Microelectronics Journal ∎ (∎∎∎∎) ∎∎∎–∎∎∎

Microelectronics Journal

www.elsevier.com/locate/mejo

Automated energy calculation and estimation for delay-insensitive digital circuits

Venkat Satagopan, Bonita Bhaskaran, Anshul Singh, Scott C. Smith*

Department of Electrical Engineering, University of Arkansas, 3185 Bell Engineering Center, Fayetteville, AR 72701

Received 4 January 2006; accepted 21 August 2007

Abstract

With increasingly smaller feature sizes and higher on-chip densities, the power dissipation of VLSI systems has become a primary concern for designers. This paper first describes a procedure to simulate a transistor-level design using a VHDL testbench, and then presents a fast and efficient energy estimation approach for delay-insensitive (DI) systems, based on gate-level switching. Specifically, the VHDL testbench reads the transistor-level design's outputs and supplies the inputs accordingly, also allowing for automatic checking of 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. The energy estimation approach produces results three orders of magnitude faster than transistor-level simulation, and has been automated and works with standard industrial design tool suites, such as Mentor Graphics and Synopsys.

Both methods are applied to the NULL Convention Logic (NCL) DI paradigm, and are first demonstrated using a simple NCL sequencer, and then tested on a number of different NCL 4-bit $\times$ 4-bit unsigned multiplier architectures. Energy per operation is automatically calculated for both methods, using an exhaustive testbench to simulate all input combinations and to check for functional correctness. The results show that both methods produce the desired output for all circuits, and that the gate-level switching approach developed herein produces results more than 1000 times as fast as transistor-level simulation, that fall within the range obtained by two different industry-standard transistor-level simulators. Hence, the developed energy estimation method is extremely useful for quickly determining how architecture changes affect energy usage.

Keywords: Asynchronous circuits; Energy; Power; Transistor-level simulation; VHDL

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 demand continues for higher performance, higher complexity, and decreased feature size, asynchronous paradigms will become more widely used in the industry. The 2003 International Technology Roadmap for Semiconductors [1] predicts that asynchronous paradigms will become more widely used in the industry to increase circuit robustness, decrease power, and alleviate many clock-related issues; and the 2005 edition [2] estimates that asynchronous circuits will account for 19% of chip area within the next 5 years, and 30% of chip area within the next 10 years.

*Corresponding author. Tel.: +1 479 575 6587; fax: +1 479 575 7967.
E-mail addresses: vsatagopan@nvidia.com (V. Satagopan), bbhaskaran@nvidia.com (B. Bhaskaran), anshulsingh@gmail.com (A. Singh), smithsco@uark.edu (S.C. Smith).
URL: http://comp.uark.edu/~smithsco (S.C. Smith).

0026-2692/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mejo.2007.08.004

Delay-insensitive (DI) asynchronous paradigms, like NULL Convention Logic (NCL) [3], have numerous advantages over their clocked Boolean counterparts, including reduced timing effort, power, noise, and electromagnetic interference, increased robustness and design reusability, and suitability for System-on-Chip 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 [4]. However, asynchronous circuits have not 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 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 predetermined points in time). Many tools including the following use this method: Cadence, Synopsys, and Mentor Graphics. However, there are numerous 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 signal outputs; thus, inputs do not change relative to a constant period, making the forcing method highly inadequate. Since asynchronous circuit inputs change relative to external handshaking signals and not relative to a constant period, these handshaking signals must be read by the tool, such that the inputs can be changed based on the status of the handshaking outputs.

A VHDL testbench allows input signals to be changed in response to output signal changes, hence, the desire to use a VHDL testbench to control transistor-level simulations. 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 additional advantages that are applicable to both asynchronous and synchronous paradigms: (1) the same VHDL testbench for testing the VHDL design can also be used to test the transistor-level design, saving time and effort to generate forced inputs; (2) circuit outputs can be automatically checked against calculated results; (3) input rise and fall times can be easily changed; and (4) a variety of waveforms can be plotted, various characteristics extracted, and calculations performed, such as calculation of energy per operation. 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.

Traditional transistor-level simulators are quite powerful and yield very accurate results; however, they require an extremely long runtime even for moderately sized circuits. DI circuits do not rely on a global clock tree network; instead, they use local handshaking signals for synchronization. Furthermore, they only consume significant energy when performing useful work; the rest of the time no switching occurs, so energy consumption is nominal, when using static CMOS gates. It is therefore feasible to utilize gate-level switching instead of transistor-level switching for estimating energy consumption of DI circuits, which promises to dramatically decrease simulation time.

This paper is divided into seven sections. Section 2 gives a brief overview of NCL. Section 3 overviews the previous work, including the derivation of an energy metric for asynchronous circuits, transistor-level simulation, and gate-level energy estimation. Section 4 describes the VHDL-controlled transistor-level simulation method developed herein; Section 5 details the gate-level energy estimation technique developed by the authors; Section 6 compares the two methods developed herein with each other and with a traditional transistor-level simulation technique, as applied to calculating energy consumption for a number of different NCL multiplier architectures; and Section 7 provides conclusions.

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 DI signaling scheme [5]. Like other DI logic methods, the NCL paradigm assumes that forks in wires are isochronic [6]. 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 [7].

2.1. Delay insensitivity

NCL utilizes symbolic completeness of expression to achieve DI behavior. A symbolically complete expression depends only on the relationships of the symbols present in the expression without reference to their time of evaluation [3]. In particular, dual-rail and quad-rail signals, or other mutually exclusive assertion groups, incorporate data and control information into one mixed control/data path to eliminate time reference. For NCL, and other DI paradigms, to be DI, assuming isochronic wire forks [6], they must meet the input-completeness and observability criteria [8].

Completeness of input requires that all outputs of a combinational circuit may not transition from NULL to DATA until all inputs have transitioned from NULL to DATA, and that all 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 [5], 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 [9]. 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 [6], 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, meaning that each gate transition is required 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 [10], which requires that completion signals only be generated such that no two adjacent DATA wavefronts can interact within any combinational component.

Most multi-rail DI 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 DI paradigms, like [5], which use only one type of state-holding gate, the C-element [7]. 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 [11]. The primary type of threshold gate is the TH*mn* gate ($1 \leqslant m \leqslant n$). 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 their 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, and N to logic 0.

3. Previous work

3.1. Energy metric

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 $I_{\mathrm{Vdd}} \times V_{\mathrm{dd}}$ and finding the average value of this waveform. The total energy is equal to the area under the power waveform, $\int (I_{\mathrm{Vdd}} \times V_{\mathrm{dd}})$.

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 asynchronous circuits, a meaningful metric is *energy per operation* [12] (e.g., the amount of energy it takes to perform one multiplication operation on average). The metric is a direct measure of the average 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 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:

$$E = \frac{\int_0^t I_{\mathrm{VDD}}(t) V_{\mathrm{DD}}\, \mathrm{d}t}{\text{number of operations}}\ \mathrm{J}. \tag{1}$$

3.2. Transistor-level simulation

Transistor-level simulation can be performed with Cadence, Synopsys, or Mentor Graphics tools, among others, for precise calculation of power or energy consumption. The following example pertains to using the Mentor Graphics toolset 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 automatically generate 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.

This process is extensive and the simulation time is very long. Another major drawback to this technique is that the designer needs to manually create and specify force files for the circuit inputs. This is a well-established method for synchronous circuits, and is easy to use when the inputs change relative to a periodic clock pulse. However, the inputs to DI systems change in response to handshaking outputs, which are not periodic. Therefore, changing the inputs at regular intervals is not appropriate.

3.3. Energy estimation

Since transistor-level simulation is an extremely time consuming process, it would be highly desirable to be able to estimate energy consumption by only considering the switching activity of gates within a circuit, without having to consider switching of the internal circuitry, or transistors, within each gate. To this end, a technique was developed in [13] for DI systems to estimate the probability, for each gate, that it would switch during the current DATA wavefront. These probabilities could then be used to calculate the number of gates that did indeed switch during the current DATA wavefront, *Num*, which could then be used in the following equation [14] to calculate energy consumption:

$$E = \sum_{\text{gates}} \frac{1}{2} C_{\text{gate-load}} V_{\text{dd}}^2 \,\text{Num}. \tag{2}$$

The paper applied the approach to calculate the switching activity of an unsigned NCL 4-bit × 4-bit dual-rail multiplier [15], and compared the calculated switching activity to that measured through VHDL simulation, showing that the calculated switching activity was fairly accurate. However, the calculated switching activity was not used to estimate the circuit's energy consumption or compared with the circuit's actual energy consumption. Another drawback of this approach is that a gate's switching probability is circuit dependent; hence each circuit would have to be extensively analyzed to calculate each gate's switching probability, before energy could be estimated.

4. VHDL-controlled transistor-level simulation

ADVance MS (ADMS), 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.

ADMS 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, Eldo/Spice, or Modelsim VHDL/Verilog on top of the design.

4.1. Files

The method presented herein uses a VHDL-on-top configuration to instantiate an Eldo/Spice child (i.e., the transistor-level circuit is the unit-under-test within the top-level VHDL testbench). The following files are required to simulate and use a top-level VHDL architecture for instantiating an Eldo/Spice subcircuit:

(a) *Spice Model File*: This is the Spice model parameter file that contains the attributes of the technology being used, and 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 [16].
(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 subcircuit(s).
(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-level VHDL architecture.
(d) *Top-level VHDL Architecture (i.e., testbench)*: This file uses the VHDL entity, described above, as the component to be simulated, generates the circuit inputs, and tests the outputs for functional correctness, if desired.
(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. Commands

The order in which commands are executed and files complied, as shown in Fig. 1, 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 ADMS is invoked, the design and command file can be loaded and the simulation run. This step can also be automated by using a simulation macro.

4.3. NCL sequencer example

To illustrate simulating a transistor-level circuit using a VHDL testbench, the method is applied to a 4-stage NCL sequencer [17,18]. The first six 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 3.2.

(1) Transform the structural VHDL description of the circuit to an EDIF netlist (sequencer.edf) using Leonardo Spectrum. The original VHDL description is shown in Fig. 2.
(2) Issue the command: **enread sequencer.edf –rcf map.cfg** to map the EDIF components to their respective transistor-level equivalent circuits. The configuration file, `map.cfg`, is shown in Fig. 3.

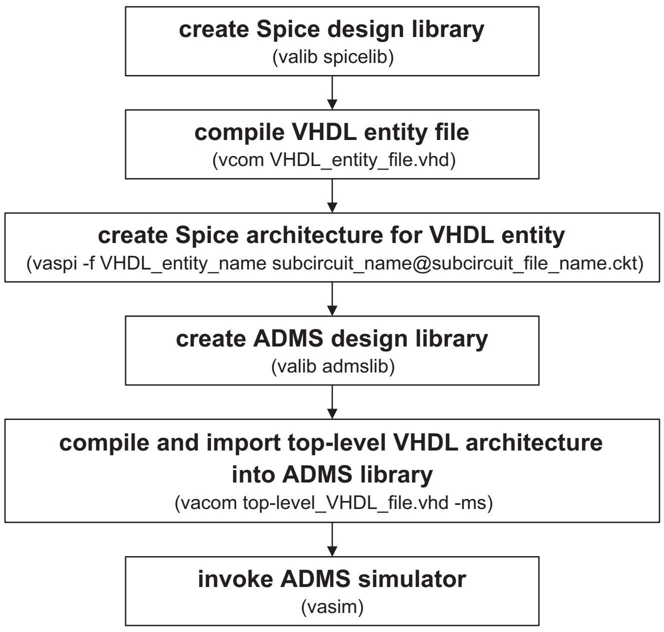

create Spice design library
(valib spicelib)
compile VHDL entity file
(vcom VHDL_entity_file.vhd)
create Spice architecture for VHDL entity
(vaspi -f VHDL_entity_name subcircuit_name@subcircuit_file_name.ckt)
create ADMS design library
(valib admslib)
compile and import top-level VHDL architecture into ADMS library
(vacom top-level_VHDL_file.vhd -ms)
invoke ADMS simulator
(vasim)

Fig. 1. ADMS command sequence.

(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; press the F2 key to load the generic library components; press the F3 key 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 Fig. 4.

(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 2nd line of the netlist file. Make sure that the port names are the same as that in the VHDL entity in Fig. 2. The 1st line of the netlist file can be used for the title of the subcircuit, so no command or declaration should be given. Note that * 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 Fig. 5.

(7) Create a VHDL entity for the Spice subcircuit created in the steps above. The entity name, port names, and port order must be the same as in the netlist file

```
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;
```

Fig. 2. Sequencer VHDL description.

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

Fig. 3. Configuration gate mapping file.

created in the previous step. The port directions must be the same as in the structural VHDL file from Step 1. The VHDL entity, `seq.vhd`, is shown in Fig. 6.

(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 to the transistor-level netlist; and port map this 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 depends on the threshold level of the A/D converters, which is specified in the command file, and on the delay through the circuit. In this case, a 5 ns delay was added after the output change was detected and before the *Ki* input changed accordingly, since the A/D converter threshold voltage was set as 1/2 V_{cc} and the circuit delay was

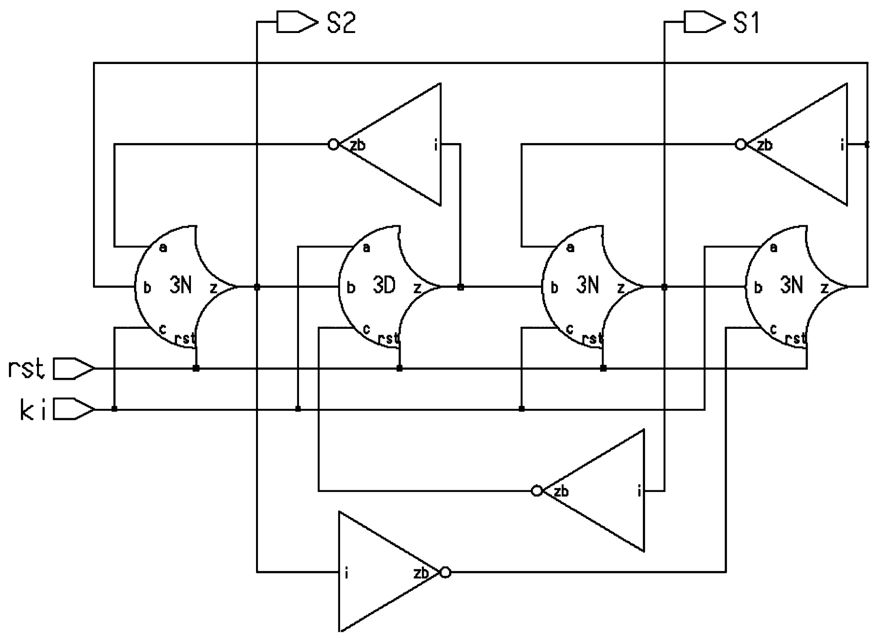


Fig. 4. Generated sequencer schematic.

```
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
```

Fig. 5. Modified sequencer subcircuit netlist.

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

Fig. 6. Sequencer VHDL entity.

small. This testbench, `tb_seq.vhd`, is shown in Fig. 7.

(9) Create a command file to control the simulation, as shown in Fig. 8. The first line includes the Spice model file. The second and third lines define V_{cc} as 3.3 V. The next four lines deal with inserting Digital to Analog and Analog to Digital converters so that the digital testbench can communicate with the analog transistor-level simulation. The D/A converters are specified as having rise and fall times of 1 ns, and the A/D converters are specified as having a threshold voltage of 1.65 V, which is 1/2 V_{cc}. The `.DEFHOOK` command allows ADMS to automatically insert the appropriate boundary elements (i.e., D/A for inputs and A/D for outputs). The following line specifies that a transient simulation is to be run for 100 ns, with a time step of

```
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;
```

Fig. 7. Modified sequencer testbench.

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`

```
.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))
```

Fig. 8. Sequencer command file.

increases measurement accuracy, but 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 total energy usage, in a file with the same name as the command file, but with extension `.aex`. The last three lines define and plot the power waveform, and extract the circuit's total energy usage from it.

(10) Create a simulation macro to control which signals are viewed. This step is optional; instead, the ADMS GUI can be used. The advantage to using a simulation macro is that it automatically configures the simulation, so this does not need to be done manually each time. The simulation macro for the sequencer, `seq.do`, is shown in Fig. 9. This is the same macro used for the VHDL simulation, except for the additional first line.

(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:
 (i) valib spicelib
 (ii) vcom seq.vhd
 (iii) vaspi –f seq seq@sequencer.ckt
 (iv) valib admslib
 (v) vacom tb_seq.vhd –ms
 (vi) 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. Fig. 10 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 average energy per operation, as specified in Eq. (1), is 19.5 pJ.

```
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
```

Fig. 9. Sequencer simulation macro.

5. Energy estimation technique for DI circuits

There are three main contributing factors to power dissipation in CMOS VLSI systems: dynamic, short-circuit, and static power. Dynamic dissipation results whenever there is a transition in the outputs due to the charging and discharging of capacitance. Short-circuit dissipation occurs when the p-type and n-type networks are both momentarily ON, leading to a direct connection path between V_{dd} and *ground*. This also occurs during output transitions. Static dissipation is due to leakage current in the transistors when they are not switching. Ideally, static CMOS circuits have very little leakage current; however, the leakage currents do become significant for very-high-density chips. In DI CMOS circuits, static power dissipation is negligible compared to the dynamic and short-circuit power; therefore, only the switching power needs to be taken into consideration when estimating energy consumption. Hence, a significant amount of energy is only used when a gate transitions.

5.1. Assumptions

Fig. 11 shows a transistor-level simulation of an NCL TH33 gate, and its corresponding energy waveform, using the Mentor Graphics Accusim II tool. There are a few key points to note that make the gate-level switching approach developed herein viable for NCL DI systems:

(a) Inputs continuously alternate between DATA and NULL, such that all wires asserted in a DATA wavefront are always deasserted in the corresponding NULL wavefront. Fig. 11 shows that NCL gates that do not transition during a DATA wavefront utilize an insignificant amount of energy, which can be ignored. Specifically, from 0 to 50 ns, two inputs, A and B, of the TH33 gate are asserted during the DATA wavefront, but the 3rd is not asserted, so the output, Z, is not asserted. The inputs are then deasserted during the NULL wavefront. Fig. 11b shows that these transitions only require 0.07 pJ, whereas approximately 33 times more energy (2.34 pJ) is required when the gate switches; hence, the non-switching energy is insignificant and can be ignored.

(b) Fig. 11b shows that the static power is indeed negligible, since the energy only changes when the inputs change, and remains relatively constant otherwise; hence, this static power can also be ignored.

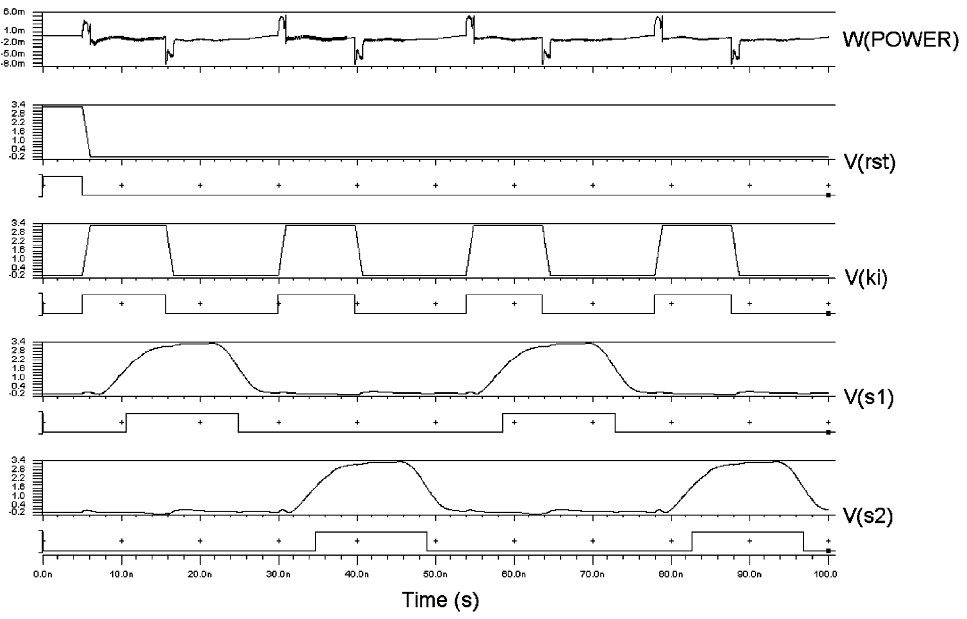


Fig. 10. Sequencer simulation waveforms.

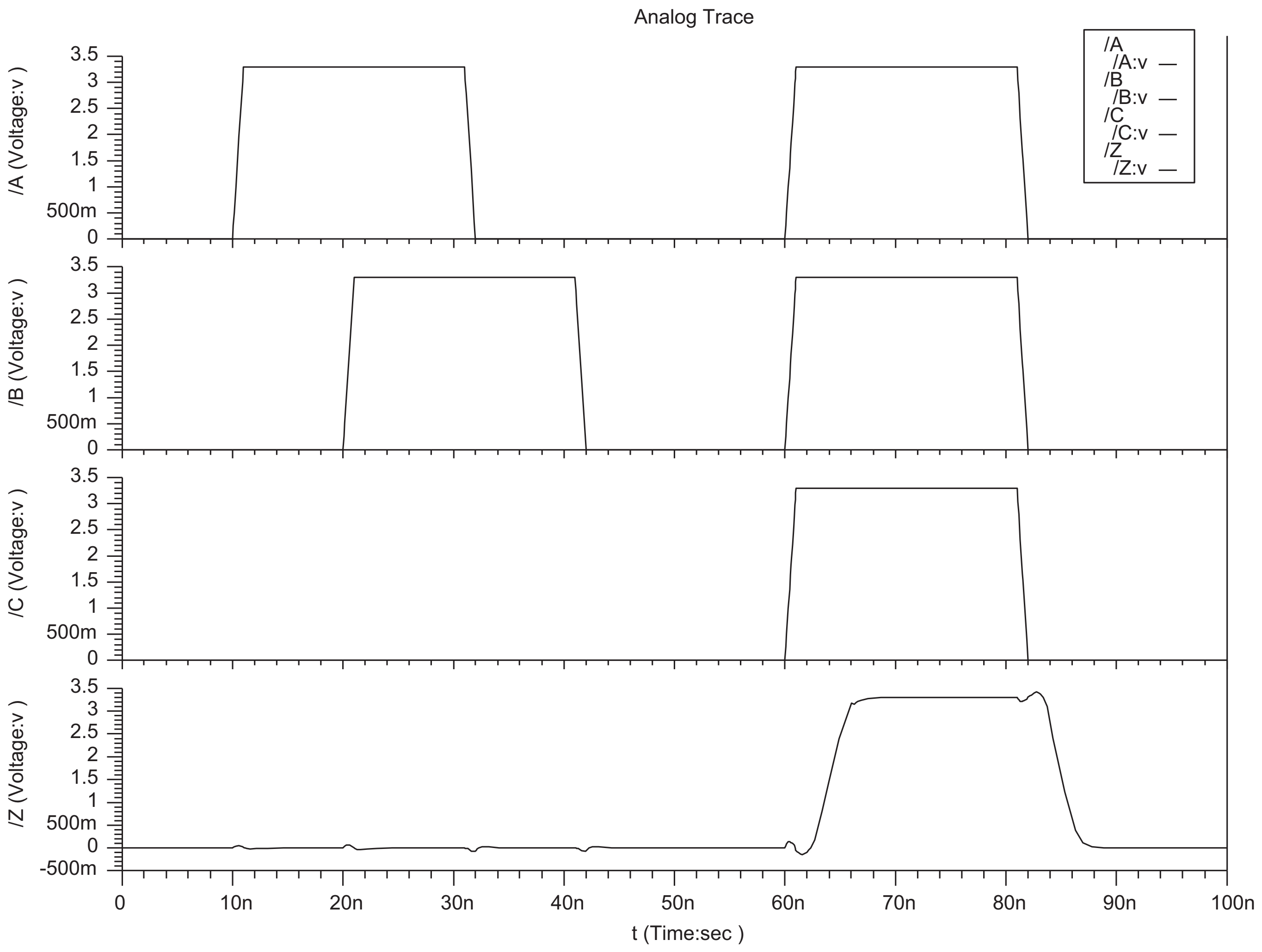


Fig. 11. (a) TH33 gate I/O waveforms; (b) TH33 gate energy waveform.

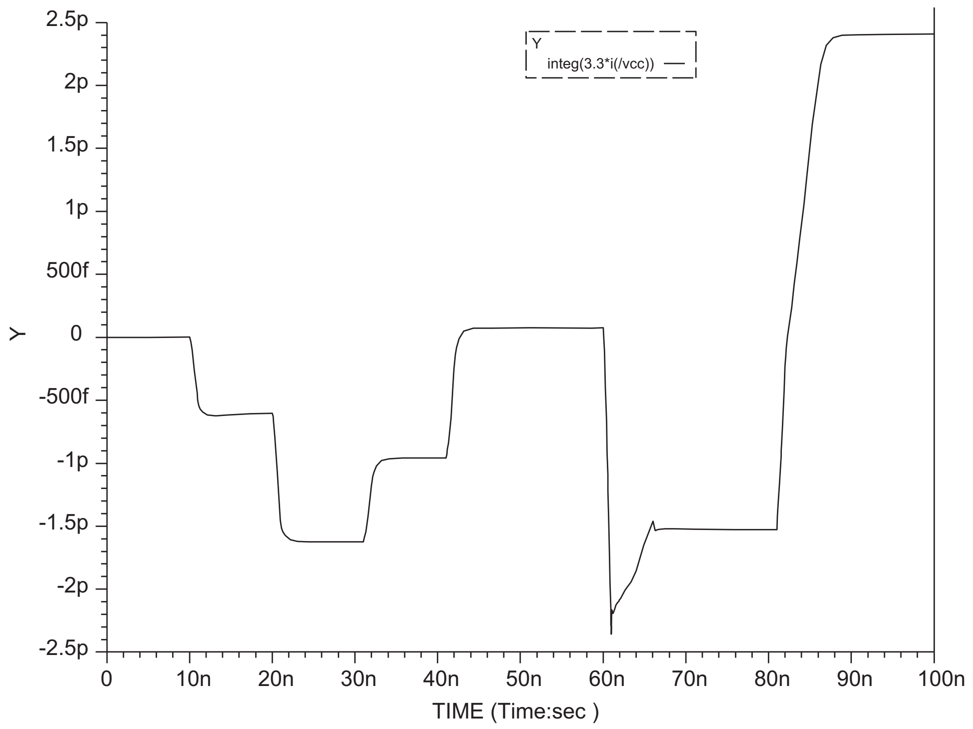


Fig. 11. (*Continued*)

(c) Since inputs continuously alternate between DATA and NULL, such that all wires asserted in a DATA wavefront are always deasserted in the corresponding NULL wavefront, all gates asserted in the DATA wavefront will be deasserted in the NULL wavefront. Therefore, the total energy for a gate to transition ($0 \rightarrow 1 \rightarrow 0$) can be added on the $0 \rightarrow 1$ transition, with no energy added for the $1 \rightarrow 0$ transition.

(d) Also note that the order and timing in which the inputs transition does not significantly change the amount of energy used in the gate transition (i.e., for the TH33 gate, asserting *A*, then *B*, then *C*, requires approximately the same amount of energy as when *A*, *B*, and *C* are all asserted simultaneously).

(e) However, for gates with multiple set conditions, the specific asserted input set causing the gate to become asserted does significantly affect the amount of energy required to assert the gate's output, because each set condition corresponds to a unique transistor network having a different capacitance to be charged and discharged. Hence, a different energy for each set condition must be used by the estimation tool for every gate. Take for example a TH23 gate. The transition due to *AB* requires 5.85 pJ; the one due to *AC* requires 5.07 pJ; and the transition due to *BC* requires 3.83 pJ. Note that asserting the 3rd input once the gate is already asserted does not require any significant energy; hence, this transition can be ignored.

(f) Finally, note that NCL systems adhere to monotonic transitions between DATA and NULL, which means that there are no glitches. This is important because glitches can require a significant amount of energy, and would be hard to incorporate into the tool using the proposed method because they could lead to gates only partially transitioning (i.e., a gate's output may start to rise, but if the inputs change, the output may then fall back to zero, or vise versa), which requires a significant amount of energy, even though the gate's output did not completely switch.

The energy estimation technique developed herein is based on gate-level transitions, instead of transistor-level transitions, making it extremely fast. It operates more than 1000 times as fast as equivalent transistor-level methods, producing results that fall within a range obtained by two different industry-standard transistor-level simulators; hence, it is very useful for quickly comparing the energy consumption of alternative circuit architectures.

Energy consumption is calculated from a VHDL simulation of a structural DI design, using modified fundamental gates that include energy and fanout information. The output of the tool is an estimate of the energy consumed during the simulation, which can then be divided by the number of operations in the simulation to generate the desired value of energy per operation. This method consists of two main parts: creating a VHDL energy library and transforming the structural VHDL model to use the energy library.

5.2. Library creation

First off, the energy for all set condition transitions ($0 \rightarrow 1 \rightarrow 0$) must be calculated for each fundamental gate, for a fanout of 0. In this paper the NCL DI paradigm is used for testing the tool, so the energy for all transitions for each of the 27 fundamental NCL gates [8], including the necessary inverting and resettable variants, was calculated. To do this, static transistor-level designs of all threshold gates were created using a 3.3 V, 0.5 µm CMOS process in Design Architect, and simulated with Accusim II using input rise and fall times of 1 ns.

After calculating the energy for all set condition transitions for each threshold gate, the VHDL energy library was created by including this information along with each gate's functional description. Specifically, an additional energy output of type *real* was added to each gate; and the gate's functional description was modified, such that whenever an output transition of $0 \rightarrow 1$ occurs, the gate's energy per transition value is output on the additional *real* type energy output.

A generic input to denote the gate's fanout was also added to each gate. This fanout information is used to increase the energy output by a gate that switches when it has a fanout greater than 0. The gate's fanout affects its transition energy because a larger fanout means that the gate is switching a larger capacitance, and therefore requires a larger amount of energy to do so. To simplify the estimation technique, the average size of a static NCL gate was calculated as 17 transistors. The TH34w32 gate, which consists of 17 transistors, was then used to test the effect that increasing a gate's fanout would have on its energy usage. It was found that increasing the fanout for any gate resulted in approximately an extra 2.0 pJ per additional gate fanout. For example, Fig. 12 contains the graph of energy usage versus fanout for the TH33 gate, connected to TH34w32 gates, showing a linear trend line with a slope of 2.03. Hence, the equation for a gate's switching energy is the transition energy for the specific input combination with a fanout of 0, plus 2.0 times the gate's fanout, *fo*. Fig. 13 shows the VHDL energy library component for the TH23 gate.

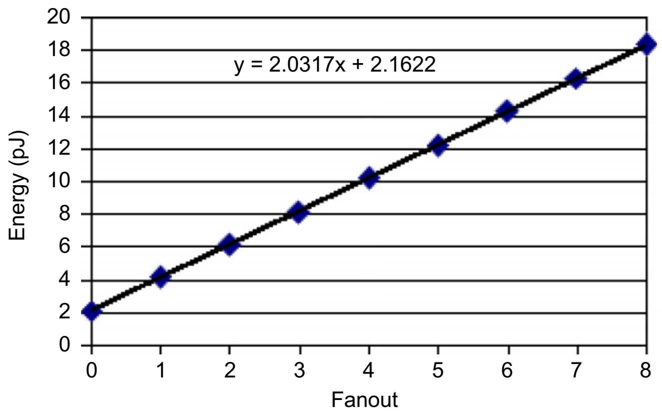


Fig. 12. Energy versus fanout for the TH33 gate.

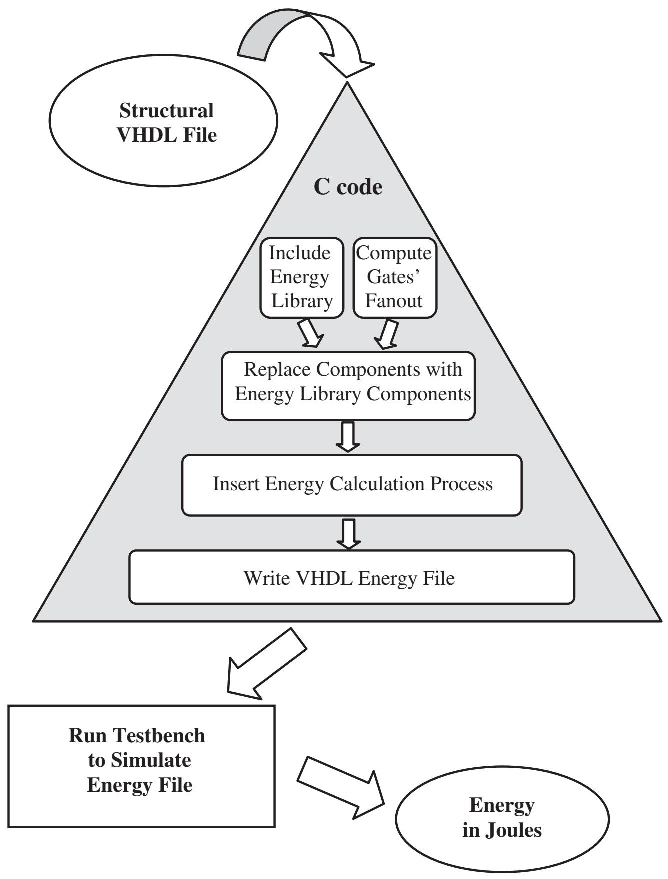
```
library ieee;
use ieee.std_logic_1164.all;
entity th23x0 is
    generic(fo: natural);
    port(energy: out real;
          a: in std_logic;
          b: in std_logic;
          c: in std_logic;
          z: out std_logic);
end th23x0;
architecture archth23x0 of th23x0 is
    signal zt: std_logic;
begin
    th23: process(a, b, c)
    begin
       if (a = '0' and b = '0' and c = '0') then
             zt <= transport '0' after tdr_th23;
       elsif (a = '1' and b = '1') then
            zt <= transport '1' after tdf_th23;
       elsif (a = '1' and c = '1') then
            zt <= transport '1' after tdf_th23;
       elsif (b = '1' and c = '1') then
             zt <= transport '1' after tdf_th23;
       end if;
    end process;
    z <= zt;
    energy_calc: process(zt)
    begin
       if zt'event and zt = '1' then
            if (a = '1' and b = '1') then
                energy <= 5.85 + (2.0 * real(fo));
            elsif (a = '1' and c = '1') then
                energy <= 5.07 + (2.0 * real(fo));
            elsif (b = '1' and c = '1') then
                energy <= 3.83 + (2.0 * real(fo));
            else
                energy <= 0.0;
            end if;
       else
            energy <= 0.0;
       end if;
    end process;
end archth23x0;
```

Fig. 13. TH23 energy library component.

5.3. Library usage

After the energy library has been created, as described in the previous section, the VHDL circuit file must be modified to use the new library. As with the previously mentioned transistor-level techniques, a structural VHDL description of the circuit is required. The method then follows the steps shown in Fig. 14. First, the structural VHDL description is loaded into Leonardo Spectrum and output as a VHDL file, so that it is in a standard format. Next, a C-language program reads this VHDL file and converts it so that it uses the new energy library. Finally, the original testbench is run on the VHDL energy file to produce the total energy consumed during the simulation.

The developed C-program executes the following steps to convert the original structural VHDL file to its energy calculating equivalent. It first calculates each gate's fanout, and includes the energy library file, which contains the



Structural VHDL File
C code
Include Energy Library
Compute Gates' Fanout
Replace Components with Energy Library Components
Insert Energy Calculation Process
Write VHDL Energy File
Run Testbench to Simulate Energy File
Energy in Joules

Fig. 14. Energy estimation flow.

component declarations for all of the energy library components, by adding the line: `use work.energy_lib.all;`. Next, it replaces each original gate with its energy library equivalent in the circuit architecture, and declares an added *real* type energy signal for each gate, and a *real* type energy accumulator signal called `energy_acc`, which is initialized to zero. The tool then inserts a process that calculates the energy consumption, which works by accumulating every transition on any gate's energy output. This process watches for a transition on any gate's energy output, and when a transition is detected, the gate's energy output is added to `energy_acc`. Finally, the program writes the VHDL energy file named `*_energy.vhd`, where * represents the original VHDL file name.

The VHDL energy file can then be compiled and simulated using the original VHDL testbench. After the simulation is complete, `energy_acc` contains the total energy consumed during the simulation. `energy_acc` can then be divided by the number of operations to calculate *energy per operation*. For many circuits an exhaustive test can be used, where all input combinations are tested. For example, a 4-bit × 4-bit multiplier requires 256 input vectors for an exhaustive test. This requires less than one minute to estimate the energy consumption using the method developed herein. For very large circuits a representative set of test vectors should be used in lieu of an exhaustive test.

5.4. NCL sequencer example

The estimation tool was first tested on the NCL sequencer in Fig. 2, comparing the results with both the ADMS and Accusim II tools. An exhaustive test (8 input combinations) was run using all three methods, resulting in an energy per operation of 15.3 pJ from Accusim II, 18.5 pJ using the developed VHDL-based estimation tool, and 19.5 pJ using the ADMS method developed herein. Note that ADMS and Accusim II are both transistor-level simulators, and that both use the same Spice model parameter file. The estimation tool's result falls within the range between the Accusim II and ADMS values.

6. Application to NCL DI multipliers

The previous section tested the developed estimation tool on a very small design, while this section tests it using various 4-bit × 4-bit unsigned multiplier architectures [15,19]. Exhaustive 256-input combination simulations were run on all designs using both the estimation tool and the ADMS process for simulating a transistor-level circuit using a VHDL testbench, presented in Section 4, with the results presented in Table 1. Note that exhaustive testing of these designs is not feasible using Accusim II due to the required non-periodic signal transitioning, as described in Section 3.2. Both simulation methods used exhaustive 256-input combination testbenches that automatically check 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 all simulations for both methods, showing that all designs produced the desired output for all 256 input combinations. Note that these multiplier designs are much larger than the sequencer, with more delay, so no additional delay had to be added in the ADMS testbench, as was necessary for the sequencer example. Each ADMS simulation took slightly more than 24 h to run on a 900 MHz Sun machine; whereas, the gate-level estimation tool simulations required less than 1 min.

As shown in the previous section, there is a range of acceptable results that falls between the ADMS and Accusim II values, so the purpose of these simulations is

Table 1
Multiplier comparison (exhaustive test)

Multiplier architecture	Energy per operation (pJ)		% error
	ADMS	Estimation tool	
Dual-rail non-pipelined	736	572	22
Dual-rail bit-wise pipelined	1376	1040	24
Dual-rail full-word pipelined	1785	1318	26
Quad-rail non-pipelined	529	449	15
Quad-rail bit-wise pipelined	795	656	17
Quad-rail full-word pipelined	1009	802	21

Table 2
Multiplier comparison (one operation)

Multiplier architecture	Energy (pJ)		
	ADMS	Estimation tool	Accusim II
Dual-rail non-pipelined	756	623	616
Dual-rail bit-wise pipelined	1520	1258	1240
Dual-rail full-word pipelined	1864	1755	1470
Quad-rail non-pipelined	559	477	434
Quad-rail bit-wise pipelined	923	851	717
Quad-rail full-word pipelined	1107	1068	857

not to compare actual energy values, but instead to compare the relative values, to show that the estimation tool can accurately estimate the relative energy requirements for various architectures. Table 1 shows that both ADMS and the developed estimation tool accurately predict the relative energy usage for the various multiplier architectures. Both methods show that the full-word pipelined designs require the most energy per operation, followed by the bit-wise pipelined designs, with the non-pipelined designs requiring the least amount of energy per operation, for both dual-rail and quad-rail logic. This is intuitively correct because the pipelined designs would be expected to require additional energy per operation compared to their respective non-pipelined versions because of the additional registers and completion logic, and the full-word pipelined designs would be expected to require more energy per operation than their respective bit-wise pipelined versions because they require additional registers and completion logic. Furthermore, both methods show that the quad-rail designs require less energy per operation compared to their respective dual-rail versions. This is also intuitively correct since 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).

To further test the absolute accuracy of the developed method, it was used to calculate the energy required for only one multiplication operation on all architectures, and the results compared with both the Accusim II and ADMS values for this operation. $1010_2 \times 0101_2$ was chosen as the test operation and was simulated on the various architectures using Accusim II, the developed estimation tool, and ADMS. The simulations show that the estimation tool's results fall between the Accusim II and ADMS values for all multiplier architectures, as summarized in Table 2.

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. The VHDL-controlled 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 at various times based on handshaking signals. The method supports automatic calculation of power and energy metrics and automated testing of functional correctness. Furthermore, this method works equally well for synchronous circuits, and can also be used for simulating digital circuits at other levels of abstraction, including post-layout simulation.

This paper also develops an automated gate-level switching method for estimating the energy required for NCL DI designs. The approach is very useful for quickly determining how architecture changes affect energy usage. For example, Table 1 shows that the quad-rail bit-wise pipelined multiplier provides a good tradeoff between energy usage and speed, since it requires slightly more energy than the dual-rail non-pipelined version and much less energy than the dual-rail pipelined versions, and is much faster than the non-pipelined versions [10,15,19], which is not intuitive. This technique was tested on a number of multiplier architectures, and the results were shown to fall within the range of values from two different industry-standard transistor-level simulators, ADMS and Accusim II, while producing the results three orders of magnitude faster than either. Furthermore, this technique is also applicable to other gate-level DI paradigms, such as [5,20–23].

References

[1] ⟨http://www.itrs.net/Links/2003ITRS/Design2003.pdf⟩ (available August 2007).
[2] ⟨http://www.itrs.net/Links/2005ITRS/Design2005.pdf⟩ (available August 2007).
[3] K.M. Fant, S.A. Brandt, NULL convention logic: a complete and consistent logic for asynchronous digital circuit synthesis, in: International Conference on Application Specific Systems, Architectures, and Processors, 1996, pp. 261–273.
[4] J. McCardle, D. Chester, Measuring an asynchronous processor's power and noise, in: Synopsys User Group Conference, 2001.
[5] C.L. Seitz, System timing, in: C. Mead, L. Conway (Eds.), Introduction to VLSI Systems, Addison-Wesley, Reading, MA, 1980, pp. 218–262.
[6] C.H. (Kees) van Berkel, M. Rem, R. Saeijs, VLSI programming, in: IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1998, pp. 152–156.
[7] D.E. Muller, Asynchronous logics and application to information processing, in: H. Aiken, W.F. Main (Eds.), Switching Theory in Space Technology, Stanford University Press, California, 1963, pp. 289–297.
[8] S.C. Smith, R.F. DeMara, J.S. Yuan, D. Ferguson, D. Lamb, Optimization of NULL convention self-timed circuits, integration, VLSI J. 37/3 (2004) 135–165.
[9] A. Kondratyev, L. Neukom, O. Roig, A. Taubin, K. Fant, Checking delay-insensitivity: 10^4 gates and beyond, in: Eighth International Symposium on Asynchronous Circuits and Systems, 2002, pp. 149–157.
[10] S.C. Smith, Completion-completeness for null convention digital circuits utilizing the bit-wise completion strategy, in: The 2003 International Conference on VLSI, 2003, pp. 143–149.
[11] G.E. Sobelman, K.M. Fant, CMOS circuit design of threshold gates with hysteresis, in: IEEE International Symposium on Circuits and Systems (II), 1998, pp. 61–65.

[12] P.A. Beerel, C. Hsieh, S. Wadekar, Estimation of energy consumption in speed-independent control circuits, IEEE Trans. Comput.-Aided Des. Integr. Circuits Systems 16/6 (1996) 672–680.
[13] J. Di, J.S. Yuan, M. Hagedorn, Switching activity modeling of multi-rail speed-independent circuits—a probabilistic approach, in: 45th Midwest Symposium on Circuits and Systems, vol. 1, 2002, pp. 475–478.
[14] P.A. Beerel, K.Y. Yun, S.M. Nowick, P-C. Yeh, Estimation and bounding of energy consumption in burst-mode control circuits, in: IEEE/ACM International Conference on Computer-Aided Design, 1995, pp. 26–31.
[15] S.C. Smith, R.F. DeMara, J.S. Yuan, M. Hagedorn, D. Ferguson, Delay-insensitive gate-level pipelining, integration, VLSI J. 30/2 (2001) 103–131.
[16] ⟨http://www.mosis.org/Technical/Testdata/⟩ (available August 2007).
[17] S.C. Smith, Speedup of NULL convention digital circuits using NULL cycle reduction, J. Syst. Archit. 52/7 (2006) 411–422.
[18] S.C. Smith, R.F. DeMara, J.S. Yuan, M. Hagedorn, D. Ferguson, Speedup of delay-insensitive digital systems using null cycle reduction, in: 10th International Workshop on Logic and Synthesis, 2001, pp. 185–189.
[19] S.K. Bandapati, S.C. Smith, M. Choi, Design and characterization of NULL convention self-timed multipliers, IEEE Des. Test Comput.: Spec. Issue Clockless VLSI Des. 30/6 (2003) 26–36.
[20] N.P. Singh, A design methodology for self-timed systems, Master's Thesis, MIT/LCS/TR-258, Laboratory for Computer Science, MIT Press, Cambridge, MA, 1981.
[21] T.S. Anantharaman, A delay insensitive regular expression recognizer, IEEE VLSI Technol. Bull. (1986).
[22] I. David, R. Ginosar, M. Yoeli, An efficient implementation of Boolean functions as self-timed circuits, IEEE Trans. Comput. 41/1 (1992) 2–10.
[23] J. Sparso, J. Staunstrup, M. Dantzer-Sorensen, Design of delay insensitive circuits using multi-ring structures, in: European Design Automation Conference, 1992, pp. 15–20.

Venkat Satagopan received his Ph.D. in Computer Engineering from the University of Missouri–Rolla (UMR) in May 2007, an M.S. degree in Electrical Engineering from UMR in December 2002, and a B.E. degree in Electronics and Communications Engineering from the University of Madras, India, in 1999. He is currently a Signal Integrity Hardware Engineer with NVIDIA in Santa Clara, CA. Dr. Satagopan's research interests include asynchronous VLSI design, VLSI testing, computer architecture, reconfigurable hardware, and signal integrity.

Bonita Bhaskaran received her Ph.D. in Computer Engineering from the University of Missouri–Rolla (UMR) in May 2007, an M.S. degree in Electrical Engineering from UMR in December 2002, and a B.E. degree in Electronics and Communications Engineering from the University of Madras, India, in May 2000. She is currently a Signal Integrity Hardware Engineer with NVIDIA in Santa Clara, CA. Dr. Bhaskaran's research interests include asynchronous VLSI design, VLSI design/testing, and signal integrity.

Anshul Singh is a Process Design Kit Engineer at Silvaco Data Systems, Massachusetts. His interests include EDA tool development, computer architecture, ASIC design, VLSI, and asynchronous logic. He received his Bachelor of Technology degree in Electronics and Communications Engineering from the Institute of Engineering and Technology, MJPR University—Bareilly, India in 2001. He then received an M.S. in Electrical Engineering from the University of Missouri—Rolla in December of 2004.

Scott C. Smith received B.S. degrees in Electrical Engineering and Computer Engineering from the University of Missouri—Columbia in May 1996, an M.S. in Electrical Engineering from the University of Missouri—Columbia in May of 1998, and a Ph.D. in Computer Engineering from the University of Central Florida, Orlando in May of 2001. He started as an Assistant Professor at the University of Missouri - Rolla in August 2001, was promoted to Associate Professor in March 2007 (to be effective September 2007), and is currently an Associate Professor at University of Arkansas. He has authored 11 journal publications, 24 conference papers, 3 US/international patents, and 2 additional international patents, all of which can be viewed from his website: http://comp.uark.edu/~smithsco/. His research interests include computer architecture, asynchronous logic design, CAD tool development, embedded system design, VLSI, FPGAs, trustable hardware, and self-reconfigurable logic. Dr. Smith is a member of Sigma Xi, Eta Kappa Nu, Tau Beta Pi, ASEE, and a Senior Member of IEEE.