

Implementing Timed Logic Simulation in VHDL

By

Zainalabedin Navabi*, Irit Dahan*+, Tedd Corman+

* Electrical and Computer Engineering Department
Northeastern University
409 Dana Research Center
Boston, Massachusetts 02115
617-437-5413

+ *VIEW*logic Systems, Inc.
293 Boston Post Road West
Marlboro, MA 01751
508-480-0881

ABSTRACT:

Conventional logic simulators calculate node logic and delay values in separate processes. The delay value is used for scheduling the logic value on the node. In this paper we will present the Timed Logic Simulation (TLS) method, which allows us to combine logic and delay computation processes into one process. In addition to a faster simulation, using this method a netlist can be converted into a set of Boolean expressions, containing detailed timing and logic information. VHDL implementation of this method allows extracting functional models from netlist descriptions, the simulation accuracy of which is identical to netlist simulation.

1. Introduction

Conventional logic circuit simulation computes logic values and delay values separately. When these values are obtained for a node, logic values are applied to the node of the circuit at the relative time at which they are supposed to happen. Such simulators' software performance is inherently slow because delayed event scheduling is more costly than zero-delay or unit delay scheduling.

To overcome such limitations, we have developed a method for concurrent computation of a node's timing and logic values. These values are attached to a node as information fields and are propagated forward to all affected circuit nodes. This method schedule values on nodes in zero delay mode. It allows for the concurrent computation of logic values and delay values, while maintaining the functional correctness of the description. We will refer to this method of simulation as Timed Logic Simulation or TLS.

VHDL is a suitable language for the implementation of this method. VHDL can be used to describe complex logic functions and can support composite data values. Composite data values makes it possible to aggregate delay values and logic values.

The TLS method uses the behavioral capabilities of VHDL. Specifically, a logic function is described as a collection of concurrent Boolean equations. This logic function uses a composite data type in order to calculate delay values at the same time as logic values. In order to integrate TLS model with other simulation techniques, full support is provided for scheduling "real" delays on output signals on the boundaries of a particular circuit. The logic simulation of the TLS model is done in zero delay mode. As a result, it is possible to achieve higher levels of simulation performance.

Implementing TLS in VHDL allows the use of Boolean expressions in place of gate instantiations in a circuit description. This enables us to convert netlists to a readable set of Boolean expressions without any loss of timing accuracy. The TLS method also support register elements.

2. TLS method -

2.1 Methodology-

At the heart of the TLS method is the understanding that zero (unit) delay simulation is faster than true delay simulation. This faster simulation is especially useful in cases where timing is important only on the boundaries (from primary inputs to primary outputs) of logic functions. In order to exploit fast unit-delay simulation while still keeping track of actual delays, TLS uses a composite data type. The data type contains the current logic value, the old logic value, the effective delay, and a time stamp (representing the time of last change) as an aggregate.

TLS uses a modified truth table, which we refer to as the Timed Truth Table. Such a table includes, in addition to a new logic value, a column for the effective delay and for the previous logic value. Using a modified table for each logical (or arithmetic) operator, it is possible to describe complex functions in which delay is an integral part. These complex functions can be evaluated quickly, without the penalty of true delay simulation, because effective delays are computed simultaneously with logic values using the modified truth table. Time stamps, old logic values, and current logic values are used to compute the new logic values and the new effective delay. Finally, the effective delays are used on the boundaries to create a "real" delayed assignment.

For an illustration of this method consider the Timed Truth Tables (TTT) for the AND and OR operators as shown in **Figure 2.1** and in **Figure 2.2** respectively.

A'	A	B'	B	C'	C	D(delay)
0	0	0	0	0	0	-
0	0	0	1	0	0	-
0	0	1	0	0	0	-
0	0	1	1	0	0	-
0	1	0	0	0	0	-
0	1	0	1	0	1	9ns
0	1	1	0	0	0	-
0	1	1	1	0	1	9ns
1	0	0	0	0	0	-
1	0	0	1	0	0	-
1	0	1	0	1	0	7ns
1	0	1	1	1	0	7ns
1	1	0	0	0	0	-
1	1	0	1	0	1	9ns
1	1	1	0	1	0	7ns
1	1	1	1	1	1	-

Figure 2.1 - TTT for a two input AND gate.

A'	A	B'	B	C'	C	D(delay)
0	0	0	0	0	0	-
0	0	0	1	0	1	8ns
0	0	1	0	1	0	6ns
0	0	1	1	1	1	-
0	1	0	0	0	1	8ns
0	1	0	1	0	1	8ns
0	1	1	0	1	1	-
0	1	1	1	1	1	-
1	0	0	0	1	0	6ns
1	0	0	1	1	1	-
1	0	1	0	1	0	6ns
1	0	1	1	1	1	-
1	1	0	0	1	1	-
1	1	0	1	1	1	-
1	1	1	0	1	1	-
1	1	1	1	1	1	-

Figure 2.2- TTT for a two input OR gate.

The input columns of a TTT consist of old and new logic value (A', B' are old values, and A, B are new values). The output of the Table are the old and new logic values of the function (C', C), and the delay (D). This delay value represents the time delay between the old value of the output and the new output. For example, entry 7 of the AND TTT indicates that when signal A (input) changes from '0' to '1' the output changes from '0' to '1' 8 nanoseconds later. An example for the OR TTT is entry 2 which indicates that when signal B changes from '1' to '0', C changes from '1' to '0'; the delay between event on B and event on C is 6 nanoseconds. The delay entries are zero ("-") when no event occurs on the output as a result of an input change (or no change on any of the inputs).

Timed Truth Tables replace the conventional Truth Tables in the logic functions of standard logic gates, and the application of data to these tables results in logic values and delay values for general Boolean expressions. In order to illustrate the method for a general logic circuit, consider the circuit shown in **Figure 2.3**.

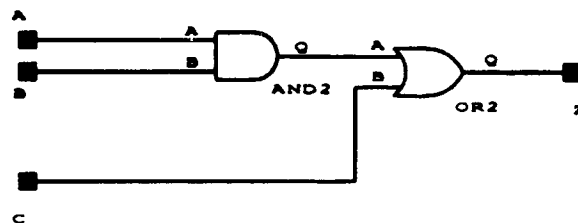


Figure 2.3 - $Z = (A \text{ AND } B) \text{ OR } C$.

Using Timed Truth Tables of **Figures 2.1** and **2.2**, logic and delay values of output Z of the above circuit can be calculated. Let's assume that at time t_0 input A changes from '1' to '0' while inputs B and C stay at '1' and '0' respectively. Since A changed from '1' to '0' and B from '1' to '1', the logic and delay values of the AND gate will be found from entry "1011" or row 11 of the TTT of AND of **Figure 2.1**. The output value in this row is '1' to '0', and the delay value is 7 ns.

In order to find the logic and delay value of output Z of the OR gate, the next lookup is in the table of **Figure 2.2**. The input of the OR gate connected to the output of the AND gate has an old-new value pair of "10". Since the C input has remained unchanged, its old-new value pair is "00". Concatenation of these value pairs results in "1000". A lookup in the OR TTT at "1000" (row 8) results in an old-new value pair of "10" with a delay value of 6 ns. The total delay at node Z from the time that A changes, is the sum of the two delays (7 ns + 6 ns), which is equal to 13 ns. As a result, output Z changes from '1' to '0' at time $t_0+13\text{ns}$.

In TLS, all logic and delay values in a circuit are calculated at the same time. A circuit consists of a set of components and the wires that interconnect these components. TLS represents circuit components as subprograms and interconnecting wires as composite data types. The subprograms take component input signals as arguments and return component output signals. The composite data types contain logic and delay values for a given signal.

2.2 TLS Database-

Parallel calculation of logic and delay values by the same subprograms allowed scheduling on the nodes in zero delay. This is the main reason for the increased speed in digital circuit simulation using the TLS method. Another factor which increases simulation speed is the relatively small data structure used for representation of circuits. The TLS method consists of two elements, components and signals. These elements must be represented in the environment of any simulator that uses the TLS method. Since components are handled by subprograms, there is no need for their representation within a data structure. Most interconnecting signals that carrying data from one component to another can be represented as a composite data type and used as arguments of subprograms that correspond to the components. Signals that are the primary inputs, primary outputs, connected to flip-flop inputs, or signals on a feedback path need to be explicitly declared, and therefore they have representation in the TLS data structure. During simulation, dynamic memory will be allocated to represent all other signals that the circuit needs in order to evaluate the circuit correctly. The memory will be freed after it is used. **Figure 2.4** shows signals that must be explicitly declared in the database, and those that do not require the assignment of an explicit data structure.

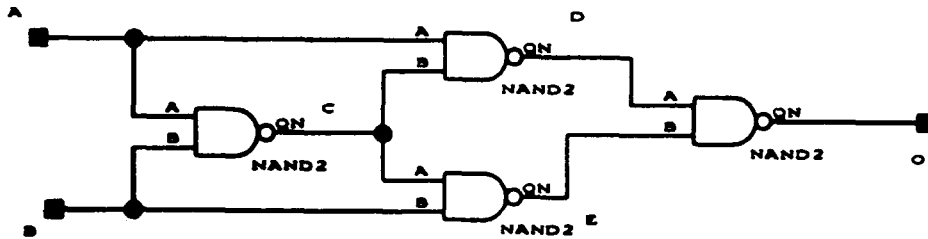


Figure 2.4 - Implementation of XOR gate.

```
C <= A NAND B;
O <= (A NAND C) NAND (C NAND B);
```

During simulation, the database will contain static composite data structures for the following signals:

- A, B - Primary inputs for this design.
- C - Signal fans out more than one gate.
- O - Circuit primary output.

When the circuit is stimulated on the primary inputs, the database will expand to include signals D and E. The memory associated with these signals (D and E) will be deallocated during simulation run time after they have are no longer required.

When circuits consists many of intermediate signals with fanouts of one, very few signals need to be explicitly declared. This results in a small data structure for the representation of the circuit.

2.3 Implementation -

Implementation of the TLS method is accomplished by creating functions for all basic logic operators. These functions will perform computations on the TTT instead of the standard truth tables. This requires the inputs and outputs of these functions to be a complex structure that includes old logic values, new logic values, delay values (to help propagate delay through components), and time stamp (time of the last change). The time stamp indicates the time of the most recent activity on a node.

A node signal representation is shown below:

$$S = \{ \text{old value, new value, delay, time} \}$$

For an illustration of the TLS of Boolean equation execution method, refer to **Figure 2.3** of the previous section.

If at time zero, A changes from '1' to '0'. The A node record will therefore be:

$$A = \{ '1', '0', \text{zero, zero} \}$$

Since the value of B remains the same, its record is:

$$B = \{ '1', '1', \text{zero, zero} \}$$

The AND function receives A and B as input. It ANDs the old logic values to obtain '1', ANDs the new logic values to obtain '0', adds the greater of the two delay values of the inputs (zero and zero) with the AND TTT's table delay value to obtain 7ns (entry 11 in the ANDs TTT), and then uses the current simulation time to form the record for node X as shown below:

$$X = \{ '1', '0', 7\text{ns, zero} \}$$

Since the value of C remains the same its record is:

$$C = \{ '0', '0', \text{zero, zero} \}$$

The OR function receives X and C as input. It ORs the old values to obtain '1', ORs the new values to obtain '0', adds the greater of the two delay values of the inputs (7ns and zero) with the OR table delay value (6ns) to obtain 13ns, and uses the current simulation time to form the record for node Z as shown below:

$$Z = \{ '1', '0', 13\text{ns, zero} \}$$

The process described above repeats for all circuit nodes, and at each step logic and delay values of internal nodes of the circuit will be calculated. Logic and time values will be reported only for signals at the boundary of a system that is being simulated.

The entries in the TTT are for a given simulation time. The signal composite data type includes a time stamp field which indicates when, during simulation time, the new logic value was last updated. The old logic value of a signal is current only if the timing of the new logic value matches the current simulation time. If the value corresponding to the new logic value of a signal has been in effect prior to the current simulation time, then the old logic value of this input is equal to the new logic value.

As an example, consider the circuit simulated earlier (**Figure 2.3**) and apply new stimulus to signals B and C at 100ns. Signal B at time 100ns changes from '1' to '0' and signal C changes from '0' to '1', and therefore the value of B and C are represented by the following records:

$$\begin{aligned} B &= \{ '1', '0', \text{zero, 100ns} \} \\ C &= \{ '0', '1', \text{zero, 100ns} \} \end{aligned}$$

Signal A remains the same as it was at zero.

$$A = \{ '1', '0', \text{zero, zero} \}$$

Since signals B and C change at 100ns, the Boolean equation will be reevaluated at 100ns. The AND function receives A and B as input. The time stamp of signal A does not match the current simulation time (signal A time stamp is zero, simulation time is 100ns). This implies that no event has occurred on this signal since time zero. We can therefore conclude that the old logic field value

(oldvalue) of signal A is old and should be updated to reflect the signal logic value during the previous simulation cycle. Signal A's old logic value is equal to the new logic value; therefore, the value of A is as shown below:

A = {'0', '0', zero, 100ns}

In order to evaluate **Figure 2.3**, the old logic value of A and B, as well as their new logic values, must be ANDed once the old logic value has been calculated. This results in a '0' for the old logic value and a '0' for the new logic value. The delay for this gate is calculated by adding the greater of the two delay values of the inputs (zero and zero) with the TTT's delay value. Since entry in the table at "0010" (A, B old and new logic values) is zero, the resulting delay value (entry two in the AND's TTT) will be zero. Finally, we include the current simulation time in the value record of signal X. The following represents this value.

X = {'0', '0', zero, 100ns }

Since there is a change on signal C at time 100ns, the OR function receives X and C as input. The time stamp of both signals are equal and match the current simulation time, therefore the Logic string is "0001". The first pair "00" represents signal X old and new logic value, and the second pairs "01" represent signal C old and new logic value. The Logic string "0001" is equivalent to entry one of the TTT of the OR function. Therefore, for the evaluation of the result of the OR operation, the greater of the two delay values of the inputs (zero and zero) should be added with the TTT's delay value (8ns). This result in 8ns for the delay of the OR gate. For the logic value results of the OR gate the old and new values of the operands are ORed together, which result in '0' and '1' respectively. Including the time stamp in the record representing Z, this value becomes the following:

Z = {'0', '1', 8ns, 100ns}

This shows that as a result of an event on signals B and C, at 100ns, an event on the output Z will occur 8ns after the changes on signals B and C happen (100ns + 8ns = 108ns). The circuit evaluation occurs in zero time, therefore delayed scheduling is not necessary. Circuits with feedback are an exception to this. In such cases, the feedback is broken and scheduling is done on the feedback path. In an event-driven simulation environment, such as VHDL, the feedback path value is applied to the input of the circuit after a small time delay (1 femtosecond).

2.3.1 Feedback Implementation -

When using the TLS method, signals that on a feedback path are evaluated infinite times. In order to handle feedback properly the feedback path must be broken and a new name must be assigned to its input side forming two explicit signals for every feedback path. This mechanism allows signal value snapshot before and after the evaluation of the expression. By comparing the old and new values of the signal on the input side of the feedback path with the old and new logic values of the output signal of the feedback path, any change in the signal values would become apparent. If there is a change in any of the logic values, the equation will be evaluated again. On the other hand, equivalent logic values on the two ends of a broken feedback path will cause the evaluation of the expression to stop and the evaluation loop to terminate. Example for feedback representation is presented in **Figure 2.7**.

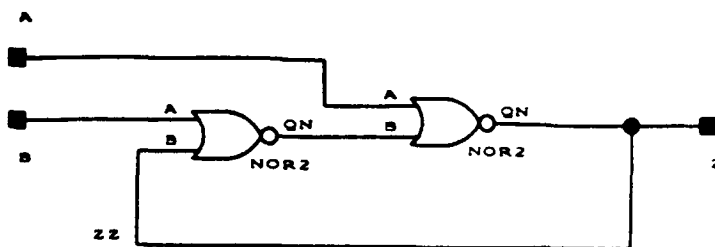


Figure 2.7 - Z <= A NOR (B NOR ZZ); feedback(Z,ZZ);

Signal Z and ZZ refer to the same signal but to different data structures. The "feedback" function will check for a change in signal Z as result of evaluation of the NOR function. If a change is found, the function will assign all Z's fields to ZZ fields, making it possible to continue the evaluation.

3. Implementation of TLS in VHDL.

In order to exercise the TLS method in an event-driven simulation environment, we have chosen to use VHDL and its simulator. VHDL has the capability to support composite data types and to describe complex logic functions as a collection of concurrent Boolean equations. Although the simulation of these equations is considered faster than simulation at other levels, e.g. gate level instantiation, calculating delay on a signal requires one simulation cycle (delta time in VHDL). Signal delays will be used to schedule values on signals until logic values are propagated to the outputs of a circuit. Each scheduling will take one simulation cycle. By using the TLS method, VHDL simulator will provide detail timing specification in simulation of Boolean equations. The results will be obtained in one simulation cycle.

Implementation of the TLS method in VHDL takes advantage of VHDL ability to overload functions. A package consisting of composite data type declarations, overloaded subprograms which support all logic operators, register elements, and Implementation of feedback, was created. Each subprogram evaluates logic and timing values in parallel.

3.1 Signal declaration in the TLS method.

Since the TLS method makes use of composite data types to represent signals, a record type NODE is declared within a VHDL package. Every field within the record NODE represent some information about the signal. The *value* field represents the signal logic value, the *oldvalue* field represents the signal logic value before last event occurred, *delay* field carry delays from previous component and the *timestamp* filed is the signal time stamp which shows when last (simulation time) event or transaction occur on the signal.

```
TYPE node IS RECORD
    value : BIT;
    oldvalue : BIT;
    delay: TIME;
    timestamp : TIME;
END RECORD;
```

Figure 3.1 signal type declaration.

All subprogram arguments and return values are of NODE type.

3.2 Supporting components in the TLS method.

Logical components have a functional description for the TLS method. Each function takes NODE types as arguments and returns a NODE type. Most logical operators consist of four main tasks for the implementation of a TTT for a specific Boolean operator. These four tasks are:

- (1) Extracting maximum delay value from inputs signals.
- (2) Generating the Logic string which provides us the right entry in the TTT's.
- (3) Calculating the current component delay and adding it to the maximum delay, propagated to the component by its arguments from previous components.
- (4) Generating accurate old and new logic value for the current component output.

Delays that are propagated into the function (component) through arguments will be considered only if the time stamp of the arguments (signals) match the current simulation time. If there is more then one signal in which the time stamp is the same as the simulation time (i.e. more then one input change

at the same time), the greater of all record *delay*'s field values will be assigned to the function returned record values. Delays that occur in earlier simulation cycles, have no effect on the current circuit evaluation and are considered to be ZERO.

When generating the function (component) delay itself, the function creates a logic string. The logic string is a combination of old and new values from all function arguments. Each pair within the logic string belongs to one of the function arguments. Concatenation of these pairs provides the TTT's entry for specific inputs. Each pair represents concatenation of the current logic value and the value of the signal during the previous simulation cycle. By testing the time stamp on signals records we can determine whether the old logic value is valid for use within the logic string. For example if the time stamp of a signal is current, the old value is valid. Otherwise, the old value to be used within the logic string will be the same as the new logic value. This occurs when the event has occurred on a signal in the previous simulation cycle.

When the combination between old and new logic values (logic string) is known, we add the proper delay to the delay propagated to the function by its arguments (task one in the function description). For example, consider the overloaded AND function of **Figure 3.2**. If the first signal (in1) changed from '0' to '1' while the second signal (in2) stayed '1', the "arr_sum" logic string remain "0111". This value of "arr_sum" causes 8ns to be added to the *delay* field of the resulting signal value.

```

FUNCTION "AND" (in1, in2 : node)      RETURN node IS
VARIABLE temp : node;
VARIABLE arr_sum : BIT_VECTOR(3 DOWNT0 0);
BEGIN
    temp.timestamp := NOW;

    -- SECTION ONE

    IF (in1.timestamp = temp.timestamp AND in2.timestamp = temp.timestamp) THEN
        IF in1.delay > in2.delay THEN
            temp.delay := in1.delay;
        ELSE
            temp.delay := in2.delay;
        END IF;
    ELSIF in1.timestamp = temp.timestamp THEN
        temp.delay := in1.delay;
    ELSIF in2.timestamp = temp.timestamp THEN
        temp.delay := in2.delay;
    ELSE
        temp.delay := 0fs;
    END IF;

    -- SECTION TWO

    IF (in1.timestamp = temp.timestamp AND in2.timestamp = temp.timestamp) THEN
        arr_sum := in1.oldvalue & in1.value & in2.oldvalue & in2.value;
    ELSIF (in1.timestamp < temp.timestamp AND in2.timestamp < temp.timestamp) THEN
        arr_sum := in1.value & in1.value & in2.value & in2.value;
    ELSIF in1.timestamp < temp.timestamp THEN
        arr_sum := in1.value & in1.value & in2.oldvalue & in2.value;
    ELSE
        arr_sum := in1.oldvalue & in1.value & in2.value & in2.value;
    END IF;

    -- SECTION THREE

    IF (arr_sum = "1010" OR arr_sum = "1011" OR arr_sum = "1110" ) THEN
        temp.delay := temp.delay + 6ns;
    ELSIF (arr_sum = "1101" OR arr_sum = "0111" OR arr_sum = "0101" ) THEN

```

```

        temp.delay := temp.delay + 8ns;
    END IF;

    -- SECTION FOUR

    temp.oldvalue := arr_sum(3) AND arr_sum(1);
    temp.value := arr_sum(2) AND arr_sum(0);

    RETURN (temp);
END;
```

Figure 3.2 - Implementation of AND gate.

3.3 Feedback Implementation-

As mentioned before, feedback is resolved by implementing a "feedback" function which takes NODE type arguments, when both arguments belong to the same signal. The first argument reflects the signal value after the equation evaluation and the second argument is the signal value before the equation was evaluated. The "feedback" function compares the two old logic values, and the two new logic values. If they are not equal, the entire first argument's record values will be assigned to the second argument after delta delay. This will cause the Boolean equation to be evaluated again in the next simulation cycle. This mechanism causes the loop to be broken. The circuit stabilizes only when both arguments are equal. The VHDL code to implement this functionality is shown in **Figure 3.3**.

```

PROCEDURE feedback (SIGNAL in1: IN node; SIGNAL in2 :INOUT node)
VARIABLE delta : TIME := 1fs;
BEGIN
    IF NOT (in1.value = in2.value AND in1.oldvalue = in2.oldvalue) THEN

        -- cause the timestamp field to match simulation time .
        in2.timestamp <= in1.timestamp + delta AFTER delta;

        in2.value <= in1.value AFTER delta;
        in2.oldvalue <= in1.oldvalue AFTER delta;
        in2.delay <= in1.delay AFTER delta;
    END IF;
END;
```

Figure 3.3- Feedback Implementation.

4. Conclusion and application of the TLS method-

While TLS significantly improves the readability of a description, the improvements in the simulation time of a TLS VHDL model were not significant. During realization of the TLS method performance, we found that the VHDL simulator spends more time scheduling signals events (of record type) than evaluating the circuit. Since TLS method is based on features which exist in most other programming languages, by implemented the TLS method in the C language for instance we can achieved a better time performance yet. In spite of the inefficiency that the VHDL simulators showed in simulating zero time delay models, we believe that by optimizing the simulator the TLS method would provide a faster simulation than simulation of structural level models. The TLS method was implemented in the C language as well, resulting in a significant improvement of simulation performance.