

# Current Fault Modeling in VITAL

J.L. Barreda, P. Sánchez and E. Villar  
Microelectronics Group. TEISA. ETSIIyT  
University of Cantabria  
Santander. Spain  
Ph: 34-42-201548 Fax: 34-42-201402

## Abstract

The recently approved VITAL standard will permit many of the drawbacks presented by the use of VHDL at gate level to be overcome. It does not, however, address one of the basic design process stages at gate level: fault simulation. With the aim of integrating this stage in the VHDL design process, several proposals have appeared in recent years for voltage fault modeling for VITAL descriptions[1]. These do not take into consideration a fault type which has appeared recently and which is of growing importance: current faults.

The main purpose of the present work is to propose a current fault model for VHDL gate-level descriptions which are VITAL compliant. To do this, it was necessary to define modeling techniques for the current flowing through a logic gate.

## 1. Introduction

Before presenting the current fault modeling, it is necessary to present current modeling techniques which were not considered within the VITAL'95 standard but which are essential for the development of the above mentioned model. These techniques are an important result of this work, since they can be used for current fault modeling and for power estimation and average/peak current determination with a maximum variation of 10% with respect to the data obtained by SPICE LEVEL 3[2], with which it has applications in fields such as low-power design, BIST scheduling, etc. Logically, the new types, signals and subprograms used in current modeling do not verify the modeling rules of the recently approved VITAL standard, constituting a proposal for a possible extension in the future.

In this paper, two types of current test are considered:

- IDDQ [5] or quiescent current test, in which the presence of faults is determined for the value of the static (quiescent) current which flows through a CMOS circuit.
- IDDT [6] or transitory current test, in which the presence of faults is observed by means of modifications in the current waveform during the switching of the logic gate.

The order of contents of this paper will be as follows. In the next section the concepts necessary for both quiescent and transitory current modeling will be introduced. Then, an example of the application of these techniques will be presented. The fourth section will describe the proposed fault model. The final section will present the conclusions of the article.

## 2. Current modeling

With the aim of developing current models for VITAL cells some non-VITAL standard types and subprograms have been introduced. In the first place, the definitions necessary for carrying out the modeling of quiescent current will be analyzed.

In order to be able to model the quiescent current through the gates of the VITAL descriptions, a new port (of a new type, Qcurrent) has been introduced into the VITAL cells, which models the quiescent current flowing through the gate. The type of this port is a resolved type with a physical (current type) base type.

```

type current is range INTEGER'LOW to INTEGER'HIGH
  units
    na ;
    ua = 1000 na;
    ma = 1000 ua;
    a = 1000 ma;
  end units;
type CurrentArray is array( NATURAL range <>) of current ;
function PowerSource( A : CurrentArray ) return current;
subtype Qcurrent is PowerSource current;

```

The resolution function calculates the current flowing through the power-supply wire (a signal connected to the new port) as the sum of the static currents of each gate (modeled as values of the new port drivers). The base current type was defined as physical so that the user should be able to use units familiar to him/her.

```

function PowerSource( A : CurrentArray ) return current is
  variable result : current := 0 na;
  begin
    IF( A'LENGTH=1) THEN return( A(A'LOW)); ELSE
      for i in A'RANGE loop
        result := result + A(i);
      end loop;
    END IF;
  return result;
end;

```

The assignation of the quiescent current values to a VITAL cell is carried out using two generic ports: a VitalTruthTableType port and a VitalQcurrentTableType port. The second port provides the value of the quiescent current associated to each row of the VitalTruthTable port. This value is calculated by means of the VitalQcurrentTable subprogram (of a similar philosophy to the VitalTruthTable, but generating quiescent current values rather than outputs). This subprogram is presented below:

```

PROCEDURE VitalQCurrentTable ( CONSTANT TruthTable : IN VitalTruthTableType;
                              CONSTANT QcurrentTable: IN VitalQcurrentTableType;
                              CONSTANT DataIn : IN std_logic_vector;
                              SIGNAL Result : OUT Qcurrent
                              ) IS
  CONSTANT InputSize : INTEGER := DataIn'LENGTH;
  Constant DefaultReturnValue : Qcurrent := 0 na;
  -- This needs to be done since the TableLookup arrays must be ascending starting with 0
  VARIABLE TruthTableAlias : VitalTruthTableType(0 TO (TruthTable'LENGTH(1)-1),
                                                    0 TO (TruthTable'LENGTH(2)-1)) := TruthTable;
  VARIABLE DataInAlias : std_logic_vector(0 TO InputSize - 1) := To_X01(DataIn);
  BEGIN
    -- search through each row of the truth table
    col_loop: FOR i IN TruthTableAlias'RANGE(1) LOOP

```

```

-- Check each input element of the entry
row_loop: FOR j IN 0 TO InputSize LOOP
  IF (j = inputSize) THEN -- This entry matches
    -- Return the Result
    Result <= QcurrentTable(i);
  RETURN;
END IF;
EXIT row_loop WHEN not(TruthTableMatch(DataInAlias(j), TruthTableAlias(i, j)));
END LOOP row_loop;
END LOOP col_loop;
Result <= DefaultReturnValue;
END;

```

Additionally a current sensor is needed. This cell evaluates the total quiescent current that is flowing through a determined number of VITAL components (decided by the user). The sensor will be modeled by means of a VITAL Level 0 cell which reads the value of the current signal (Figure 1).

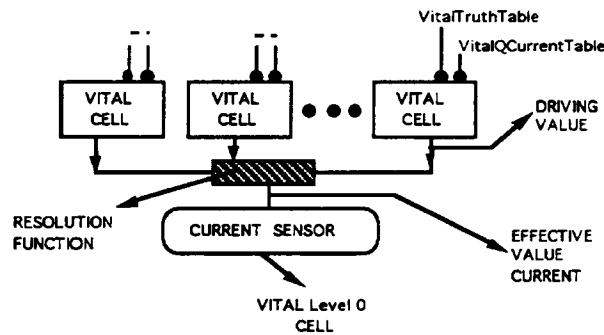


Figure 1

The types presented for the quiescent current modeling are not applicable to the dynamic current modeling. In order to be able to use this modeling, it is necessary to model the current flowing from the circuit to the sensor in a continuous form, considering all behavior to be transitory. This is not possible using an event-driven simulator. For this reason, the current waveform has been modeled by means of a piece-wise linear current waveform, in a similar way to [2][3][4]. This enables results to be obtained which are in disagreement with, at most, 10% of those obtained with SPICE [2]. The basic idea is that the current waveforms for any change in the inputs of the VITAL cell are modeled by the foundry as a set of time-current pairs as shown in Figure 2.

The VHDL simulator must be able to add up all of these piece-wise linear current waveforms to obtain the total current waveform. The new approach introduced in this paper is that the VHDL simulator does not work with current values, but rather with the points at which the slope of the waveform varies (break points). In this way, the total waveform can be calculated with great speed and accuracy. The basic idea is that in each cell a set of time-current pairs is specified, which model the consumption of this cell upon a certain change in the inputs (Figure 3).

Input change => current waveform =  $\{(0, i_0), (t_1, i_1), (t_2, i_2), \dots, (t_n, i_n)\}$

i = current  
t = Time

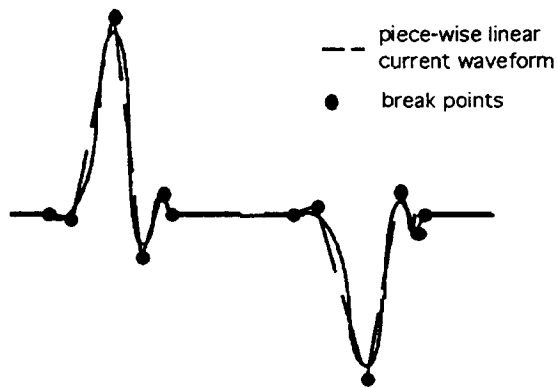


Figure 2

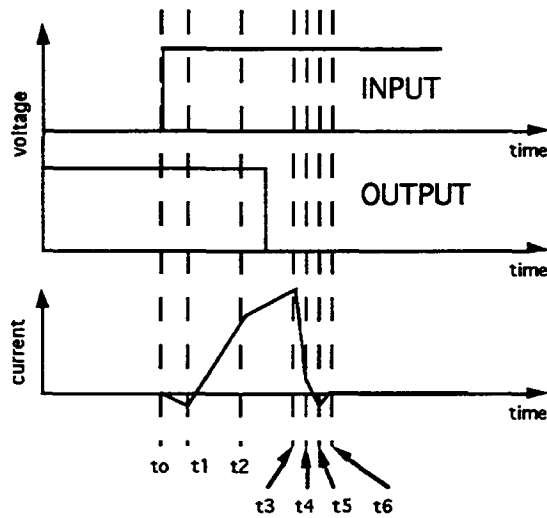


Figure 3

It should be pointed out that the model presented for quiescent current is a particular case of the dynamic model, in which “ $i_0$ ” is the quiescent current and  $n = 0$ . In our approach, the list of pairs presented previously is transformed into a set of linear equations as follows :

$$\begin{array}{ll}
 \text{Input change} \Rightarrow \text{Current} = & P_1 * T + C_1 & t_0 < T < t_1 \\
 & P_2 * T + C_2 & t_1 < T < t_2 \\
 & \dots & \\
 & P_N * T + C_N & t_{n-1} < T < t_n
 \end{array}$$

Where:

$$\begin{array}{l}
 T = \text{Continuous Time} \\
 P_j = (i_{j-1} - i_j) / (t_{j-1} - t_j) \\
 C_j = i_{j-1} - (P_j * t_{j-1})
 \end{array}$$

The current is modeled in VHDL by means of resolved signals of record type (Dcurrent) in which one field models the parameter P (slope) and the other the parameter C (initial point) of the linear equation.

```

type current is record
    P : REAL;
    C : REAL;
end record;

type CurrentArray is array( NATURAL range <>) of current;
function DynamicPowerSource( A : CurrentArray ) return current;
subtype Dcurrent is DynamicPowerSource current;

```

The calculation of the total current is defined as the sum of the currents of each cell. These are modeled by means of equations of the following equation type, in which (P, T) vary as discrete events.

$$\text{Cell\_current} = P_i * \text{Time} + C_i$$

Hence, the total current is calculated by means of the following expression :

$$\text{Total\_current} = \sum \text{Cell\_current} = (\sum P_i) * \text{Time} + (\sum C_i)$$

This equation is implemented in the resolution function of the Dcurrent type. Thus, this will be the type of signal used to model the power source.

```

function DynamicPowerSource( A : CurrentArray ) return current is
    variable result : current := (0.0 , 0.0);
    begin
        IF( A'LENGTH=1) THEN return( A(A'LOW)); ELSE
            for i in A'RANGE loop
                result.P := result.P + A(i).P;
                result.C := result.C + A(i).C;
            end loop;
        END IF;
    return result;
end;

```

The advantage of this type is that it allows the current waveform to be modeled with relative accuracy. From the point of view of VITAL, a new port ( of the Dcurrent type) is introduced in the cells which models the dynamic current consumption of the cell. Also, a generic port is added, enabling the {time-P-C} sets to be specified by means of the VitalCurrentTableType type. Every row of this table is associated to the transaction modeled in the same row of a VitalStateTableType constant. These tables are processed inside the cell by means of a VitalDynamicCurrent subprogram.

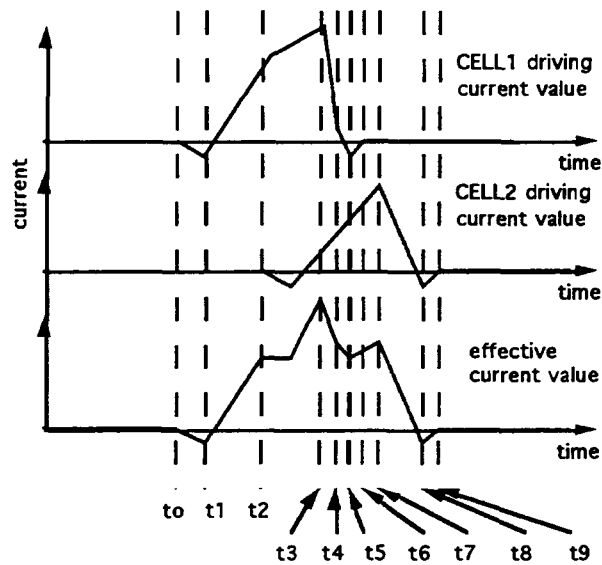


Figure 4

The types necessary for the simulation and the VitalDynamicCurrent subprogram are shown below:

```

type VitalCurrent is record
  Point : current;
  T : TIME;
end record;
type VitalCurrentTableType is array(NATURAL range <>, NATURAL range <>) of
VitalCurrent;
procedure VitalDynamicCurrent ( constant CurrentTable : in VitalCurrentTableType;
                                constant StateTable : in VitalStateTableType;
                                signal DataIn : in std_logic_vector;
                                signal Result : out current ) is

  constant InputSize : INTEGER:= DataIn'LENGTH;
  variable DataInAlias : std_logic_vector(0 to InputSize - 1);
  variable StateTableAlias : VitalStateTableType(0 TO (StateTable'LENGTH(1)-1),
                                                    0 TO (StateTable'LENGTH(2)-1)) := StateTable;
  variable CurrentTableAlias: VitalCurrentTableType(0 TO (CurrentTable'LENGTH(1)-
1),
                                                    0 TO (CurrentTable'LENGTH(2)-1)) := CurrentTable;
  type EventCurrentTableType is array (Natural range <>) of VitalCurrent;
  variable EventCurrentTable : EventCurrentTableType( 0 TO
(StateTable'LENGTH(1)-1));
  variable maxEventIndex : Natural := 0;
  variable nextEventIndex : Natural := 0;
  variable Index : INTEGER;
  variable currentTime : TIME;
  variable nextTime : TIME := TIME'HIGH;
  variable nextEvent: TIME;
  variable PrevTime : TIME;
  variable temp : current;
  variable PrevDataIn : Std_logic_vector(0 to DataIn'LENGTH-1):= (others=>'X');

```

```

begin

infinite: LOOP          -- Infinite loop
if(maxEventIndex /= nextEventIndex) then
    wait on DataIn for nextEvent;
    else
        wait on DataIn;
    end if;
DataInAlias:= To_X01(DataIn);
currentTime := now;
if(DataInAlias/=PrevDataIn) then
    if(NextEvent /= TIME'HIGH and nextEventIndex < maxEventIndex) then
        if(nextTime > currentTime) then
            NextEvent := nextTime - currentTime;
        else
            assert false
            report "Time computation error in VitalDynamicCurrent";
        end if;
    end if;
    assert( StateTable'LENGTH(1)=CurrentTable'LENGTH(1))
        report "Incorrect current table";
col_loop: FOR i IN StateTableAlias'RANGE(1) LOOP
    -- Check each input element of the entry
row_loop: FOR j IN 0 TO InputSize LOOP
    IF (j = inputSize) THEN -- This entry matches
        nextEventIndex:=0;
        maxEventIndex:=0;
        nextTime:=currentTime;
        PrevTime:= 0 ns;
        current_loop: for k in 0 to (CurrentTable'LENGTH(2)-1) LOOP
            EventCurrentTable(maxEventIndex).point.A :=
                CurrentTableAlias(i,k).point.A;
            EventCurrentTable(maxEventIndex).point.B :=
                CurrentTableAlias(i,k).point.B -
                (CurrentTableAlias(i,k).point.A
                 *(TimeToReal(currentTime)+ TimeToReal(PrevTime)));
            EventCurrentTable(maxEventIndex).T := CurrentTable(i,k).T;
            maxEventIndex := maxEventIndex +1;
            if(CurrentTable(i,k).T = TIME'HIGH) then
                exit current_loop;
            else
                PrevTime:= CurrentTableAlias(i,k).T + PrevTime;
            end if;
        end loop current_loop;
    exit col_loop;
    end if;
    exit row_loop when not
        StateTableMatch(PrevDataIn(j), DataInAlias(j), StateTableAlias(i,j));
    end LOOP row_loop;
end LOOP col_loop;
PrevDataIn := DataInAlias;
end if;
if(nextTime=currentTime and maxEventIndex > nextEventIndex) then -- update
events
    temp.A := EventCurrentTable(nextEventIndex).point.A;
    temp.B := EventCurrentTable(nextEventIndex).point.B;

```

```

    if(EventCurrentTable(nextEventIndex).T = TIME'HIGH) then
        nextTime := TIME'HIGH;
        nextEvent:= TIME'HIGH;
    else
        nextTime := EventCurrentTable(nextEventIndex).T + currentTime;
        nextEvent:= EventCurrentTable(nextEventIndex).T;
    end if;
    nextEventIndex := NextEventIndex +1;
    Result <= temp;
end if;
end LOOP ;
end;

```

### 3.Example of current modeling

As an example of quiescent current modeling, the description of a VITAL component (an inverter without delay) is proposed. In this cell, three new ports ( a Qcurrent type port and two generic ports of VitalQcurrentTableType and VitalTruthTableType types) are defined.

```

entity INV is
    generic( InputValues : VitalTruthTableType(0 to 3, 0 to 1):=      (('0','1'),      --1
                                                                    ('1','0'),      --2
                                                                    ('0','X'),      --3
                                                                    ('1','X'));      --4
            QcurrentValues : VitalQcurrentTableType(0 to 3):= (20 na,      --1
                                                                30 na,          --2
                                                                3 ma,          --3
                                                                1 ma));        --4

    port( A : in  Std_logic;
          Y : inout Std_logic;
          vdd: out Qcurrent := 0 na);
end;
architecture op2 of INV is
    signal DataIn : std_logic_vector(0 to 1);
    begin
        VitalINV(Y,A,OPEN,OPEN);
        DataIn <= A & Y;
        VitalQCurrentTable (InputValues, QcurrentValues, DataIn, vdd);
    end;
end;

```

Each row of the InputValue constant corresponds to the same row of the QcurrentValue constant. In this way, when the gate input value is '0' and the output is '1', the static current is 20 nA.

On the other hand, as an example of dynamic current modeling, the model of current flow at a two-input AND gate is proposed. For this, it will be necessary to define a set of parameters  $C_i$  (slope  $i$ ),  $P_i$  (initial point  $I$ ) and  $t_i$  (time between two changes of the current flow). These parameters are defined by the VitalDCurrentTableType type generic port. Each row is associated to a transition of the VitalTruthTableType type generic port. In this way, when an input of the AND gate changes from '0' to '1'(symbol '/'), the output being '0', the form of the current will be given by the following pairs:

[0,0 A],[500ps, 10 mA], [1000 ps, -1mA], [1200 ps, 0 A]

```

entity and2 is
  generic( InputTransition : VitalStateTableType( 0 to 3, 0 to 2):=( (      '/',      '1',
      '0'),      -- 1
      ( '1',      '/',      '0'),      -- 2
      ( 'B',      'B',      'r'),      -- 3
      ( 'B',      'B',      'f')));      -- 4

  CurrentWaveform      : VitalCurrentTableType( 0 to 3, 0 to 3):=(
  ( ((2.0e7,0.0), 500 ps),      ((-2.2e7,10.0e-3), 500 ps ),      --1
  ((5.0e6,-1.0e-3), 200 ps),      ((0.0,20.0e-9), TIME'HIGH)),
  ( ((2.0e7,0.0), 500 ps),      ((-2.2e7,10.0e-3), 500 ps ),      --2
  ((5.0e6,-1.0e-3), 200 ps),      ((0.0,20.0e-9), TIME'HIGH)),
  ( ((1.0e6,0.0), 1 ns),      ((0.0,1.0e-3),TIME'HIGH),((0.0,0.0),0 ns),((0.0,0.0),0 ns)),      --3
  ( ((1.0e6,0.0), 1 ns), ((0.0,1.0e-3),TIME'HIGH),((0.0,0.0),0 ns),((0.0,0.0),0 ns)) )      --4
  );
  port( InA,Inb : in Std_logic;
        Y : InOut Std_logic;
        vdd: Out Dcurrent := (0.0,0.0));
end ;

```

#### 4. Fault modeling

The previous sections define a quiescent and dynamic modeling technique of the current flowing through the VITAL cells. The methods of current testing are based on the variation of current consumption when a fault is produced. For this, it is necessary to introduce current sensors in the design, modeled as Level 0 VITAL cells. These sensors act as fault observation points.

The faults detectable by quiescent current test can be divided into two types:

- Internal faults in the cells. In these cases, the injection of faults in the gate is carried out by means of change in constant value in the VitalQcurrentTableType generic port (switch constant [1]). This possibility is very useful in embedded test systems, in which a VITAL cell may represent a very complex component (such as a microprocessor), whose test by means of IDDQ is modeled using the faulty values of the generic port which models the static power dissipation.

- Short-circuit faults. The consideration of the outputs in the calculation of quiescent current enables short-circuit faults between the gate outputs to be modeled. The fault is modeled as the identification of the two short-circuiting signals as a single signal. Its value will be defined by the resolution function of the "std\_logic" type. When this resolution provokes discrepancies in the output value at the port compared to that expected from the inputs, the values of short-circuit current will be used. These values are defined in the constant associated to the VitalQcurrentType type generic port which models this discrepancy (Figure 5).

In a similar way faults can be detected using IDDT. The only difference is that the fault simulator replaces the constant associated to the VitalQcurrentType type port of the correct circuit by one which models the internal fault behavior detectable by IDDT. In the short-circuit case, the detection is produced by considering the output in the calculation of the dynamic current. The injection of the short-circuit fault is produced in the same way as in the quiescent current case.

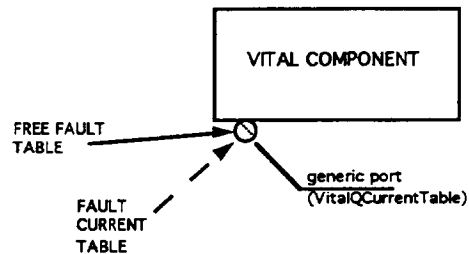


Figure 5

## 5. Conclusions

In this paper a current fault model has been presented for VHDL structural descriptions which follow the rules laid down by the VITAL standard. This fault model includes IDDQ and IDDT tests. For this, it has been necessary to define types and subprograms which model the current flow in the cell. These elements have not been introduced in the VITAL standard.

On the other hand, the simulator results seem to show little discrepancy compared to those obtained with electrical simulators like SPICE[2]. This permits the current modeling and the application of IDDQ and IDDT to test circuits. As a consequence of the application of the models presented here, better results have been achieved than those obtained by some non-VHDL commercial logic simulators [7], since:

- 1.- It enables the separate parts of the circuit under test to be modeled, since as many Qcurrent and Dcurrent type signals can be used as there are separate power-supply parts of the circuit.
- 2.- As well as short-circuit faults at the outputs, it also considers internal faults in the cells which are detectable by IDDQ or IDDT.
- 3.- It enables the maximum value of the static and dynamic current in the fault free circuit to be estimated, thus facilitating the design of the current sensor. In fact, the sensor could be incorporated as a Vital Level 0 cell in the system description.

At the moment effort is being made in two lines of development, to improve the current modeling part of this work. On one hand, to improve the modeling in the case where a signal changes while the port is still affected by a previous change, and on the other hand, to study the propagation of 'X'-values through the circuit.

## 6. References

- [1] J.L. Barreda et al.. "Fault Modelling in VITAL". Workshop on Libraries, Component Modelling, and Quality Assurance. 1995.
- [2] Rouatbi, B. Haroun, A. Al-khalili. "Power Estimation for Sub-Micron CMOS VLSI circuits". ICCAD'92. 1992 .
- [3] H. Kriplani, F. Najm and I. Hajj. "Maximum current estimation in CMOS circuits". 29th Design Automation Conference. 1992.
- [4] T. Krodel. "PowerPlay-Fast Dynamic Power Estimation Based on Logic Simulation". ICCD'91. 1991.
- [5] R. Aitken. "A Comparison of Defect Models for Fault Location with Iddq Measurements". ITC'93. 1993.
- [6] J. Arguelles et al. "Iddt Testing of Continuous-Time Filters". VLST'95. 1995.
- [7] SystemHilo 4.5. Release Notes. VEDA Design Automation. December 1994.