Modeling Of Combinational Circuits Based On Ternary Multiplexer Using VHDL

A. Sathish kumar
Dept. of Electronics & Communication Engineering,
Amrita Vishwa Vidyapeetham,
Amrita School of Engineering,
Bangalore, India.

A. Swetha Priya
Dept. of Electronics & Communication Engineering,
Amrita Vishwa Vidyapeetham,
Amrita School of Engineering,
Bangalore, India.

***Abstract*—This paper presents a novel method for defining, analyzing, testing and implementing the basic combinational circuitry with VHDL Simulator. This paper shows the potential of VHDL modeling and simulation that can be applied to Ternary switching circuits to verify its functionality and timing specifications. A novel method is brought out for implementing the basic combinational circuitry with minimum number of multiplexers. It also includes 1-bit and 2-bit position shifter and Barrel shifter. Method of coding is illustrated with respect to block diagram. An intention is to show how proposed simulator can be used to simulate MVL circuits and to evaluate system performance.**

Keywords-MVL; 9-state logic system; Reliability-Unreliability model; VHDL

I. INTRODUCTION

The binary logic is limited to only two states '1' and '0', where as Multi-Valued Logic (MVL) is a set of finite or infinite number of values [1]. The MVL is implemented in two modes i.e. current mode and voltage mode. In current mode, 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. Today's VLSI technology offers ways to realize MVL circuits in order to bring their full potential into many operational circuits. Many authors have directed their efforts to the implementation of Multi-Valued logic looking for benefit from all advantages it possess over the binary logic. It is possible for ternary logic to achieve simplicity and energy efficiency in digital design since the logic reduces the complexity of interconnects and chip area, in turn, reducing the chip delay [2, 3]. It also offers better utilization of transmission channels because of the higher information content carried by each line. It gives more efficient error detection and correction codes and possesses potentially higher density of information storage. One of the main advantages of ternary logic is that it reduces the number of required computation steps. Furthermore, serial and serial-parallel arithmetic operations can be carried out faster if the ternary logic is employed.

In 1964, Alexander showed that natural base e=2.71828, where e is called Euler constant is the most efficient radix for implementation of switching circuits [4,5]. The base $r=e^2=7.389056099$ is considered to be more advantageous and most often used in electronic computers till 20^{th} century with digits 0 and 1 only. The 20^{th} century brought an alternative to base-2(binary) with focus on ternary radix $r=e^3=20.08553692$. As the value of radix increases, the information carrying capacity of each connection also increases. Hence, multi-valued logical systems, for instance, a three-valued (radix 3) digital realization would be more appropriate than binary. Ternary(or three-valued) means a switching element, which switches among 3 levels namely true, false and intermediate or correspondingly 0, 1 and 2 voltage levels [6].

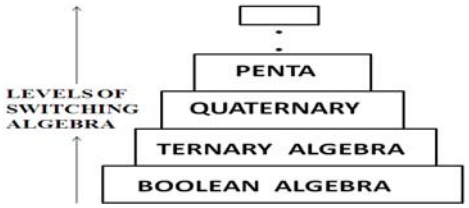


Figure 1. Levels of switching Algebra

Expanding the existing logic levels to ternary and higher levels as shown in Fig.1, higher processing rates could be achieved in various applications like memory management, communication throughput and domain specific computation. An evident advantage of a ternary representation over binary is economy of digits. To represent a number in binary system, one needs 58% more digits than that of ternary. For example, to represent a 15-digit decimal number, one requires 34 ternary digits instead of 54 binary digits. Ternary representation admits sign convention also. This is the reason why ternary is casting its applications in the field of Fuzzy logic, Machine Learning, Artificial Intelligence, Data Mining, Robotics, Digital signal processing, Digital control systems and Image Processing [7]. It is mainly applied in new transforms for encoding and compression, error correction, state assignment, representation of discrete information and in automatic telephony. These benefits have been shown to be useful for the design of ternary computers, for digital filtering.

The proposed work in this paper deals with the use of Very High-Speed Integrated Circuit Hardware Description Language (VHDL) as a logic simulator to evaluate the performance of MVL circuits [8]. Here we present a concept to model the ternary combinational and arithmetic circuits with minimum number of multiplexers showing the potential

and ease in the design of ternary circuits as well as its modeling using EDA tool for its performance with respect to verification of desired functionality of MVL. In Section II, preliminaries to ternary algebra and VHDL for MVL circuits are introduced. In Section III, 3x1 MUX is taken as a basic building block to explore the realization of circuits like 9x1 MUX, 27x1 MUX, Half Adder, Half Subtractor, Full Adder, Full Subtractor, Multiplier, 1-bit and 2-bit Comparator, Ripple Carry adder and Carry Save adder, 1-bit and 2-bit Position shifter and Barrel shifter. Conclusion is given in Section VI and Annexure gives the corresponding codes for circuits given in section III.

II. VHDL FOR TERNARY ALGEBRA

VHDL can be used to model, simulate and describe ternary system where signals inside the circuit can take tristate logic i.e., the usual true and false states, with a third transparent high impedance state. The third (intermediate) state can be metaphorically thought of as either an unambiguously true or unambiguously false. In real life situations also uncertainty in decisions (probabilistic decisions) occur rather than a clear cut decision of yes/no requirement such as yes/no/may be, open/close/half open or half close, up/down/straight or left/right/straight. This emphasizes that there is always an uncertainty in deciding the values. Rather, one would come across situations wherein one has to accept multiple decisions like the extent to which the decision may be true or may be false. The Reliability-Unreliability model [9] is sufficient enough to support the decision methodology statistically as shown in Fig. 2. Mathematically, it implies that reliability will be interpreted by the inequality $0 \le R \le 1$. If R=1, it is interpreted to be absolutely true and if R=0, it is interpreted to be absolutely false. If R being in the vicinity of 0.5 could be interpreted as neither true nor false and above 0.5 is said to be reliability level and below 0.5 is said to be unreliability level.

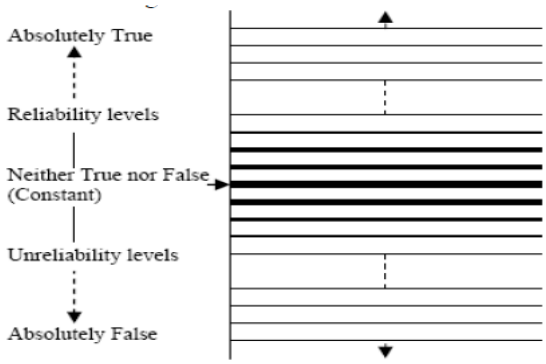


Figure 2. Truth value decision based on Reliability-Unreliability model

The proposed VHDL simulator can be used to synthesize & to verify the performance of ternary logic circuits with the help of technology dependent package called 9-state StdLogic_1164 package [10] whose levels are listed in Table I which allows the description of circuits based on TTL, CMOS, GaAs, NMOS, PMOS and ECL devices. To demonstrate the use of VHDL as a ternary logic simulator, we have used Logic-0 to represent 0 volt, High impendence Z to represent 1volt and Logic-1 to represent 2 volts.

VHDL provides an effective way to connect several logic outputs to a single input, where all but one is forced to the high impedance state, allowing the remaining outputs to operate in the normal binary sense. This concept is commonly used for memory bank connection in computers and other similar devices to a common data bus where a large number of devices can communicate over the same channel, simply by ensuring only one is enabled at a time.

TABLE I. 9-STATE LOGIC SYSTEM

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

III. DESIGN AND VERIFICATION OF TERNARY CIRCUITS

In this paper, 3x1 MUX is taken as a basic building block to explore the realization of circuits with minimum number of ternary 3X1 MUX (Multiplexer) with techniques of EDA tools. The design concept is implemented based on ternary K-map method for ternary function minimization.

A. Design of Basic Gates

To design ternary multiplexer, we start with the basic gates and design of decoder which are building blocks for any circuitry. The basic building gates are Ternary Inverter, Ternary OR (TOR), Ternary AND (TAND) and Ternary XOR (TXOR) [11-13] which is symbolized and represented as given in Fig.3 and Eq. (1-7) respectively.

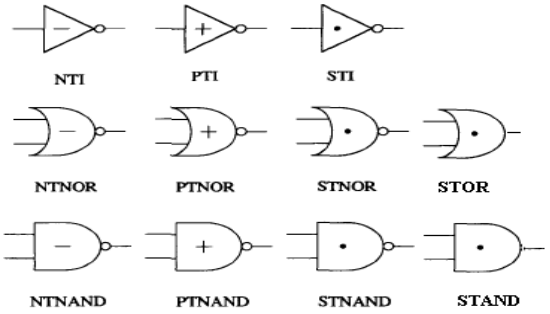


Figure 3. Symbols for basic gates

$$STI = \overline{X^{1}} = 2 - X \quad (1)$$

$$PTI, NTI = \overline{X^{i}} = \begin{cases} i & \text{if } X \neq i \\ 2 - i & \text{if } X = i \end{cases} \quad (2)$$

The VHDL package is built as shown in Annex I. The VHDL code given in Annex II-case I for unary functions is built as shown in Fig. 4(a) and Table II. The simulation results are shown in Fig. 4(b). Similarly, Annex II-case II shows the code for ternary Inverter which functions as tabulated in Table II and Fig. 5 shows the simulation result.

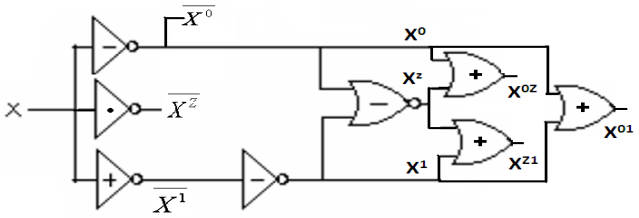


Figure 4(a). Circuit for obtaining unary functions

TABLE II. OPERATING TABLE OF UNARY FUNCTIONS

X	X^0	X^Z	X^1	X^{0Z}	X^{Z1}	X^{01}	$\overline{X^0}$ NTI	$\overline{X^Z}$ STI	$\overline{X^1}$ NTI
0	1	0	0	1	0	1	1	1	1
Z	0	1	0	1	1	0	0	Z	1
1	0	0	1	0	1	1	0	0	0

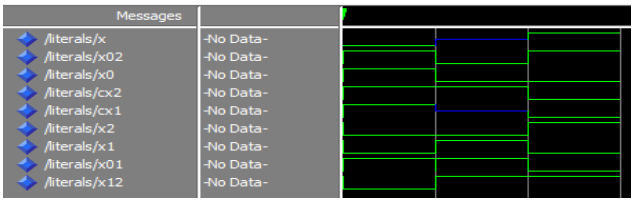


Figure 4(b). Simulation result of unary functions

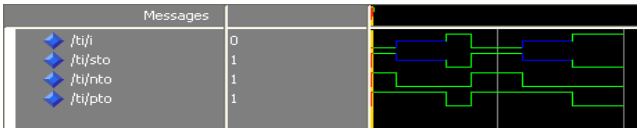


Figure 5. Simulation result of Ternary Inverter

Logic Sum or TOR:

$$X1 + X2 + \ldots + Xn = MAX(X1, X2, \ldots, Xn) \quad (3)$$

Logic Product or TAND:

$$X1 \bullet X2 \bullet \ldots \bullet Xn = MIN(X1, X2, \ldots, Xn) \quad (4)$$

Similarly, TNAND is

$$\overline{X1 \bullet X2 \bullet \ldots \bullet Xn} = MIN(\overline{X1 \bullet X2 \bullet \ldots \bullet Xn}) \quad (5)$$

TNOR is

$$\overline{X1 + X2 + .. + Xn} = MAX(\overline{X1 + X2 + .. + Xn}) \quad (6)$$

Annex II-case III shows the code for basic ternary gates which functions as tabulated in Table III and Fig. 6 shows the simulation results.

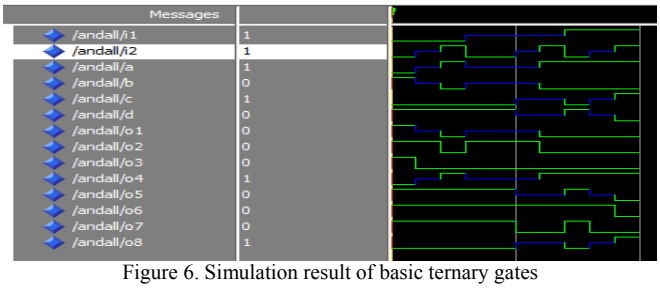


Figure 6. Simulation result of basic ternary gates

Ternary Ex-OR function is mod-3 addition of ternary numbers and is represented as shown in Fig.7(a) and Eq. (7). Modulo-3 sum [14] is the sum of two integers ignoring the carry digits in the addition as shown in Table IV. Annex II-case IV shows the coding of Modulo-Sum and simulation result is shown in Fig. 7(b). Redundant code techniques use half-adding functions which require Modulo-3 addition.

$$X \oplus Y = MODSUM(X, Y) = (X + Y) \bmod 3 \quad (7)$$

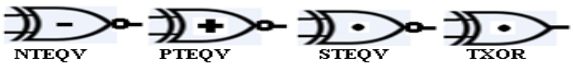


Figure 7(a). Symbols of TXOR & TEQV gates

TABLE IV. OPERATING TABLE OF MODULO-3 SUM OPERATORS

A	B	TXOR	STEQV	PTEQV	NTEQV
0	0	0	1	1	1
0	Z	Z	Z	1	0
0	1	1	0	0	0
Z	0	Z	Z	1	0
Z	Z	1	0	0	0
Z	1	0	1	1	1
1	0	1	0	0	0
1	Z	0	1	1	1
1	1	Z	Z	1	0

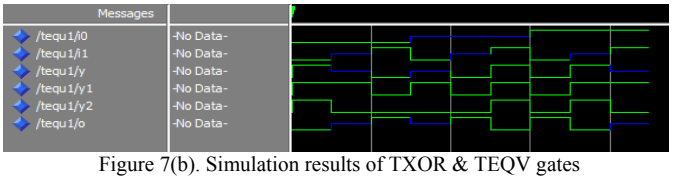


Figure 7(b). Simulation results of TXOR & TEQV gates

B. *Design of 3x1Multiplexer*

An approach for implementing ternary function is to convert given ternary variable into unary variable using ternary to unary decoder as shown in Fig.8. A decoder shown in Fig.9(a) is a combinational circuit that converts the ternary information from n input lines to a maximum of 3^n unique output lines [15-18]. The code is given in Annex II-case V which functions as given in Table II and the simulation results as shown in Fig. 9(b).

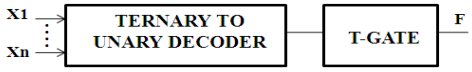


Figure 8. Implementation of ternary function

A ternary multiplexer is a combinational circuit that selects one of the 3^n input lines based, on a set of n selection lines and directs it to a single output line. Normally, there are 3^n inputs which come from a decoder and n select lines whose bit combinations determine which input to select. The design of 3X1 multiplexer (MUX) is as presented in Fig.10 and operates as given in Table V. The structural model is given in Annex II-case VI and the simulation results in Fig. 10(b).

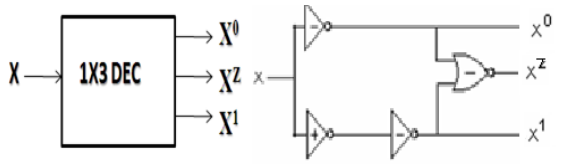


Figure 9(a). Block diagram and Circuit diagram of 1x3 Decoder

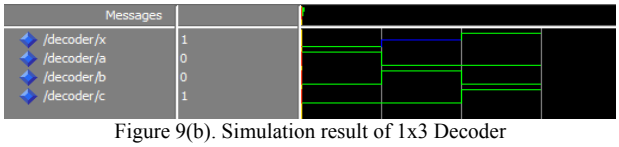


Figure 9(b). Simulation result of 1x3 Decoder

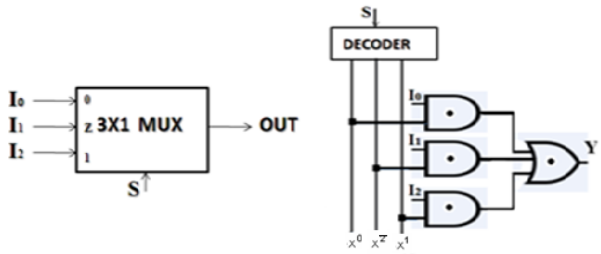


Figure 10(a). Block and circuit diagram for 3x1 Multiplexer

TABLE V. FUNCTION TABLE OF 3X1 MULTIPLEXER

S	Y
0	I0
Z	I1
1	I2

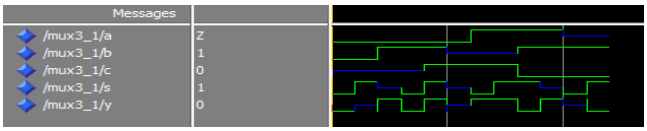


Figure 10(b). Simulation result of 3x1 Multiplexer

C. Design of 9x1Mux Using 3x1MUX

A 9x1 MUX is built using four 3x1 MUX as shown in Fig.11(a). A 9x1 MUX has 2 select lines to give an output among 9 inputs as given in Table VI and Fig.11(a) shows the simulation result for structural coding given in AnnexIII-caseI.

D. Design of Half Adder and Subtractor using 3x1 MUX

A ternary half adder (HA) is a circuit that adds two bits and generates a sum and carry using Modulo-3 addition. The HA functions as shown in Table VII and can be realized as given in Fig.12 (a). Annex III-case II shows the coding model which on simulation gives waveform shown in Fig 12(b). Similarly, ternary half subtractor (HS) is a circuit that will subtract one from the other number (i.e., A-B) and generate a difference and borrow using ternary logic. It functions as shown in Table VIII and can be realized as given in Fig.13(a). Annex-III-case III gives the code for HS which is illustrated with wave form as shown in Fig.13(b).

E. Design of Full Adder & Full Subtractor using 3x1 MUX

A full adder (FA) is a circuit that will add three bits and generates a sum and a carry. Fig.14(a-b) shows the design of FA and simulation results respectively as given in Table IX. Annex III-case IV gives the code for FA. Similarly, Full Subtractor (FS) is a circuit that will subtract three bits (i.e., (A-(B-Bin)), and generates a difference and borrow. Fig.15(a-b) shows the design of FS and simulation results respectively, as given in Table X. Annex III-case V gives the code for FS.

F. Design of 1-bit Multiplier using 3x1 MUX

A Multiplier multiplies two bits and generates the product as shown in Table XI. The structural modeling is given in Annex III-case VI according to the design in Fig.16(a) and the simulation results are given in Fig.16(b).

G. Design of 1-bit Comparator using 3x1 MUX

A magnitude comparator is a combinational circuit that compares two bits A & B and determines their relative magnitudes. The comparison of two bits is an operation that determines if one number is greater than, less than or equal to other number as tabulated in Table XII. The design of 1-bit comparator is shown in Fig.17(a). It gives Y=f(A>B) when en=0, Y=f(A=B) when en=1 and Y=f(A<B) when en=2. Fig.17(b) shows the simulation results with respect to Annex III-case VII.

H. Design of 2-bit Comparator using 27x1 MUX

The design of 27x1 MUX is shown in Fig.18(a) functioning as in Table XIII. The structural modeling is given in Annex III-case VIII and the simulation result is given in Fig.18(b). The design of 2-bit Comparator needs three 27x1 MUX and one 3x1 MUX which selects one of the function as given in the Fig.19(a). If en=0, Y=f(A_1A_0>B_1B_0), if en=1, Y=f(A_1A_0=B_1B_0) and if en=2, Y=f(A_1A_0<B_1B_0) and Table XIV shows the operation. The structural modeling is given in Annex III-case IX and simulation result is shown in Fig.19(b).

I. Design of Ripple Carry Adder and Carry Save Adder

Two binary words, each of n bits, can be added using ripple carry adder as shown in Fig.20(a). The carry input is connected to the least significant bit and the carry output of each full adder is connected to carry input of the next most significant FA. It is a typical example of iterative circuit which is slow, since in the worst case a carry must propagate from least significant FA to the most significant one. The structural modeling is given in Annex III-case X and simulation result is shown in Fig.20(b). Carry save adder computes the sum of 3 or more n-bit numbers which gives the sequence of partial sum and carry bits. The structural modeling is given in Annex III-case XI and simulation result is shown in Fig.21(b).

J. Design of 1-bit & 2bit position shifter and Barrel Shifter

The shifter circuit is designed as shown in Fig. 22(a), shifts the bits of n input vectors by 1-bit position to the right. It fills the vacant bit on the left side with zero. Table XV gives the function table of shifter circuit. If S=0, the input is loaded as output which is said to work in parallel mode, if S=1, the input is shifted by one bit position and if S=2, the input is shifted by two-bit positions padded by zeros in the left position. The

more versatile shifter circuit will be able to shift by more bit positions at a time. If the bits that are shifted out are placed into the vacated positions on the left, then the circuit effectively rotates the bits of input vector by a specified number of bit positions. Such a circuit is often called a Barrel shifter. Barrel shifter [19] works as given in Table XVI and design is as given in Fig. 23(a). The shifter takes the parallel load for S=0 and for S=1, the output is shifted by one position and values are rotated. Similarly for S=2, the output is shifted by two bit position and rotated circularly. Annex II-case XII and case XIII shows the VHDL code for position shifter and barrel shifter respectively. The simulation result for shifter circuitry and Barrel shifter is shown in Fig.22(b) and Fig.23(b), respectively.

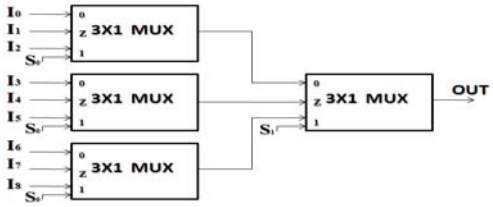


Figure 11(a). Block diagram using 3x1 MUX

TABLE VI. FUNCTION TABLE OF 9X1MULTIPLEXER

S1	S0	OUT
0	0	I0
0	Z	I1
0	1	I2
Z	0	I3
Z	Z	I4
Z	1	I5
1	0	I6
1	Z	I7
1	1	I8

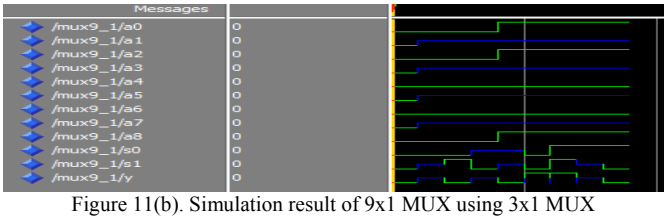


Figure 11(b). Simulation result of 9x1 MUX using 3x1 MUX

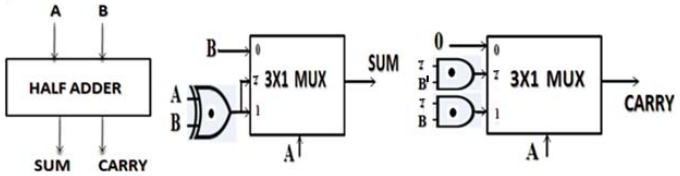


Figure 12(a). Block diagram of half adder using 3x1 MUX

TABLE VII. FUNCTION TABLE FOR HALF ADDER

A	B	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

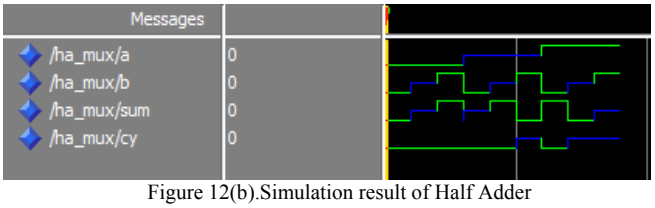


Figure 12(b).Simulation result of Half Adder

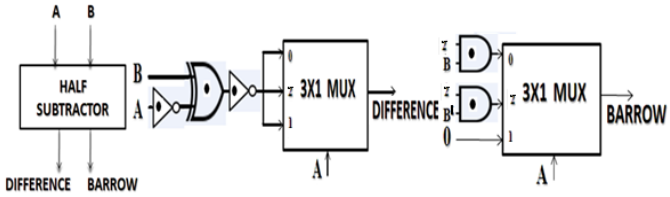


Figure 13(a). Block diagram of Half Subtractor using 3x1 MUX

TABLE VIII. FUNCTION TABLE FOR HALF SUBTRACTOR

A	B	DIFF	BORR
0	0	0	0
0	Z	1	Z
0	1	Z	Z
Z	0	Z	0
Z	Z	0	0
Z	1	1	Z
1	0	1	0
1	Z	Z	0
1	1	0	0

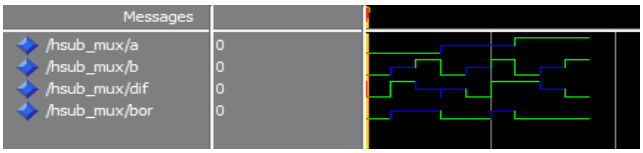


Figure 13(b). Simulation result of Half Subtractor

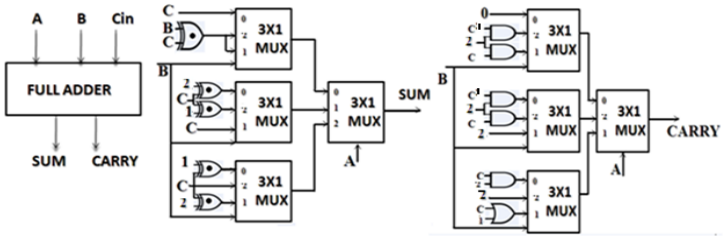


Figure 14(a). Block diagram of Full Adder using 3x1 MUX

TABLE IX. FUNCTION TABLE FOR FULL ADDER

A	B	Cin	SUM	CARRY
0	0	0	0	0
0	0	Z	Z	0
0	0	1	1	0
0	Z	0	Z	0
0	Z	Z	1	0
0	Z	1	0	Z
0	1	0	1	0
0	1	Z	0	Z
0	1	1	Z	Z
Z	0	0	Z	0
Z	0	Z	1	0
Z	0	1	0	Z
Z	Z	0	1	0
Z	Z	Z	0	Z
Z	Z	1	Z	Z
Z	1	0	0	Z
Z	1	Z	Z	Z
Z	1	1	1	Z
1	0	0	1	0
1	0	Z	0	Z
1	0	1	Z	Z
1	Z	0	0	Z
1	Z	Z	Z	Z
1	Z	1	1	Z
1	1	0	Z	Z
1	1	Z	1	Z
1	1	1	0	1

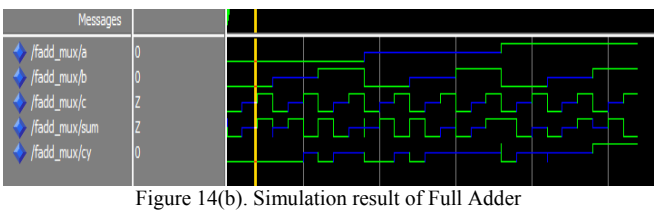


Figure 14(b). Simulation result of Full Adder

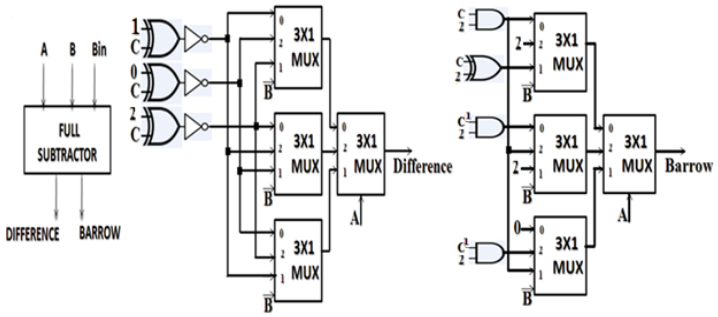


Figure 15(a). Block diagram of Full Subtractor

TABLE X. FUNCTION TABLE FOR FULL SUBTRACTOR

A	B	Bin	DIFF	BORR
0	0	0	0	0
0	0	Z	1	Z
0	0	1	Z	Z
0	Z	0	1	Z
0	Z	Z	Z	Z
0	Z	1	0	Z
0	1	0	Z	Z
0	1	Z	0	Z
0	1	1	1	1
Z	0	0	Z	0
Z	0	Z	0	0
Z	0	1	1	Z
Z	Z	0	0	0
Z	Z	Z	1	Z
Z	Z	1	Z	Z
Z	1	0	1	Z
Z	1	Z	Z	Z
Z	1	1	0	Z
1	0	0	1	0
1	0	Z	Z	0
1	0	1	0	0
1	Z	0	Z	0
1	Z	Z	0	0
1	Z	1	1	Z
1	1	0	0	0
1	1	Z	1	Z
1	1	1	Z	Z

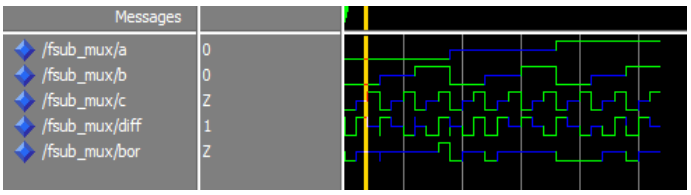


Figure 15(b). Simulation result of Full Subtractor

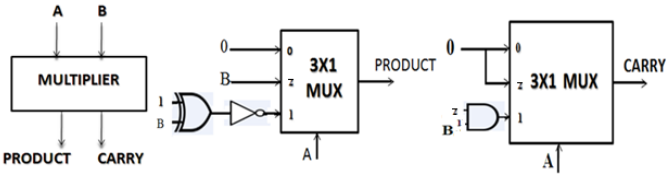


Figure 16(a). Block diagram of 1-bit Multiplier using 3x1 MUX

TABLE XI. FUNCTION TABLE FOR 1-BIT MULTIPLIER

A	B	PROD	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	Z	Z

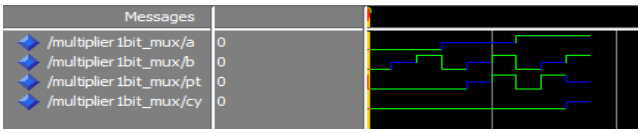


Figure 16(b). Simulation result of 1-bit Multiplier using 3x1 MUX

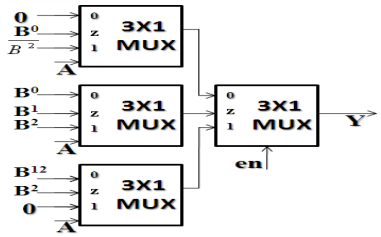


Figure 17(a). Block diagram of 1-bit Comparator

TABLE XII. FUNCTION TABLE FOR 1-BIT COMPARATOR

A	B	A>B	A=B	A<B
0	0	0	1	0
0	Z	0	0	1
0	1	0	0	1
Z	0	1	0	0
Z	Z	0	1	0
Z	1	0	0	1
1	0	1	0	0
1	Z	1	0	0
1	1	0	1	0

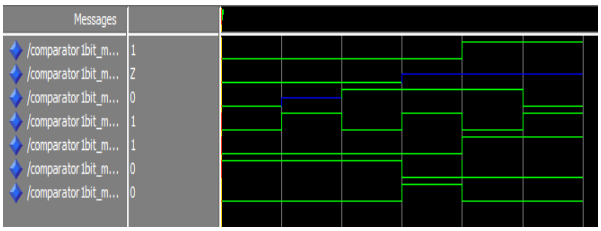
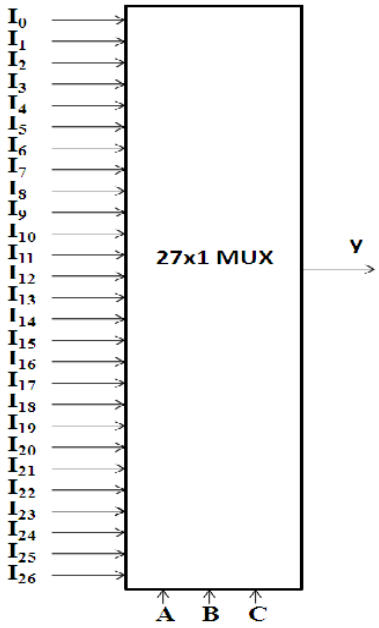


Figure 17(b). Simulation results of 1-bit Comparator using 3x1 MUX

TABLE XIII. FUNCTION TABLE FOR 27X1 MUX

A	B	C	Y
0	0	0	I0
0	0	Z	I1
0	0	1	I2
0	Z	0	I3
0	Z	Z	I4
0	Z	1	I5
0	1	0	I6
0	1	Z	I7
0	1	1	I8
Z	0	0	I9
Z	0	Z	I10
Z	0	1	I11
Z	Z	0	I12
Z	Z	Z	I13
Z	Z	1	I14
Z	1	0	I15
Z	1	Z	I16
Z	1	1	I17
1	0	0	I18
1	0	Z	I19
1	0	1	I20
1	Z	0	I21
1	Z	Z	I22
1	Z	1	I23
1	1	0	I24
1	1	Z	I25
1	1	1	I26

Figure 18(a). Block diagram of 27x1 MUX



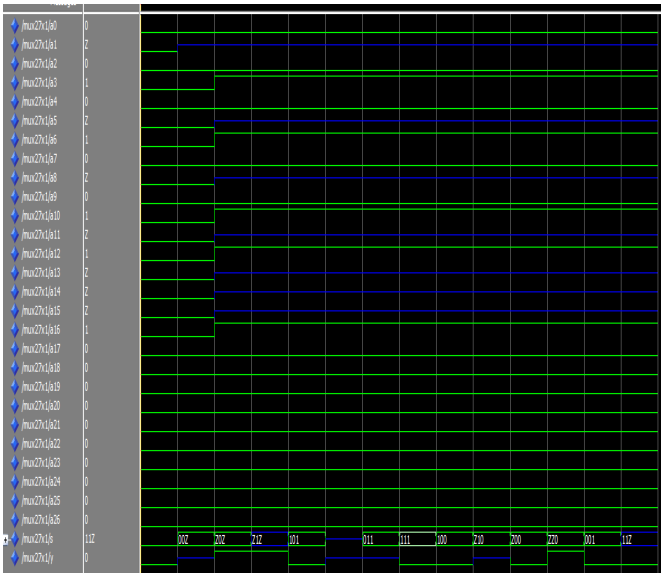


Figure 18(b). Simulation result of 27x1 MUX

TABLE XIV. FUNCTION TABLE FOR 2-BIT COMPARATOR

A1	A0	B1	B0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	Z	0	0	1
0	0	0	1	0	0	1
0	0	Z	0	0	0	1
0	0	Z	Z	0	0	1
0	0	Z	1	0	0	1
0	0	1	0	0	0	1
0	0	1	Z	0	0	1
0	0	1	1	0	0	1
0	Z	0	0	1	0	0
0	Z	0	Z	0	1	0
0	Z	0	1	0	0	1
0	Z	Z	0	0	0	1
0	Z	Z	Z	0	0	1
0	Z	Z	1	0	0	1
0	Z	1	0	0	0	1
0	Z	1	Z	0	0	1
0	Z	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	Z	1	0	0
0	1	0	1	0	1	0
0	1	Z	0	0	0	1
0	1	Z	Z	0	0	1
0	1	Z	1	0	0	1
0	1	1	0	0	0	1
0	1	1	Z	0	0	1
0	1	1	1	0	0	1
Z	0	0	0	1	0	0
Z	0	0	Z	1	0	0
Z	0	0	1	1	0	0
Z	0	Z	0	0	1	0
Z	0	Z	Z	0	0	1
Z	0	Z	1	0	0	1
Z	0	1	0	0	0	1
Z	0	1	Z	0	0	1
Z	0	1	1	0	0	1
Z	Z	0	0	1	0	0
Z	Z	0	Z	1	0	0
Z	Z	0	1	1	0	0
Z	Z	Z	0	1	0	0
Z	Z	Z	Z	0	1	0
Z	Z	Z	1	0	0	1
Z	Z	1	0	0	0	1
Z	Z	1	Z	0	0	1
Z	Z	1	1	0	0	1
Z	1	0	0	1	0	0
Z	1	0	Z	1	0	0
Z	1	0	1	1	0	0
Z	1	Z	0	1	0	0
Z	1	Z	Z	1	0	0
Z	1	Z	1	0	1	0
Z	1	1	0	0	0	1
Z	1	1	Z	0	0	1
Z	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	Z	1	0	0
1	0	0	1	1	0	0
1	0	Z	0	1	0	0
1	0	Z	Z	1	0	0
1	0	Z	1	1	0	0
1	0	1	0	0	1	0
1	0	1	Z	0	0	1
1	0	1	1	0	0	1
1	Z	0	0	1	0	0
1	Z	0	Z	1	0	0
1	Z	0	1	1	0	0
1	Z	Z	0	1	0	0
1	Z	Z	Z	1	0	0
1	Z	Z	1	1	0	0
1	Z	1	0	1	0	0
1	Z	1	Z	0	1	0
1	Z	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	Z	1	0	0
1	1	0	1	1	0	0
1	1	Z	0	1	0	0
1	1	Z	Z	1	0	0
1	1	Z	1	1	0	0
1	1	1	0	1	0	0
1	1	1	Z	1	0	0
1	1	1	1	0	1	0

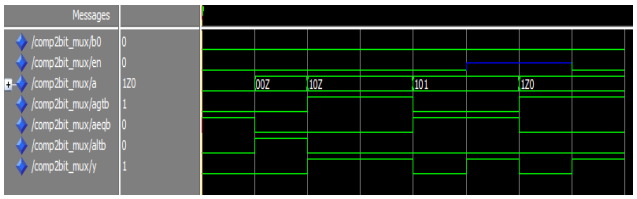


Figure 19(b). Simulation result of 2-bit Comparator

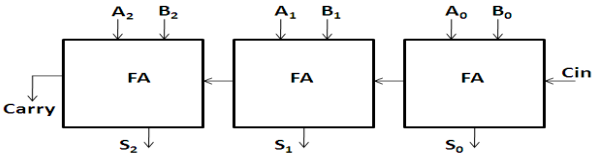


Figure 20(a). Block diagram of Ripple Carry Adder

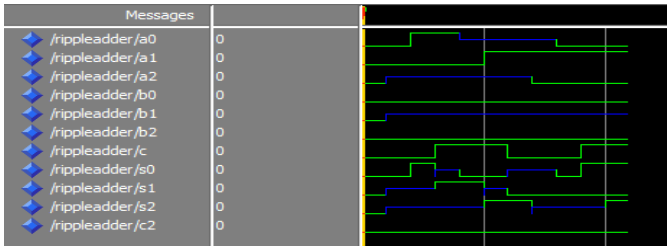


Figure 20(b). Simulation result of Ripple Carry Adder

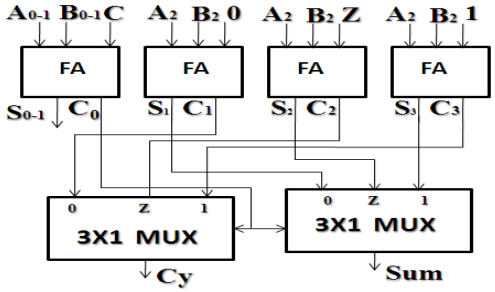


Figure 21(a). Block diagram of Carry Save Adder

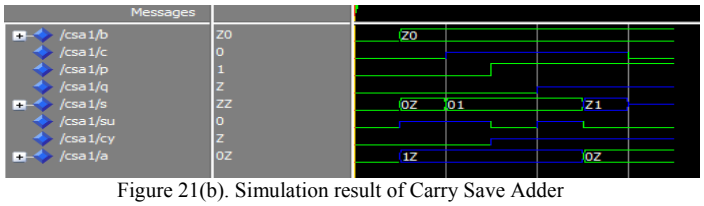


Figure 21(b). Simulation result of Carry Save Adder

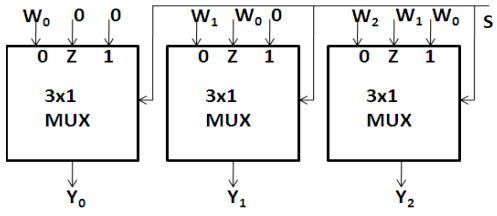


Figure 22(a). Block diagram of 1-bit & 2-bit Position Shifter

TABLE XV. FUNCTION TABLE FOR 1-BIT & 2-BIT POSITION SHIFTER

X	Y0	Y1	Y2
0	W0	W1	W2
Z	0	W0	W1
1	0	0	W0

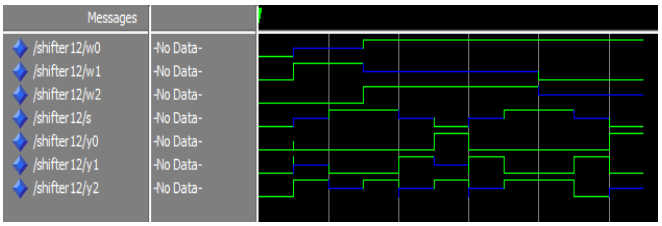


Figure 22(b). Simulation result of 1 & 2 bit Position Shifter

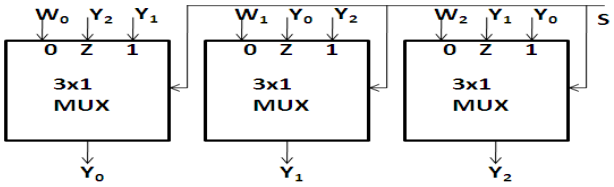


Figure 23(a). Block diagram of Barrel Shifter

TABLE XVI. FUNCTION TABLE FOR BARREL SHIFTER

S	Y0	Y1	Y2
0 (PARALLEL LOAD)	W0	W1	W2
Z	W2	W0	W1
(SHIFTS BY 1-BIT POSITION TO RIGHT)	W1	W2	W0
	W0	W1	W2
1	W1	W2	W0
(SHIFTS BY 2-BIT POSITION TO RIGHT)	W2	W0	W1
	W0	W1	W2

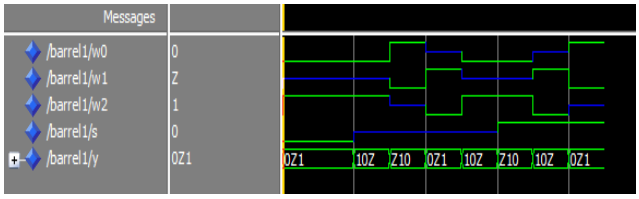


Figure 23(b). Simulation result of Barrel Shifter

IV. CONCLUSION

MVL provides means of increasing data processing capability per unit chip area. One of the main advantage of ternary logic is that it reduces the number of required computation steps. The number of digits required in a ternary family is $\log_3 2$ times less than that required in binary logic. It is assumed that ternary-logic elements can operate at a speed approaching that of the corresponding binary-logic elements. However, if the ternary and binary logic gates are used to take advantage of their respective merits, performance could be

significantly improved because ternary logic gates are a good candidate for decoding block since it requires less number of gates while binary logic gates are a good candidate for fast computation modules. Thus, ternary logic gate design technique combined with the conventional binary logic gate design technique also provides an excellent speed and power consumption characteristics in data path circuit such as full adder and multiplier. In this paper, design of combinational and arithmetic circuits is designed with minimum number of ternary multiplexers. VHDL simulator has been used to simulate MVL Systems which provide enough information to verify functionality and timing specifications. The simulation results with the code are furnished.

REFERENCES

[1] A.P.Dhande, R.C.Jaiswal and S.S.Dudam, "Ternary Logic Simulator Using VHDL", 4th International Conference: Sciences of Electronic, SETIT 2007-TUNISIA, March 25-29, pp.1-6, 2007.

[2] Sheng Lin, Yong-Bin Kim, and Fabrizio Lombardi, "CNTFET-Based Design of Ternary Logic Gates and Arithmetic Circuits", *IEEE Transactions on Nanotechnology*, Vol. PP, Issue.99, Nov.23, 2009, pp.1-1.

[3] S.L.Hurst, "Multivalued logic - Its status and its future", *IEEE Trans.On Computers,* vol. C-33, 1984, pp. 1160-1179.

[4] C. Rozon, On the Use of VHDL as a Multi-Valued Logic Simulator, 26th International Symposium on Multiple-Valued Logic (ISMVL '96), IEEE, 1996, pp.110-115.

[5] Raymond E.Miller," Switching Theory, Vol. I, John Wiley & Sons,1966, pp.8-9.

[6] M. Yoeli and G. Rosenfeld, "Logical Design of Ternary Switching Circuits", *IEEE Transactions on Electronic Computers*, Vol.EC-14, Issue.1, Feb.1965, pp.19-29.

[7] Marek Perkowski, Presentation on "Introduction to multivalued logic". Available as: http://web.cecs.pdx.edu/~mperkows/ temp/JULY/006.Introduction-to-MV-logic.ppt

[8] Sudhakar Yalamanchili, Presentation on "Basics of VHDL", Georgia Institute of Technology, 2006.

[9] D. Venkat Reddy, Ch. D. V. Paradesi Rao and E. G. Rajan, "Sequential Circuits In The Framework Of (2n+1)-ary Discrete Logic", *IJCSNS International Journal of Computer Science and Network Security*, Vol.8 No.7, July 2008, pp.175-181.

[10] A. P. Dhande and V. T. Ingole, "Design And Implementation Of 2 Bit Ternary ALU Slice", 3rd International Conference: Sciences of Electronic (SETIT 2005), *IEEE Transc.*, March 2005, pp.1-11.

[11] H.T. Mouftah, "A Study On The Implementation Of Three-valued Logic", University of Toronto, Toronto, Ontario, Canada, pp.123-126.

[12] Kenneth C. Smith, "Multiple-Valued Logic: A Tutorial and Appreciation", Survey & Tutorial Series, *IEEE Transc.* in computers,Vol.21, Issue.4, Apr.1988, pp.17-27.

[13] D. I. Porat, "Three-valued digital systems", *PROC. IEE Inst. Elect Eng,* Vol. 116, No. 6, JUNE 1969, pp.947-954.

[14] R. P. Hallworth and F. G. Heath, "Semiconductor Circuits For Ternary Logic", The Institution of Electrical Engineers, Monograph No. 482 E, *IEEE trans.*,VOL. 109, PART C.,Nov. 1961, pp.219-225.

[15] Richard F.Tinder, "Engineering Digital Design", Second Edition, Academic Presss,2000.

[16] Neil H.E.Weste,David Harris, Ayan Banerjee, "CMOS VLSI Design",A circuit and systems perspective, third Edition, pearon Education,2007.

[17] John F.Wakerly, "Digital Design, principles & practices, third edition, prentice Hall of India Private Limited (EEE),2005.

[18] Nazeih M.Botros, "HDL Programming VHDL and verilog",dreamtech press, 2006.

[19] Stephen Brown and Zvonko Vranesic, "Fundamentals of Digital Logic with Verilog Design", second edition, Tata McGraw-Hill Publishing Company Limited, New Delhi,2008.

A.Sathish Kumar received Bachelors in Electronics and Communication Engineering from Sona College of Technology, Salem, Tamilnadu in the year 2009 and currently pursuing Masters in VLSI Design at Amrita School of Engineering, Bangalore, India. He is a member of IETE.

A. Swetha Priya received Bachelors in Electronics and Instrumentation Engineering from Amrita School of Engineering, Bangalore in the year 2009 and currently pursuing Masters in VLSI Design at Amrita School of Engineering, Bangalore, India.

ANNEXURE

ANNEX I : CODE OF TERNARY PACKAGE

```
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 litzero (l:ternarylogic) return ternarylogic;
function litone (l:ternarylogic) return ternarylogic;
function littwo (l:ternarylogic) return ternarylogic;
function lit01 (l:ternarylogic) return ternarylogic;
function lit12 (l:ternarylogic) return ternarylogic;
function lit02 (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 ptnand (l: ternarylogic; r: ternarylogic) return ternarylogic;
function stor3 (l: ternarylogic; r: ternarylogic; u:ternarylogic) return ternarylogic;
function stand3 (l: ternarylogic; r: ternarylogic; u:ternarylogic) return ternarylogic;
function ntnand (l: ternarylogic; r: ternarylogic) return ternarylogic;
function stnor (l: ternarylogic; r: ternarylogic) return ternarylogic;
function ptnor (l: ternarylogic; r: ternarylogic) return ternarylogic;
function ntnor (l: ternarylogic; r: ternarylogic) return ternarylogic;
function txor (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;
type t_logic_table3 is array (ternarylogic, ternarylogic,ternarylogic) of ternarylogic;
constant stand_table : t_logic_table := (('0', '0', '0'),('0', 'Z', 'Z'),('0', 'Z', '1'));
constant stnand_table : t_logic_table :=(('1', '1', '1'),('1', 'Z', 'Z'),('1', 'Z', '0') );
constant ptnand_table : t_logic_table := (('1', '1', '1'),('1', '1', '1'),('1', '1', '0'));
constant ntnand_table : t_logic_table := (('1', '1', '1'),('1', '0', '0'),('1', '0', '0'));
constant stor_table : t_logic_table :=(('0', 'Z', '1'),('Z', 'Z', '1'),('1', '1', '1') );
constant stnor_table : t_logic_table :=(('1', 'Z', '0'),('Z', 'Z', '0'),('0', '0', '0'));
constant ptnor_table : t_logic_table :=(('1', '1', '0'),('1', '1', '0'),('0', '0', '0'));
constant stor3_table : t_logic_table3 :=((('0', 'Z', '1'),('Z', 'Z', '1'),('1', '1', '1')),(('Z',
'Z', '1'),('Z', 'Z', '1'),('1', '1', '1')),(('1', '1', '1'),('1', '1', '1'),('1', '1', '1')));
constant stand3_table : t_logic_table3 :=((('0', '0', '0'),('0', '0', '0'),('0', '0', '0')),(('0',
'0', '0'),('0', 'Z', 'Z'),('0', 'Z', 'Z')),(('0', '0', '0'),('0', 'Z', 'Z'),('0', 'Z', '1')));
constant ntnor_table : t_logic_table :=(('1', '0', '0'),('0', '0', '0'),('0', '0', '0'));
constant txor_table : t_logic_table :=(('0', 'Z', '1'),('Z', '1', '0'),('1', '0', 'Z'));
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');
constant litzero_table : t_logic_ld := ('1','0','0');
constant litone_table : t_logic_ld := ('0','1','0');
constant littwo_table : t_logic_ld := ('0','0','1');
```

```
constant lit01_table : t_logic_ld := ('1','1','0');
constant lit12_table : t_logic_ld := ('0','1','1');
constant lit02_table : t_logic_ld := ('1','0','1');

--lit zero
function litzero (l: ternarylogic)
return ternarylogic is
begin
return (litzero_table(l));
end litzero;
--lit one
function litone (l: ternarylogic)
return ternarylogic is
begin
return (litone_table(l));
end litone;
--lit two
function littwo (l: ternarylogic)
return ternarylogic is
begin
return (littwo_table(l));
end littwo;
--lit01
function lit01 (l: ternarylogic)
return ternarylogic is
begin
return (lit01_table(l));
end lit01;
--lit02
function lit02 (l: ternarylogic)
return ternarylogic is
begin
return (lit02_table(l));
end lit02;
--lit12
function lit12 (l: ternarylogic)
return ternarylogic is
begin
return (lit12_table(l));
end lit12;
--stand
function stand (l: ternarylogic; r: ternarylogic)
return ternarylogic is
begin
return (stand_table(l, r));
end stand;
--stor
function stor (l: ternarylogic; r: ternarylogic)
return ternarylogic is
begin
return (stor_table(l, r));
end stor;
--stnand
function stnand (l: ternarylogic; r: ternarylogic)
return ternarylogic is
begin
return (stnand_table(l, r));
end stnand;
--stor3
function stor3 (l: ternarylogic; r: ternarylogic; u: ternarylogic)
return ternarylogic is
begin
return (stor3_table(l, r, u));
end stor3;
--stand3
function stand3 (l: ternarylogic; r: ternarylogic; u: ternarylogic)
return ternarylogic is
begin
return (stand3_table(l, r, u));
end stand3;
--ptnand
function ptnand (l: ternarylogic; r: ternarylogic)
return ternarylogic is
begin
return (ptnand_table(l, r));
end ptnand;
--ntnand
function ntnand (l: ternarylogic; r: ternarylogic)
return ternarylogic is
begin
return (ntnand_table(l, r));
end ntnand;
--stnor
function stnor (l: ternarylogic; r: ternarylogic)
return ternarylogic is
begin
return (stnor_table(l, r));
end stnor;
--ptnor
function ptnor (l: ternarylogic; r: ternarylogic)
return ternarylogic is
begin
return (ptnor_table(l, r));
end ptnor;
--ntnor
function ntnor (l: ternarylogic; r: ternarylogic)
return ternarylogic is
begin
return (ntnor_table(l, r));
end ntnor;
--txor
function txor (l: ternarylogic; r: ternarylogic)
return ternarylogic is
begin
return (txor_table(l, r));
end txor;
--simple not
function stnot (l: ternarylogic)
return ternarylogic is
begin
return (stnot_table(l));
end stnot;
--positive not
function ptnot (l: ternarylogic)
return ternarylogic is
begin
return (ptnot_table(l));
end ptnot;
--ntnot
function ntnot (l: ternarylogic)
return ternarylogic is
begin
return (ntnot_table(l));
end ntnot;
END ternary_types;
```

ANNEX II : CODE FOR BASIC BUILDING GATES

CASE I : PROGRAM FOR UNARY OPERATORS

```
library ieee;
use ieee.std_logic_1164.all;
use work.ternary_types.ALL;
entity literals is
port(x:in ternarylogic;
x02,x0,cx2,cx1,x2,x1,x01,x12:inout ternarylogic);
end literals;
architecture dataflow of literals is
begin
x0<=ntnot(x); -- gives X^0 & X^0
cx2<=ptnot(x); -- gives X^1
cx1<=stnot(x); -- gives X^Z
x2<=ntnot(cx2); -- gives X^1
x1<=ntnor(x0,x2); -- gives X^Z
x01<=stor(x0,x1); -- gives X^0Z
x12<=stor(x2,x1); -- gives X^Z1
x02<=stor(x0,x2); -- gives X^01
end dataflow;
```

CASE II: PROGRAM FOR TERNARY INVERTERS

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.ternary_types.ALL;
ENTITY TI IS
PORT (I : IN ternarylogic; STO,NTO,PTO : OUT ternarylogic);
END TI;
ARCHITECTURE BEH OF TI IS
BEGIN
NTO <= ntnot(I);  -- Negative Ternary Inverter
PTO <= ptnot(I);  -- Positive Ternary Inverter
STO <= stnot(I);   -- Simple Ternary Inverter
END BEH;
```

CASE III: PROGRAM FOR BASIC TERNARY GATES

```
library ieee;
USE IEEE.std_logic_1164.ALL;
USE WORK.ternary_types.ALL;
ENTITY ANDALL IS
PORT (I1,I2 : IN ternarylogic;
a,b,c,d:INOUT ternarylogic;
o1,o2,o3,o4,o5,o6,o7,o8: OUT ternarylogic);
END ANDALL;
ARCHITECTURE BEH OF ANDALL IS
BEGIN
--ternary or
a <= stor(I1,I2);
o1 <= stnot(a); --simple nor
o2<= ptnot(a); --positive nor
o3 <= ntnot(a); --negative nor
--ternary nor
b <= stnor(I1,I2);
o4<= stnot(b);--simple or
-- ternary  and
c <= stand(I1,I2);
o5 <= stnot(c); --simple nand
o6 <= ptnot(c); --positive nand
o7 <= ntnot(c); --negative nand
--ternary nand
d<= stnand(I1,I2);
o8 <= stnot(d); --simple and
END BEH;
```

CASE IV: PROGRAM FOR MODULO-3 SUM OPERATORS

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.ternary_types.ALL;
ENTITY tequ1 IS
PORT (I0,I1 : IN ternarylogic;
y,y1,y2: OUT ternarylogic);
END tequ1;
ARCHITECTURE BEH OF tequ1 IS
signal O:ternarylogic;
BEGIN
O <= txor(I0,I1);y<=stnot(O);
y1<=ptnot(O);y2<=ntnot(O);
END BEH;
```

CASE V: PROGRAM FOR DECODER

```
library ieee;
use ieee.std_logic_1164.all;
use work.ternary_types.ALL;
entity decoder is
port(x:in ternarylogic; a,b,c:inout ternarylogic);
end decoder;
architecture archdecoder of decoder is
signal d:ternarylogic;
begin
a<=ntnot(x);  -- gives X^0
d<=ptnot(x);c<=ntnot(d);  -- gives X^1
b<=ntnor(a,c); -- gives X^2
end archdecoder;
```

CASE VI: PROGRAM FOR 3x1 MULTIPLEXER

```
library ieee;
use ieee.std_logic_1164.all;
USE WORK.ternary_types.ALL;
ENTITY mux3_1 IS
port(a,b,c:in ternarylogic;
s:in ternarylogic;
y:out ternarylogic);
end mux3_1;
architecture struct of mux3_1 is
signal s0,s1,s2,a1,b1,c1,y1:ternarylogic;
component decoder
PORT (x : in ternarylogic; a,b,c: inout ternarylogic);
END component;
component stand1
PORT (I0,I1 : IN ternarylogic; O : OUT ternarylogic);
END component;
component stor1
PORT (I0,I1 : IN ternarylogic; O : OUT ternarylogic);
END component;
begin
a01:decoder port map(s,s0,s1,s2);
a11:stand1 port map(a,s0,a1);
a21:stand1 port map(b,s1,b1);
a31:stand1 port map(c,s2,c1);
a41:stor1 port map(a1,b1,y1);
a51:stor1 port map(y1,c1,y);
end struct;
```

ANNEX III : CODE FOR COMBINATIONAL CIRCUIT MINIMIZATION USING 3X1 MUX

CASE I: PROGRAM FOR 9x1 MUX USING 3x1 MUX

```
library ieee;
use ieee.std_logic_1164.all;
USE WORK.ternary_types.ALL;
ENTITY mux9_1 IS
port(a0,a1,a2,a3,a4,a5,a6,a7,a8:in ternarylogic;
s0,s1:in ternarylogic; y:out ternarylogic);
end mux9_1;
architecture beh of mux9_1 is
signal y1,y2,y3:ternarylogic;
component mux3_1
port(a,b,c:in ternarylogic; s:in ternarylogic; y:out ternarylogic);
end component;
begin
a00:mux3_1 port map(a0,a1,a2,s1,y1);
a11:mux3_1 port map(a3,a4,a5,s1,y2);
a22:mux3_1 port map(a6,a7,a8,s1,y3);
a33:mux3_1 port map(y1,y2,y3,s0,y);
end beh;
```

CASE II: PROGRAM FOR HALF ADDER USING 3x1 MUX

```
library ieee;
use ieee.std_logic_1164.all;
USE WORK.ternary_types.ALL;
ENTITY ha_mux is
port(a,b:in ternarylogic; sum,cy:out ternarylogic);
end ha_mux;
architecture beh of ha_mux is
signal x,x1,y1,y2:ternarylogic;
component txor1
PORT (I0,I1 : IN ternarylogic; O : OUT ternarylogic);
END component;
component stand1
PORT (I0,I1 : IN ternarylogic; O : OUT ternarylogic);
END component;
component mux3_1
port(a,b,c:in ternarylogic; s:in ternarylogic; y:out ternarylogic);
end component;
begin
y1<=littwo(b);
z0:txor1 port map(a,b,x);
z12:stand1 port map('Z',y1,y2);
z11:stand1 port map('Z',b,x1);
z1:mux3_1 port map(b,x,x,a,sum);  --Sum
z2:mux3_1 port map('0',y2,x1,a,cy);  --Carry
end beh;
```

CASE III: PROGRAM FOR HALF SUBTRACTOR USING 3x1 MUX

```
library ieee;
use ieee.std_logic_1164.all;
USE WORK.ternary_types.ALL;
ENTITY hsub_mux is
port(a,b:in ternarylogic;
dif,bor:out ternarylogic);
end hsub_mux;
architecture beh of hsub_mux is
signal x,x2,x3,x1,y1,y2:ternarylogic;
component txor1
PORT (I0,I1 : IN ternarylogic; O : OUT ternarylogic);
END component;
component stand1
PORT (I0,I1 : IN ternarylogic; O : OUT ternarylogic);
END component;
component mux3_1
port(a,b,c:in ternarylogic; s:in ternarylogic; y:out ternarylogic);
end component;
begin
y1<=stnot(a);
z0:txor1 port map(y1,b,x);
y2<=stnot(x);
z1:mux3_1 port map(y2,y2,y2,a,dif); --Difference
x1<=stand('Z',b); x2<=littwo(b); x3<=stand('Z',x2);
z2:mux3_1 port map(x1,x3,'0',a,bor);  --Borrow
end beh;
```

CASE IV: PROGRAM FOR FULL ADDER USING 3x1 MUX

```
library ieee;
use ieee.std_logic_1164.all;
USE WORK.ternary_types.ALL;
ENTITY fadd_mux is
port(a,b,c:in ternarylogic;
sum,cy:out ternarylogic);
end fadd_mux;
architecture beh of fadd_mux is
signal x,x2,x1,x3,x4,x01,x11,sum1,sum2,sum3,cy1,cy2,cy3:ternarylogic;
component mux3_1
port(a,b,c:in ternarylogic; s:in ternarylogic; y:out ternarylogic);
end component;
begin
x<=txor('0',c); x01<=txor('1',c);  x11<=txor('Z',c);
z1:mux3_1 port map(x,x11,x01,b,sum1);
z2:mux3_1 port map(x11,x01,x,b,sum2);
z3:mux3_1 port map(x01,x,x11,b,sum3);
z03:mux3_1 port map(sum1,sum2,sum3,a,sum); --Sum
x1<=stand('Z',c); x2<=littwo(c); x3<=stand('Z',x2);
x4<=stor(c,'1');
z4:mux3_1 port map('0',x3,x1,b,cy1);
z5:mux3_1 port map(x3,x1,'Z',b,cy2);
z6:mux3_1 port map(x1,'Z',x4,b,cy3);
z7:mux3_1 port map(cy1,cy2,cy3,a,cy); --Carry
end beh;
```

CASE V: PROGRAM FOR FULL SUBTRACTOR USING 3x1 MUX

```
library ieee;
use ieee.std_logic_1164.all;
USE WORK.ternary_types.ALL;
ENTITY fsub_mux is
port(a,b,c:in ternarylogic;
diff,bor:out ternarylogic);
end fsub_mux;
architecture beh of fsub_mux is
signal
x0,x2,x1,x3,x4,x5,y0,y1,y2,y3,y4,bor1,bor2,bor3,diff1,diff2,diff3:ternarylogic;
component mux3_1
port(a,b,c:in ternarylogic; s:in ternarylogic; y:out ternarylogic);
end component;
begin
x0<=txor('1',c); x1<=stnot(x0); x2<=txor('0',c);
x3<=stnot(x2); x4<=txor('Z',c); x5<=stnot(x4);
z1:mux3_1 port map(x1,x3,x5,b,diff1);
z2:mux3_1 port map(x5,x1,x3,b,diff2);
z3:mux3_1 port map(x3,x5,x1,b,diff3);
z4:mux3_1 port map(diff1,diff2,diff3,a,diff); --Difference
y0<=stand('Z',c);  y1<=stor('Z',c); y3<=littwo(c); y4<=stand('Z',c);
z11:mux3_1 port map(y0,'Z',y1,b,bor1);
z12:mux3_1 port map(y4,y0,'Z',b,bor2);
z13:mux3_1 port map('0',y4,y0,b,bor3);
z14:mux3_1 port map(bor1,bor2,bor3,a,bor); --Barrow
end beh;
```

CASE VI: PROGRAM FOR 1-BIT MULTIPLIER USING 3x1 MUX

```
library ieee;
use ieee.std_logic_1164.all;
USE WORK.ternary_types.ALL;
ENTITY multiplier1bit_mux is
port(a,b:in ternarylogic;
pt,cy:out ternarylogic);
end multiplier1bit_mux;
architecture struct of multiplier1bit_mux is
signal x0,x2,x1,x3:ternarylogic;
component mux3_1
port(a,b,c:in ternarylogic;
s:in ternarylogic; y:out ternarylogic);
end component;
begin
x0<=txor('1',b); x1<=stnot(x0); x2<=littwo(b); x3<=stand(x2,'Z');
z1:mux3_1 port map('0',b,x1,a,pt);  --Product
z2:mux3_1 port map('0','0',x3,a,cy);  --Carry
end struct;
```

CASE VII: PROGRAM FOR 1-BIT COMPARATOR USING 3x1 MUX

```
library ieee;
use ieee.std_logic_1164.all;
USE WORK.ternary_types.ALL;
ENTITY comparator1bit_mux is
port(a,b,en:in ternarylogic; y:out ternarylogic; agtb,aeqb,altb:inout ternarylogic);
end comparator1bit_mux;
architecture beh of comparator1bit_mux is
signal x0,x2,x1,x3,x4:ternarylogic;
component mux3_1
port(a,b,c:in ternarylogic; s:in ternarylogic; y:out ternarylogic);
end component;
begin
x0<=littwo(b); x1<=litzero(b); x2<=litone(b); x3<=lit01(b); x4<=lit12(b);
z1:mux3_1 port map('0',x1,x3,a,agtb); -- A greater than B
z2:mux3_1 port map(x1,x2,x0,a,aeqb); --A equal to B
z3:mux3_1 port map(x4,x0,'0',a,altb); --A less than B
z4:mux3_1 port map(agtb,aeqb,altb,en,y); --selecting one function out of three
end beh;
```

CASE VIII: PROGRAM FOR 27x1 MUX

```
library ieee;
use ieee.std_logic_1164.all;
USE WORK.ternary_types.ALL;
ENTITY mux27x1 is
port(a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,a17,a18,a19,a20,a2
1,a22,a23,a24,a25,a26:in ternarylogic; s:in t_logic_vector(2 downto 0);
y:out ternarylogic);
end mux27x1;
architecture beh of mux27x1 is
begin
process(a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,a17,a18,a19,a2
0,a21,a22,a23,a24,a25,a26,s)
begin
case s is
when "000"=>y<=a0; when "00Z"=>y<=a1; when "001"=>y<=a2;
when "0Z0"=>y<=a3; when "0ZZ"=>y<=a4; when "0Z1"=>y<=a5;
when "010"=>y<=a6; when "01Z"=>y<=a7; when "011"=>y<=a8;
when "Z00"=>y<=a9; when "Z0Z"=>y<=a10; when "Z01"=>y<=a11;
when "ZZ0"=>y<=a12; when "ZZZ"=>y<=a13; when "ZZ1"=>y<=a14;
when "Z10"=>y<=a15; when "Z1Z"=>y<=a16; when "Z11"=>y<=a17;
when "100"=>y<=a18; when "10Z"=>y<=a19; when "101"=>y<=a20;
when "1Z0"=>y<=a21; when "1ZZ"=>y<=a22; when "1Z1"=>y<=a23;
when "110"=>y<=a24; when "11Z"=>y<=a25; when "111"=>y<=a26;
when others=>
end case;  end process;
```

```
end beh;
```

CASE IX: PROGRAM FOR 2 BIT COMPARATOR USING 27x1 MUX

```
library ieee;
use ieee.std_logic_1164.all;
USE WORK.ternary_types.ALL;
ENTITY comp2bit_mux is
port(b0,en:in ternarylogic;
a:in t_logic_vector(2 downto 0); agtb,aeqb,altb:inout ternarylogic;
y:out ternarylogic);
end comp2bit_mux;
architecture beh of comp2bit_mux is
signal y1,y2,y3,y4,y5:ternarylogic;
component mux27x1 is
port(a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,
a17,a18,a19,a20,a21,a22,a23,a24,a25,a26:in ternarylogic;
s:in t_logic_vector(2 downto 0); y:out ternarylogic);
end component;
component mux3_1
port(a,b,c:in ternarylogic; s:in ternarylogic; y:out ternarylogic);
end component;
begin
y1<=litzero(b0);y2<=littwo(b0);y3<=stnot(y2);y4<=lit12(b0);y5<=litone(b0);
z0:mux27x1 port map ('0','0','0',y1,'0','0',y3,'0','0','1','0','0','1',y1,'0','1',y3,
'0','1','1','0','1','1',y1,'1','1',y3,a,agtb);
z1:mux27x1 port map (y1,'0','0',y5,'0','0',y2,'0','0','0',y1,'0','0',y5,'0','0',y2,'0','0',
'0',y1,'0','0',y5,'0','0',y2,a,aeqb);
z2:mux27x1 port map (y4,'1','1',y2,'1','1','0','1','1','0',y4,'1','0',y2,'1','0',
'0','1','0','0',y4,'0','0',y2,'0','0','0',a,altb);
z3:mux3_1 port map(agtb,aeqb,altb,en,y);
end beh;
```

CASE X: PROGRAM FOR RIPPLE CARRY ADDER

```
library ieee;
use ieee.std_logic_1164.all;
USE WORK.ternary_types.ALL;
ENTITY rippleadder is
port(a0,a1,a2,b0,b1,b2,c:in ternarylogic; s0,s1,s2,c2:out ternarylogic);
end rippleadder;
architecture beh of rippleadder is
signal c0,c1:ternarylogic;
component fulladd
port(x,y,z:in ternarylogic; sum,cy:out ternarylogic);
end component;
begin
z0:fulladd port map(a0,b0,c,s0,c0);
z1:fulladd port map(a1,b1,c0,s1,c1);
z2:fulladd port map(a2,b2,c1,s2,c2);
end beh;
```

CASE XI: PROGRAM FOR CARRY SAVE ADDER

```
library ieee;
use ieee.std_logic_1164.all;
USE WORK.ternary_types.ALL;
ENTITY csa1 is
port(a,b:in t_logic_vector(1 downto 0); c:in ternarylogic;
p,q:in ternarylogic; s:out t_logic_vector(1 downto 0);
sum,cy:out ternarylogic);
end csa1;
architecture beh of csa1 is
signal s1,s2,s3,c1,c2,c3,c0:ternarylogic;
component fulladd
port(x,y,z:in ternarylogic; sum,cy:out ternarylogic);
end component;
component fulladd2bit
port(a,b:in t_logic_vector(1 downto 0); c:in ternarylogic;
s:out t_logic_vector(1 downto 0); c1:out ternarylogic);
end component;
component mux3_1
port(a,b,c:in ternarylogic; s:in ternarylogic; y:out ternarylogic);
end component;
begin
z0:fulladd2bit port map(a,b,c,s,c0);
u0:fulladd port map(p,q,'0',s1,c1);
u1:fulladd port map(p,q,'Z',s2,c2);
u2:fulladd port map(p,q,'1',s3,c3);
u3:mux3_1 port map(s1,s2,s3,c0,sum);
u4:mux3_1 port map(c1,c2,c3,c0,cy);
end beh;
```

--2-bit Fulladder

```
library ieee;
use ieee.std_logic_1164.all;
USE WORK.ternary_types.ALL;
ENTITY fulladd2bit is
port(a,b:in t_logic_vector(1 downto 0); c:in ternarylogic;
s:out t_logic_vector(1 downto 0); c1:out ternarylogic);
end fulladd2bit;
architecture beh of fulladd2bit is
signal c0:ternarylogic;
component fulladd
port(x,y,z:in ternarylogic; sum,cy:out ternarylogic);
end component;
begin
u0:fulladd port map(a(0),b(0),c,s(0),c0);
u1:fulladd port map(a(1),b(1),c0,s(1),c1);
end beh;
```

CASE XII: PROGRAM FOR 1-BIT & 2-BIT POSITION SHIFTER

```
library ieee;
use ieee.std_logic_1164.all;
use work.ternary_types.ALL;
entity shifter12 is
port(w0,w1,w2,s:in ternarylogic; y0,y1,y2:inout ternarylogic);
end shifter12;
architecture beh of shifter12 is
component mux3_1
port(a,b,c:in ternarylogic; s:in ternarylogic; y:out ternarylogic);
end component;
begin
z0:mux3_1 port map(w0,'0','0',s,y0);
z1:mux3_1 port map(w1,w0,'0',s,y1);
z2:mux3_1 port map(w2,w1,w0,s,y2);
end beh;
```

CASE X: PROGRAM FOR BARREL SHIFTER CIRCUIT

```
library ieee;
use ieee.std_logic_1164.all;
use work.ternary_types.ALL;
entity barrelshifter is
port(w0,w1,w2,s:in ternarylogic; y:inout t_logic_vector(0 to 2));
end barrelshifter;
architecture beh of barrelshifter is
type i is range 0 to 2;
component mux3_1
port(a,b,c:in ternarylogic; s:in ternarylogic; y:out ternarylogic);
end component;
begin
if(s='0') then
z0:mux3_1 port map(w0,w2,w1,'0',y(0));
z1:mux3_1 port map(w1,w0,w2,'0',y(1));
z2:mux3_1 port map(w2,w1,w0,'0',y(2));
elsif(s='Z')then
begin
i<=0;
lop1:  for i in 0 to 2 loop  begin
if(i=0) then
z10:mux3_1 port map(w0,w2,w1,'Z',y(0));
z11:mux3_1 port map(w1,w0,w2,'Z',y(1));
z12:mux3_1 port map(w2,w1,w0,'Z',y(2));
i:=i+1;
else y<=ror(y);   i:=i+1;
end if;  end loop;
else   i<=0;
lop2:   for i in 0 to 2 loop
  begin
if(i=0) then
z110:mux3_1 port map(w0,w2,w1,'1',y(0));
z111:mux3_1 port map(w1,w0,w2,'1',y(1));
z112:mux3_1 port map(w2,w1,w0,'1',y(2));
```

```
i:=i+1;
else
y<=rol(y);  i:=i+1;
end if;
end loop;  end if;
```

```
end beh;
```

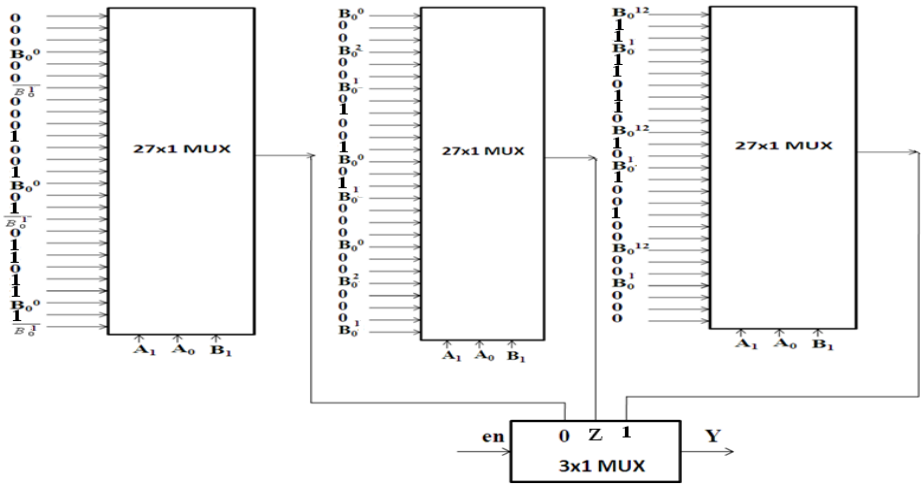


Figure 19(a). Block diagram of 2-bit comparator

TABLE III. OPERATING TABLE OF BASIC GATES

A	B	TAND	STNAND	PTNAND	NTNAND	TOR	STNOR	PTNOR	NTNOR
0	0	0	1	1	1	0	1	1	1
0	Z	0	1	1	1	Z	Z	1	0
0	1	0	1	1	1	1	0	0	0
Z	0	0	1	1	1	Z	Z	1	0
Z	Z	Z	Z	1	0	Z	Z	1	0
Z	1	Z	Z	1	0	1	0	0	0
1	0	0	1	1	1	1	0	0	0
1	Z	Z	Z	1	0	1	0	0	0
1	1	1	0	0	0	1	0	0	0