FDL '98

Modeling Layout Extracted MOS Transistor Circuits in VHDL

Ekaterini Gikas, Stefan Hendricx, Luc Claesen

IMEC vzw/Katholieke Universiteit Leuven
Kapeldreef 75, B-3001 Heverlee, Belgium
e-mail: katerina@imec.be

Abstract

The ever-increasing complexity of today's electronic systems has made the design of correct circuits a challenging undertaking. The electronic design process could be a lot easier if designers were able to readily reuse pre-designed modules. In this paper, work is presented which enables the reuse of pre-designed MOS-transistor circuits in larger systems. In our approach, low-level layout extracted transistor netlist specifications of MOS-transistor circuits are modeled in VHDL. The resulting models can then be used for verification purposes or they can be embedded in more complex electronic system designs. Our modeling technique is, essentially, style independent and can deal with detailed characteristics of MOS circuits, such as ratioed logic, charge sharing, bi-directional pass transistors and pre-charged logic.

Keywords: VHDL, Design Reuse, IP, Intellectual Property, VHDL modeling, Core based design, Formal Verification

1 Introduction

Technological advances in microelectronics have enabled today's digital systems to be of great complexity, containing possibly millions of transistors. As technology continuously progresses, this complexity increases day by day, making the design process itself more difficult, time consuming, costly and error prone. Therefore, it is necessary for design automation tools to improve and embody design principles helpful for the designers. For instance, tomorrow's tools should support "design reuse", which refers to the use of pre-designed modules in new designs, and "formal verification", which refers to a rigid proof of a system's correctness. In this paper, we present a pragmatic methodology which promotes both design reuse and formal verification for systems specified in the hardware description language VHDL [8].

In essence, our methodology aims to model MOS-transistor circuits, specified at the layout level, at a higher level of abstraction – i.e. at the structural level – using VHDL. For this purpose, a transistor netlist is first extracted from the circuit's layout. From this netlist, a logic description or *lgc*-description is then derived for the functional behaviour of the circuit, using the tool COSMOS (a Compiled Simulator for MOS Circuits) [1]. Finally, this logic description is compiled into a VHDL model for the given circuit.

In general, existing functional extraction tools are style dependent and operate by identifying a limited set of transistor patterns. Our methodology, however, is style independent and, being based on COSMOS, can deal with more detailed characteristics of MOS circuits – such as ratioed logic, charge sharing, bi-directional pass transistors and pre-charged logic. Consequently, the VHDL models generated by our approach are accurate functional models of the original circuits. Our circuit models may also be reused in bigger, more complex electronic systems. Libraries of such cells and reusable cores can also be

created. Simulation of the systems embedding these models is possible in a standard VHDL simulation system (e.g. Mentor Graphics, Synopsys, Cadence, etc.).

As we have mentioned before, automatic formal verification becomes essential to verify electronic designs rapidly and rigorously, because the complexity of today's electronic designs has made verification through exhaustive simulation a very tedious – if not impossible – task. Today, the circuit layout is still mostly being verified using a Layout-versus-Schematic (LvS) verification technique. Clearly, the usefulness of such a verification process is strongly impaired by both error prone schematics and slow SPICE simulations. A tool for Layout-versus-RTL (LvRTL) verification, which gets rid of both the schematics and the SPICE simulations, can be created by combining a functional extraction tool, like the one presented in this paper, with a functional equivalence checker. Another functional extraction tool, that is commercially available, is *Laybool* [7]. *Laybool* is implemented by SGS Thomson Electronics and has been incorporated in VFormal (an equivalence checker from Compass DA) to enable another LvRTL verification methodology.

In the next section, we present a few basic concepts on the switch-level model of MOS-transistor circuits. In section 3, we subsequently describe our methodology for the automatic generation of VHDL specifications for MOS circuits. To illustrate our approach, section 4 introduces a small example. Finally, section 5 presents some experimental results and we conclude our presentation in section 6.

2 Concepts of modeling MOS circuits

In our approach, MOS-transistor circuits are modeled at the switch level, a level between the very detailed transistor level and the more abstract gate level. The interested reader may refer to [2] for a survey of switch-level algorithms. In this section, we present a few basic concepts of the MOS-transistor circuit model, as it was introduced by R. E. Bryant [3,4] for symbolic simulation in COSMOS [1].

At the switch level, a MOS transistor circuit is modeled as a network of switches – i.e. a set of nodes interconnected by transistor switches. The nodes may be presented in three different states, encoded as '0', '1' and 'X'. A node is in state '0' when its voltage is low, in state '1' when its voltage is high and in state 'X' when its voltage is invalid (a voltage between 0 and 1), uninitialized or unknown.

The interconnection structure of a switch-level network can be represented by a channel graph. The channel graph defines a partitioning of the network in channel-connected subnetworks. Furthermore, the functional behavior of each of these subnetworks can be represented by a steady-state response function.

The Dual-Rail Encoding is employed to enable writing of an expression of a MOS circuit's functionality in the switch-level model in terms of Boolean formulas. According to this encoding, a state value $y \in \{0,1,X\}$ is encoded in terms of two Boolean values y.1 and $y.0 \in \{0,1\}$ as follows:

y	y.1	y.0
1	1	0
0	0	1
X	1	1

Table 1: The Dual-rail encoding

Hence, the steady-state response function representing the behavior of a subnetwork can be described as a sequence of Boolean operations, which define a mapping of the subnetwork's inputs and old state to its outputs and new state. The functionality of the whole circuit is then expressed as a set of systems of Boolean equations.

ANAMOS – one of the subprograms of COSMOS – is responsible for the actual symbolic analysis of the circuits. For a given MOS circuit, the symbolic analysis procedure generates a functional description of the circuit in terms of a set of Boolean (or symbolic) functions. This set of symbolic functions forms the input for our modeling methodology, which is explained in more detail in the following section.

3 Modeling MOS transistor circuits in VHDL

Although VHDL was not initially intended for modeling at the switch level, there have already been various approaches to introduce the switch-level model in VHDL [5,6]. In [5], for instance, a Switch-Level Package has been built, which provides numerous functions for the support of a 46-value system. This 46-value system is used to model the bi-directional behavior of MOS switches. Our approach is quite different. We do not attempt to create VHDL models for transistor switches and use these as primitive components for the design of MOS circuits. On the contrary, in our methodology, the MOS circuit has been designed in advance and we want to derive a VHDL model of it. So our methodology relies on a symbolic analyzer to analyze a layout extracted MOS-transistor circuit netlist and to generate a symbolic description of the circuit's functionality. Based on this symbolic description, VHDL constructs are then used to create a VHDL model of the MOS circuit.

Let us consider our approach in more detail. First of all, a VHDL-package *Dual_Rail_Encoding* has been defined. This package declares a new type *X01*, which includes the three values 'X', '0' and '1', reminiscent of the three-valued (ternary) logic used in the switch-level model of MOS-transistor circuits (see section 2). In addition, this package also defines several functions to implement transformations from the three-valued encoding to binary encoding, and from binary back to the three-valued encoding. For example, the functions *Low_Rail* and *High_Rail* determine the low rail *y*.0 and high rail *y*.1 of a three-valued signal (or variable) *y*. Likewise, the function *Ternary* returns the ternary value corresponding to a dual-rail encoded value.

A system design in VHDL has one external view and one or more internal views. The *entity* describes a system's external view or interface. The *architecture* describes the system's internal view; it describes either the structure, the behavior or the data flow of the system. Sometimes an architecture can not be concretely classified in one of the above categories, but rather it describes the system in a mixed manner.

Our model of a MOS-transistor circuit in VHDL is composed of an entity and an architecture too. The input and output signals declared in the entity's port declaration are of type *X01*. These signals take the value '0', '1' or 'X', depending on the voltage in the corresponding node. Although the input and output signals take the above values externally (in the entity), internally (in the architecture) we introduce a pair of binary signals for each input and output signal, as the dual-rail encoding imposes.

The architecture describing the MOS circuit can be abstractly divided in three sections (see Figure 1). In the first section, all pairs of binary signals, encoding the circuit's input signals, are computed. Using the inputs' dual-rail encoding, the second section computes the dual-rail encoding for each of the outputs. Finally, in the last section, the ternary values of the output signals are calculated. But how are the signals that comprise the dual-rail encoding of

the output signals computed?

1. Three-valued input signals ⇒ Dual-rail input signals
2. Component instances ⇒ Compute dual-rail of output signals
3. Dual-rail output signals ⇒ Three-valued output signals

Figure 1: Global outline of a MOS circuit's VHDL-architecture

Our VHDL-models preserve the original partitioning of the MOS circuit as a set of interconnected sub-circuits – i.e. the channel-connected subnetworks extracted by COSMOS. Each distinct subcircuit defines a unique VHDL component. In the architecture of our final VHDL-model, instances of these components are appropriately interconnected among each other, thus modelling the MOS circuit's interior structure.

The components' inputs and outputs are bit signals. These are declared in each component's entity port declaration. Output signals may need to be accessed within the component for some calculation; this is allowed in the *lgc* language but not in VHDL. Therefore the component's output signals are declared as *buffers* (because in VHDL, this mode permits reading the value of an *output* signal within the component). The reader should note that this doesn't impose any restriction on the usability of our model, because these signals are not visible in the MOS system's interface.

In *lgc*, the behavior of each subsystem is described by an ordered sequence of Boolean operations in a *leaf* module. These operations may be primitive or composite. Primitive operations are, for instance, the logic *and* operation (when transistors are connected in series), the logic *or* operation (transistors connected in parallel) and simple assignment operations. Composite operations are simply operations defined in terms of primitive and/or other composite operations. In VHDL, the concurrent *process* construct is used to define a set of actions to be executed in sequence. As such, a VHDL process can be used to model the behaviour of individual components. Evidently, a process describing a component's behavior must be sensitive to all the input signals of the component.

There exist cases where the new value of an output is needed for some calculations within the component. Inside VHDL processes, however, signal assignments are not executed immediately; in VHDL, signal assignments merely define a projected value for signals, which are only updated at the end of the δ-cycle. To solve this discrepancy, we always define a variable inside a process for each output signal. Since variable assignments are executed immediately, the desired output value is always available inside the component. At the end of the δ-cycle, we then update the output signal of the component with the value of the corresponding variable.

The VHDL model of MOS transistor circuits explained above is in principle the same as the one used in the symbolic simulator COSMOS. However, the extensive use of VHDL in system design makes our work valuable. In the next section a small example is used to further illustrate what has been introduced in this section.

4 A very simple MOS circuit

The simple circuit we consider in this section has three inputs *a, b* and *c* and a single output *Output*. The desired output of this small system can be specified by the following Boolean expression : $Output = a \cdot b + c$

A transistor level implementation of this MOS circuit is shown in Figure 2.

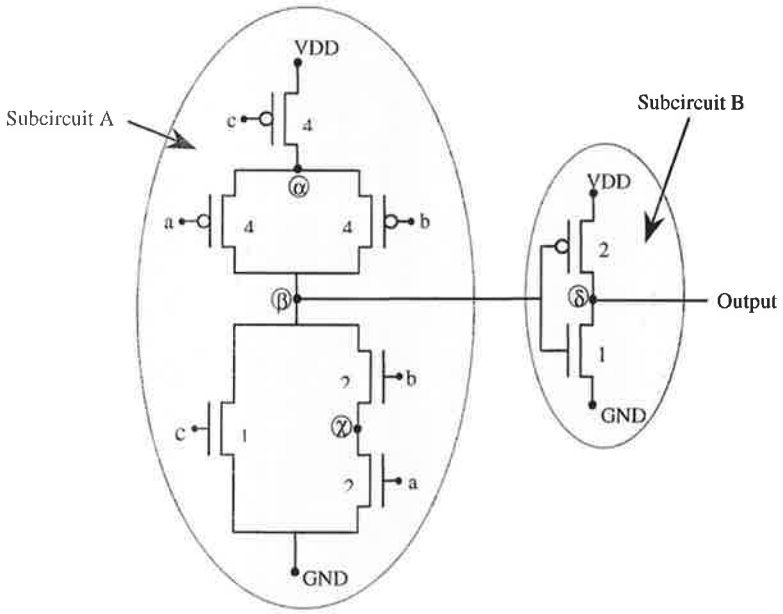

Figure 2: A possible MOS implementation of the function *Output* = *a.b* + *c*

At the physical level, we can describe the circuit in Figure 2 by the following transistor netlist:

```
|units:150 tech:cmos
p  a    alpha     beta    2  4
p  b    alpha     beta    2  4
p  c    Vdd       alpha   2  4

n  a    gamma     GND     2  2
n  b    beta      gamma   2  2
n  c    beta      GND     2  1

p  beta Vdd       Output  2  2
n  beta Output    GND     2  1

A  Output  Sim : Output
```

Figure 3: Transistor netlist for the MOS circuit of Figure 2

Each line in this netlist specifies the transistor type, the names of gate, drain and source pins, and the relative length and width of the channel of the transistor. Starting from this netlist description, COSMOS first creates a *.ntk* file for the switch-level network and then an *.lgc* file, containing the (symbolic) description of the circuit's behaviour. The *lgc*-description, which is the actual input to our own tools, for the example circuit looks like:

```
suffix H, L ;
prim id, and, or ;

/* Lgc-description of subcircuit A */
leaf sn_A (n_3_0, n_3_1; n_4_0, n_4_1,
           n_5_0, n_5_1, n_6_0, n_6_1)
{
   and(\0; n_4_1, n_5_1)
   or(n_3_0; n_6_1, \0)
   or(\1; n_4_0, n_5_0)
   and(n_3_1; n_6_0, \1)
}
```

```
/* Lgc-description of subcircuit B */
leaf sn_B(n_1_0, n_1_1; n_2_0, n_2_1)
{
   id(n_1_0; n_2_1)
   id(n_1_1; n_2_0)
}

/* Final Structure Body */
{
   beta, a, b, c, Output;
   sn_A"sn_A/0/"(beta:L, beta:H; a:L, a:H, b:L, b:H, c:L, c:H)
   sn_B"sn_B/1/"(Output:L, Output:H; beta:L, beta:H)
}
```

In the *lgc*-description, leaf *sn_A* describes subcircuit A and leaf *sn_B* describes subcircuit B. The argument list for each leaf-definition denotes the dual-rail encoding for the outputs and the inputs correspondingly. The final structure body calls these modules with the actual node names. The LGCC* program (the revised *lgc*-compiler LGCC as it has been extended by our work) is used to generate the VHDL-model for the above circuit. A part of this model, consisting of the entity and the architecture of the component corresponding to leaf *sn_A* and the final body, has been included in appendix A.

Despite its limited size, the example above displays all concepts of MOS-circuit modeling that we have presented in the previous sections. The generated VHDL code clearly illustrates how VHDL processes model channel-connected subcircuits. One can also observe how the functions *Low_Rail*, *High_Rail* and *Ternary*, defined in the *Dual_Rail_Encoding* package, are used.

5 Experimental results

We have generated VHDL models for a number of different MOS-transistor circuits, ranging from a simple carry ripple full-adder to a BCD-recogniser, a simple microprocessor and a cryptographic coprocessor. Each of these models has been simulated in the Mentor Graphics design environment, and were shown to be fault-free with respect to the original behaviour. Table 2 illustrates some practical data for these experiments: the number of transistors for each of the circuits, and the CPU time (in seconds) needed to generate the VHDL model on a DEC 5000/120.

	Numbers of Transistors	CPU time (sec)
Carry ripple full-adder	184	0.4
Simple ALU	70	0.3
BCD-recogniser	104	0.2
Tamarack microprocessor	15034	1.2
Subterranean [9] (a cryptographic Co-Processor)	43000	2.1

Table 2: Some experimental results for generating VHDL-code for various transistor-level circuits

6 Conclusions

This paper illustrates a method for automatically generating VHDL-descriptions for MOS-transistor circuits, starting from layout extracted transistor netlists. The software implementing the above method has been used for modeling systems with varying size and functionality, with satisfactory results in simulation. This gives us the certainty that the outcome of our work is useful to designers, who need to incorporate pre-designed MOS circuits in their designs. Furthermore, the derived functional models can be used for a Layout versus RTL equivalence checking.

We should admit, however, that because of maintaining the dual-rail encoding and the excessive amount of signals it imposes, our model is rather cumbersome and therefore its utilisation is restricted. For instance, it is difficult to use it for purposes like re-synthesis, documentation etc. unless it is further refined. Because of the same reasons, refinement is necessary for the use of our models in existing equivalence checking programs.

Although the simulation time of the derived models is not as short as the simulation time of the RTL-equivalents, it is nevertheless worthwhile to have at our disposal these accurate models automatically, instead of designing them from the beginning each time. Hence, to reduce design time, hardware design companies could use this methodology to generate VHDL models for MOS circuits they need to incorporate in specific designs. Alternatively, they could build libraries of VHDL models for MOS circuits, which they have at their disposal, for later use.

Using our tool in combination with an equivalence checking program, designers can verify that their RTL-model before synthesis is equivalent to the model extracted from the final generated layout. An important advantage is that our methodology does not try to identify specific patterns of transistors in the extracted netlist; on the contrary, it can deal with more general transistor interconnections and it can account for transistor characteristics, such as dynamic pass transistors, pre-charged logic etc.

7 Acknowledgments

The authors would like to thank Randal E. Bryant and the members of the COSMOS team for making the COSMOS system available to them. The Research presented in this paper was partially supported by a scholarship from the Flemish Institute for the promotion of Scientific-Technological Research in Industry.

8 References

1. D. Beatty, K. Brace, R. E. Bryant, K. Cho, L. Huang. User's Guide to COSMOS a Compiled Simulator for MOS Circuits.
2. R. E. Bryant. A Survey of Switch-Level Algorithms. IEEE Design & Test of Computers, August 1987.
3. R. E. Bryant. A Switch Level Model and Simulator for MOS Digital Systems, IEEE Transactions on Computers, Vol C-33, No 2, February 1984. pp.160-177.
4. R. E. Bryant. Boolean Analysis of MOS Circuits, IEEE transactions on computer aided design vol. CAD-6, No 4, July 1987.

5. A. G. Stanculescu, A. S. Tsay, A. N. D. Zamfirescu, D. L. Perry. Switch Level VHDL Descriptions, ICCAD89.

6. R. D. Acosta, S. P. Smith, Jeff Larson. Mixed-Mode Simulation of Compiled VHDL Programs, ICCAD89.

7. Dino Caporossi and Geof Barrett. Formal Verification of a Large Design. http://www.eedesign.com/Editorial/1996/CoverStory9601.html

8. R. Lipsett, C. Schaeffer and C. Ussery. VHDL: Hardware Description and Design. Kluwer Academic Publisher, 1989.

9. L. Claesen, J. Daemen, M. Genoe, G. Peeters. Subterranean: A 600 Mbit/sec Cryptographic VLSI Chip. Proceedings ICCD'93, International Conference on Computer Design, VLSI in Computers & Processors, Cambridge Massachusetts, October 3-6, 1993.

9 Appendix A

In this appendix, the VHDL-model generated by our methodology for the example in section 4 is listed. In order to contain its size, the definition for component *sn_B* has been omitted from this VHDL-description.

```
-- VHDL description "circuit.vhd" generated from LGC-file "circuit.lgc"
ENTITY sn_A IS
PORT (
        n_3_0, n_3_1 : BUFFER bit;
        n_4_0, n_4_1, n_5_0, n_5_1, n_6_0, n_6_1 : IN bit );
END sn_A;

ARCHITECTURE sn_A_behavior OF sn_A IS
BEGIN
  PROCESS
    variable n_3_0_new, n_3_1_new : bit;
    variable tmpvar1 : bit;
  BEGIN
    tmpvar1 := n_4_1 AND n_5_1;
    n_3_0_new := n_6_1  OR  tmpvar1;
    tmpvar1 := n_4_0  OR  n_5_0;
    n_3_1_new := n_6_0  AND  tmpvar1;
    n_3_0 <= n_3_0_new;
    n_3_1 <= n_3_1_new;
    WAIT ON n_4_0, n_4_1, n_5_0, n_5_1, n_6_0,  n_6_1;
  END PROCESS;
END sn_A_behavior;


LIBRARY D_R_E;
USE D_R_E.Dual_Rail_Encoding.ALL;

ENTITY circuit IS
PORT (
     beta      : OUT X01;
     a, b, c   : IN  X01;
     Output    : OUT X01 );
END circuit;
```

```
ARCHITECTURE Structure of circuit IS
   SIGNAL beta_L, beta_H     : BIT;
   SIGNAL a_L, a_H           : BIT;
   SIGNAL b_L, b_H           : BIT;
   SIGNAL c_L, c_H           : BIT;
   SIGNAL Output_L,Output_H  : BIT;
-- Component declaration statements
BEGIN
     a_L <= Low_Rail(a);
     a_H <= High_Rail(a);
     b_L <= Low_Rail(b);
     b_H <= High_Rail(b);
     c_L <= Low_Rail(c);
     c_H <= High_Rail(c);
     U0 : sn_A PORT MAP(beta_L, beta_H, a_L, a_H,
                        b_L, b_H, c_L, c_H);
     U1 : sn_B PORT MAP(Output_L, Output_H,
                        beta_L, beta_H);
     beta <= Ternary(beta_L,beta_H);
     Output <= Ternary(Output_L,Output_H);
END Structure;
```