generic Processing Unit Using SystemVerilog

Authors: Wolfgang Ecker, Volkan Esen, Peter Jensen, Lars Schönberg, Thomas Steininger

Abstract

The past decades of system design have shown that generic modeling is an efficient instrument for reducing the overall design time and allows IP-reuse more easily.

In this paper we describe the usage of the lately evolved hardware description and verification language (HDVL) SystemVerilog with regard to implementing a highly generic processing unit.

Our analysis shows that SystemVerilog allows writing generic designs in a more convenient manner. However it is shown that some of the useful features are still not supported by the current stage of EDA tool development. Nevertheless workarounds for the unsupported features are presented.

1. Introduction

Generic modeling has successfully made its way through the past decades of system design and has proven to be an efficient instrument for reducing the overall design time. By implementing designs in a generic way the reusability of components is increased and thus IP-based design is facilitated. Both VHDL87/93 and VERILOG95/2001 incorporate features which allow to model generic units.

In this paper we describe the usage of the recently evolved hardware description and verification language (HDVL) SystemVerilog [1] with regard to describing a highly generic multi register processing unit. SystemVerilog is a super set of the VERILOG language as well as VHDL. It is intended to replace both VERILOG and VHDL in the future and to unify design and verification languages as well.

After discussing related topics, we introduce some example features of our implementation while keeping focus on the generic elements. Subsequent to that we discuss some key features of the design part of SystemVerilog, like e.g. `packed unions` and `interfaces`. Benefits and difficulties are presented, which occur when these features are employed in our design. New ways of modeling common units, as e.g. an instruction decoding unit, are introduced.

Our analysis shows that SystemVerilog enables the designer to write more clear and concise code on one hand and on the other that some minor difficulties occur compared to e.g. VHDL, which still could be eliminated by further improvement of SystemVerilog tool support.

2. Related Work

Reusable modeling strategies have focused on making models as generic as possible [2]. To overcome language barriers and to increase flexibility in modeling generic aspects, generator and preprocessor approaches [3] have been proposed. Although they have been successfully employed for nearly a decade, generator and preprocessor techniques suffer from a common disadvantage: everything that might be used later on must be modeled upfront.

Starting at a higher level of description [4][5][6] reduces coding effort and automatically handles things like clock and reset signals, but does not completely alleviate the problems. A similar approach in generating instruction set architectures was introduced with the LISA concept [7].

A completely new style of hardware description called “rule based design” is introduced by Bluespec Inc. Rule based design allows to produce easily modifiable code, since large parts of the necessary control logic are either generated automatically (multiplexers, enable signals for registers) or quite simple to infer (FSMs, pipeline structures). The underlying

HDL “Bluespec SystemVerilog” [8] is based on VERILOG / SystemVerilog.

3. Specification Overview

The CPU is implemented as a Harvard architecture and uses the pipeline structure of the DLX processor [9].

It was desired that the processor could be easily changed in size by the use of several parameters which are specified by the user.

One of the first questions was to decide which of these parameters are independent from each other, which are not and in case of the latter in which boundaries they are to be chosen.

Our CPU uses five parameters which are (mostly) independent from each other and three which are derived from those:

The independent parameters are as follows:

- Width of data memory cells
- Available address space for data memory
- Available address space for code memory
- Available address space for register file
- Length of immediate values used by some instructions

The derived parameters are as follows:

- Width of code memory cells
- Width of registers in register file
- Maximum number of bits shifted by shift instructions

The length of the various operands used by the different instructions depends on the value of one or more of the independent parameters. Therefore the length of an instruction word is not fixed and even differs between the instruction classes. Thus, the maximum instruction length (which equals the width of a code memory cell) has to be computed.

In a similar way the width of the internal registers depends on the data width and the data address width, since a register must be able to hold both data words and addresses for the data memory.

The last dependent parameter is the maximum number of bits that can be shifted using shift / rotate instructions. This multiple-shift operation was mainly added for the case of data width greater than code width. In this case constants (which are stored in the code memory) have to be distributed among several memory cells. In order to reassemble them for computation, a sequence of load and shift operations is needed. Hereby a special load operation is used which only overwrites the lowermost n bits of a register and doesn’t alter the remaining bits (in this case n equals the code width). For this it is required that the processor is able to shift a complete instruction word.

In order to keep the design scalable, it is essential to avoid any fixed values. Thus, all signal declarations, etc. refer to those parameters.

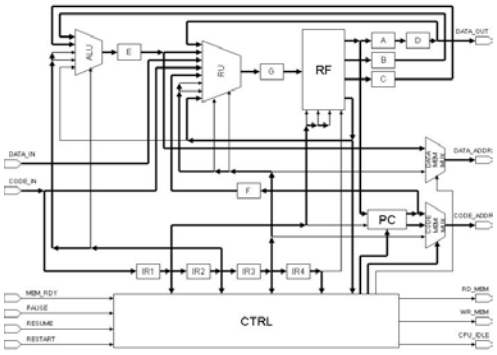

Figure 1: Block diagram of the CPU

4. Implementation

One feature that was introduced first by SystemVerilog is the use of type parameters, i.e. a type is passed rather than a value. This allows the declaration of signals with generic types. One advantage is the ease of switching between the usual four-state-logic used by VERILOG and respectively the newly

introduced two-state-logic, which is used in VHDL. When this paper was written though, the software tools we used to simulate/synthesize our design (regardless of the vendors) did not yet support the type passing feature.

Our generic concept had a strong impact on two components of the CPU:

The register file (RF) contains a variable number of data registers, which is determined by the parameter `register_address_width`. The width of those registers is equal to the greater of the `data_width` and `data_address_width` parameters. In addition to the data registers and a small register for several status flags, the RF includes one demultiplexer unit for a single write access and three multiplexers for simultaneous read accesses. These components are required to work with a varying number of input and respectively output ports. Both VHDL and VERILOG do not provide a feature for accomplishing this. SystemVerilog solves this problem by allowing the use of multi-dimensional arrays as ports as described later on.

The second component is only necessary due to the required generic nature of the CPU. Since multiple data formats have to be handled (content and addresses of code memory, content and addresses of data memory, content of internal registers, etc.) several conversion functions are needed in order to ensure the correct processing of data. In order to make the design structure as transparent as possible, all of those conversions are included in the so called 'Resizing Unit' (RU). This unit is basically an enhanced multiplexer which performs truncation and sign / zero extension on the input data words in order to cope with all possible data formats.

In this context we encountered another problem due to a lack of tool-support for new SystemVerilog features. Since the relationship between the input data width and output data width have to be evaluated, several comparisons between the corresponding parameters are performed. Depending on the result, the conversion uses different index values in order to access the necessary bits. The values in an unused branch might be illegal for obvious reasons:

```
bit [a-1:0] input;
bit [b-1:0] output;
if (a >= b)
  output = input[b-1:0];
else
  output[a-1:b] = 'b0;
  output[b-1:0] = input;
end
```

Some tools evaluate IF and CASE clauses only during runtime without taking into account whether the values being checked are already known (as in case of a parameter comparison). As a direct consequence both branches of the IF clause are checked for index errors, which causes the compiler to exit with an error message.

The only way we could avoid this behavior was to manually include macro definitions in the code which in turn contradicts some benefits of parameterized designs.

5. New language features in SystemVerilog

SystemVerilog offers several new features which are very useful for the development of generic / scalable designs.

There are several improvements in the way modules can be connected to each other:

Even though it is not possible in SystemVerilog to design a module with a variable number of ports, it is possible to use multi-dimensional arrays as ports (the number of elements of each dimension can be parameterized) which allows one to emulate the desired effect:

```
module mux
  #(parameter byte unsigned
      sel_width = 2,
      data_width = 8)
   (input bit [2**sel_width-1:0]
      [data_width-1:0] data_in,
    input bit [sel_width-1:0]
      sel,
    output bit [data_width-1:0]
      data_out);

always_comb
begin
```

```
  data_out = data_in[sel];
end
endmodule
```

VERILOG and VHDL require the user to manually connect all of the signals shared between higher level module and instantiated component (mapping by either name or position). SystemVerilog enhances the 'mapping by name' principle with the '.*' operator that automatically connects all signals of the component not explicitly specified to signals on the higher level that have the same name.

Another improvement is the newly introduced interface concept. Interfaces allow the declaration of so called 'modports', basically a combination of several ports and their direction. E.g. you could declare an interface for a simple handshake protocol consisting of a request and an acknowledge signal. Afterwards you declare a modport A in which REQ is an input port and ACK an output as well as a modport B with reversed directions. Later, instead of being required to connect the wires manually you just connect your module to the corresponding modport.

In addition, interfaces allow the implementation of almost all functionality that can be used in a module, namely the use of tasks and functions. By the inclusion of tasks in modports it is possible to specify which connected module has access to one task or the other. E.g. an interface could provide the tasks 'send' and 'receive' to all connected modules. This allows designing all modules completely independent of a specific communication protocol, which is only used in the interface. By using several interfaces that implement different protocols the interface concept is very valuable in terms of IP-reuse.

Another new feature of SystemVerilog is the introduction of a much more sophisticated type system. In contrast to basic VERILOG which only provides one type (a four-value type) and one-dimensional arrays with elements of this type, SystemVerilog adopted many of the concepts already used in VHDL and C++.

In addition to the introduction of multi-dimensional arrays and a two-value data type, it is now possible to use composed data types such as structs and unions.

We expected unions to be especially interesting for our implementation since by using unions we could easily implement our different instruction classes in one structure, which in turn eliminates the need for an explicit instruction decoder. Due to tool related problems we were not able to follow this approach without restrictions.

Yet the use of unpacked unions is not supported by any of the available tools while packed unions come with a small but very inconvenient requirement – all elements of a packed union must have the same length. Since all instruction classes of our design have a different length (depending of several parameters) we needed to add a 'dummy' field to fill the missing bits in case of an instruction too short. For the longest element the dummy field normally is of length zero:

```
logic[maxLength-currentLength-1:0]
```

In VHDL a bit vector defined as (-1 downto 0) will not be implemented. Unfortunately VERILOG does not support manual declaration of vector directions, which results in case of the example above in a bit vector of length two. Thus VERILOG does not allow the declaration of a bit vector of length zero. This restriction applies to SystemVerilog as well since it is completely backwards compatible. As a result, we were required to enlarge the maximum instruction length by one bit, which has no functional use for the design.

If the elements of a packed union are not of the same size, the size correction could be done by the tools instead of the user. The problem could be solved by either modifying the LRM accordingly (i.e., drop the equal length requirement) or by extending tool functionality to cover automatic bit stuffing.

6. Language comparison

Feature	**VHDL**	**VERILOG**	**SV**
multiple component / module instantiation	√	√	√
value parameters	√	√	√

type parameters	X	X	√
preprocessing	X	√	√
interconnects	*	X	+

√: language provides this feature

X: language does not provide this feature

*: VHDL allows the use of unconstrained bit_vectors in a port list. This feature is not available in SystemVerilog.

+: SystemVerilog introduces several new features not present in either VHDL or VERILOG. They include:

- use of multi-dimensional arrays as ports
- more powerful port mapping by use of the '.*' operator
- very powerful interface concept that even allows the use of functions and tasks [10][11]. By using interfaces it is very simple to implement programmable protocols.

7. Conclusion & Outlook

Tool support for several key features of generic modeling was very poor when our study was performed. Nevertheless the study so far shows that SystemVerilog enables the designer to specify generic models in a more comfortable way than most if not all traditional HDLs.

Given complete tool support for the LRM, SystemVerilog has the potential to become the leading hardware description language for all future designs especially those developed under the aspect of generic functionality or reusability. Our examples showed that SystemVerilog in most cases can really be considered a superset of both VHDL and VERILOG. These conclusions were made completely leaving aside that SystemVerilog is also a full power verification language, which further enhances its usability.

Regardless of these conclusions, there are still some valuable features that could be adopted from other HDLs in order to improve SystemVerilog flexibility.

8. REFERENCES

[1] Accellera Inc.: *SystemVerilog 3.1a Language Reference Manual*, May 2004.

[2] Preis, V.: *An Approach to Complex and Self-Generating VHDL Models for Simulation and Synthesis*. In Proceedings of the Spring '94 Meeting of the VHDL-Forum for CAD in Europe, pp.39ff. Tremezzo, April 1994.

[3] Bauer, M.; Ecker, W.; Gsottberger, Y.: *Generative modeling of synthesizable IPs using a VHDL Preprocessor*. In: Proceedings of the IP '99, Santa Clara, USA, March 1999.

[4] Hohenauer, M.; Scharwaechter, H.; Karuri, K.; Wahlen, O.; Kogel, T.; Leupers, R.; Ascheid, G.; Meyr, H.; Braun, G.: *Compiler-in-loop Architecture Exploration for Efficient Application Specific Embedded Processor Design*. In Design & Elektronik, February 2004, Germany.

[5] Schliebusch; Chattopadhyay, A.; Leupers, R.; Ascheid, G.; Meyr, H.; Steinert, M.; Braun, G.; Nohl, A.: *RTL Processor Synthesis for Architecture Exploration and Implementation*. DATE04.

[6] King, H.; Kission, P.; Rahmouni, M.; Jerraya, A.A.: *Behavioral Synthesis and Component Reuse With VHDL*. Kluwer Academic Publishers, November 1996.

[7] Stefan Pees, Andreas Hoffmann, Vojin Zivojnovic, Heinrich Meyr: *LISA-Machine Description Language for Cycle-Accurate Models of programmable DSP Architectures*. Internet source: http://www.sigda.org/Archives/ProceedingArchives/Dac/Dac99/papers/1999/dac99/pdfiles/51 2.pdf.

[8] Bluespec, Inc., Waltham, MA. *Bluespec SystemVerilog Version 3.8 Reference Guide*. November 2004.

[9] Hennessy, John L.; Patterson, David A.: *Computer Architecture, A Quantitative Approach*. Morgan Kaufmann Publishers, 3rd Edition, 2003.

[10] Ecker, Wolfgang; Jensen, Peter; Kruse, Thomas; Zambaldi Martin: *SystemVerilog: Interface Based Design*. FDL 2004.

[11] Ecker, Wolfgang; Jensen, Peter; Kruse, Thomas: SystemVerilog in Use: *First RTL Synthesis Experiences with Focus on Interfaces*. ESNUG 2004.