INFORMATICA — UNIVERSITEIT VAN AMSTERDAM

BACHELOR INFORMATICA

UNIVERSITEIT VAN AMSTERDAM

[] System Architecture Simulation using SystemC

R.A.J. Wacanno

April 20, 2020

Supervisor(s):
drs. T.R. Walsta,
drs. A. van Inge,
ing. E.H. Steffens

Signed: Signees

Abstract

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Contents

CHAPTER 1

Introduction

One of the many advantages of the act of software design—when compared to traditional hardware design, is the relative low bar of entry. Anyone with a computer, some time and the right amount of motivation, is able to start working on projects or teach him or herself the ins and outs of various programming languages or theories in programming. This used to be in stark contrast to hardware design. Trying to design custom hardware components or learning about the inner workings of certain components was done in either a strictly theoretical fashion, or in a more involved practical fashion. The first of these tactics typically entails reading up on the needed technical specifications from manuals and applying this to an abstract implementation in the form of diagrams. This way, testing of the functional correctness of a design had to be tested purely based on deduction from this theoretical implementation, without real world testing. Practical testing could also be a possibility, but this came with other disadvantages, like having to buy necessary components, equipment, and needing to have the required technical knowledge to handle said components and equipment.

With the dawn of more capable computational hardware, it becomes more and more feasible to design hardware in a more software-like fashion and test its functionality in a purely software based simulation framework. Applying these techniques to designing and understanding hardware eliminates the need for buying actual hardware, which, like software design, dramatically lowers the bar of entry.

Apart from this, hardware simulation introduces more possible advantages when compared to using real components. When running more complex components, like integrated circuits, the inner workings are hidden in a black box-like fashion. Without the use of specialised probing equipment, it is impossible to see what goes on inside such parts. With further increased complexity, having a look inside the a used components might even become impossible. By using simulated hardware,

This research in particular focuses on the simulation of computer system architectures. Central to these architecture is of course the central processing unit of CPU, which is inherently a complex components comprised of a number of discrete components brought together on a single piece of silicon.

In the end, one might still want to test a hardware design written in software in the real world.

1.1 Research question

CHAPTER 2

Theoretical background

2.1 Model simulation

2.1.1 Continuous models

2.1.2 Discrete models

2.1.3 Transaction-level modelling vs. register-transfer level modelling

2.2 SystemC

Central to this research is the C++ class library SystemC. SystemC is a library aimed at allowing the user to implement hardware functionality on various different levels of abstraction. [mist nog spul]

What makes SystemC especially suitable for this research is the aforementioned way it allows for multiple levels of abstraction. On the one hand, using SystemC, an entire system architecture can be constructed by combining atomic logic gates who's individual implementation involves trivial bit operations. On the other side of the spectrum, SystemC allows for the implementation of a components complete functionality using C/C++ code, while still using SystemC as its interface to communicate with other components.

With the use of the example module in listing 2.1, the sections below will illustrate the basic principles and functionality of SystemC.

```
1  SC_MODULE(example)
2  {
3      sc_in<bool> clock{ "CLOCK" }, poll{ "POLL" };
4      sc_out<sc_uint<20>> value{ "VALUE" };
5
6      and_g and1{ "AND1" };
7      not_g not1{ "NOT1" };
8
9      sc_signal sig{ "SIG" };
10
11     void handle_poll()
12     {
13         out.write(1337);
14     }
15
16     void handle_clock()
17     {
18         while (true)
19         {
20             out.write(5318008);
21             wait();
22         }
```

```
23      }
24
25      SC_CTOR(example)
26      {
27          SC_METHOD(handle_poll)
28          sensitive << poll;
29
30          SC_THREAD(handle_clock)
31          sensitive << clock.pos();
32      }
33  }
```

Listing 2.1: SystemC example module

2.2.1 SystemC components and constructs

Data types

Apart from the new classes it introduces, SystemC also comes with some data types which are useful in a hardware design context.

`sc_int<w>` and `sc_uint<w>` are the respective signed and unsigned fixed width integer types in SystemC. Using the template variable `w`, the amount of bits used to represent the value of the integer in question can be specified. These data types ensure the correct bit width independent on the architecture it is compiled on, as opposed to C/C++ built-in types like `int`. The only limitation of these types is the fact that they are limited to a width of 64 bits. In order to use integer values with arbitrary bit width, `sc_bigint<w>` and `sc_biguint<w>` can be used as the respective counterparts of the aforementioned types.

In order to steam line the passing of and the operations on groupings of individual bits, SystemC includes a so called bit vector in the form of `sc_bv<n>`. With this vector comprising of `n` bits, each individual bit can be addressed and manipulated. This eliminates the need to perform bit masks on existing data types like `int` and allows for the formation of more flexible amounts of bits—other than the usual 8, 32 and 64 bits.

When dealing with true hardware signals, value analogies using only two values—i.e. 'true' or 'false' or 1 and 0—actually don't suffice. In the real world, the values of a signal are characterised using one of the following possible categories: false (0), true (1), intermediate value (X) and floating value (Z). These possible values can be encoded in SystemC using the `sc_logic` data type. Several logic values can be combined using the `sc_lv<w>` type.

Modules

The fundamental building block of any SystemC-described hardware component is the module.

Ports

Signals

2.2.2 Processes

Methods and threads

Simulation scheduling

2.2.3 Constructing hierarchy

2.3 System architectures

2.3.1 MIPS

2.4 Existing simulation solutions

2.5 Used libraries

2.6 VHDL

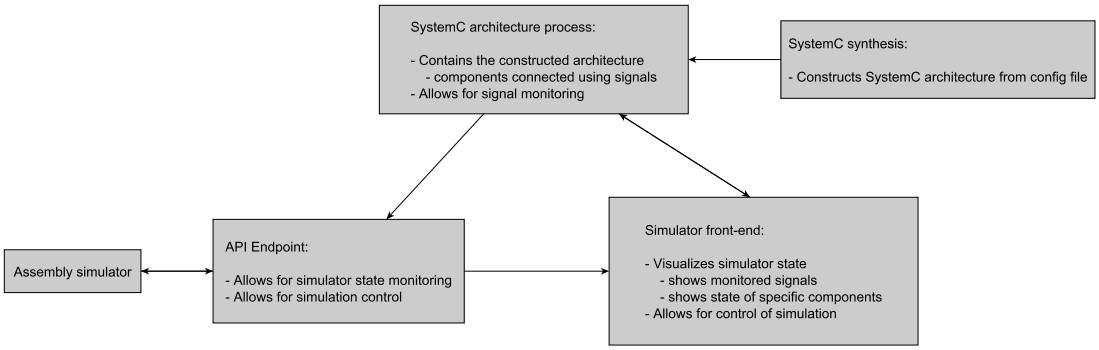
2.7 Ethical implications

CHAPTER 3

Implementation

3.1 SystemC synthesis

3.2 Architecture simulation



CHAPTER 4

Results

CHAPTER 5

Conclusions