

Ternary Logic Simulator Using VHDL

A.P.Dhande^{*}, R.C.Jaiswal^{**} and S.S.Dudam^{*}

^{*} Pune Institute of computer Technology, Pune, INDIA, Pune Institute of computer Technology, Pune, INDIA

ashwpict@yahoo.com

shridhar_dudam@yahoo.co.in

^{**} Pune Institute of computer Technology, Pune, INDIA

jaiswalrc@yahoo.com

Abstract: Today, the complexity of binary digital hardware system is steadily increasing. Due to this fact, new approaches for efficiently describing and implementing the digital systems are investigated. Multivalued logic approach offers several advantages over existing binary digital system. Today's Existing simulation tools do not permit the simulation of MVL circuits. An objective of proposed work is to describe how existing VHDL can be used as potential EDA tool for the simulation of MultiValuedLogic circuits and system. Here we have considered signal 'Z' as one of the state of MultiValuedLogic System along with signals '0' & '1' (0 being ground potential, Z intermediate state and 1 as +5v state). The VHDL modeling and simulation of T-Gates and 1-bit multiplier circuit is described and commented.

Key words: VHDL, T-Gates, MultiValuedLogic, literals, CMOS, Ternary.

INTRODUCTION

Since last few decades Multiple-Valued Logic (MVL) has been possible alternative to binary logic. The binary logic is limited to only two states '1' and '0', where as MVL is a set of finite or infinite number of values. An MVL system has a radix greater than 2 [1][2].

The MVL is implemented in two modes i.e. current mode and voltage mode. In current operation, MVL states are defined in terms of output current, which is an integral multiple of reference current and in voltage mode, MVL states are in terms of distinct voltage levels [3]. Today's VLSI technology offers ways to realize MVL circuits in order to bring their full potential into many operational circuits [4].

The proposed work deals with the use of Very High-Speed Integrated Circuit Hardware Description Language as a logic simulator to evaluate the performance of MVL circuits [5]. Here we present a concept to model the MVL circuits and discuss its performance with respect to verification of desired functionality of MVL, gate propagation delay and loading effect. The concept is elaborated with some examples considered here.

An idea is to provide a way to synthesis MVL system using VHDL is to feed both circuit descriptions and circuit specification to a synthesis tool.

VHDL can be used to model, simulate and describe binary system where signals inside the circuit can take the other values than 0 & 1 [6][7][8]. The 9-

state StdLogic_1164 package level values are listed in table 1.

Table 1: 9 state logic systems

Symbols	Values
U	Uninitialized
X	Unknown
0	Logic 0
1	Logic 1
Z	High impedance
W	Weak Unknown
L	Weak Zero
H	Weak One
-	Don't Care

To demonstrate the use of VHDL as a ternary logic simulator we have used Logic 0 to represent logic 0 state, High impedance Z to represent logic 1 state and Logic 1 to represent logic 2 state.

1. Ternary logic: Preliminaries

Ternary logic is 3-valued switching digital system. For supply voltage of 5 v, 3 levels of ternary system are defined as: logic 0 = ground potential, logic 1 as 2.5V and logic 2 as 5v.

Ternary logic gates are the basic building blocks of any ternary circuits, which switch among above three levels. Basic ternary gates is ternary inverter, which is used for constructing ternary AND/NAND, ternary OR/NOR etc.

Yoeli-Rosinfeld algebra [9] defines three type of inverter, the STI (simple ternary inverter) PTI (positive ternary inverter) and NTI (negative ternary inverter) such that,

$$\begin{aligned} \text{STI} &= \overline{X^1} = 2 - X \\ \text{PTI, NTI} &= \overline{X^i} = i \quad \text{if } X \neq i \\ &\quad 2 - i \quad \text{if } X = i \end{aligned}$$

Where i take the value 2 for PTI and 0 for NTI. Truth table for three types of inverter is shown in table 2.

Based on this inverter other T-gates i.e. T-AND/NAND, ternary OR/NOR etc. are constructed [10][11].

Ternary AND/NAND function is defined as:

$$\begin{aligned} \text{T-AND} &= X_1 X_2 \dots X_n = \min [X_1 X_2 \dots X_n] \\ \text{T-NAND} &= \overline{X_1 X_2 \dots X_n} = \min [\overline{X_1 X_2 \dots X_n}] \end{aligned}$$

And

Ternary OR/NOR function is defined as:

$$\begin{aligned} \text{T-OR} &= X_1 X_2 \dots X_n = \max [X_1 X_2 \dots X_n] \\ \text{T-NOR} &= \overline{X_1 X_2 \dots X_n} = \max [\overline{X_1 X_2 \dots X_n}] \end{aligned}$$

Table 3 shows truth table for AND/NAND and OR/NOR T-gates.

Table 2: Truth table for Ternary Inverter

Input X	Output		
	STI	PTI	NTI
0	1	1	1
Z	Z	1	0
1	0	0	0

Table 3: Truth table for T-gates

Input		Output			
X1	X2	AND	NAND	OR	NOR
0	0	0	1	0	1
0	Z	0	1	Z	Z
0	1	0	1	1	0
Z	0	0	1	Z	Z
Z	Z	Z	Z	Z	Z
Z	1	Z	Z	1	0
1	0	0	1	1	0
1	Z	Z	Z	1	0
1	1	1	0	1	0

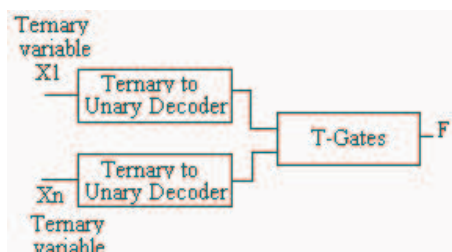


Fig.1 Ternary function implementation

For implementing ternary function, an approach is to convert given ternary variable into unary variable using ternary to unary decoder [12], and implement the circuit using T-gates [13]. Fig.1 shows implementation of ternary function.

2. Implementation of T-Gates Using VHDL

The procedure for implementing T-gates can be summarized as follows:

Step 1: Establish the truth table for T-gates.

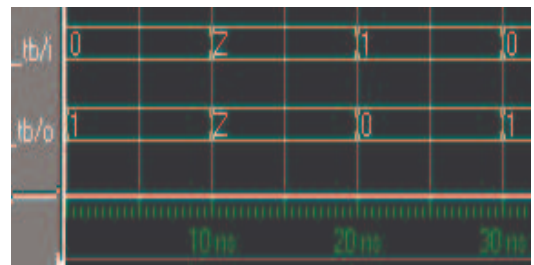
Step 2: Decide the description style of VHDL. This can be behavioral, Dataflow, structural or mixed of all offered by VHDL.

Step 3: Develop the VHDL constructs i.e. packages, libraries, components and entities.

Step 4: Use the construct developed in step 3 to implement ternary functions.

Step 5: Use the features of the simulator to conduct the verification of the functionality and timing specifications of the ternary system. The simulator normally displays signal waveforms using one level for logic 1, one level for logic 0, and a third level (shown half-way in between logic 1 and logic 0) for all other levels, strengths, disconnection, don't cares, unknowns, etc. (normally distinguished by colors and/or other attributes). One way to circumvent this problem is to build the VHDL constructs such that Z is intermediate level, which is displayed by blue color in the simulation [14].

The VHDL construct written for STI is given **Appendix I** and simulation result is shown below.



Simulation result for STI

3. Design Examples

The utility of proposed simulator is explained with the design examples of ternary half adder and 1-Bit ternary multiplier. An intention is to show how proposed simulator can be use to simulate MVL circuits and to evaluate system performance.

CASE 1: Design of ternary Half-Adder.

Ternary adder is a circuit that adds two ternary digits and generates sum and carry. Truth table for ternary adder is shown in table 4 along with k-map for

sum and carry in Fig.2 & implementation circuit in Fig.3.

Table4: Truth table for Half-Adder

X1	X2	Sum	Carry
0	0	0	0
0	Z	Z	0
0	1	1	0
Z	0	Z	0
Z	Z	1	0
Z	1	0	Z
1	0	1	0
1	Z	0	Z
1	1	Z	Z

The logic design procedure is:

From input variable decide number of decoders [12] requires for function implementation.

- 1) Construct truth table and K-map for input variable
- 2) Find cell grouping if possible i.e. grouping of 1x3, 2x3&3x3 of 2's &1's terms. For grouping of 1's 2 can be considered as don't care terms.
- 3) Deduce, Minimize and Realize the switching functions in the form of $F=F_2+1.F_1$, where F_2 is terms containing 2's only & F_1 is terms of 1's only. Logical AND operation is represented by \cdot .

From above procedure output equation of Half-Adders is:

$$F_{sum} = X^1_Z X^0_1 + X^Z_Z X^Z_1 + X^0_Z X^1_1 + Z [X^Z_Z X^0_1 + X^0_1 X^Z_1 + X^1_Z X^Z_1]$$

$$F_{carry} = Z [X^1_Z X^Z_1 + X^1_Z X^1_1 + X^Z_Z X^1_1]$$

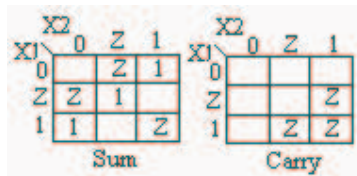


Fig.2: K-Map for Half-Adder

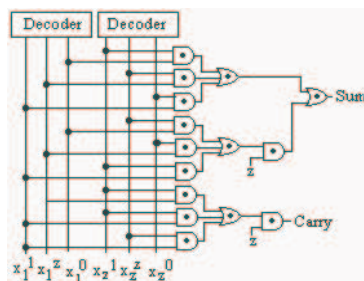


Fig.3:Circuit for Half-Adder

CASE 2:Design of 1-Bit Multiplier

Ternary multiplier is a circuit that multiplies two one bit numbers and generates product & carry. Truth table for multiplier is shown in table 5 & fig.4 shows K-Map. From K-map output equations for product & carry are:

$$F_{prod} = X^Z_Z X^1_1 + X^1_Z X^Z_1 + Z [X^Z_Z X^1_1]$$

$$F_{carry} = Z [X^1_Z X^1_1]$$

Fig .5 shows implementation of multiplier.

Table 5: Truth table for 1-Bit Multiplier

X1	X2	Product	Carry
0	0	0	0
0	Z	0	0
0	1	0	0
Z	0	0	0
Z	Z	Z	0
Z	1	1	0
1	0	0	0
1	Z	1	0
1	1	1	1

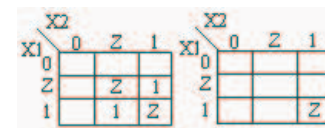


Fig.4: K-Map for 1-Bit multiplier

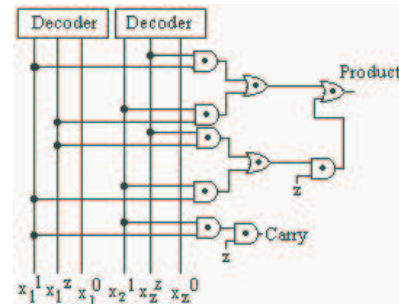


Fig .5:Implementation of multiplier

The VHDL code for ternary Half-adder & multiplier is given in **Appendix II**.

4. Simulation Results

Simulation results for above examples are shown in fig.6 & 7 respectively. The intermediate state is shown by blue color. The simulation result verifies truth table of the Half-Adder and 1-Bit multiplier. From simulation, overall delay for STI is 9.62 ns. & For multiplier is 9.76ns.

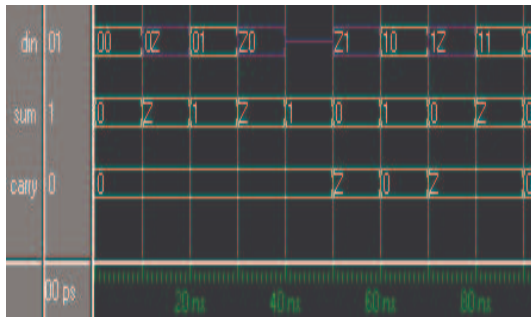


Fig.6: Simulation result for Half-Adder.

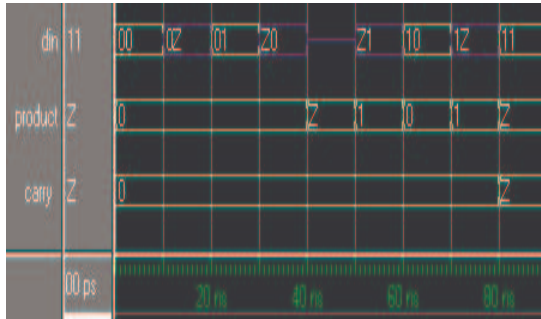


Fig.7: Simulation result for 1-Bit Multiplier.

5. Conclusion

To develop ternary circuits & to verify their performance, tools for digital simulation, layout generation & extraction are necessary.

Supported by two typical examples the proposed work describes basic procedure & establishes VHDL as potential tool for simulation of ternary circuits and systems. Further, in proposed work additional features can be incorporated to provide enough information to verify timing information.

The proposed simulator can be used to synthesis & to verify the performance of ternary logic circuits.

Work proposed in [9] is based on C-MOS implementation of T-gates where as in proposed work we made the use of three logic levels as ternary logic levels.

7. References:

- [1] O.Mirmotahari & Y.Berg 'A novel D-latch in multiple-valued semi-floating gate recharged logic' Proc.ISMVL 2004.
- [2] K.C.Smith 'Multiple Valued Logic: A tutorial & application' IEEE Tran. Computer Vol.21 p.p.17-21 April 1988.
- [3] R.Mariani, F.Pessolano & R.Saletti 'A new CMOS ternary logic design for low power low voltage circuit' Tutorial University of Pisa, Italy.
- [4] C.Rozon 'On the use of VHDL as a Multi-valued Logic Simulator' Proc. ISMVL 1996, p.p.110-115.
- [5] IEEE 'IEEE standard VHDL language reference manual, IEEE standard 1076-

1987, march 1988.

[6] Armstrong J.R. 'Chip level modeling with VHDL' PHI 1989.

[7] Harr R. & Staneulescu A. 'Applications of VHDL to circuit design' Kluwer academic publishers, Boston, 1991.

[8] Lipsett R., Schaefer C. & Vssery C. 'VHDL: Hardware description language & Design' Kluwer academic publishers, Boston, 1989.

[9] M.Yoeli, G.Rosenfeld 'Logic design of ternary switching circuit' IEEE Tran.Computer, Vol. C-14, P.P.19-29, Feb.1965.

[10] A.P.Dhande & V.T.Ingole 'Design of 3-Valued R-S & D flip-flops based on simple ternary gates' International journal of software engineering & knowledge engineering, Vol.15, No.2, pp.411-417, April 2005.

[11] A.P.Dhande & V.T.Ingole 'Design of Ternary R-S & D- Flip-Flops' International Academy of sciences: Enformatika Vol.4 Feb.2005 ISSN 1305-5313, p.p.61-65.

[12] P.C.Balla & A.Antonioniou 'Low power dissipation MOS ternary logic family' Proc.IEEE jour. on solid state circuits, vol.sc-19.no.5, p.p.739-749, october 1984.

[13] A.P.Dhande & V.T.Ingole 'Synthesis & implementation of 1-bit ternary ALU' Proc.IEEE-ACE 2003.

[14] S.Brown & Z.Vranesic 'Fundamentals of digital logic with VHDL design' TMH.

APPENDIX I: PACKAGE & CODE FOR STI

Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;

```
PACKAGE ternary_types IS
TYPE ternarylogic IS ('0','Z','1');
TYPE t_logic_vector is array (natural range <>) of
ternarylogic;
function stnot (l: ternarylogic) return ternarylogic;
function ptnot (l: ternarylogic) return ternarylogic;
function ntnot (l: ternarylogic) return ternarylogic;
function stand (l: ternarylogic; r: ternarylogic)
return ternarylogic;
function stor (l: ternarylogic; r: ternarylogic)
return ternarylogic;
function stnand (l: ternarylogic; r: ternarylogic)
return ternarylogic;
function stnor (l: ternarylogic; r: ternarylogic)
return ternarylogic;
function ntnor (l: ternarylogic; r: ternarylogic)
return ternarylogic;
END ternary_types;
PACKAGE BODY ternary_types is
type t_logic_ld is array (ternarylogic) of ternarylogic;
type t_logic_table is array (ternarylogic, ternarylogic)
of ternarylogic;
constant stand_table : t_logic_table :=
((('0', '0', '0'), ('0', 'Z', 'Z'), ('0', 'Z', '1'));
```

```

constant stor_table : t_logic_table :=
((0, 'Z', '1'), ('Z', 'Z', '1'), ('1', '1', '1') );
constant stnand_table : t_logic_table :=
((1, '1', '1'), ('1', 'Z', 'Z'), ('1', 'Z', '0') );
constant stnor_table : t_logic_table :=
((1, 'Z', '0'), ('Z', 'Z', '0'), ('0', '0', '0'));
constant ntnor_table : t_logic_table :=
((1, '0', '0'), ('0', '1', '0'), ('0', '0', '0'));
constant stnot_table : t_logic_ld := ('1', 'Z', '0');
constant ptnot_table : t_logic_ld := ('1', '1', '0');
constant ntnot_table : t_logic_ld := ('1', '0', '0');
Function stand (l: ternarylogic; r: ternarylogic)
return ternarylogic is
begin
return (stand_table(l, r));
end stand;
function stor (l: ternarylogic; r: ternarylogic)
return ternarylogic is
begin
return (stor_table(l, r));
end stor;
function stnand (l: ternarylogic; r: ternarylogic)
return ternarylogic is
begin
return (stnand_table(l, r));
end stnand;
function stnor (l: ternarylogic; r: ternarylogic)
return ternarylogic is
begin
return (stnor_table(l, r));
end stnor;
function ntnor (l: ternarylogic; r: ternarylogic)
return ternarylogic is
begin
return (ntnor_table(l, r));
end ntnor;
function stnot (l: ternarylogic)
return ternarylogic is
begin
return (stnot_table(l));
end stnot;
function ptnot (l: ternarylogic)
return ternarylogic is
begin
return (ptnot_table(l));
end ptnot;
function ntnot (l: ternarylogic)
return ternarylogic is
begin
return (ntnot_table(l));
end ntnot;
END ternary_types;

```

TERNARY NOT (STI) GATE

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.ternary_types.ALL;
ENTITY ternary_not IS
    PORT (I : IN ternarylogic;
          O : OUT ternarylogic);
END ternary_not;
ARCHITECTURE not_fn OF ternary_not IS
BEGIN

```

```

    O <= stnot(I);
END not_fn;

```

APPENDIX II: EXAMPLES

CASE I: TERNARY HALF-ADDER

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.ternary_types.ALL;
entity ter_half is
    Port ( din : in t_logic_vector(1 downto 0);
          sum, carry : out ternarylogic );
end ter_half;
architecture halfadder of ter_half is
    component ternary_decoder
        PORT(a: IN ternarylogic;
             X: INOUT t_logic_vector (2 downto 0));
    end component;
    component adder_part
        Port ( D1,D2,D3,D4,D5,D6 : in ternarylogic;
              HAP : out ternarylogic );
    end component;
    signal DX, DY: t_logic_vector (2 downto 0);
    signal hap1, hap2, hap3 : ternarylogic;
begin
    U1: ternary_decoder port map (din(0),DX);
    U2: ternary_decoder port map (din(1), DY);
    U3: adder_part port map (DY(0),DX(2),
                           DY(1),DX(1),DY(2),DX(0),hap1);
    U4: adder_part port map (DY(1),DX(0),
                           DY(0),DX(1),DY(2),DX(2),hap2);
    U5: adder_part port map (DY(1),DX(2),
                           DY(2),DX(1),DY(2),DX(2),hap3);
    carry <= stand(hap3,'Z');
    sum <= stor(hap1, stand(hap2,'Z'));
end halfadder;

```

Case II: TERNARY MULTIPLIER

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.ternary_types.ALL;
entity ter_multiplier is
    Port ( Din : in t_logic_vector(1 downto 0);
          Product, carry : out ternarylogic );
end ter_multiplier;
architecture multiplier of ter_multiplier is
    component ternary_decoder IS
        PORT(a: IN ternarylogic;
             X: INOUT t_logic_vector (2 downto 0));
    END component ternary_decoder;
    component multi_part is
        Port ( D1,D2,D3,D4 : in ternarylogic;
              MProd : out ternarylogic );
    END component multi_part;
    signal DX, DY : t_logic_vector (2 downto 0);
    signal p1,p2,: ternarylogic;
begin
    U1: ternary_decoder port map (Din(0), DX);
    U2: ternary_decoder port map (Din(1), DY);
    carry <= stand(stand(DX(2),DY(2)), 'Z') ;
    U3: multi_partport
        map(DX(1),DY(2),DX(2),DY(1),p1);

```

```
U4:multi_partport  
map(DX(1),DY(1),DX(2),DY(2),p2);  
product <= stor(p1, stand(p2,'Z'));  
end multiplier;
```

* Code for ternary decoder is not given here but can be obtained from author on request.

