Logical Modeling of Adiabatic Logic Circuits using VHDL with Examples

Lee A. Belfore II *
Old Dominion University, Norfolk, Virginia, 23529, USA

Abstract

The underlying nature of adiabatic circuits is most accurately characterized at the circuit level as it is for traditional technologies. However, in order to scale system designs for adiabatic logic technologies, modeling of adiabatic circuits at the logic level is necessary. Logic level models of adiabatic logic circuits can facilitate the design, development, and verification of large scale digital systems that may be infeasible using circuit simulators. Adiabatic logic circuits can be powered with a four stage power clock consisting of idle, charge, hold, and recover stages that provides for adiabatic charging and charge recovery to give adiabatic circuits their low power operation. By both discretizing the temporal aspects of the power clock and the logic values, a logical model of adiabatic circuit operation is proposed. Using the expressive capabilities of Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL), the salient aspects of adiabatic circuit models can be captured. In this work, a VHDL framework is defined for modeling adiabatic logic circuits & systems and its use is demonstrated in several example adiabatic logic circuits.

Key Words: Low power electronics; Digital circuits; Logical Model; Digital simulation; VHDL.

1 Introduction

Adiabatic logic circuit technology offers lower power consumption compared with CMOS technologies by energizing circuits adiabatically and then adiabatically recovering stored energy from the circuit for later reuse [3, 7, 8, 10]. The efficiency and behavior is established by the circuit level behaviors that are quantified in circuit simulations and measured in actual circuits. Once the circuits are suitably characterized, the overall operation can be described symbolically. This description of their operation is the basis for logical models of adiabatic circuits.

With current digital system design requirements and modeling practices, it is impractical to rely solely on circuit simulations to validate a design because the circuit simulations require substantial computational resources. As a result, the design process employs conservative models with conservative margins, substitutes vetted approximate models for high fidelity models, and/or, limits circuit simulation to special cases requiring circuit level fidelity. The fidelity is not reduced in an arbitrary fashion, but rather aspects of the circuit operation are modeled symbolically. Circuit level properties associated with the symbolic representations can be included using a circuit extraction step to improve the fidelity of the model. Such is the case with circuit delays, for example.

Approaches for modeling adiabatic and partially adiabatic circuits appear in the literature [14, 15]. In Varga et al., the adiabatic pipeline is modeled using the IEEE `std_logic` type for logic values and guarded blocks to manifest the timing of the power clock [14]. The motivation is to model the pipeline structure in anticipation of synthesis. Finally, approaches for modeling in Verilog are developed [15] with the observation that VHDL is similarly capable. The clear intent of these approaches is to facilitate modeling larger scale models and support synthesis based on the logical behavior of the models.

More generally in the literature, adiabatic circuit dynamics can be modeled with VHDL by one of two methods. First, VHDL libraries can be created with the specific capacity to model analog signals [9]. In addition, the VHDL standard has been extended to support mixed analog/digital modeling in VHDL-AMS [1]. In both of these approaches, the circuit is ultimately represented by a system of differential equations. In these works, it would be necessary to develop libraries to support adiabatic circuit models. The principle disadvantage in these approaches is the significant simulation time necessary for large circuits. During system development, it is more pragmatic to focus on logical modeling, constrained by conservative performance metrics, to facilitate iterative design. Once the design approaches the final phase, then it may become necessary to shift to higher fidelity circuit simulations.

In this work, an approach is introduced for modeling adiabatic circuits. The model defines a multivalued logic value definition consistent with adiabatic circuit operating modes. The logic values facilitate developing adiabatic logic pipelines and troubleshooting of logic circuits. Importantly, the model preserves the dual rail nature of adiabatic signals.

This paper is organized into five sections including an introduction, an overview of the operation of adiabatic circuits, a presentation of adiabatic VHDL models, simulation results for

*Department of Electrical and Computer Engineering. Email: LBelfore@odu.edu

several examples, and a summary.

2 Adiabatic Logic Circuits Operation

In this section, the basis for logical models of adiabatic circuit operation is presented. The intention is to identify modes of circuit operation that can be represented symbolically rather than actual circuit level behaviors. The interested reader can find the details of adiabatic circuit operation elsewhere [2, 3, 7, 8, 10].

Adiabatic circuits are capable of low power operation by providing the energy to the circuit adiabatically and then later adiabatically retrieving the energy for subsequent reuse. Note that adiabatic operation implies that no heat is dissipated during circuit operation. Since the circuit operation is not ideally adiabatic, some energy will be dissipated, but can be greatly reduced compared to traditional CMOS technologies.

2.1 Power Clocks

Adiabatic circuit operation can be divided into four segments reflecting the modes of circuit operation. The segments are *idle*, *charge*, *hold*, & *recover*, or abbreviated by I, C, H, & R. The nature of each segment captures an adiabatic circuit's mode of operation and is manifest by the nature of the power source during the segment as a function of time. Repeating the segments in the order presented enables adiabatic operation. Since segments are repeated periodically, the circuit's power source is henceforth termed the *power clock*.

In more detail, the segment operation is described as follows. In the *idle* mode, the circuit voltage source is 0V, the circuit is unpowered, and thus consumes no power. In the *charge* mode, the voltage supplied slowly increases, charging capacitive elements in the circuit that when fully charged, enabling the circuit to provide its designated function. A key aspect of the charge mode is the "slow" increase in the voltage supplied. By "trickle charging" the capacitive elements, the net power consumed can be shown to be reduced [7]. In the limit where the voltage increases over an indefinitely long time interval, the circuit operates in a truly adiabatic fashion. In the *hold* mode, the circuit is fully charged. With no current entering the circuit, the circuit consumes no power. Finally, in the *recover* mode, the circuit is discharged through its voltage source, returning its charge for later reuse. Similar to the charge mode, the slow ramp down of the voltage source retrieves the charge adiabatically. An important observation is that, unlike traditional CMOS circuits where such charge is resistively dissipated, the charge is recovered through the voltage source for later reuse.

Further, to simplify the discussions, a trapezoidal clock is assumed, although many adiabatic circuits operate using sinusoidal or other periodic shapes that are more easily generated. Figure 1 shows four power clock phases, shifted 90° with respect to one another.

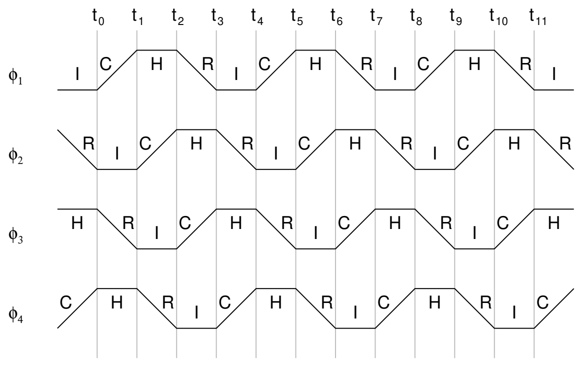
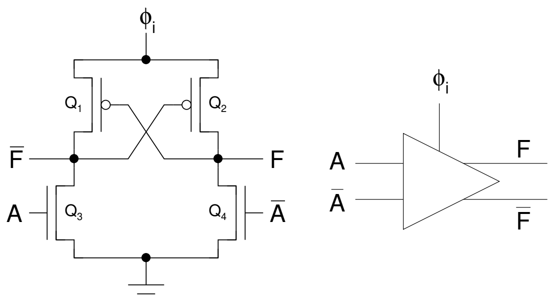


Figure 1: Four power clock phases

2.2 Adiabatic Circuit Dynamic Behavior

Figure 2 gives a circuit level schematic for the simplest adiabatic logic gate, the buffer-inverter, along with its schematic symbol. High fidelity models of the buffer-inverter



(a) Circuit (b) Schematic symbol

Figure 2: Buffer-inverter adiabatic logic circuit

unsurprisingly also require high fidelity models of transistors modeled by nonlinear differential equations. Describing the adiabatic circuits in terms of its logical modes of operation requires several simplifying assumptions. First, transistors operate as simple switches that are either open or closed depending on the gate voltage. The circuit schematic includes two types of transistors, NMOS (Q_3 & Q_4) and PMOS (Q_1 & Q_2) making the fabrication of adiabatic logic circuits compatible with traditional CMOS circuits. The transistor has three terminals: gate, source, and drain. For NMOS transistors, when a voltage across the gate and source, V_{GS}, exceeds a characteristic threshold voltage, the transistor turns on. PMOS transistors operate similarly with polarities reversed. To simplify interpreting the circuit models that follow, the NMOS transistors are on when $V_{\mathrm{GS}}^{\mathrm{NMOS}} > 0V$ and the PMOS transistors are on when $V_{\mathrm{GS}}^{\mathrm{PMOS}} < 0V$. Second, when transistors are on, they have a constant characteristic resistance R_{on}. Third, transistors have zero leakage currents when off. Fourth, all parasitic capacitances and resistances are ignored. Fifth and finally, only the transistor's gate capacitance is considered. The different simplified modes of transistor operation are shown in Figure 3. Applying the transistor models in Figure 3 to Figure 2(a) for A=`'1'`, the different modes of operation are illustrated in

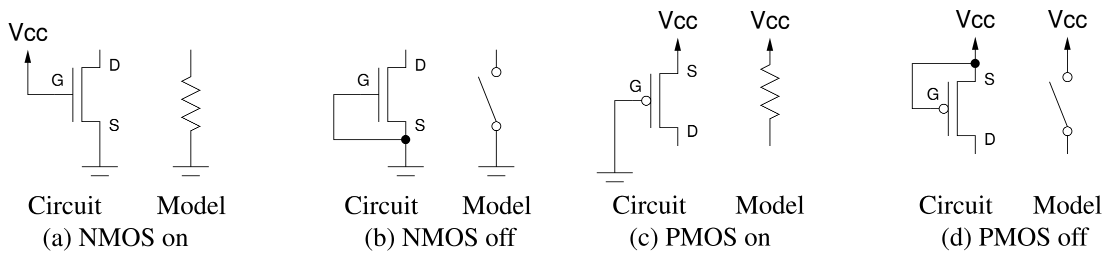


Figure 3: Simplified transistor operational modes

Figure 4. Note that C_F and $C_{\overline{F}}$ are the lumped capacitances for the circuit fanout. More concretely observed from the figure is the circuit charging and charge recovery in Figures 4(c) & 4(e) with no power consumed in Figures 4(b) & 4(d).

2.3 Adiabatic Combinational Circuit Architecture

The architecture of adiabatic combinational logic circuits is organized into several consecutive layers of logic powered by power clocks that are shifted in phase to facilitate the transfer of signal values from one layer to the next. Unlike traditional CMOS circuits where a wire conveys the logical signal, adiabatic circuits require two wires that operate in a complementary fashion when the circuit's power clock is operating in any segment except idle. This manner of signal is often referred to as dual-rail. Furthermore, the evaluation of adiabatic logic gates is synchronous with respect to its power clock unlike CMOS circuits which are entirely asynchronous. Indeed, adiabatic combinational logic circuits can be naturally pipelined with new inputs accepted at suitable times during the respective power clock phase. Figure 5 presents the architecture of a simple multilayered adiabatic logic circuit consisting of four buffer-inverters powered by four power clocks operated at phases displaced by 90°. Note transfer of values from one layer to the next when the current power clock is in the hold segment while the subsequent layer is in the charge segment.

3 Adiabatic VHDL Models

Refining the ideas introduced in §2, VHDL models for adiabatic circuits are presented here. Recall, hardware description languages (HDLs) are programming languages that model complex digital systems and are the source for hardware synthesis. HDLs facilitate specification of digital systems by modeling systems logically rather than at the circuit level. In addition, HDLs include familiar high-level language (HLL) programming capabilities for computing static constants, to support system modeling, and to provide desired modeling capabilities. For adiabatic logic circuits modeled at a temporal resolution where the influence of the power clock is important, the HLL features will be used to implement the logical behaviors of adiabatic logic circuits.

3.1 Anatomy of a VHDL Model

A VHDL model consists of an entity and an architecture [5]. The entity defines the model interface including the entity's signals, signal types, and signal modes (input, output, etc.). Further, model meta-information can be passed through optional generic parameters. Not unexpectedly, VHDL built-in types include the `bit` and `bit_vector` types. In addition, IEEE Standard 1164 [4] defines the more comprehensive `std_logic` type that better models use cases that occur in traditional digital circuits. For example, the `std_logic` type handles high impedance connections and wired logic connections for passive logic that are circuit level effects that extend to logic circuits. Considering the operation of adiabatic circuits described in §2, neither `bit` nor `std_logic` provides suitable models for adiabatic logic circuits.

Figure 6 shows an example of a VHDL model for a two-input AND gate using the `std_logic` type. The AND gate model shows declarations consisting of two inputs & one output and includes the behavioral model code for a two-input AND gate. Delays, extracted using a separate circuit analysis process, can be inserted consistent with the synthesized circuit.

3.2 Adiabatic Logic Values

At the circuit level, adiabatic logic values are more complicated than traditional logic values for several reasons. First, in adiabatic circuits, gate outputs are "dual railed" where a circuit structure generates both the true and complementary output values. Second, due to the effect of the power clock, the circuit output value is only valid at certain times as previously shown in Figure 5. Indeed, a logic one is a pulse that coincides with the circuit's power clock on the true sense gate output and a logic zero is a pulse on the complementary sense gate output while the circuit's power clock is active. While this operation is inherently analog, the circuit outputs can be categorized as logic one and logic zero. Taking a broader view of timing and circuit state, a suitable discretization of the behavior can be proposed in a manner that is consistent with adiabatic circuit operation. What follows is a discussion of the discretizing of the timing and circuit logic values.

The nature of the power clock provides straightforward guidance for discretizing time. With the dynamics of adiabatic circuits naturally falling into four distinct operating modes, it

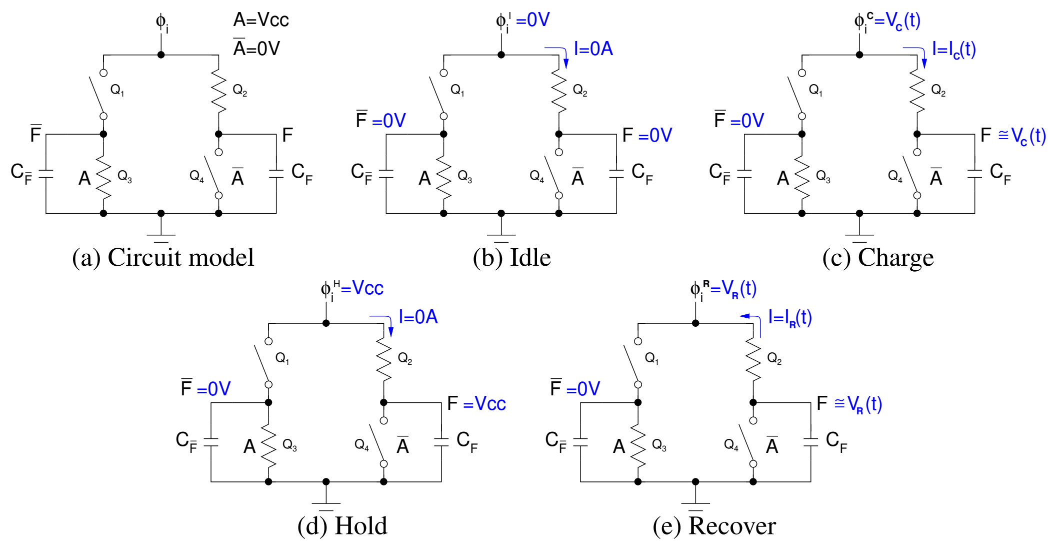


Figure 4: Simplified buffer-inverter circuit models for input A='1'

makes sense to discretize the phase into four segments. The following type declaration reflects the discretization suitable for adiabatic VHDL models.

```
type simplePhaseSegment is
        ('I','C','H','R');
```

The simplePhaseSegment type specifies the values 'I', 'C', 'H', and 'R', representing idle, charge, hold, and recover respectively. For segments where the power clock is changing (in 'C' and 'R'), no circuit dynamics are modeled, rather the logical result reflecting the values at the end of the segment are reported. Any varying circuit level quantities will be represented symbolically in that segment. Extending simplePhaseSegment is phaseGeneral which is a record including a simplePhaseSegment and phase index fields.

A new basic type, aBitSimple, is an eleven valued logic system defined to represent the range of adiabatic signal values that reflect the logic value, nature of the circuit, and value in relation to the phase. In this work, we have chosen to not differentiate the signal strengths during the charge and hold phases to facilitate interpretation of timing diagrams. Including these are straightforward and results in five additional signal values covering respective activities during the charge phase. The permissible values for this type are summarized in Table 1.

The fully qualified signal VHDL model is defined record type that includes both the signal value and the phase:

```
type aBit is record
  val     : aBitSimple;
  myPhase : phaseGeneral;
end record;
```

Including the phase in the signal definition enables run time checking to confirm the aBit phase is consistent with the assigned phase of the gate's power clock.

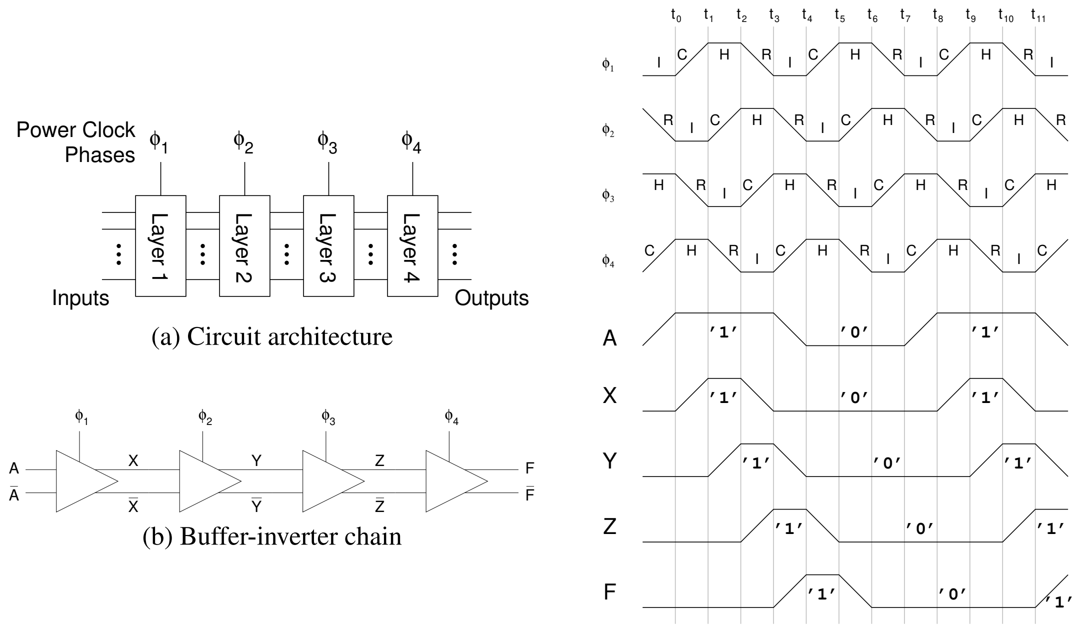
Several utility routines have been created to help manage signal values and phases. Some routines facilitate the conversion between standard signal types (bit and std_logic) and the new aBit type. Furthermore, operator overloading for the new logic type has been implemented to permit the natural composition of logic expressions. In the event indeterminate inputs or phase errors occur, the logic operations evaluate to unknown values ('X' or 'x') to facilitate troubleshooting. Finally, the logic values 'Z' and 'z', along with the requisite bus resolution functions, permit high impedance bus modeling.

Table 1 Summary of adiabatic signal values for the aBitSimple type

Value	Description
'U'	driving uninitialized value
'X'	driving unknown value
'0'	driving logic zero
'1'	driving logic one
'Z'	high impedance
'u'	recover uninitialized value
'x'	recover unknown value
'L'	recover logic zero
'H'	recover logic one
'z'	recover high impedance
'*'	fully discharged

3.3 Logical Adiabatic Gate Model

The logical adiabatic gate model requires changes both to the gate entity and to the behavior defined in its architecture compared with conventional gate models. The adiabatic gates perform logic functions, so one reasonable approach would be to adopt traditional logic values in the gate model. In this approach, phase information would be lost. Furthermore, adiabatic gates are dual rail, whose representation is not as important as the power clock phase in logical modeling.



(a) Circuit architecture

(b) Buffer-inverter chain

(c) Timing diagram for buffer-inverter chain.

Figure 5: Adiabatic circuit architecture and operation

```
entity and2 is
  port(a,b: in std_logic;z:out std_logic);
end entity and2;
architecture behavioral of and2 is
begin
   if a='0' or b='0' then       z<='1' after 500 ps;
   elsif a='1' and  b='1' then  z<='0' after 200 ps;
   else                         z<='X' after 350 ps;
   end if;
end architecture behavioral;
```

Figure 6: Example VHDL model

However, their explicit inclusion provides an opportunity to have visibility of all signals in the circuit. Apropos, the entity for the AND gate shown in Figure 7 includes dual rail input & output logic signals and the clock phase driving the gate.

```
entity adbAnd2 is
  port(
       phi : in generalPhase;
       a,an: in aBit;
       b,bn: in aBit;
       z,zn:out aBit
  );
end entity adbAnd2;
```

Figure 7: Entity for two-input adiabatic AND gate

Determining the gate outputs is no longer a simple matter of evaluating the gate's logic function based on the circuit inputs because of the dependence on the power clock segment. The model presented in Figure 8 implements the behavior for the two-input adiabatic AND gate that accounts for the power clock. When the clock phase changes, the gate inputs are verified to be in phase and to be correctly lagging the gate's power clock phase. When a phase error is detected, the output signal is assigned an 'X' value. Since logical operations have been overloaded, the gate logic function is expressed in a natural fashion, permitting logic equations to model the respective MOS switching networks. Logic operations are evaluated in their respective common phase, facilitating the composition of complex logic functions. The resulting output value is stored in a temporary variable so that the phase can be correctly updated to be consistent with the power clock for the gate. In transitioning to and during the hold segment, the logic gate outputs remain constant in the model.

3.4 Extending to Other Logic Gates

The dual rail nature of the logic gates simplifies creating families of logic gates. Signal inversion is accomplished simply by swapping the true and complementary signal rails requiring no additional circuitry. Indeed, with DeMorgan's Theorem, it is easy to show that by swapping dual rail signals to complement inputs & outputs, the two-input AND gate can also serve as an

Table 2 Utility functions and procedures

Name	Purpose
`isCharging`	function, returns true when power clock is charging
`isHolding`	function, returns true when power clock is maximal
`isRecovering`	function, returns true when power clock is discharging
`isIdle`	function, returns true when power clock is off
`deenergize`	procedure, reduces the strength of signal while retaining logic value
`assignToPhase`	procedure, assigns a phase to a signal

```
process(phi)
  variable zInt,znInt:aBit;
begin
  -- check for valid input and output
  -- phase segments
  if(isCharging(phi)) then
    zInt  <= a  AND b;
    zIntn <= an OR  bn;
  elsif isHolding(phi) then
    -- by VHDL semantics,
    -- no update-no signal change
  elsif isRecovering(phi) then
    zInt  := deenergize(zInt);
    znInt := deenergize(znInt);
  else -- idle
    zInt.val  := '*';
    znInt.val := '*';
  end if;
  assignToPhase(zInt, phi);
  assignToPhase(znInt, phi);
  z  <= zInt;
  zn <= znInt;
end process;
```

Figure 8: Behavioral model for two-input adiabatic AND gate

OR, NAND, or NOR gate. In addition, more complex logic functions can be modeled using the logic equation for the true input values and the dual logic equation for the complementary input values.

For example, the logic equations for a full adder are

$$\begin{aligned} S &= A \oplus B \oplus C_i \\ C_o &= A \cdot B + A \cdot C_i + B \cdot C_i \end{aligned} \tag{1}$$

With traditional CMOS logic, the full adder can be implemented with several gates. In adiabatic logic, each logic function can be implemented with an NMOS switching network, so the full adder can be implemented with two adiabatic logic gates. The logic equations for the complementary networks are

$$\begin{aligned} \overline{S} &= \overline{A} \oplus \overline{B} \oplus \overline{C}_i \\ \overline{C}_o &= (\overline{A} + \overline{B}) \cdot (\overline{A} + \overline{C}_i) \cdot (\overline{B} + \overline{C}_i) \end{aligned} \tag{2}$$

The second example is a multiplexer with dedicated, mutually exclusive select lines. The general true and complementary logic equations are

$$Z = \sum_{i=0}^{N-1} S_i D_i \qquad \overline{Z} = \prod_{i=0}^{N-1} (\overline{S}_i + \overline{D}_i), \tag{3}$$

where N is the number of data inputs. It is also easy to show that, for $N = 2$, (3) can specify two-input XOR and XNOR gates.

3.5 Test Bench

A test bench is a special VHDL model which is used to verify the circuit model. The test bench instantiates the unit under test, generates all stimulus, and can include code to verify the model's outputs are correct. Figure 9 gives the VHDL process that generates the i^{th} power clock. For four power clock phases, each power clock phase i has the same period T and is delayed by $(i-1) \times 90°$, or $\mathrm{T}(i-1)/4$ with respect to a reference time at the start of the simulation. This can be easily generalized for a different number of power clock phases. In Figure 9, the power clock process includes one full clock period interval at the beginning of the simulation with no activity among all clocks. The first `wait` statement ensures that all power clocks are inactive for at least one full period of the power clock and the start of each is delayed to ensure each clock will be in the appropriate relative phase.

```
...
constant T: time := 100 ns;
...
process
-- generate the ith power clock phase
-- i in {1,2,3,4}
begin
  Phi_i <= ('I',i-1);
  wait for T*(3+i)/4; -- See narrative
  loop
    Phi_i.segment<='C'; wait for T/4;
    Phi_i.segment<='H'; wait for T/4;
    Phi_i.segment<='R'; wait for T/4;
    Phi_i.segment<='I'; wait for T/4;
  end loop;
end process;
```

Figure 9: Generating the i^{th} phase of the power clock

In order for outputs to conform to proper adiabatic operation, inputs must be set in the appropriate manner to ensure the adiabatic operation of the gate receiving the input. In addition, it is possible that different inputs may be required at different logic layers, and hence must be synchronized to the correct power clock phase. This can be accommodated in one of two ways. First, the inputs can be provided at the same time and always on the same phase. In this case, buffers will need to be inserted to delay the signal until it has the required phase for its respective input layer. Second, the inputs can be provided and synchronized to the required phase. The modeling satisfies either case.

4 Examples

Three examples of adiabatic logic circuit models are presented here. In the first, a full adder model is presented. In the second, a Kogge-Stone adder model is presented. In the third, a more complex model of the AES S-Box is presented. The models were verified using GHDL Version 0.33 under the IEEE-1164 1993 release of the VHDL standard on Ubuntu 16.04. In addition, while the modeling is based on the 1993 standard, no issues are anticipated for later VHDL standard releases. Waveforms are displayed using the GTKWave V3.3 waveform viewer.

4.1 Full Adder

A simple but useful example to consider is the full adder. The full adder is a key building block used to implement computer arithmetic hardware. The full adder model consists of two logic gates and operates using one power clock phase using the logic functions defined in (1) and (2). The behavior is modeled by modifying the code in Figure 8 by substituting the logic equations for the sum and carry functions respectively in place of the AND gate logic equations. The simulation results are presented in Figure 10. The inputs provided to the full adder sequence through all eight input combinations in successive power clock cycles, noted with cursors A-H.

4.2 Kogge-Stone Adder

The next example is a Kogge-Stone adder (KSA) [2, 6] and demonstrates the operation of a more complex multilayered combinational circuit. The KSA adds two binary integers and is among the fastest combinational adders, whose implementation requires $log_2 N + 2$ layers of adiabatic logic. The KSA adder can be fully implemented with commonly known gates such as AND, OR, XOR, & etc. By implementing certain composite functions to provide carry generates & propagates as individual logic gates, the circuit architecture can be simplified. Indeed, these composite gates are part of the formulation of KSA adders and are summarized in Table 3. Note that the Buffer cell is not a part of the traditional KSA adder formulation. Rather, the Buffer cell is included in this model to support proper adiabatic circuit operation to match the power clock phase for the values propagating from layer to layer in the adder. Note also that subscripts on gate input values are nominal and are related to the local interconnections required to implement the KSA adder.

Table 3 Kogge-Stone logic cells

Cell	Logic Equation	
Black cell	$G_{out} = (P_1 \cdot G_0) + G_1$	$P_{out} = P_1 \cdot P_0$
Gray cell	$G_{out} = (P_0 \cdot G_0) + G_1$	
White cell	$G_{out} = P_1 \cdot P_0$	$P_{out} = P_1 \oplus P_0$
Buffer cell	$G_{out} = G_0$	$P_{out} = P_0$

The VHDL model for the KSA adder has been implemented in a generic fashion so that the same architecture can implement any power-of-2 sized KSA adder. Figure 11 gives the entity used to model the KSA adder. In order to simplify the presentation of results, a four-bit KSA adder is demonstrated. Specifically, the VHDL model for the four-bit adiabatic KSA adder modeled here requires $log_2 N + 2 = 4$ layers of logic to implement. Figure 12 shows the simulation beginning at 4 μs for a circuit powered by power clocks with 100 ns periods. Note that signal complements have been omitted. At cursor A (4.0875 μs), the input $Op1_A$=0101, $Op2_A$=1001 and Cin_A=1. The output layer is charging at cursor C (4.1875 μs) and $Cout_C$=0, and Sum_C=1111. In addition, at cursor C, the inputs are changed to $Op1_C$=1010, $Op2_C$=0101 and Cin_C=0 resulting in $Cout_F$=0, and Sum_F=1111 at 4.2875 μs.

4.3 Advanced Encryption Standard (AES) Substitution Box

In this section, we present a significantly more complex model which is the logic for the Advanced Encryption Standard (AES) substitution box (S-box). The purpose of the S-box is to introduce a nonlinear, but difficult to reverse, transform to enhance the security of the encryption. The interested reader can find more information by consulting the AES standard [12]. The S-box is a complex combinational logic function devised by others [11, 13]. Their proposed circuit, however, cannot be directly implemented using adiabatic logic circuits because of the multiphase nature of the adiabatic logic circuits.

The logic for the S-Box follows from the implementation method proposed by Satoh et al. [13] and detailed combinational logic S-Box implementation described by Mui [11]. The respective authors note the efficiency of their implementation in terms of hardware. Figure 13 gives an overview of the S-Box and inverse S-Box logic. A transformation, (δ), is applied to the original $GF(2^8)$ system to permit decomposition in terms of a $GF(2^4)$ system, and subsequently a $GF(2^2)$ system to permit derivation of logic functions for intermediate values [11, 13]. Once the system is expressed in terms of a $GF(2^2)$ system, the logic functions at this level can be expressed directly as four-input, two-output logic expressions. From the $GF(2^2)$ logic functions, logic expressions for the $GF(2^4)$ and then ultimately $GF(2^8)$ can be derived.

Figure 14 shows the individual logic blocks that are used in both the S-Box and inverse S-Box transformations. The number of layers of adiabatic logic are indicated above each block. The logic is mostly implemented with two-input gates along with a handful of three-input gates. The δ and affine transforms T are matrix/vector operations on individual bits using AND and exclusive-OR operations, i.e. $GF(2)$.

Inspection of Figures 13 & 14 reveal a complex hardware organization with a multitude of paths for logical results that flow through varying numbers of layers of logic. For proper operation, the phases of inputs received for any block must be identical and the block must be energized by the next sequential

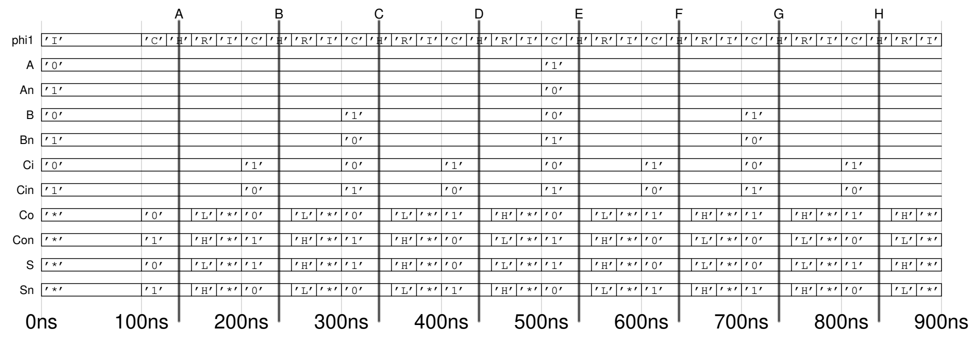


Figure 10: Full adder simulation results

```
entity KsaGeneric is
  generic(order : integer := 2); -- width=2**2=4
  port (
    phi1,phi2,phi3,phi4 : in  phaseGeneral;
    adbCin,  adbCinN    : in  aBit;
    adbOp1,  adbOp1n    : in  aBit_vector(2**order-1 downto 0);
    adbOp2,  adbOp2n    : in  aBit_vector(2**order-1 downto 0);
    adbSum,  adbSumN    : out aBit_vector(2**order-1 downto 0);
    adbCout, adbCoutN   : out aBit
  );
end KsaGeneric;
```

Figure 11: Entity for Kogge-Stone adder. Note that `**` has been overloaded for integer types

power clock phase. Figure 15 gives the annotated block diagram for the adiabatic logic implementation. Differing from previous examples, the logic circuit is implemented using six power clock phases so that complementary power clock phases, power clocks exactly 180° out of phase, are nonoverlapping. Further, because the input phases to a logic block must match, the phase for an unmatched signal is matched with its destination by adding a suitable number of buffer-inverter gates and are denoted by the Φ^N blocks. The solution attempts to optimize the hardware by not fully pipelining the forwarding in some cases.

Both the S-Box and inverse S-Box models were simulated for all possible input combinations and verified for correctness in the test bench. Figure 16 gives an example timing result for an S-Box input of `11000011`.

5 Summary and Future Work

A modeling framework has been presented that is consistent with the logical operation of adiabatic logic circuits. A new type, `aBit`, was defined that captures the main modes of operation for adiabatic circuits. The type models the principle adiabatic signal features and ties the operation of the logic circuits to the power clock. The framework for defining logic functions was presented. Finally, three modeling examples with their respective simulation results were presented.

Future work will include verifying the operation of the modeling framework on a wider variety of adiabatic and reversible circuits. In addition, applicability to different clocking schemes & timing, energy modeling, and transistor level synthesis will be investigated as well.

References

[1] E. Christen and K. Bakalar. "VHDL-AMS – A Hardware Description Language for Analog and Mixed-Signal Applications." *IEEE Transactions on Circuits and Systems–II Analog and Digital Signal Processing*, 46(10): 1263–1272, October 1999.

[2] M. Cutitaru. *IDPAL A Partially-Adiabatic Energy-Efficient Logic Family: Theory and Applications to Secure Computing*. PhD Thesis, Old Dominion University, Norfolk, Virginia, USA, August 2014.

[3] J. S. Denker. "A Review of Adiabatic Computing." *IEEE Symposium on Low Power Electronics*, 94–97, San Diego, California, USA, pp. 94–97, September 1994.

[4] IEEE Computer Society. *IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164)*, March 1993.

[5] IEEE Computer Society. *IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076^{TM}-2008, January 2009.

[6] P. M. Kogge and H. S. Stone. "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations." *IEEE Transactions on Computers*, C-22(8): 783–791, August 1973.

[7] J. Koller and W. Athas. "Adiabatic Switching, Low Energy Computing, and the Physics of Storing and Erasing Information." *Workshop on Physics and Computation, 1992. PhysComp '92*, Dallas, Texas, USA, pp. 267–270, October 1992.

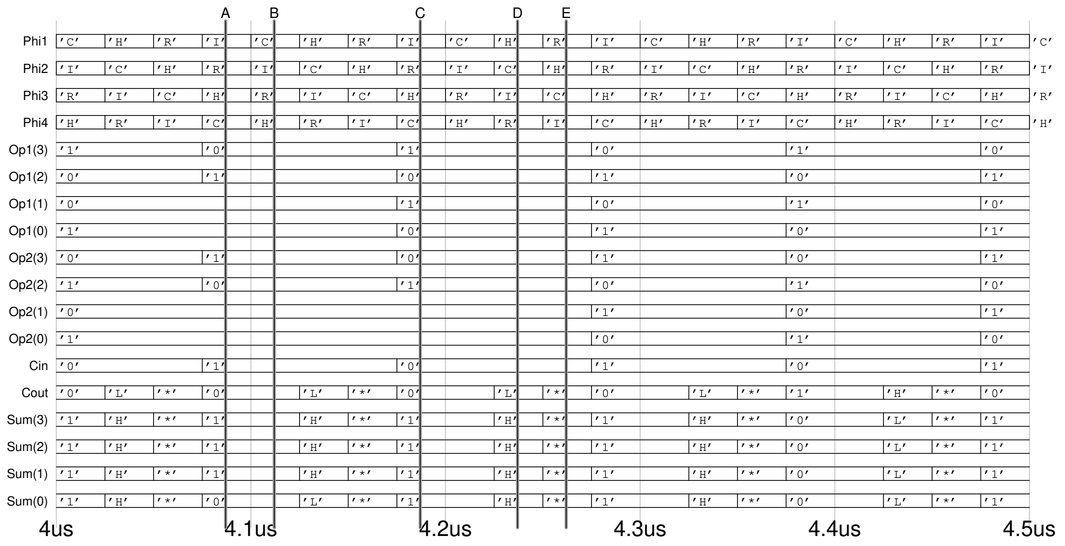
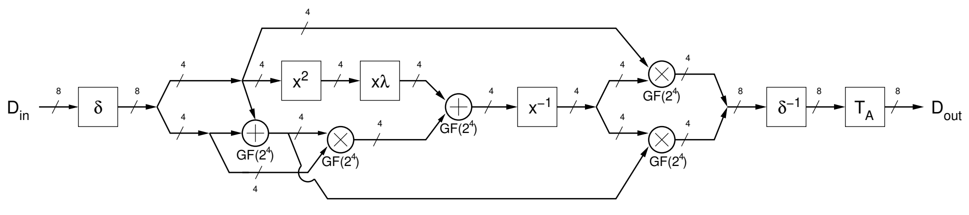
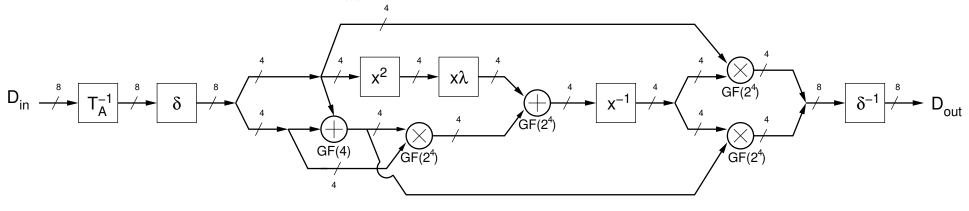


Figure 12: KSA Simulation results at 4us



(a) Forward S-Box transformation



(b) Inverse S-Box transformation

Figure 13: S-Box block diagram based on composite field decomposition [11, 13]

[8] A. Kramer, J. S. Denker, B. Flower, and J. Moroney. “2nd Order Adiabatic Computation with 2n-2p and 2n-2n2p Logic Circuits.” *Proceedings of the International Symposium on Low Power Design ISLPD'95*, Dana Point, California, USA, pp. 191–196, 1995.

[9] R. Mita and G. Palumbo. “Modeling of Analog Blocks by Using Standard Hardware Description language.” *Analog Integrated Circuits and Signal Processing*, 48(2):107–120, August 2006.

[10] Y. Moon and D.-K. Jeong. “An Efficient Charge Recovery Logic Circuit.” *IEEE Journal of Solid-State Circuits*, 31 (4):514–522, April 1996.

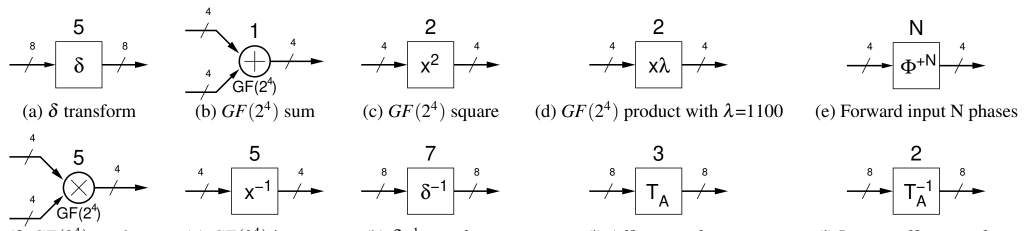
[11] E. N. Mui. “Practical Implementation of Rijndael S-Box Using Combinational Logic.” unpublished technical report, 2007.

[12] NIST. “FIPS PUB 197, Advanced Encryption Standard (AES),” U.S. Department of Commerce/National Institute of Standards and Technology, 2001.

[13] A. Satoh, S. Morioka, K. Takano, and S. Munetoh. “A Compact Rijndael Hardware Architecture with S-Box Optimization.” *7th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2001)*, Gold Coast, Australia, pp. 239–254, December 2001.

[14] L. Varga, G. Hosszú, and F. Kovács. “Two-level Pipeline Scheduling of Adiabatic Logic.” *International Spring Seminar on Electronics Technology (ISSE 2006)*, St. Marienthal, Germany, pp. 390–394, May 2006.

[15] D. J. Willingham. *Asynchrobatic Logic for Low-Power VLSI Design*. PhD thesis, University of Westminster, London, England, March 2010.



(a) δ transform (b) $GF(2^4)$ sum (c) $GF(2^4)$ square (d) $GF(2^4)$ product with λ=1100 (e) Forward input N phases

(f) $GF(2^4)$ product (g) $GF(2^4)$ inverse (h) δ^{-1} transform (i) Affine transform (j) Inverse affine transform

Figure 14: S-Box block components. The number above each component is its respective number of logic layers.

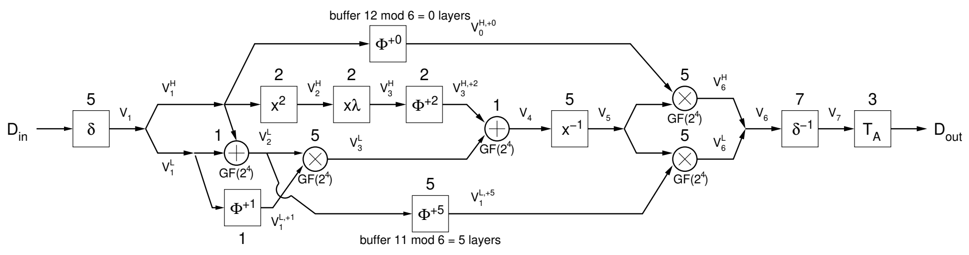


Figure 15: Annotated S-Box block diagram.

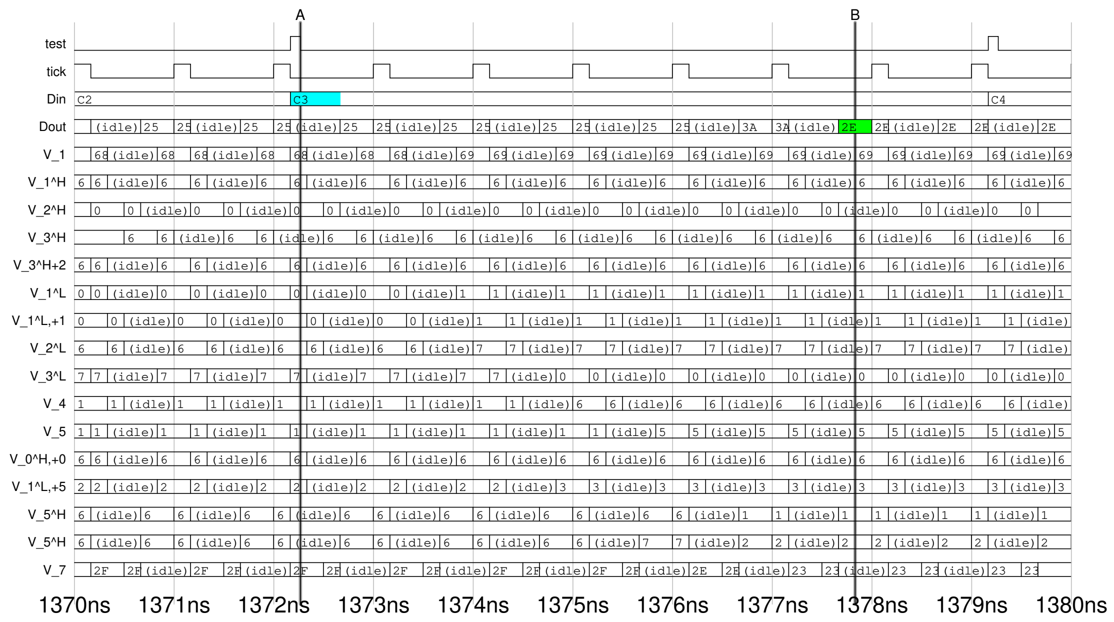


Figure 16: S-Box simulation result for an input of `11000011`

Lee A. Belfore II joined the Department of Electrical and Computer Engineering at Old Dominion University, Norfolk, Virginia, in 1997 and is currently an Associate Professor. He received his Ph.D. in Electrical Engineering from The University of Virginia, his MSE in Electrical Engineering/Computer Science from Princeton University, and his BSEE in Electrical Engineering from Virginia Tech. His research interests include modeling and analysis of low power digital electronics technologies, custom data processing systems using FPGAs, machine learning, and autonomous vehicles.