Logical Modeling of Adiabatic Logic Circuits using VHDL

Lee A. Belfore II

Old Dominion University, Norfolk, Virginia, U.S.A.
`lbelfore@odu.edu`

Abstract

The underlying nature of adiabatic circuits is most accurately characterized at the circuit level as it is for traditional technologies. 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.

1 Introduction

Adiabatic logic circuit technology offers lower power consumption compared with CMOS technologies by energizing circuits adiabatically energized and then adiabatically recovering stored energy from the circuit for later use [7, 3, 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, including reduced fidelity, to enable the verification of circuit operation while substituting high fidelity circuit level models for vetted approximation methodologies or limiting 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 what is done with circuit delays, for example.

Approaches for modeling adiabatic and partially adiabatic circuits appear in the literature [11, 12]. 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 [11]. The motivation is to model the pipeline

structure in anticipation of synthesis. Finally, approaches for modeling in Verilog are developed [12] with the observation that VHDL is similarly capable. The clear intent of these approaches is to facilitate modeling larger scale models and facilitate 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 for large circuits. During 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 several examples, and a summary.

2   Adiabatic Logic Circuits Operation

In this section, a logical model for adiabatic circuit operation is presented. The intention is to create a symbolic representation of adiabatic circuit operation and not model actual circuit level behaviors. The interested reader can find the details of adiabatic circuit operation elsewhere [7, 3, 8, 10, 2].

Adiabatic circuit operation is both powered and coordinated by the power clock. The power clock is a periodic signal that powers circuits performing logic operations, hold the values for a sufficient time to propagate logic values to the next layer of logic, and then recovers as much energy from the circuit to be recycled for future operations. From a logical modeling perspective, the power clock discretizes time.

The power clock is structured into four segments reflecting the different stages adiabatic circuit operation. In this paper, the segments will be designated as Charge (C), Hold (H), Recover (R), and Idle (I). The first segment is the charge segment (often also called the evaluate). During the charge segment, the circuit is adiabatically energized by gradually increasing the power clock voltage while its inputs are held constant. At the end of the charge segment, the circuit is fully energized with the outputs at full scale presenting the result of the gate's logic function. During the hold segment, the power clock is constant, holding the outputs constant for the duration of next layer of logic's charge segment. In the recover segment, the power clock voltage is gradually reduced while at concurrently, through the power clock line, adiabatically recovering charge from the circuit. Finally, in the idle stage, the circuit is not energized and the power clock is at its minimum. Further, the duration of the idle segment depends on the number of power clock phases.

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. Circuits generally require two or more clock phases for operation with Figure 1 presenting an example logic circuit architecture with its accompanying timing diagram. In this example, each logic layer is a buffer-inverter with signals connected so logic values propagate in a non-inverting fashion. Logic values are propagated layer by layer, synchronized by their respective power clock phases. For proper operation, the phases must be consecutive, with the successive phases lagging by 90°, assuming four power clock phases. As a

consequence of the power clocks organization, layers of logic form a pipeline of sorts, each powered by the successive power clock phase. In Figure 1, the input that is applied alternates `'1's` and `'0's` on successive power clock cycles.

(a) Circuit architecture

(b) Buffer-inverter chain
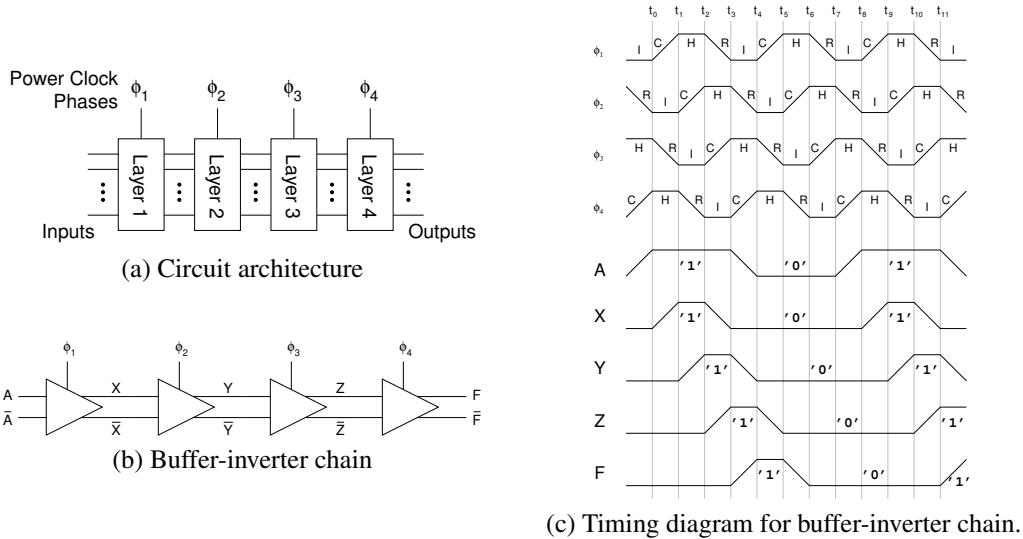
(c) Timing diagram for buffer-inverter chain.

Figure 1: Adiabatic circuit operation.

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 serve as 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 create 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` provide suitable models for adiabatic logic circuits.

Figure 2 shows an example of a VHDL model for a two-input AND gate using the `std_logic` type. The AND gate model shows declarations of has 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.

```
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 2: Example VHDL model

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 1. 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 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 is 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;
```

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'	recovery uninitialized value
'x'	recovery unknown value
'L'	recovery logic zero
'H'	recovery logic one
'z'	recovery high impedance
'*'	fully discharged

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 an unknown value to facilitate troubleshooting. Finally, the logic values 'Z' and 'z', along with the requisite bus resolution functions, permit high impedance bus modeling.

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. 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 not as important as the power clock phase in logical modeling, their explicit inclusion provides an opportunity to have visibility for all signals in the circuit. Apropos, the entity for the AND gate shown in Figure 3 will require 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 3: 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 in the circuit inputs because of the dependence on the power clock segment. The model presented in Figure 4 implements the behavior for the two-input adiabatic AND gate that accounts for the power clock. When the clock phase changes, input phase segments are verified to be the same and also one

segment preceding the gate's power clock phase segment where the output value is determined in the charge segment. A phase error results in the output signal being 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 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.

```
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 4: Behavioral model for two-input adiabatic AND gate

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

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 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 a MOS 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

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

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 test the model. Figure 5 gives the VHDL process that generates the ith 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 $T(i-1)/4$ 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 5, 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 5: Generating the ith 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, hence must be synchronized to different power clock phases. 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 presented works for either method.

4   Examples

Two 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. 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.

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 4 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 6. The inputs provided to the full adder sequence through all eight input combinations in successive power clock cycles, noted with cursors A-H.
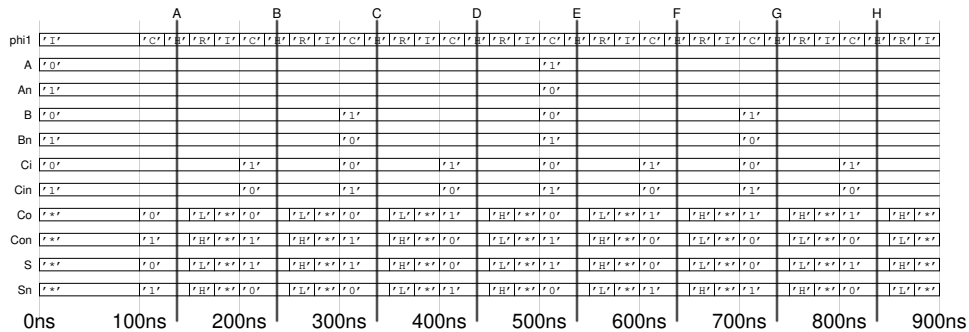

Figure 6: Full adder simulation results

The next example is a Kogge-Stone adder (KSA)[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 $log_2 N + 1$ layers of adiabatic logic. In addition to basic gates (AND & XOR), the implementation includes four special cells to manage carry generates & propagates and their respective functions are summarized in Table 3. Note that the Buffer cell is not a part of the traditional KSA adder. 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.

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 7 gives the entity used to model the KSA adder. In order to simplify the presentation of results, a four-bit KSA adder is demonstrated.

Table 3: Kogge-Stone logic cells

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

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

Specifically, the VHDL model for the four-bit adiabatic KSA adder modeled here requires $log_2N+1 = 4$ layers of logic to implement. Figure 8 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.
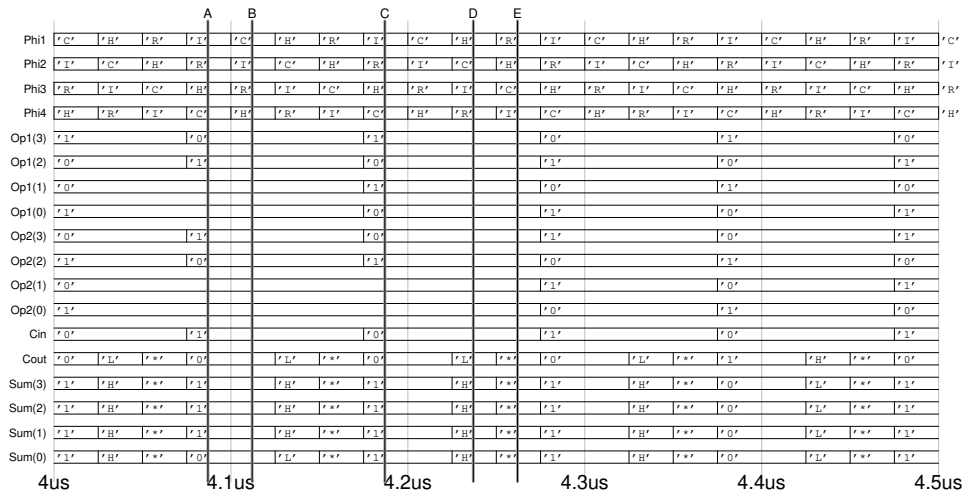

Figure 8: KSA Simulation results at 4us

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 examples simulation results were presented for a full adder and a Kogge-Stone adder.

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 and energy modeling will be investigated as well.

References

[1] Ernst Christen and Kenneth 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] Mihail 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. In *IEEE Symposium on Low Power Electronics*, pages 94–97, San Diego, California, USA, 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*, January 2009. IEEE Std 1076^{TM}-2008.

[6] Peter M. Kogge and Harold 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.G. Koller and W.C. Athas. Adiabatic switching, low energy computing, and the physics of storing and erasing information. In *Workshop on Physics and Computation, 1992. PhysComp '92*, pages 267–270, Dallas, Texas, USA, October 1992.

[8] A. Kramer, J. S. Denker, B. Flower, and J. Moroney. 2nd order adiabatic computation with 2n-2p and 2n-2n2p logic circuits. In *Proceedings of the International Symposium on Low Power Design ISLPD'95*, pages 191–196, Dana Point, California, USA, 1995.

[9] Rosario Mita and Gaetano Palumbo. Modeling of analog blocks by using standard hardware description language. *Analog Integrated Circuits and Signal Processing*, 48(2):107–120, August 2006.

[10] Yong Moon and Deog-Kyoon Jeong. An efficient charge recovery logic circuit. *IEEE Journal of Solid-State Circuits*, 31(4):514–522, April 1996.

[11] László Varga, Gábor Hosszú, and Ferenc Kovács. Two-level pipeline scheduling of adiabatic logic. In *International Spring Seminar on Electronics Technology (ISSE 2006)*, pages 390–394, St. Marienthal, Germany, May 2006.

[12] David John Willingham. *Asynchrobatic logic for low-power VLSI design*. PhD thesis, University of Westminster, London, England, March 2010.