

# Using a VHDL Testbench for Transistor-Level Simulation and Energy Calculation

Anshul Singh and Scott C. Smith

*University of Missouri – Rolla, Department of Electrical and Computer Engineering*

*133 Emerson Electric Co. Hall, 1870 Miner Circle, Rolla, MO 65409*

*Phone: (573) 341-4232, Fax: (573) 341-4532,*

*E-mail: [smithsco@umr.edu](mailto:smithsco@umr.edu), [anshulsingh@gmail.com](mailto:anshulsingh@gmail.com)*

**Keywords:** Asynchronous circuits, delay-insensitive circuits, power, energy, NULL Convention Logic (NCL)

## Abstract

*This paper describes a procedure to simulate a transistor-level design using a VHDL testbench. Specifically, the VHDL testbench reads the transistor-level design's outputs and supplies the inputs accordingly. This setup also allows the testbench to check for functional correctness. This type of transistor-level simulation is absolutely necessary for asynchronous circuits because the inputs change relative to handshaking signals, which are not periodic, instead of changing relative to a periodic clock pulse, as do synchronous systems. The method further supports automated calculation of power and energy metrics.*

*This method is first demonstrated using a simple NULL Convention Logic (NCL) sequencer. It is then applied to two more complex NCL circuits, 4-bit  $\times$  4-bit unsigned dual-rail and quad-rail non-pipelined multipliers. Energy per operation is automatically calculated and compared for the two different multiplier architectures. An exhaustive testbench is used for both designs to simulate all input combinations; and the testbench checks for functional correctness, showing that both designs produce the desired output for all 256 input combinations.*

## 1. Introduction

For the last two decades the focus of digital design has been primarily on synchronous, clocked architectures. However, as clock rates have significantly increased while feature size has decreased, clock skew and fabrication process variations have become major problems. High performance chips must dedicate increasingly larger portions of their area for clock drivers to achieve acceptable skew, assuming normal (expected)

fabrication process variations, causing these chips to dissipate increasingly higher power, especially at the clock edge, when switching is most prevalent. As these trends continue, the clock is becoming more and more difficult to manage. This has caused renewed interest in asynchronous digital design. As the demand continues for higher performance, higher complexity, and decreased feature size, asynchronous paradigms will become more widely used in the industry. The International Technology Roadmap for Semiconductors predicts a likely shift from synchronous to asynchronous design styles in order to increase circuit robustness, decrease power, and alleviate many clock-related issues [1].

Delay-insensitive asynchronous paradigms, like NULL Convention Logic (NCL) [2], have numerous advantages over their clocked Boolean counterparts, including reduced timing effort, power, noise, and EMI, increased robustness and design reusability, and suitability for System-on-Chip (SoC) design. Delay-insensitivity also yields average-case, versus worst-case performance, no glitch power, and distributes the demand for power over time and area, reducing the occurrence of hot spots and peak power demand. However, asynchronous circuits haven't yet gained widespread industrial popularity due mostly to the lack of industry-standard CAD tool support.

Since low power is one of the main advantages of asynchronous systems, power/energy usage is an important benchmark for delay-insensitive (DI) circuits. The standard method for transistor-level simulation and power/energy calculation is to manually force the inputs (i.e. change the inputs at certain points in time). Many tools including the following use this method: Cadence, Powermill, and Mentor Graphics. However, there are many drawbacks for applying this approach to asynchronous circuits. For synchronous circuits, the inputs can be changed relative to a constant clock period,

making this forcing method acceptable. But for asynchronous circuits, the inputs must be changed based on handshaking signals; thus they do not change relative to a constant period, making the forcing method highly inadequate. Since asynchronous circuit inputs change relative to handshaking signals and not relative to a constant period, these handshaking signals must be read, such that the inputs can be changed based on the status of the handshaking signals. A VHDL testbench can do exactly this; thus the desire to use a VHDL testbench to control transistor-level simulations.

A VHDL testbench can read outputs and change inputs accordingly. This will allow the output handshaking signals to be able to control when a circuit's inputs are changed. Since an asynchronous circuit's outputs can change at any time, this ability to be able to control the circuit inputs based on the circuit outputs is absolutely necessary, especially for large designs. This VHDL testbench method of controlling a transistor-level simulation also has many other advantages that are applicable to both asynchronous and synchronous paradigms. First, the same VHDL testbench used to test a VHDL design can now also be used to test a transistor-level design, saving the time it would take to generate the forced inputs. Another advantage is the ability to automatically check the circuit outputs against calculated results, instead of doing this manually like in the forcing method. There are also several other advantages, such as being able to easily change input rise and fall times, automatically plot a variety of waveforms, automatically extract various characteristics and perform calculations, such as calculating energy per operation, etc. Hence, this tool can be used for both synchronous and asynchronous circuits, but is imperative for asynchronous circuits, where the inputs do not change relative to a constant clock frequency.

## 2. NCL Overview

NCL is a self-timed logic paradigm in which control is inherent in each datum. NCL follows the so-called weak conditions of Seitz's delay-insensitive signaling scheme [3]. Like other delay-insensitive logic methods, the NCL paradigm assumes that forks in wires are isochronic [4]. Various aspects of the paradigm, including the NULL (or spacer) logic state from which NCL derives its name, have origins in Muller's work on speed-independent circuits in the 1950s and 1960s [5].

### 2.1 Delay-Insensitivity

NCL uses symbolic completeness of expression to achieve delay-insensitive behavior. A symbolically complete expression depends only on the relationships of the symbols present in the expression without reference to

their time of evaluation [2]. In particular, dual-rail and quad-rail signals, or other mutually exclusive assertion groups (MEAGs), can incorporate data and control information into one mixed control/data path to eliminate time reference. For NCL and other circuits to be purely delay-insensitive, assuming isochronic wire forks [4], they must meet the input-completeness and observability criteria [6].

Completeness of input requires that all the outputs of a combinational circuit may not transition from NULL to DATA until all inputs have transitioned from NULL to DATA, and that all the outputs of a combinational circuit may not transition from DATA to NULL until all inputs have transitioned from DATA to NULL. In circuits with multiple outputs, it is acceptable, according to Seitz's weak conditions [3], for some of the outputs to transition without having a complete input set present, as long as all outputs cannot transition before all inputs arrive. Observability requires that no *orphans* may propagate through a gate [7]. An orphan is defined as a wire that transitions during the current DATA wavefront, but is not used in the determination of the output. Orphans are caused by wire forks and can be neglected through the isochronic fork assumption [4], as long as they are not allowed to cross a gate boundary. This *observability* condition, also referred to as *indicatability* or *stability*, ensures that every gate transition is observable at the output; which means that every gate that transitions is necessary to transition at least one of the outputs. Furthermore, when circuits use the bit-wise completion strategy with selective input-incomplete components, they must also adhere to the completion-completeness criterion [8], which requires that completion signals only be generated such that no two adjacent DATA wavefronts can interact within any combinational component.

Most multi-rail delay-insensitive systems, including NCL, have at least two register stages, one at both the input and the output. Two adjacent register stages interact through request and acknowledge lines,  $K_i$  and  $K_o$ , to prevent the current DATA wavefront from overwriting the previous DATA wavefront by ensuring that the two are always separated by a NULL wavefront.

### 2.2 Logic Gates

NCL differs from other delay-insensitive paradigms, like [2], which use only one type of state-holding gate, the C-element [5]. A C-element behaves as follows: when all inputs assume the same value, the output assumes this value; otherwise, the output does not change. On the other hand, all NCL gates are state-holding. NCL uses threshold gates as its basic logic elements [9]. The primary type of threshold gate is the  $TH_{mn}$  gate ( $1 \leq m \leq n$ ).  $TH_{mn}$  gates have  $n$  inputs. At least  $m$  of the  $n$  inputs must be asserted before the output becomes

asserted. Because NCL threshold gates are designed with *hysteresis*, all asserted inputs must be deasserted before the output is deasserted. Hysteresis ensures a complete transition of inputs back to NULL before asserting the output associated with the next wavefront of input data. NCL threshold gates may also include a *reset* input to initialize the output. Circuit diagrams designate resettable gates by either a *D* or an *N* appearing inside the gate, along with the gate's threshold. *D* denotes the gate as being reset to logic 1; *N*, to logic 0.

### 3. Previous Work

Since low power is one of the main advantages of asynchronous systems, power/energy usage is an important benchmark for DI circuits. Previous methods use Spice transistor-level tools for precise calculation of power or energy consumption. This can be done with Cadence, Powermill, or Mentor Graphics tools, among others. The following example pertains to using the Mentor Graphics tools for power/energy calculation. First, a structural VHDL description of the circuit is transformed to an EDIF netlist using the synthesis tool, Leonardo Spectrum. This file consists solely of threshold gates, which are in the form of black boxes. Next, the EDIF 200 Netlister tool is used to map the black box threshold gates to their equivalent transistor-level circuits. Schematic Generator is then used to create the transistor-level circuit diagram. Using Design Architect, the transistor-level circuit diagram is opened and its viewpoint is created. Finally, the transistor-level circuit is simulated using Accusim II, an analog circuit simulator, which yields the necessary voltage and current waveforms used to calculate power and energy with the Waveform Calculator.

Energy consumption and power dissipation are two metrics of assessing performance of VLSI systems, and are closely related to each other. Power is an instantaneous value, and is defined as the source voltage,  $V_{dd}$ , multiplied by the current through  $V_{dd}$ ,  $I_{vdd}$ . Average power can then be calculated by plotting  $I_{vdd} \times V_{dd}$  and finding the average value of this waveform. The total energy is equal to the area under the power waveform,  $\int (I_{vdd} \times V_{dd})$ .

A popular metric in synchronous circuits is *energy per clock cycle*. This is calculated by dividing the total energy for the simulation by the number of clock cycles in the simulation. Since there is no clock in DI circuits, a meaningful metric is *energy per operation* [10] (i.e. how much energy does it take to do one multiplication operation on average). The metric is a direct measure of the amount of work done per operation, providing a measure of how battery life will be affected, and is good for determining the effects that different architectures have on energy usage. A comparable value for

synchronous circuits can be calculated by multiplying the energy per clock cycle by the average number of clock cycles per operation, to obtain energy per operation. The equation for energy per operation is given below.

$$E = \frac{\int_0^t I_{vdd}(t) \times V_{dd} dt}{\text{number of operations}} \text{ Joules} \quad (1)$$

## 4. VHDL Testbench Simulation Method

*ADVance MS*, an extension of Mentor Graphics Eldo simulator, can be used to simulate a VHDL-AMS (analog/mixed-signal) model that includes Eldo/Spice subcircuits as components, an Eldo/Spice netlist with VHDL-AMS, VHDL, Verilog-AMS, and/or Verilog modules as components, or even a pure Eldo (or Eldo RF)/Spice netlist. An Eldo command file is necessary to define simulation parameters when the design to simulate contains analog or mixed analog/digital components, even if no Eldo netlist or subcircuits are used.

*ADVance MS* allows for multiple languages within the description of a single design. However, this is not a co-simulation. There is only one netlist (or description) even if different languages are used at different levels of hierarchy in the design. This produces a unified hierarchy, or netlist, using VHDL-AMS, Verilog-AMS, SPICE, or VHDL/Verilog on top of the design. The following configurations are possible:

- (a) VHDL-AMS-on-top
- (b) Eldo/Spice-on-top
- (c) ModelSim VHDL/Verilog-on-top
- (d) Verilog-AMS-on-top

### 4.1 Required Files

The method presented herein focuses on using a VHDL-on-top configuration to instantiate an Eldo/Spice child. The following files are required to simulate and use a top-VHDL entity for instantiating an Eldo/Spice subcircuit:

- (a) Spice Model File: This is the spice model parameter file that should be the same model file used to create viewpoints for each transistor-level schematic and main design schematic. This file can be obtained from the MOSIS website [11].
- (b) SUBCKT File: This file contains the Eldo/Spice netlist of the transistor-level circuit to be simulated. A subcircuit file has the extension `.ckt` and contains one or more Eldo subcircuits.
- (c) VHDL Entity: This file contains the entity description of the circuit to be simulated, including

the ports, which must be equivalent to the SPICE netlist ports in terms of name, number, and order. This VHDL entity does not contain a corresponding architecture. The SPICE subcircuit architecture will be attached to the VHDL entity. This entity will be used as a component in the top-VHDL entity.

- (d) Top-VHDL entity/testbench: This file uses the VHDL entity, described above, as the component to be simulated.
- (e) Command File: The command file is used for simulation commands, options, inserting A/D and D/A boundary elements, including the spice model library, declaring global variables, defining waveforms, extracting characteristics, etc.

## 4.2 Compilation and Simulation

The order in which commands are executed and files compiled is extremely important. The Mentor Graphics environment must also be set before compiling or simulating (i.e. set user library and working directory variables). Once the following commands are issued in the order specified below, ADVance MS can be invoked and the design can be loaded and simulated.

1. **valib spicelib**  
Creates a SPICE design library
2. **vcom VHDL\_entity\_file.vhd**  
Compiles the VHDL entity file
3. **vaspi -f VHDL\_entity\_name subcircuit\_name@subcircuit\_file\_name.ckt**  
Creates a SPICE architecture for the VHDL entity previously compiled with the VHDL compiler using the vcom command
4. **valib admslib**  
Creates an ADMS design library
5. **vacom top-level\_VHDL\_file.vhd -ms**  
Compiles and imports the top-level VHDL entity into the ADMS design library
6. **vasim**  
Invokes ADVance MS

Once ADVance MS is invoked, the design and command file can be loaded and the simulation run. This can also be automated by using a simulation macro.

## 5. Application to NCL Sequencer

To illustrate simulating a transistor-level circuit using a VHDL testbench, the method is applied to a 4-stage NCL sequencer [12]. The first 6 steps consist of obtaining the transistor-level design of the circuit to simulate. This can be generated from the VHDL structural model, as overviewed in Section 3.

1. Transform the structural VHDL description of the circuit to an EDIF netlist using Leonardo Spectrum. The original VHDL description is shown in Figure 1.

```
library ieee;
use ieee.std_logic_1164.all;
entity sequencer is
    port (ki, rst: IN std_logic;
          s1, s2: OUT std_logic);
end sequencer;
architecture arch of sequencer is
    signal d0, d1, d2, d3: std_logic;
    signal r0, r1, r2, r3: std_logic;
    component th33nx0
        port (a: in std_logic;
              b: in std_logic;
              c: in std_logic;
              rst: in std_logic;
              z: out std_logic);
    end component;
    component th33dx0
        port (a: in std_logic;
              b: in std_logic;
              c: in std_logic;
              rst: in std_logic;
              z: out std_logic);
    end component;
    component invx0
        port (i: in std_logic;
              zb: out std_logic);
    end component;
begin
    g0: th33nx0 port map(ki, d3, r1, rst, d0);
    g1: th33dx0 port map(ki, d0, r2, rst, d1);
    g2: th33nx0 port map(ki, d1, r3, rst, d2);
    g3: th33nx0 port map(ki, d2, r0, rst, d3);
    i0: invx0 port map(d0, r0);
    i1: invx0 port map(d1, r1);
    i2: invx0 port map(d2, r2);
    i3: invx0 port map(d3, r3);
    s1 <= d2;
    s2 <= d0;
end arch;
```

Figure 1. Sequencer VHDL description.

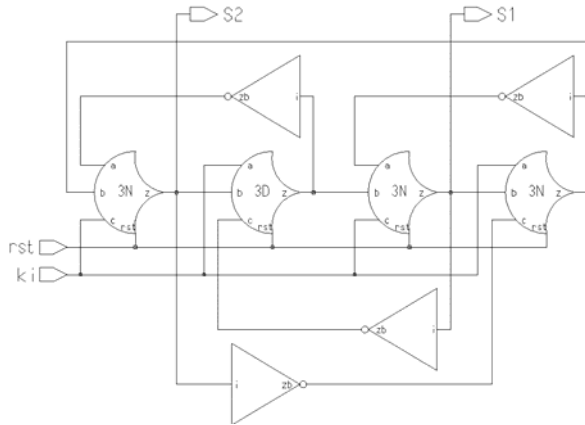
2. **enread sequencer.edf -rcf map.cfg**  
Maps the EDIF components to their respective transistor-level equivalent circuits. The configuration file, map.cfg, is shown in Figure 2.

```
map library $USERLIB/th33dx0 work/th33dx0 -external
map library $USERLIB/th33nx0 work/th33nx0 -external
map library $USERLIB/invx0 work/invx0 -external
```

Figure 2. Configuration gate mapping file.

3. Generate the transistor level circuit diagram. Open Schematic Generator and choose File → Open component from Model from the pull down menu. Click on the Navigator button and choose work/sequencer/arch. Click OK twice; hit F2 to load the generic library components; then hit F3 to generate the schematic. Save the design and exit.

4. Create the design viewpoint for the generated schematic, work/sequencer/arch, using the Spice model file. For large circuits, multi-sheet designs may be generated; hence, a viewpoint must be created for each sheet. The generated schematic is shown in Figure 3.



**Figure 3. Generated sequencer schematic.**

5. Open AccuSim II from within Design Manager and select work/sequencer/default. From the drop down menu select Report → Netlist → Write Netlist. A Report Netlist/Write Netlist to Named File window will popup. Enter sequencer.ckt and click OK.
6. Open this netlist/subcircuit file in a text editor for modification. Delete the .options, .temp, and V\_Dcinit lines. Add the subcircuit name and ports to the 2<sup>nd</sup> line of the netlist file. Make sure that the subcircuitname, port names, and order is the same as that in the VHDL entity in Figure 1. The 1<sup>st</sup> line of the netlist file can be used for the title of the subcircuit, so no command or declaration should be given. \* is used for commenting in netlist and command files. Next, delete all contents after and including .op NO\_SMALL\_SIGNAL. Finally, add .ENDS as the last line. The modified sequencer.ckt file is shown in Figure 4.

```
Design: /home/g2/astxb/NCL/SEQ_TRAN/sequencer.default
.SUBCKT seq ki rst s1 s2
M_I$577_M103 I$577_N$217 ki 0 0 cmosn W=12u 2u
M_I$577_M110 I$577_N$11 rst 0 0 cmosn W=4u L=2u
.
.
.
M_I$576_M2 I$576_N$9 d1 I$576_N$4 VCC cmosp W=24u L=2u
.ENDS
```

**Figure 4. Modified sequencer subcircuit netlist.**

7. Create a VHDL entity that lists the ports and parameters of the SPICE subcircuit created in the steps above. This is shown in Figure 5.

```
Library IEEE;
use IEEE.std_logic_1164.all;
entity SEQ is
port(ki, rst : in std_logic;
      s1, s2: out std_logic);
end SEQ;
```

**Figure 5. VHDL entity.**

8. Augment the VHDL testbench to use the transistor-level circuit instead of the VHDL version. Include the spicelib library; change the unit-under-test (UUT) component; and port map the component. Note that if the design is small, additional wait statements may have to be added in the testbench to let the outputs stabilize before changing the inputs. This testbench is shown in Figure 6.

```
Library IEEE;
use IEEE.std_logic_1164.all;
library spicelib;
entity tb_seq is
end;
architecture TESTBENCH of tb_seq is
signal rst,ki,s1,s2: std_logic;
component SEQ
port(ki, rst : in std_logic;
      s1, s2: out std_logic);
end component;
begin
  UUT:ENTITY spicelib.seq
  port map(ki=>ki,rst=>rst,s1=>s1,s2=>s2);
  OUTPUTS: process
  begin
    rst <= '1';
    ki <= '0';
    wait until S1 = '0' and S2 = '0';
    wait for 5 ns;
    rst <= '0';
    ki <= '1';
    wait until S1 = '1';
    wait for 5 ns;
    ki <= '0';
    wait until S1 = '0';
    wait for 5 ns;
    ki <= '1';
    .
    .
    .
    wait until S2 = '1';
    wait for 5 ns;
    ki <='0';
    wait until S2 = '0';
    wait;
  end process;
end TESTBENCH;
```

**Figure 6. Modified sequencer testbench.**

9. Create a command file to control the simulation, as shown in Figure 7. The first line includes the Spice model file. The second and third lines define  $V_{cc}$  as 3.3V. The next four lines deal with inserting Digital to Analog (D2A) and Analog to Digital (A2D) converters so that the digital testbench can communicate with the analog transistor-level simulation. The `.DEFHOOK` command allows *ADVance MS* to automatically insert the appropriate boundary elements (i.e. D2A for inputs and A2D for outputs). The following line specifies that a transient simulation is to be run for 100 ns, with a time step of 0.1 ns. The `.OPTPWL` command is used to set the tolerance parameter, `RELTOL`, to  $10^{-5}$  s, instead of using the default value,  $10^{-3}$  s. Decreasing `RELTOL` increases measurement accuracy, but it also increases simulation time. For the sequencer example, decreasing `RELTOL` resulted in a much smoother power curve without spurious transitions; however, the energy calculation was within 2% using the default value. The command `.OPTION AEX` is used to save extracted waveform characteristics, such as the total energy usage, in a file with the same name as the command file, but with the extension `.aex`. The last three lines define and plot the power waveform, and extract the circuit's total energy usage from it.

```
.LIB spice
.GLOBAL VCC
VCC VCC 0 DC 3.3
.MODEL d2a bit D2A MODE=std_logic
+vhi=3.3 vlo=0.0 trise=1n tfall=1n
.MODEL a2d bit A2D MODE=std_logic vth=1.65
.DEFHOOK d2a bit a2d_bit
.TRAN 0.1ns 100ns
.OPTPWL RELTOL=(0,1e-5s)
.OPTION AEX
.defwave power= (3.3*(I(VCC)))
.plot tran w(power)
.extract integ(w(power))
```

**Figure 7. Sequencer command file.**

10. Create a simulation macro to control which signals are viewed. This step is optional; instead the *ADVance MS* GUI can be used. The advantage to using a simulation macro is that it automatically configures the simulation, so you don't have to manually do it each time. The simulation macro for the sequencer is shown in Figure 8. This is the same macro used for the VHDL simulation, except for the additional first line.

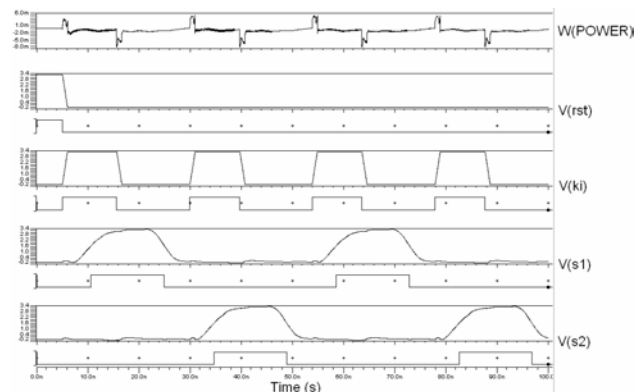
```
view structure nets wave
add wave :tb_seq:uut:rst
add wave :tb_seq:uut:ki
add wave :tb_seq:uut:s1
add wave :tb_seq:uut:s2
run -all
```

**Figure 8. Sequencer simulation macro.**

11. The circuit is now ready to simulate. Issue the commands given in Section 4.2 in the specified order. For the sequencer example, the following commands would be run:

- `valib spicelib`
- `vcom seq.vhd`
- `vaspi -f seq seq@sequencer.ckt`
- `valib admslib`
- `vacom tb_sequencer.vhd -ms`
- `vasim -cmd com.cmd tb_seq -do seq.do`

The last command will automatically load the design, simulate it as specified in the command file, and plot the waveforms specified in the command file and simulation macro. Figure 9 shows the resulting sequencer simulation. Notice that both the digital and analog waveforms are displayed, along with the power waveform, as specified in the command file. The total energy used during the simulation was extracted, as specified in the command file, and stored in `com.aex`. This value was 78 pJ. Since there are four operations, the energy per operation, as specified in Equation 1, is 19.5 pJ.



**Figure 9. Sequencer simulation waveforms.**

## 6. Application to NCL Multipliers

The previous section detailed the step-by-step process for simulating a transistor-level circuit using a VHDL testbench. This section applies that method to two unsigned 4-bit  $\times$  4-bit non-pipelined multipliers, a dual-rail [13] and quad-rail [14] version. Both simulations used an exhaustive, 256 input combination, testbench that

automatically checks for functional correctness. If any output result was not correct, an *incorrect* signal would be asserted and remain asserted throughout the simulation. *incorrect* remained deasserted throughout the entire simulation for both architectures, showing that both designs produced the desired output for all 256 input combinations. The energy per operation for the quad-rail and dual-rail multipliers was calculated as 529 pJ and 736 pJ, respectively. Analyzing these results shows that the dual-rail version requires 39% more energy than the quad-rail version. This is expected because there are half as many signals transitions for quad-rail logic (i.e. two dual-rail signals transition for each corresponding quad-rail signal transition). The dual-rail version is however 8% faster. Note that since these are larger designs, with more delay, no additional delay had to be added in the testbench, as was necessary for the sequencer example.

## 7. Conclusions

This paper describes one approach for simulating a transistor-level design with a VHDL testbench, using the Mentor Graphics digital design tool suite. This transistor-level simulation method is absolutely necessary for asynchronous circuits because the inputs do not change relative to a periodic clock pulse, but instead change value based on handshaking signals. The method supports automatic calculation of power and energy metrics and automated testing of functional correctness. Furthermore, this method will work equally well for synchronous circuits.

The approach developed herein was applied to three different NCL circuits, and their transistor-level designs were successfully simulated using self-checking exhaustive VHDL testbenches. Two of the designs were non-pipelined unsigned 4-bit  $\times$  4-bit dual-rail and quad-rail NCL multipliers. As expected, the quad-rail multiplier consumed less energy per operation than the dual-rail version.

## References

- [1] [public.itrs.net/Files/2003ITRS/Home2003.htm](http://public.itrs.net/Files/2003ITRS/Home2003.htm) (available February 16, 2005).
- [2] K. M. Fant and S. A. Brandt, "NULL Convention Logic: A Complete and Consistent Logic for Asynchronous Digital Circuit Synthesis," *International Conference on Application Specific Systems, Architectures, and Processors*, pp. 261-273, 1996.
- [3] C. L. Seitz, "System Timing," in *Introduction to VLSI Systems*, Addison-Wesley, pp. 218-262, 1980.
- [4] C. H. (Kees) van Berkel, M. Rem, and R. Saeijs, "VLSI Programming," *1988 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 152-156, 1998.
- [5] D. E. Muller, "Asynchronous Logics and Application to Information Processing," in *Switching Theory in Space Technology*, Stanford University Press, pp. 289-297, 1963.
- [6] S. C. Smith, R. F. DeMara, J. S. Yuan, D. Ferguson, and D. Lamb, "Optimization of NULL Convention Self-Timed Circuits," *Integration, The VLSI Journal*, Vol. 37/3, pp. 135-165, August 2004.
- [7] A. Kondratyev, L. Neukom, O. Roig, A. Taubin, and K. Fant, "Checking Delay-Insensitivity:  $10^4$  Gates and Beyond," *Eighth International Symposium on Asynchronous Circuits and Systems*, pp. 149-157, April 2002.
- [8] S. C. Smith, "Completion-Completeness for NULL Convention Digital Circuits Utilizing the Bit-wise Completion Strategy," *The 2003 International Conference on VLSI*, pp. 143-149, June 2003.
- [9] G. E. Sobelman and K. M. Fant, "CMOS Circuit Design of Threshold Gates with Hysteresis," *IEEE International Symposium on Circuits and Systems (II)*, pp. 61-65, 1998.
- [10] P. A. Beerel, C. Hsieh and S. Wadekar, "Estimation of Energy Consumption in Speed-Independent Control Circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 16/ 6, pp. 672-680, 1996.
- [11] <http://www.mosis.org/Technical/Testdata/> (available February 16, 2005).
- [12] S. C. Smith, R. F. DeMara, J. S. Yuan, M. Hagedorn, and D. Ferguson, "Speedup of Delay-Insensitive Digital Systems Using NULL Cycle Reduction," *The 10<sup>th</sup> International Workshop on Logic and Synthesis*, pp. 185-189, June 2001.
- [13] S. C. Smith, R. F. DeMara, J. S. Yuan, M. Hagedorn, and D. Ferguson, "Delay-Insensitive Gate-Level Pipelining," *Integration, The VLSI Journal*, Vol. 30/2, pp. 103-131, 2001.
- [14] S. K. Bandapati, S. C. Smith, and M. Choi, "Design and Characterization of NULL Convention Self-Timed Multipliers," *IEEE Design and Test of Computers: Special Issue on Clockless VLSI Design*, Vol. 30/6, pp. 26-36, November-December 2003.