Modeling Multi-Valued Circuits in SystemC*

Daniel Große Görschwin Fey Rolf Drechsler

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{grosse,fey,drechsle}@informatik.uni-bremen.de

Abstract

The complexity of todays hardware systems steadily increases. Due to this fact new ways of efficiently describing systems are investigated. A very promising approach in this area is SystemC which is a C++-library. To take advantage of SystemC in the multi-valued domain, the concept of multi-valued logic has to be embedded in SystemC.

In this paper such a concept is introduced and details of the implementation are given. This creates a powerful development environment to model and efficiently simulate complex multi-valued circuits and systems. Due to C++-concepts, like operator overloading and templates, the task of modeling circuits becomes very convenient and handling of multi-valued signals is elegant. This gives the opportunity to design large circuits that can be mapped onto physically multi-valued gates. A scalable arithmetic logic unit is studied and experimental results are given.

1. Introduction

With the increasing complexity of circuits and systems having several million gates it becomes more and more important to efficiently describe and simulate those designs. The two most popular hardware description languages, namely VHDL and Verilog, are used to describe the RT-level, while a first reference design usually is modeled in C or C++. This gap can be closed by the use of C-like hardware description languages. One very promising candidate is SystemC which allows to model a design at different levels of abstraction, starting at the functional level and ending at a cycle-accurate model [3]. The well-known concept of a hierarchical description of a hardware system is transferred to SystemC by modeling a module as a C++-class. Any SystemC-description can be compiled to an executable with a common C++-compiler and results in a very efficient simulator for the specified system.

Also a first commercial tool "CoCentric™SystemC" to synthesize SystemC is already available from Synopsys™, though only a subset of the language may be used in the description [7]. Research results on synthesizing more complex programming constructs [8] and on formal verification of SystemC-descriptions [1, 2] have recently been proposed.

On the other hand a lot of work is done on circuit design based on multi-valued logic. Modeling circuits on a multi-valued basis instead of being based upon binary logic can lead to advantages in the verification [4]. But also the physical realization of multi-valued gates is object of current research. This leads to the need of a development environment that supports modeling multi-valued circuits. A first approach in this direction was introduced in [6] where a package to model ternary circuits in VHDL was provided. But this was not generic, so for each radix a separate VHDL-package is required.

In this paper we present a concept to model multi-valued circuits in SystemC and discuss the implementation of two multi-valued datatypes. Since SystemC is a C++-library it can be extended to support multi-valued logic in an efficient manner by using the facilities of C++. This creates a powerful development environment for the design of multi-valued circuits. The first of the new datatypes is the one-digit-type `sc_multval` which is parameterized such that the radix is chosen at instantiation. Second, `sc_mv` is a multi-valued vector type. Using these new types handling of multi-valued signals becomes very easy, due to operator overloading in C++. This means, the designer decides about the radix of a signal at instantiation and uses it like any other signal afterwards. Adding two quarternary signals a and b is literally as easy as writing $a+b$. Using the new datatypes an arithmetic logic unit (ALU) has been modeled and simulation results are given, that show the influence of the parameters width and radix of operands.

*This work was supported in part by DFG grant DR 287/8-1.

The remaining part of the paper is structured as follows: In Section 2 the algebra we chose to implement is defined and some concepts of SystemC are shortly introduced. Section 3 describes the concepts and details of the implementation of the two new datatypes. On top of this implementation a multi-valued ALU is constructed in Section 4 and simulation results are reported. The conclusions and an overview of future work can be found in Section 5.

2. Preliminaries

2.1. Multi-Valued Circuits

Multi-valued networks can simply be modeled as graphs. Edges in these graphs correspond to signals while vertices correspond to primary inputs or outputs, to flip-flops or to basic gates. Usually cycles in the network extend across flip-flops, while the combinatorial parts of the network are free of cycles.

All of the basic gates can be used as operators in higher level descriptions of the circuit. More complex constructs can be defined on these gates and can be used in the high level description as well.

Given k as the radix, the operators (that correspond to basic gates) are defined over a set $\mathcal{P} = \{0, \ldots, k-1\}$. We chose a functional complete set of operators [5], all having one output and one or two operands x and y ($x, y \in \mathcal{P}$): $EQUAL(x, y)$, $MIN(x, y)$, $MAX(x, y)$, $INV(x)$, $LITERAL_{a,b}(x)$. MIN and MAX correspond in the binary case to AND and OR, respectively. INV corresponds to the complement and is defined as

$$INV(x) := (k-1) - x.$$

The $LITERAL$-operator has two fixed parameters $a, b \in \mathcal{P}, a \leq b$ and is defined by:

$$LITERAL_{a,b}(x) = \begin{cases} k-1, & \text{if } a \leq x \leq b \\ 0, & \text{otherwise} \end{cases} .$$

All binary operators are defined on two operands of the same radix only. The generalization of the definition to operands of different radices can be done if necessary.

2.2. SystemC

SystemC comes as a C++-library, its source code is freely available[1]. It provides the facilities to model a system at different levels of abstraction. Since the model can be compiled into an executable the simulation of the design is very efficient. For this task a standard C++-compiler can be used.

Classes to implement modules and their interconnect are provided by SystemC. In an early design step the description of functionality and data flow can use all features of C++, but should already reflect the coarse structure of the latter design. This description can then be refined to a synthesizable one. While doing this different approaches to realize particular modules can be explored.

While SystemC-constructs, like modules or clocked processes within modules, can be synthesized already, some others can not. This is due to the fact that some constructs only make sense in the area of simulation, like e.g. routines to trace waveforms. On the other hand several C++-constructs, like e.g. pointers are not supported by the only SystemC-Compiler available so far [7], but some promising approaches to overcome limitations of this kind have been published recently [8].

Using C++ all basic datatypes of SystemC are implemented as classes that provide a set of operators, conversion routines and output routines. Amongst the basic datatypes for signals are `sc_bit` which is binary and `sc_logic` that extends `sc_bit` by an unknown ('X') and a high impedance value ('Z'). For either type a corresponding vector type is also provided, namely `sc_bv` and `sc_lv`.

Some notion of multi-valued logic is also already inherent to SystemC. It includes the type `sc_int` which allows to specify a number of bits for its value and arithmetic operations are provided for this type. Therefore this can be interpreted as being a multi-valued signal, having a radix that is a power of two. But in this case the signal is synthesized to binary constructs. Also modeling circuits with other radices is a tedious task, when only the native types of SystemC are used.

For more details on SystemC see e.g. [3] or the SystemC-website.

3. Multi-Valued Modeling

To allow for easy modeling of multi-valued circuits two datatypes are needed. The first one is a one-digit type and corresponds to a signal or wire in a physical multi-valued design. Details about this type, called `sc_multval` are given in Section 3.1. The second is a vector type `sc_mv` that is constructed from the one-digit type, it is introduced in Section 3.2. For example buses or arithmetic operations can be conveniently modeled using this type.

3.1. One-Digit Type

The new multi-valued type `sc_multval` corresponds to the binary type `sc_bit` which is native to SystemC.

[1] The source code and several documents on SystemC can be found at www.SystemC.org.

```
friend const sc_multval operator | (
   const sc_multval& a, const sc_multval& b )
{
   return sc_multval( max(a.m_val,b.m_val) );
}
```

Figure 1. Code of the |-operator

`sc_bit` is defined on the set $\{0, 1\}$. The common logical operators as well as conversion, input and output routines are defined on this type.

Summarized the features of the new type are:

- The radix can be chosen at instantiation.
- Operators can be applied as on other C++-datatypes.
- The underlying operations can easily be changed.

The remainder of this section shows how these features are put into practice, therefore parts of the implementation have to be discussed in more detail.

The type `sc_multval<int k>` is defined as a C++-template class having the desired radix k as its parameter. Therefore it is defined on the set of values $\{0, \ldots, k-1\}$. Where possible the usual C++-operators were overloaded to realize their multi-valued counterparts. For example the multi-valued operator MAX is the general case of the binary OR-operator and therefore intuitively corresponds to it. So the C++-instruction "`a | b`" applied to two operands a and b of type `sc_multval` calculates $MAX(a, b)$. Figure 1 shows the C++-code for overloading the operator "|", the internal values of the operands are compared. The same correspondence is true for the multi-valued operators $EQUAL$, MIN and INV that overload the C++-operators "==", "`&`" and "∼", respectively.

The matter is different with $LITERAL_{a,b}$ for two reasons. First, there is no unique correspondence to an operator on a binary value. Second, in this case there can be several instances of $LITERAL_{a,b}$ in use, having different parameters a and b. Therefore we decided to implement this operator as a member function of `sc_multval`. Since it is a unary operator with only one output the assignment $x := LITERAL_{2,4}(y)$ can be written in the C++-source as "`x = y.literal(2,4);`". The generalization of the two binary operators MIN and MAX to handle two operands of different radices can be done, if necessary and if the semantics is defined.

For simulation it is necessary to retrieve signal values from the circuit. To achieve this, SystemC uses a method `sc_trace`, that takes a parameter of type `sc_multval`. This method is also part of the implementation and simply scans the actual value of the variable.

This concept of modeling multi-valued circuits also allows for a change of the underlying algebra without touching the design itself. Simply by modifying the routines that implement the operators their semantics can be redefined. This way an arbitrary arithmetic can be chosen.

3.2. Vector Type

The vector-type `sc_mv<int k, int n>` is defined as a template class as well. The two template parameters specify the radix k and the width n of the vector. Basically there are two ways to implement this type in SystemC. Internally the value of the vector can either be mapped to an integer value or stored in an array of `sc_multvals`.

Holding the value internally as an integer has the advantage of fast arithmetic operations, since the native integer operations can be used. But this is paid by disadvantages in the access to components of the vector. The multi-valued operators are usually carried out on the digits separately. Therefore carrying out any of the operations would mean, first to convert the vectors value to the multi-valued domain, second carry out the operation component-wise, third convert the value back. The same problem would arise when accessing a digit of the multi-valued vector. The internal representation would have to be converted to a multi-valued one at first.

Therefore the implementation as an array of `sc_multvals` was chosen. Then the application of the multi-valued operators is efficient and so is the access to the vector's components. For example the code for $LITERAL_{a,b}$ makes use of the operator "`[]`" to access single elements of the vectors:

```
sc_mv literal(int a, int b) {
 sc_mv<k,n> tmp;
 for (int i=0;i<n;i++)
  tmp[i] = (*this)[i].literal(a,b);
 return tmp;
}
```

This comes at the cost of less efficient arithmetic operations. But it is not desirable, since SystemC is used as a fast simulator. We consider it acceptable to a certain extent, since mostly the vectors components will be accessed or basic operations will be carried out. Without overloading the arithmetic operators three steps would be necessary

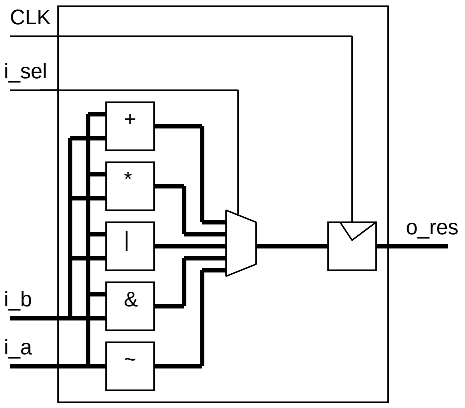

Figure 2. A simple ALU

to carry out an operation: conversion to integer, carrying out the operation and conversion back to `sc_mv`. Therefore the overloaded addition is realized using the school-method, i.e. adding component-wise. So it becomes more efficient than the simple forewards and backwards conversion. The multiplication is more difficult and indeed it does not become more efficient, but by overloading the operator the conversion steps before and after the multiplication can be done internally. They do not occur in the design and this leads to "cleaner" desriptions.

Another advantage of overloading the arithmetic operators occurs during synthesis of the description. The arithmetic operation can easily be recognized as working on multi-valued operands and an efficient hardware realization can be chosen from a library.

Tracing the waveforms of vectors is done component-wise by using the method for tracing `sc_multval`.

4. Case Study: ALU

To demonstrate the efficiency of our approach we studied a scalable multi-valued ALU for different radices and width of the operands. The block diagram of the ALU can be seen in Figure 2. The module has four inputs and one output. Besides the necessary clock-input there is a select-signal and two inputs for operands `i_a` and `i_b`. The radix of the `clock` is two of course and a datatype native to SystemC is used.

The ALU realizes the operations INV, MIN, MAX, addition and multiplication. Therefore the radix of the input `i_sel` to select the desired operation is five. The result always has the same radix as the two operands. The width of both operands is equal and the result has the double width, to take the maximum result of a multiplication. Radix and width of the operands can be separately defined, so the ALU is scalable. Its functionality is covered by the routine `compute` (see Figure 3), which is called at every rising edge of the clock. As can be seen the description is convenient and easy to understand.

```
void ALU::compute()
{
  sc_mv<RADIX_k,WIDTH_n*2> res;

  switch (i_sel.read().toInt()) {
  case 0: // MIN
    res = i_a.read() & i_b.read();
    break;
  case 1: // MAX
    res = i_a.read() | i_b.read();
    break;
  case 2: // INV of i_a
    res = ~i_a.read();
    break;
  case 3: // ADDITION
    res = i_a.read() + i_b.read();
    break;
  case 4: // MULTIPLICATION
    res = i_a.read() * i_b.read();
    break;
  }
  // write result to output
  o_res.write(res);
}
```

Figure 3. The routine `compute` performs the basic operation of the ALU

4.1. Simulation Results

An exemplary trace of a simulation run can be seen in Figure 4. The widely used VCD (value change dump) file format[2] is used to store the traced waveforms, so for displaying any tool that is able to read this format can be used. We used the freely available waveform-viewer Dinotrace[3].

In this simulation the radix of operands and result (`i_a`, `i_b`, `o_res`) was set to five, the width of the operands was four and therefore the width of the result was eight. Each multi-valued vector is split into its components, the components are displayed as integers and therefore as if being 32-bit-values. The result of an operation is delayed by one clock cycle.

Runtimes of longer simulation runs are given in Table 1. All these runs were carried out on a Pentium II at 400 MHz with 256 MB of physical RAM running under Linux.

[2]This is included in the description of the IEEE Standard 1364-1995.
[3]Dinotrace can be downloaded at www.veripool.com/dinotrace/.

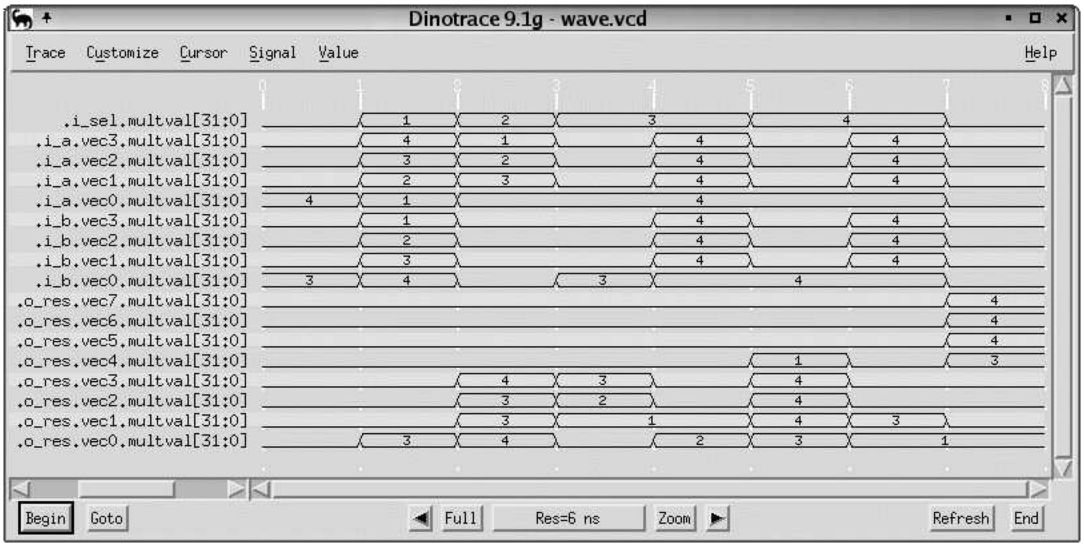

Figure 4. In- and Outputs of the ALU

In each case 1 million cycles were simulated with either tracing waveforms or no tracing. The stimuli were created randomly. We investigated runtimes for different radices and widths of the operands.

The empty fields are due to overflow in the multiplication. The operator itself is coded with 64-bit accuracy, therefore a check in this operation takes care that no overflow occurs. Having operands of width 10 and radix 8 leads to a binary representation of 30 bit. Therefore the result has a width of 60 bit and is still in the range of the operator. For efficiency the check is implemented such that it does not calculate a sharp upper bound of the operators range, so the multiplication on radix 16 and width 8, leading to a 64-bit result is not carried out.

It is remarkable that the simulation of 1 million cycles does only take about 40 seconds on operands of width 10 on a Pentium II. Furthermore it can be observed that the radix of the operands is of minor influence.

Of course the time needed increases when tracing of waveforms is activated. But if the size of the waveform-file of more than 200 MB for operands of width 10 is taken into account, 200 seconds are still very efficient.

5. Conclusions and Future Work

In this paper an extension to SystemC for modeling multi-valued circuits was introduced. This allows to model and implement complex multi-valued systems by a high level hardware description and also to efficiently simulate it. The concept of overloading operators in C++ allows to mix the new datatypes `sc_multval` and `sc_mv` with the native SystemC- and also C++-datatypes, e.g. to directly assign an integer to one of the new types or vice versa. The semantics to synthesize the new datatypes can be chosen to be either truly multi-valued or to be any two-valued encoding scheme like e.g. logarithmic or one-hot.

Besides this, future work is to define the semantics of operators on operands with different radices and include this in the basic multi-valued types. To allow for unknown ('X') and high impedance ('Z') states during simulation another multi-valued type that supports these values has to be defined.

References

[1] R. Drechsler and D. Große. Reachability analysis for formal verification of SystemC. In *EUROMICRO*, pages 337–340, 2002.

[2] D. Große and R. Drechsler. Formal verification of LTL formulas for SystemC designs. 2003.

[3] T. Grötker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[4] S. Höreth and R. Drechsler. Formal verification of word-level specifications. In *Design, Automation and Test in Europe*, pages 52–58, 1999.

[5] J. Muzio and T. Wesselkamper. *Multiple-Valued Switching Theory*. Adam Hilger, Bristol and Boston, 1986.

[6] C. Rozon. On the use of VHDL as a multi-valued logic simulator. In *Int'l Symp. on Multi-Valued Logic*, pages 110–115, 1996.

[7] Synopsys. *Describing Synthesizable RTL in SystemC^{TM}, Vers. 1.1*. Synopsys Inc., 2002. Available at http://www.synopsys.com.

[8] N. Vanspauwen, E. Barros, S. Cavalcante, and C. Valderrama. On the importance, problems and solutions of pointer synthesis. In *Symposium on Integrated Circuits and System Design*, pages 317–322, 2002.

Table 1. CPU-Time in seconds for the simulation of 1 million clock cycles

TRACE	RADIX	WIDTH 1	2	3	4	5	6	7	8	9	10
no	3	12.14	15.53	18.96	21.54	24.99	27.42	29.96	32.49	38.16	40.61
	4	12.42	15.76	19.14	21.69	25.10	27.60	30.08	32.64	38.36	40.82
	5	12.56	15.86	19.28	21.78	25.17	27.65	30.22	32.75	38.38	40.84
	6	12.66	15.91	19.30	21.78	25.22	27.66	30.30	32.84	38.49	40.93
	7	12.74	15.94	19.33	21.80	25.24	27.70	30.29	32.84	38.55	41.00
	8	12.77	15.98	19.35	21.85	25.24	27.76	30.29	32.84	38.57	41.04
	9	12.81	16.01	19.36	21.88	25.26	27.80	30.33	32.90	38.56	
	10	12.85	16.02	19.37	22.03	25.27	27.95	30.38	32.87	38.58	
	11	12.91	16.04	19.41	21.87	25.30	27.83	30.39	32.92		
	12	12.93	16.06	19.40	21.89	25.38	27.85	30.37	32.92		
	13	12.94	16.04	19.41	21.93	25.39	27.85	30.39	32.93		
	14	12.97	16.06	19.42	21.91	25.38	27.89	30.37	32.90		
	15	12.97	16.06	19.42	21.93	25.40	27.89	30.40			
	16	12.97	16.07	19.44	21.91	25.41	27.86	30.43			
yes	3	47.27	61.68	76.17	89.27	104.89	118.25	131.57	145.83	165.02	178.92
	4	49.07	64.63	80.88	95.42	112.72	126.60	141.38	156.73	177.67	193.32
	5	49.72	66.79	83.74	99.65	116.79	132.20	147.80	164.38	185.33	201.17
	6	50.87	67.98	85.46	101.39	119.85	135.48	151.89	168.47	190.77	207.57
	7	51.55	68.73	86.88	103.03	122.01	138.28	154.78	171.67	194.61	211.59
	8	51.94	69.66	87.77	104.73	123.71	140.19	157.16	174.54	197.79	214.92
	9	52.13	70.22	88.43	105.47	124.86	141.86	158.85	176.50	199.69	
	10	52.36	70.44	89.34	106.44	125.79	143.01	160.25	178.25	201.76	
	11	52.72	71.12	89.56	107.03	126.77	143.86	161.79	179.71		
	12	52.95	71.25	90.24	107.62	127.55	145.05	163.07	181.26		
	13	53.12	71.66	90.69	108.38	128.41	146.07	163.80	182.63		
	14	53.29	71.83	91.27	108.76	129.00	146.40	164.72	182.79		
	15	53.44	72.19	91.22	109.21	129.39	147.04	165.17			
	16	53.58	72.12	91.61	109.68	129.95	148.30	165.64			